Table of Contents
Table of Contents ........................................................................................................... ii
Papers ............................................................................................................................. ii
Preface ............................................................................................................................ v
Thank you ..................................................................................................................... vi
Copyright ..................................................................................................................... vii

Papers

Writers Workshop A – Business

Workshop Leader: Claudius Link
A1 Creating Customer Value Propositions for Technology Products
   Michael Weiss
A3 Social Entrepreneurship Patterns
   Eri Shimomukai, Takashi Iba
A4 Audit of Knowledge flows and Critical business processes
   Elissaveta Gourova, Kostadinka Toteva, Yanka Todorova
A5 The Adoption of Software Patterns in the Dutch Software Industry
   Gerard Nijboer, Jaap Kabbedijk, Slinger Jansen
A6 A Case Study of the Variability Consequences of the CQRS Pattern in Online Business Software
   Jaap Kabbedijk, Slinger Jansen, Sjaak Brinkkemper

Writers Workshop B – Engineering

Workshop Leader: Veli-Pekka Eloranta
B1 Ultra-Fast and Lots of Choices - A Pattern Approach to NoSQL
   Tim Wellhausen, Highly Scalable
B2 Human Machine Interface Patterns For Distributed Machine Control Systems
   Veli-Pekka Eloranta, Marko Leppänen
B3 Patterns for PLE
   Dietmar Schütz
B4 A Pattern Language for Service-Based Platform Integration and Adaptation
   Ioanna Lytra, Stefan Sobernig, Huy Tran, Uwe Zdun
B5 Applying Patterns to Model-Driven Development of Automation Systems: An Industrial Case Study
   Christopher Preschern, Nermin Kajtazovic, Christian Kreiner
B6 Family of Bridges pattern
   Vallidevi Krishnamurthy, Chitra Babu
Writers Workshop C – Interaction

Workshop Leader: Klaus Marquardt

C2a Lessons from EuroPLoP: About Workshop Moderation
   Allan Kelly

C2b Lessons from EuroPLoP: Two patterns for Shepherding
   Allan Kelly

C3 Presentation Patterns: A Pattern Language for Creative Presentations
   Takashi Iba, Aya Matsumoto, Kaori Harasawa

C4 A Costume Pattern Language for Costumes in Films
   David Schumm, Johanna Barzen, Frank Leymann, Lutz Ellrich

C5 A Proven Pattern Integration Process
   Damien Gouteux, Brahim Hamid, Christian Percebois

C6 A Real-Time Design Pattern for Advanced Driver Assistance Systems
   Hela Marouane, Achraf Makni, Rafik Bouaziz, Claude Duvallet, Bruno Sadeg

Writers Workshop D – Education

Workshop Leader: Christian Köppe

D1 Patterns and Paths - Introducing the path metaphor
   Christian Kohls

D3 A Pattern Language for Teaching in a Foreign Language - Part 1
   Christian Köppe, Mariëlle Nijsten

D4 Assessment Patterns - A pattern language for assessing students’ work in computer science courses
   Uwe van Heesch, Richard van den Ham, Pieter van den Hombergh

D5 A Pattern for Learning Phraseology of a Foreign Language
   Karen Velázquez

D6 A Pattern Language for Designing Pattern Dialogue Workshops
   Takashi Iba

D7 Game Language
   Anne Hoffmann

Focus Groups

FG1 Crossing The Jungle - A Retrospective of Web 2.0
   Andreas Rüping

FG2 Evaluating the Consequences of Applying Architectural Patterns
   Jaap Kabbedijk
Preface

Welcome to the proceedings of the 17th European conference on Pattern Languages of Programs (EuroPLoP). It was held in the Bavarian Kloster Irsee, Germany from 11-15 July 2012. Pattern authors and pattern enthusiasts have met to discuss, explore and apply patterns.

This volume contains the post-conference-versions of the patterns reviewed in the Writers Workshops at the conference. There were four parallel workshops this year, each with up to seven papers. The domains covered at the conference have spread out from software development to management, education, movie production and human interaction.

Software development has always been the heart of EuroPLoP. However, over the past years the domains covered have spread out and gained a wide variety. While Eri Shimomukai et al. have documented patterns to improve social live (A3), David Schumm et al. demonstrate that patterns can also be used to describe and design costumes for film production. Education has become another main stream for pattern mining. For example Karen Velázquez uses the pattern approach to promote phraseological units for teaching a foreign language.

EuroPLoP is a conference where academics and practitioners meet and bring together the best of their worlds. To support the authors who might be experts in their domain but have little pattern experience, a special way of coaching takes place: After submitting a paper, every author gets assigned to an experienced pattern author. This so-called shepherd gives the author tailor-made, constructive feedback for their paper. This happens in at least three iterations. Often the shepherds are domain experts themselves so the paper can profit from the coaching even in two ways.

After the shepherding process the papers are reviewed by the program committee and in this quality gate it is judged whether the paper is accepted for the conference or not. The goal of the shepherding process is to bring the paper to a state of maturity that can be discussed in a workshop at the conference. This workshop consists of mostly five to seven participants. All authors are participants themselves and bring their feedback to the other authors of their workshop. During the session in which a paper is discussed, the author of that paper remains silent and concentrates on taking notes. This prevents defensive discussions and has proven to provide a maximum of constructive, thorough and useful feedback. The comments given through this review process often reach a depth that is hardly seen in other formats.

After the conference the author does another iteration to include the workshop comments. These versions are what you can read in this volume. In addition to the workshops there have also been pre-announced focus groups and an Open Space slot to allow for short notice discussions. Outside of scheduled slots there is also plenty of opportunity to have vivid discussions in the garden, at the bar or while walking around in the village of Irsee.

EuroPLoP is a writers’ conference. So judging a paper is not the most important thing. We rather focus on feedback, progress and learning. This is reflected by the fact that fostering creativity and fun get a lot of space during the conference. Before every workshop session there is a slot where games are played. We also had the “paint and create” session where all participants cooperate in creating a piece of art work and – as a counterpart to all the mental activities – there are also slots for swimming, soccer
or sauna. In 2012 the regular activities were complemented by an outdoor group event: solving the riddles of a ropes garden in the forest around Kloster Irsee.

So if you are interested in patterns, go ahead and submit a paper for the next EuroPLoP! Even if you do not have an idea for a paper yet, there is a lot to learn about patterns and the domains they cover. Find more details at www.europlop.net.

There are many participants who return year after year and a considerable number of first timers every year.

**Thank you**

While there are two chairs assigned to conduct the conference every year, there are a lot of people without whom EuroPLoP would not be possible. First of all we would like to thank everyone who has submitted a paper and those who attended the conference!

Our special thanks go to all the shepherds of EuroPLoP 2012: Aliaksandr Birukou, Allan Kelly, Andreas Fießer, Andreas Rueping, Bettina Biel, Brahim Hamid, Christian Kohls, Christian Köppe, Christoph Hannebauer, Claudius Link, Cyrille Martraire, Dietmar Schuetz, Dirk Schnelle-Walka Eduardo Fernandez, Ernst Oberortner, Georgina Holden, Hans Wegener, Klaus Marquardt, Linda Rising, Michael Weiss, Neil Harrison, Paris Avgeriou, Peter Sommerlad, Stefan Sobernig, Tim Wellhausen, Uwe Zdun, Uwe van Heesch, Veli-Pekka Eloranta, and Yishay Mor.

Your time and effort is greatly appreciated!

When deciding on the submission we got valuable feedback from the program committee: Aliaksandr Birukou, Allan Kelly, Andreas Rüping, Claudius Link, Klaus Marquardt, Neil Harrison, Paris Avgeriou, Tim Wellhausen, Uwe van Heesch, Uwe Zdun, and Veli-Pekka Eloranta.

Uwe van Heesch was a great help as Focus Group Chair and organized the Open Space.

We would also like to thank Michael Weiss for creating, maintaining and refining the submission system for many years.

An important part of the unique character of our conference are the more playful activities performed by our conference Querdenker George Platts. Our thanks also go to Ralf Trinkwalder and Thomas Weinmüller for introducing us to the secrets of balancing through the woods! We would also like to thank the staff of Kloster Irsee for providing us with a peaceful environment free of all the concerns of everyday live and full of wonderful food!

EuroPLoP is an event run by Hillside Europe e.V. We would like to thank the board members and the members of the EuroPLoP Support Committee for their support in organizational and long-term matters.

May 2013

Andreas Fiesser, Program Chair  
Christian Kohls, Conference Chair
Copyright

All copyright is retained by individual authors. All authors have granted Hillside Europe limited rights to publish papers as part of the conference proceedings.

EuroPLoP 2012 is organized by Hillside Europe e.V., Munich
www.hillside-europe.net

Hillside Europe

www.europlop.net
Creating Customer Value Propositions for Technology Products

Michael Weiss
Carleton University, Ottawa, Canada
weiss@sce.carleton.ca
Version 0.8

“In the middle of difficulty lies opportunity.” (Einstein)

1 Introduction

The development of a technology product requires a great number of decisions and involves many different stakeholders. Decisions include the choice of technology, architecture, release schedule, and staffing of the project team. Stakeholders include end users, customers, businesses, developers, domain experts, component suppliers and service providers.

For each of the stakeholders, the success of a technology product will, ultimately, be judged by the value it creates for them. Value can be thought of in terms of benefits. Examples of benefits obtained from using a technology product are reduced cost, faster time to market, meeting needs, ease of use, return on investment (ROI), and satisfaction (Coplien, 2010).

The customer is a particularly important stakeholder. He or she is the one who pays for the technology product. The goal of this paper is to articulate patterns for the creation of value propositions for customers and end users. The paper is written from the perspective of a technology company.

Technology companies are often founded by developers, who are prone to look at products from a technology, not a customer perspective.

A value proposition is the reason why customers buy a product or service from one company over another. It describes how a company delivers value, ie, a set of benefits, to its customers. It also describes how the company is better at delivering the value than its competitors. Value propositions are about how a company differentiates itself from other companies.

Value is what customers get in exchange for buying a product. For example, when someone buys a cell phone, they may do so because they want to be able to receive calls from wherever they happen to be. A value proposition is at the core of a business model. Other components of a business model include customers, channels, and revenue, on the one hand, and activities, partners, and costs, on the other. Good starting points for reading more about value, value propositions, and business models are Osterwalder & Pigneur (2010), Anderson et al. (2007), and Meyer (2007).

1 Licensed under a Creative Commons BY-SA license.
2 Patterns

To define your customer value proposition, ask yourself these questions:

- What customer problem are you solving?
- What frustrations do customers experience with current solutions? Are they able to articulate their needs?
- How are customers solving their problem today? What is it that customers actually want to achieve?
- Are there better ways of solving the customer’s problem? Do you have the required background and skills to deliver?
- How is your solution different from other solutions in the market?

When you define a value proposition, you should start by putting yourself in the customer’s shoes. Do not assume that what excites you about a solution (eg that your application runs on Android as well as iOS) will also excite your customers. Instead, try to understand what problems they face, and how you can use your skills and technical capabilities to solve them. You may need to separate between the end user of your technology (who uses it on a daily basis) and the customer (who pays for the technology).

A good source of value propositions are frustrations that customers currently experience. These frustrations lead you to latent needs that the customers may have, but are unable to articulate. Customers have either gotten used to workarounds using current solutions, or are simply unaware of which alternatives are technically feasible. Of course, you also want to understand what makes a product more enjoyable. Thus, you should also look for ways to please the customer by exceeding their expectations.

Next, you need to understand what current solutions the customer can access to solve their problem. To that end, you want to identify the job to be done, ie what is it that the customer actually wants to achieve. If you only focus on products that directly compete with your solution (eg your product is a car, and you are just comparing it to other cars), you may fail to recognize the customer’s underlying need (eg they may be looking for a means of transportation, which increases the range of potential solutions to include trains, airplanes, boats, or running shoes, for that matter).

What you bring to the table as the founder of a technology company are your background, skills, and the people you know. You are looking for opportunities to match your background and skills to the customer’s needs. These enable you to imagine solutions that customers cannot conceive; their experience is limited to products that exist in the market. They may not be able to imagine solutions that are within your reach. In other words, you are a peddler of possibilities. To deliver the solution, you often need to work with partners, as customers prefer a whole product from one source (Kelly, 2012).

However, it is not enough merely to solve the problem as well as other solutions. Your solution must be better (ie cheaper, faster, easier to use, etc.) in some dimensions. Look for points of difference that set you apart from your competition. In fact, if you are doing this well, what you want to emphasize are the points of difference where you demonstrate an intimate understanding of your customer. You can do this through a resonating focus on just the dimensions that matter most. This also means that you might end with multiple value propositions, one for each of your customer segments.

---

2 Having access to partners is perhaps not as obvious as the founder’s background and skills, but key to being able to deliver your product (Read et al., 2010).
The audience for these patterns include entrepreneurs in the technology domain, as well as developers of technology products in established companies who need to interact with customers. They were motivated by the author’s role as a mentor to startups spun out from a master’s program on technology entrepreneurship. Having observed the same conceptual problems with what defining a value proposition in pitch after pitch, the author hopes that these patterns will help guide new entrepreneurs better articulate what value their products or services create for their customers.

A map of the patterns showing their relationships is shown in Figure 1. Links between patterns X and Y should be interpreted as “after pattern X you may also use pattern Y”. Patterns in bold face are described in this paper. The thumbnails of the other patterns can be found in the Appendix. The figure also shows which questions are linked to which patterns.

Figure 1: Patterns for creating customer value propositions
2.1 In the Customer's Shoes

“Consider what your customers will look, smell, touch and feel - from the start to end of your experience.” (Lim, 2011)

**Context**
Any business model starts with a value proposition. The first thing you need to do is to put yourself in the customer's shoes.

**Problem**
What customer problem are you solving?

**Forces**
Customers face problems in their daily lives or businesses.

Customers may not be able to articulate the problem they have.

The customer's imagination of what solutions can be provided to their problems is limited by what they know.

Asking customers about their needs will lead to incremental improvements, not new ways of solving their problems.

Customers are not experts on ways to solving their problem. Don't look to them to tell you how to solve the problem.

Therefore,

**Solution**
Try to understand what problems customers face, and how you can use your skills and technical capabilities to solve them.

In order to understand what problem the customer faces, you need to take the customer's perspective. Your product must address needs the customer has. Customer needs come in two types: needs that the customer can articulate, or perceived needs, and needs that are difficult to articulate, or latent needs. Surveys and focus groups used in traditional market research can only tell you about perceived needs. To learn about your customers' latent needs, you need to observe customers as they go about achieving their tasks.

As customers and end users are often different people, you will need to conduct this analysis for both groups. You should have an answer to what problem you are solving for the people who pay for your technology (customers), and how you are helping the people who use your solution on a daily basis (end users). While your end user may define their need in terms of specific features that your solution should have, your customer (someone who is more senior
than your end user) may be mainly interested in how much money they will be able to save if their organization uses the solution.

**Consequences**

You may be surprised by the needs customers really have.

Learn to speak the language of your customers. This is essential for gathering requirements and writing documentation.

You may need to help your customers discover latent needs.

Time spent upfront on gaining a good understanding of your customer's needs pays off when you develop your solution.

**Known uses**

One of the companies the author deals with sells software for visualizing complex data. It solves the problem of small companies that collect significant amounts of data (eg through customer surveys), but have no good ways of making sense of it, and that do not have access to expensive Business Intelligence solutions. They offer the visualization capability as a simple-to-use service.

**Related patterns**

This pattern complements the marketing patterns described in Kelly (2012), especially CUSTOMER UNDERSTANDING. The unique aspect of this pattern is on needs that customers cannot articulate.

**Sources**

Meyer (2007) and the author's experience.

**Perceived and latent needs**

An example of a perceived need is a user looking for a faster portable scanner or one with greater memory capacity. An example of a latent need is that users really want to limit the number of gadgets they have to carry with them.

Currently, most portable gadgets have a single purpose. So, an industrial designer may need to take a potpourri of gadgets wherever he goes, including a digital camera for taking photos, a voice recorder for conducting interviews or sampling sounds, a portable scanner to scan photos and articles, a sketchbook for capturing ideas when inspiration strikes, and a collection of pencils of different strength.

I happened to sit next to a well-known designer once at an event, when he emptied his bag on the table to make this very point. The designer's latent need is: there are too many gadgets to carry, but if he leaves any one of them at home, it may be the one he needs most on a given day. So, he has learned to live with this constraint; he is not content, but he lacks a viable alternative (Weiss, 2012).
2.2 Job to be Done

"The fact that you’re 18 to 35 years old with a college degree does not cause you to buy a product," Christensen says. "It may be correlated with the decision, but it doesn’t cause it. We developed this idea because we wanted to understand what causes us to buy a product, not what’s correlated with it. We realized that the causal mechanism behind a purchase is, ‘Oh, I’ve got a job to be done.’" (Nobel, 2011)

Context You put yourself in the customer’s shoes to understand what problem you are solving. Now, you need to understand what current solutions the customer can access today to solve their problem, and what it is they are actually trying to achieve.

Problem What task are customers trying to achieve with your product?

Forces Customers are already solving their problems, although it may be in a less efficient way than using your product. You need to get to the bottom of what task your customers are trying to achieve, and how you can improve on the current solutions they use.

Many technology companies limit their attention to product features, and with that, to products that directly compete with their solution in terms of those features (eg other cars). By doing so, they fail to recognize the customer’s underlying need (eg they may be looking for any kind of means of transportation).

Therefore,

Solution Identify the job to be done, ie what job they would “hire” your solution for, and the alternative solutions available to them.

To identify the job to be done observe your customers as they do their work. You will see them encounter problems (also known as “breakdowns” in the user-centered design literature, because they interrupt the customer’s normal flow of activity), and understand how they help themselves when they face them. Doing so will make you aware of the alternative solutions available to customers. When those solutions are suboptimal, this creates an opportunity for you.

Additionally, the Internet is an excellent resource for finding information about competing solutions, not only in terms of their features, but in terms of customer feedback and the frustrations
customers experience using those competing solutions.

**Consequences**
You understand how customers currently achieve their tasks.
You do not limit your attention to products that directly compete with your solution, but are aware of all competing solutions.

**Known uses**
Apple's products are designed with a customer's job in mind. When Apple introduced the iPod, there were existing products that customers were using to listen to music. However, many of these devices were difficult to use and it was hard to obtain music for them. Apple recognized the job-to-be done as customers looking for a convenient way to entertain themselves (listen to music). The competition for the iPod was not other MP3 players, but the whole distribution model of how customers would get to the music.

The evolution of e-readers is another good example (Adner, 2012). One of the first e-readers was introduced by Sony. The PRS-500 excelled in terms of hardware features: it was the first e-reader to use e-ink technology which dramatically improved readability and reduced battery consumption, problems faced by previous entrants to the e-reader market. However, what Sony failed to address was the real job customers wanted to achieve.

Customers did not only want a portable device for reading books, but also needed access to a selection of content comparable to existing bookstores, at a competitive price. The job to be done could be described as “carry a library of books with you and conveniently acquire new content”. The solution to this problem was later provided by Amazon through its Kindle and associated service for downloading books directly from the Kindle device.

**Related patterns**
Some users will also come up with their own solutions. Such **user innovators** are a rich source of product ideas.

**Sources**

**Alternative solutions**

Let's apply the lessons from this pattern to the scanner example. If your product is a portable scanner, you might just be comparing it to other portable scanners on the market. However, your real competition may be far broader than originally conceived, but so are your solutions. A new solution to a problem that the customer faces may involve another type of technology or an alternative approach.

Solutions competing with a portable scanner include copiers (if one is nearby), the user's memory (often unreliable), pen and paper (slow and tedious), as well as a camera-equipped smartphone (a very viable alternative, see below). Not all of these solutions are necessarily good solutions. However, once a customer has adopted a solution, it can create a “lock in” effect. This means that your solution needs to be sufficiently compelling for customers to switch to it (Weiss, 2012).
Acknowledgements

Thank you to my shepherd, Hans Wegener, for his constructive suggestions that made me rethink what I wrote, and his shepherding-by-skype style. I also want to thank my writers' workshop at EuroPLOP 2012. I received many good comments that helped me further improve my patterns.

In formatting these patterns I owe a tremendous amount to the format Allan Kelly has used in his own papers, which I tried to emulate.

Appendix – Pattern thumbnails

Here are short forms of the patterns not described in this paper.

<table>
<thead>
<tr>
<th>LATENT NEEDS</th>
<th>Often, customers cannot articulate their needs. Therefore, look for frustrations that customers experience.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEDDLER OF POSSIBILITIES</td>
<td>Your background and skills enable you to imagine solutions that users cannot conceive.</td>
</tr>
<tr>
<td>POINTS OF DIFFERENCE</td>
<td>Your solution must be better than alternatives. Therefore, look for points of difference that set you apart.</td>
</tr>
<tr>
<td>RESONATING FOCUS</td>
<td>Resonate with your customer’s needs by emphasizing the points of difference that matter most to your customer.</td>
</tr>
<tr>
<td>USER INNOVATORS</td>
<td>Look for customers who solve their problems themselves. This makes them a rich source of new product ideas.</td>
</tr>
<tr>
<td>WHOLE PRODUCT</td>
<td>Bundle your product with additional products or services that are required for customers to realize its value (Kelly, 2012).</td>
</tr>
</tbody>
</table>

References

I tried to limit the number of references, but the ones below are needed to give proper attribution. Key references are highlighted with a (*).


Coplien, J. (2010), Lean Architecture, Addison Wesley.


Nobel, C. (2011), Clay Christensen's milkshake marketing, hbswk.hbs.edu/item/6496.html


Photo credits

Shoes, taken by Ömer Ünlü and shared under a CC-BY license, www.flickr.com/photos/55293400@N07/5811660807

Car, taken by Nouhailler and shared under a CC-BY SA license, www.flickr.com/photos/patrick_nouhailler/5507110001

Airplane, taken by viZZZual.com and shared under a CC-BY license, www.flickr.com/photos/vizzzual-dot-com/2274726982

Train, taken by Bright Meadow and shared under a CC-BY SA license, www.flickr.com/photos/brightmeadow/56844105

Thinker, taken by deSeneste.dk and shared under a CC-BY license, www.flickr.com/photos/dseneste/5912382808
Abstract

Social Entrepreneurship is believed to be wrapped in mystery due to the reason that patterns occurring in this context consecutively have never been written in languages. As a result, it is difficult to teach tips for making social changes. However, pattern language, which is a method for sharing tacit knowledge for designing, can act as a medium showing the implicit knowledge of social entrepreneurs. This paper presents a pattern language, which we named “Change Making Patterns.” The objective of these patterns is to encourage more individuals to take own actions in making a better world with less social problems. Furthermore, this paper presents five essential patterns for the educational programs.

1. Introduction

The field of social entrepreneurship is growing rapidly and attracting attention from many different sectors. People are attracted to social entrepreneurs such as the Nobel Peace Prize winner, Muhammad Yunus, and business entrepreneurs like Steve Jobs, who are extraordinary people came up with great ideas and created new products and services that dramatically improved people lives. The reason behind the popularity of social entrepreneurship is that “there’s something inherently interesting and appealing about entrepreneurs and the stories of why and how they do what they do” [1].

In the meantime, not only certain talented people such as “social entrepreneurs” need to tackle variety of problems in society, but also each individual is required to embody “social entrepreneurship” in order to make a better world. There are, however, few opportunities and tools which help people to learn how to solve social issues. The contemporary writers in management and business such as Jean Baptiste Say, Joseph Schumpeter, and Peter Drucker have presented a wide range of theories of entrepreneurship based on observing entrepreneurs’ behaviors and their minds [2]. It means that they found some sort of patterns which appeared repeatedly in the same contexts. Because social entrepreneurs share a lot of characteristics [2], we assume that there should be certain patterns which are shared by a variety of people in different fields in order to solve problems in society. Our goal is simply to find and write the patterns for making changes on social issues, and provide them as “tips”. If we can achieve this goal, then people who strive to tackle the certain problems can find the solution and take actions. Consequently, the
number of social issues would be decreased in the future.

2. Issues: Difficulty in Replication of Social Entrepreneurship

Although the concept of “social entrepreneurship” is gaining the popularity, its definition is still vague. It means different things to different people. Many believe that it is exclusively related to non-profit organizations starting for-profit or earned-income ventures. Some use it to describe to anyone who starts a non-profit organization. However, we would like to apply the definition which David Bornstein emphasizes in his book, and define the word for this patterns [3]. Social entrepreneurs are creative individuals who question the status quo, exploit new opportunities and ideas, and refuse to give up in order to solve important social problems. Therefore, social entrepreneurship is defined by all the knowledge and ways of thinking that “social entrepreneurs” possess or exercise.

As we mentioned in the introduction, every single person is now required to take social entrepreneurial actions due to numerous social problems around the world. However, it is still difficult to share action guidelines applying to a variety of people in different situations. Those people who defined the word “social entrepreneurship,” such as David Bornstein, discovered their common characteristics throughout careful observations and analysis on them. Clearly, there are a number of solutions and rules which have appeared consecutively in the same contexts. The reason why sharing tips remains difficult is that those solutions and rules rarely have been expressed in some sort of languages. As a result, even if we can recognize the mature social entrepreneurs and their impacts on society, it is difficult to replicate or scale out social entrepreneurship because there is no media or material consisting of guideline leading people to take actions.

3. Approach: The Need for a Pattern Language

In order to overcome the difficulty in replication of social entrepreneurship, it is necessary to verbalize problems which social entrepreneurs have been consecutively facing with in certain contexts, and their tacit knowledge for solving them. Now, we would like to apply a pattern language which is a language for writing “tips” proposed by Christopher Alexander, an architect, to verbalize them. This paper presents social entrepreneurship patterns that is a pattern language for making changes on a variety of social issues around the world. The objective of this pattern language is to clarify how social entrepreneurs discover problems in society, think about solutions, and solve them. To achieve this objective, we firstly need to understand social entrepreneurial mindset and observe their behaviors in certain contexts. Then, we have to mine their motivation and knowledge on change-makings which they have in common. Through oral history interviews, we collected “seeds” of this pattern languages.

3.1 Significance of Pattern Language

A pattern language is a method which Christopher Alexander, an architect, proposed in order to describe the tacit knowledge of local acts and create global order [4][5]. A pattern language is composed by a number of elements called patterns. Each pattern is written in a set format which
consists of a name, a context, a problem which occurs consecutively in a certain context, a force as a premise that causes the problem, a solution, and a action of the pattern. Even though social entrepreneurs have a lot of different types of tacit knowledge, they repeat in the certain contexts, so we can find patterns. Above all, social entrepreneurs always grasp problems in society, exploit solutions, and take creative actions toward the better world, so the format of a pattern language perfectly suits verbalizing social entrepreneurs’ tips.

3.2 Interviews for Social Entrepreneurship Patterns

Interviews, which we implemented, consist of two parts: an oral history interview and a problem centered interview, a type of semi-structured interview proposed by Witzel [8]. In the first part, interviewees can reflect on their experiences without any bias, so interviewers are able to understand their recognition structure on their actions toward social problems [9], because oral history interviews are that interviewers do not interrupt and summarize what the interviewees say. In the second part, interviewers focus on a certain problem that an interviewer have faced, and a solution that he or she took. We also tried to connect those problems, solutions and actions during interviews in order to make pattern writing easier and more precise.

Additionally, interviewees speak with their vivid and living words on oral history interviews. This reflection on experiences with vivid expressions is meaningful to a pattern language. Christopher Alexander, the first proponent of a pattern language, emphasizes that a pattern language contributes for constructing the living and growing whole [10].

4. Methods: The Process of Making the Patterns

The process of mining and writing social entrepreneurship patterns is based on oral history interviews as this paper mentioned in Section 3. It is important to show how we collect the tacit knowledge and write patterns in living words. The process of making this pattern language is clarified below and in Figure 1.

1. Make an appointment. It is important to briefly inform the purposes of the interview. We have implemented interviews to ten social entrepreneurs up to now.
2. Research on the interviewees in order not to ask basic information and consume time. Additionally, it is helpful for the next step, which is preparation of questions.
3. Draw up a questionnaire which contains questions that the interviewers want to ask the interviewees. Questions should not be specific which is like closed questions, because it disturbs interviewees to recall their experiences [7]. Preferably, a questionnaire has topics related to their backgrounds and childhood in order to understand their motivation of their activities which they have done. Send a questionnaire more than a day before the interviews in order to make sure that the interviewees have chances to read through it and brainstorm the interview.
4. Set up a comfortable environment for an interview. Preferably, an office which the
interviewer often uses can be taken place. Prepare a printed questionnaire which is the same one which you have sent to the interviewee before the interview. Also, it is better to bring a recorder, a pen and post notes (sticky notes) to take notes what the interviewee says.

5. Start talking about the questions in the questionnaire. During an oral history part, try to be silent in order to prevent the interviewer from summarizing what they say. In a problem centered interview, focus on certain episode which they encounter problems and solution.

6. Write down important points especially 5W1H; What (Which), Where, When, Why, and How on the post notes. The reason is that these points are highly possible to lead making patterns [12]. In many cases, a solution toward social issues is firstly talked as a “tip” of social entrepreneurship, so the interviewers or pattern writers need to clarify the contexts and problems. Often, the intention of the actions and solutions are deeply related to the context and problems, so asking Why is tremendously effective.

7. Organize and converge the important points from interviews with the affinity diagram. Some groups are usually formed according to a variety of situations and contexts. Decide on which pattern the interviewer will write.

8. Write patterns using the result from the affinity diagram. If necessary, the interviewers can review the transcriptions.

9. Ask the interviewees to revise the patterns which are produced as a result of an interview. In case of the patterns that have been created by several interviewees’ experiences, the interviewer needs to ask all of them to revise. Encourage them to confirm the wording whether the nuance is expressed correctly in the patterns. It is possible to interview the same interviewees again to improve the patterns, if necessary.

10. Write up social entrepreneurship patterns and reflect on the revises.
5. Patterns: Social Entrepreneurship Patterns

Change Making Patterns consists of 31 patterns. Those 31 patterns are categorized into two main levels and six phases. Each level and each phase have its own objective. Figure 2 shows the two main levels and six phases of the pattern language and the corresponding patterns in each phase. This paper present 5 essential patterns for social entrepreneurial education.

The first level is Self-Empowerment Level, and the second level is Change Making Level. Throughout the process of making the patterns, we discovered that social entrepreneurs not only share tips for making changes or taking actions on social issues, but also commonly have self-empowerment patterns. In short, they reflect on and empower themselves before they take action toward social problems. Often, social entrepreneurs start with self-empowerment, yet they come move back and
forth between two levels.


In **Change Making Level**, there are four phases: *Preparation*, *Change Construction*, *Implementation*, and *Scale-out*. Patterns in *Preparation* promote the reader to step forward into the field of their own issues. *Preparation* phase consists of a pattern: *Quick Action*. Patterns in *Change Construction* show how to think up the theory of change based on social mission and vision. The phase consists of a pattern: *Blueprint*. Patterns in *Implementation* teach some specific strategy for shaping ideas into form. This phase consists of a pattern: *Success Prototyping*. Finally, patterns in *Scale-out* phase provide tips for expanding projects.

![Figure 2. The Structure of the Social Entrepreneurship Patterns](image)

**6. Concluding Remarks**

The complexities of social issues all over the world are challenging the existing organizations and government which have been committing to take action and responsibility to create a better world with less social problems. As a solution, more individuals or civic sector are required to take initiatives to solve issues around themselves rather than relying on existing approaches. Therefore, there is a need for more accessible and effective methods for social entrepreneurship education. Change Making Patterns might be developed as a pedagogical tool for social entrepreneurship education.
Moreover, the completion of the patterns is not our final goal because it is going to be crucial to
device plans for encouraging readers to take action with the patterns. Therefore, we suggest that
patterns should be used in projects or some sort of action while it is essential to improve
comprehensibility of the patterns. Action planning with patterns leads the reader to go beyond the level
of current significance; the reader will be able to plan their future actions with the patterns.

Acknowledgment
I am deeply grateful to Yumeno Nito, Hiroshi Ogawa, Rikako Sano, Takako Yamada,
Takamasa Matsuura, Ryo Imamura, and Junto Oki who were generous enough to volunteer as
interviewees and revise the patterns. Also, my heartfelt appreciation goes to Aliaksandr Birukou whose
comments and suggestions were on inestimable value for my studies.

References
revised.
Chukoshinsyo.
Science Association Ritsumeikan University.
Nature of the Universe, Book 1 – The Phenomenon of Life. CES Publishing.
Knowledge” in 3rd Conference on Collaborative Innovation Networks
(COINS2011).
Patterns

Microvision
Do not think big, look around you.

Context: You are interested in starting a project related to social issues.

Problem: You have no idea what you should start with, because you do not see any personal connections with the social issue.

- “Social” sounds too big.
- People tend to have a big picture in the beginning.
- If people do not see any personal connections with an issue, it is hard to come up with their own approaches.

Solution: Understand the issue on a more personal basis.

Review your life story and look at the issues around you. Focus on the emotions you have experienced, especially anger, fear, and sadness. Then, identify the people who are involved and their stories. Search for people who have the same problem in order to trigger you to start taking actions.

Or, reflect on your failures and analyze them using 3W1H (No.12) questions. Then, draw your own Blueprint (No.17) which show how they would change the situation, and tell your past experiences or your vision to others.

Consequence: You will be able to tell the target issue as your own with empathy, so it helps to engage, potential team members, and supporters.

Example: Establishing a fashion brand, Over the Rainbow, which advocates social issues, Ms. Rikako Sano was traumatized by her childhood memory of witnessing a coup in Jakarta. With her desires to bridging the social gap, Ms. Sano joined Amnesty International, which protects human rights. However, she realized that there is a need for a system that attracts people who has little interests rather attracting
interested people. Ms. Sano narrowed her focus on fashion, appreciating her parents’ career as former fashion models and networks in the industry. She succeeded in solving a social issue by questioning her surroundings and looking closer into the issue, exploiting her interest in fashion.
Detective

Be skeptical about every premise.

Context: You are trying to identify your mission and goals of your project that respond to the social issue.

▼

Problem: You cannot come up with influential solutions leading to fundamental change on the social issue.

- People are blind to the gut issue, being influenced by stereotypical opinions.
- Because people see only superficial situations, they do not realize the fundamental causes of the issue.

▼

Solution: Have a skeptical perspective about the origins of the social problem, and doubt the general opinions toward it.

Think persistently about inquisitive questions on the causes of the problem, such as “Why is this problem not to be solved?” and “Why isn’t there a system solving the issue?”

▼

Consequence: As a result, you are able to obtain a reasonable approach, which makes “the “impossible” to “possible.”

Example: Although NPO Katariba has been aiming to transform high school education, just by discussing the concept did not help to create opportunities to meet high school students. Mr. Ryo Imamura always questioned himself “why he has little chance to meet high school students and searched for possible tactics.” As a result, he created opportunities to see them in person and held classes thus increasing the amount of interacting with high school students.
Quick Actions
Act on your intuition.

Context: You are faced with numerous ways to proceed and cannot decide on which way to go on your project.

▼

Problem: You think too much and hesitate to make progress in the project.
- When people cannot find a logical way of thinking, people tend to hesitate taking actions.
- New ideas are normally believed to be difficult to implement.
- When pondering over issues people hesitate to take action.

▼

Solution: Trust your instinct and take actions as soon as you hit upon a great idea.
Let your inner feeling overcome your logical mind. As soon as you have even a tiny question, do not hesitate to research. If you feel curious about something such as people or places, *Dive into the Field (No. 15)*. If you want to try something for the solution, it is highly recommended to start it small or make *Success Prototyping (No.25)*.

▼

Consequence: Rapid progress attracts new encounters and opportunities for your project to grow. Therefore your quick actions will guide you to achieve your goal sooner than before.

Example: In 2009, witnessing the regime change in Japan, Mr. Ryo Imamura thought that this major turning point in government policy would be an opportunity for Katariba to advance forward in the education field. To quickly respond to this chance, he quit his job and decided to dedicate his time and effort to Katariba. Thanks to his comeback, Katariba was moving forward with new projects and rapidly making progress, such as business expansions.
Blueprint
Put your ideas into shape.

**Context:** You have created Sustainable System (No.16). You are now planning on how to implement it.

**Problem:** Your project drags on, and it takes time to accomplish the goal, so you give up tackling the issue.

- People easily give up on challenging activities.
- People become less productive and postpone things without a clear plan.

**Solution:** Make a blueprint of your project to envision your future success.
First of all, be optimistic and start with imaging your ideal vision even if there are tremendous risks, be optimistic and start with imaging your ideal vision. Think in the framework of theory of change which is like “If I did ~, then people would be ~.” After brainstorming the theory of change, do not hesitate to revise it by writing down or telling them to others. Then, understand what kind of activities or actions you need to take in order to reach your goal, and set deadlines. Then, express your actions and deadlines to others.

**Consequence:** Firm determinations are appealing to others while they motivate yourself to take actions. As a result, they are able to persuade the supporters’ commitment.

**Example:** Ms. Takako Yamada wanted the youth in the Philippines that are under protection of NGOs to be independent and create their own ways. To achieve this, she started a business that hired the young Filipinos as English instructors. Not only teaching English, they also committed to manage and run the business. As a result, earning money on their own led them to build self-confidence. Ms. Yamada discovered a method to make her vision come true by closely working with and talking to the coworkers.
in the Philippines.
Success Prototyping

Keep moving your hand to create success for convincing supporters.

Context: You have already initiated your own project and want to gather more supporters and customers to expand your project. Even though you have come up with many ideas for creating more impacts, you have not implemented any of them.

Problem: Since you have not tested your ideas, there is no reliable outcomes. Therefore, you cannot deliver your project to other communities.

- You hardly obtain trust from people until you have reliable outcome and results.
- It is hard to create great success at the beginning.
- Resources such as time and budget are limited, so you are afraid of loosing them.

Solution: Prototype your plan in your project and attempt making the some successful cases.

Find a specific field, and implement even a small program or a rough plan. For instance, you can try a pilot program, or provide a prototype service for free.

Ask the initial users to evaluate your prototype and receive some good points and rooms to improve.

Improve those prototypes and gain credibility by reflecting on feedbacks from the initial users.

Consequence: You will be able to gain tangible outcomes, which creates reliability. The more you have credibility, the better you have a chance in obtaining support, collaborating with investors, and expanding your project. However, it does not mean that your first prototype succeed at all times, so you have risks to waste resources.

Example: Mr. Ryo Imamura, a senior executive manager at a NPO, Katariba is playing a vital role in building career awareness of students by providing a place to interact. However, he was stuck in a situation where he had little opportunities to interact with high school students thus having trouble testing his ideas in the market. As an exit strategy, he targeted one school and called for undergrad volunteers to share their stories with high school students, which became a successful prototype that became the base of
his business model. Soon after, such business spread due to its accomplishments and results by word of mouth.
Audit of Knowledge flows and Critical business processes

Elissaveta Gourova¹, Kostadinka Toteva², Yanka Todorova³

¹ Sofia University, Faculty of Mathematics and Informatics, Sofia, Bulgaria, elis@fmi.uni-sofia.bg
² TechnoLogica LTD, Chervena Stena 46, 1421 Sofia, Bulgaria, kossy.toteva@gmail.com
³ Sofia University, Faculty of Mathematics and Informatics, Sofia, Bulgaria, yanka@todoroff.co.uk

1. INTRODUCTION

Knowledge has become one of the important resources for competitiveness and growth of enterprises world-wide. Recent trends in Information and Communication Technology (ICT) development have provided organizations and individuals with new opportunities for creativity, collaboration and knowledge transfer. The emergence of Web 2.0 and the Semantic Web (Web 3.0) has reflected in changes of business and innovation models. At the same time, employees have more opportunities to access information and knowledge and use them in the business processes of their organization. The proper Knowledge Management (KM) needs to ensure that the critical business processes are supported with the required up-to-date information and knowledge. It is essential, therefore, during the Knowledge Audit (KA) to investigate the knowledge flows in the organization and the support to the implementation of company business goals through emphasis on knowledge supply and use in business processes.

<table>
<thead>
<tr>
<th>PATTERN NAME</th>
<th>PATTERN PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNOWLEDGE AUDIT PLAN, [EuroPLoP'2009]</td>
<td>Planning of Knowledge Audit scope, activities and resources</td>
</tr>
<tr>
<td>KNOWLEDGE AUDIT TEAM, [EuroPLoP'2009]</td>
<td>Selecting the right Knowledge Audit Team, with the desired mix of skills and knowledge</td>
</tr>
<tr>
<td>KNOWLEDGE AUDIT METHODOLOGY, [EuroPLoP'2009]</td>
<td>Develop methodology for successfully performing specific Knowledge Audit tasks and activities</td>
</tr>
<tr>
<td>KNOWLEDGE AUDIT QUESTIONNAIRE, [EuroPLoP'2009]</td>
<td>Select, compose or adapt Knowledge Audit Questionnaire according to specific company needs</td>
</tr>
<tr>
<td>KNOWLEDGE ASSETS MAPPING [EuroPLoP'2011]</td>
<td>Identify, locate, and assess knowledge assets, and on this base set priorities and identify action needs</td>
</tr>
<tr>
<td>KNOWLEDGE LANDSCAPE MAPPING [EuroPLoP'2011]</td>
<td>Assess available KM practices, programs, projects, infrastructure elements, policies and procedures, etc., and determine actions for improvement</td>
</tr>
<tr>
<td>KNOWLEDGE FLOWCHARTS [EuroPLoP'2012]</td>
<td>Identify existing paths, means of knowledge flows between individuals, groups and in the organization as a whole, aimed at improving knowledge flows</td>
</tr>
<tr>
<td>COMPETITIVE KNOWLEDGE ANALYSIS [EuroPLoP'2011]</td>
<td>Identify areas of expertise and important knowledge assets providing competitors strengths and opportunities</td>
</tr>
<tr>
<td>KNOWLEDGE DIAGNOSTICS [EuroPLoP'2012]</td>
<td>Understand knowledge-related mechanisms and processes, both at individual, group and organizational levels</td>
</tr>
<tr>
<td>CRITICAL KNOWLEDGE FUNCTION ANALYSIS [EuroPLoP'2012]</td>
<td>Identify critical operational, professional or managerial functions, and determine the potential value of their knowledge-related improvements</td>
</tr>
<tr>
<td>KNOWLEDGE MANAGEMENT BENEFITS ASSESSMENT [EuroPLoP'2011]</td>
<td>Focus on potential effects of KM initiatives as a base for planning, action, and monitoring of KM implementation</td>
</tr>
<tr>
<td>KNOWLEDGE AUDIT DATA GATHERING [EuroPLoP'2010]</td>
<td>Ensure high-quality data gathering and involvement of staff in the KA process</td>
</tr>
</tbody>
</table>
The concept of business patterns for Knowledge Audit was presented at workshops of EuroPLoP 2009, 2010 and 2011 [3, 4, 5]. This paper is a follow-up of these patterns and deepens the Knowledge Audit methodology pattern outlining the tools which can be used for its implementation.

These patterns are intended to codify business practices in the area of Knowledge Audit implementation in a pattern language so that they may be better understood, communicated, applied and studied. A summary of all patterns considered by the authors is given in Table 1. The patterns are intended for Knowledge Management practitioners, for managers of Small and Medium Enterprises (SMEs), entrepreneurs, students, experts and consultants. The patterns may be applied in the context of SMEs or knowledge-intensive public or private organizations.

2. THE PATTERNS

The Knowledge Audit implementation follows different phases and processes, and each process is linked to a pattern to be followed by practitioners (Figure 1). This paper deepens the pattern KNOWLEDGE AUDIT METHODOLOGY, considers the tools which may be used [1, 2, 7] and proposes related patterns:

- KNOWLEDGE FLOWS ANALYSIS – aimed at identifying existing paths, means of knowledge flows between individuals, groups and in the organization as a whole, and thus, improving knowledge flows;
CRITICAL KNOWLEDGE FUNCTION ANALYSIS – aimed at identifying critical operational, professional or managerial functions, and determining the potential value of their knowledge-related improvements;

KNOWLEDGE DIAGNOSTICS – aimed at understanding knowledge-related mechanisms and processes, both at individual, group and organizational levels, in order to link them better to core business processes.

3. KNOWLEDGE FLOWS ANALYSIS

As a manager of an organization, you are aware that tacit knowledge is disseminated within the social contacts and communications between people. You like to identify the real knowledge flows in your organizations, and on this basis, to improve the knowledge sharing among your employees, and to ensure that knowledge is disseminated properly among people and departments. Therefore, you launched a project for evaluating social networks in your organization.

3.1. Problem: How is knowledge disseminated in your organization?

The main factors to take into account for solving this problem include:

- Usually, organizations have internal rules and procedures for business processes and communications. However, the everyday communications of employees often have deviations based on their personal connections established.
- There is an ambiguity of employees who are important for knowledge flows in the organization. However, in order to facilitate knowledge flows to reach those who need that knowledge, objective information for knowledge-related connections of employees is needed.
- Knowledge flows are different for each area of knowledge and work in the organization. Focusing on the specific knowledge flows in each area can increase the difficulties for understanding them and can lead to a long data gathering process.
- Investigating knowledge flows in an organization is a time-consuming and resource-consuming process, as it needs to make clear all connections between employees regardless of the formal organizational structure. However, it gives valuable information to managers and can support higher efficiency of the organization.
- The success of the analysis depends not only on the social and analytical skills of the team that conducts it, but also on its ability to motivate and involve other people, as well as to get the necessary support by top managers.
- The social networks analysis highlights problems in knowledge sharing within the organization. However, it can provide also insights to managers for possible solutions to be used in the knowledge strategy of the organization.

3.2. Solution: Create a knowledge flowchart which shows how knowledge is disseminated in your organization

The knowledge flowchart represents a graph which shows knowledge flows in your organization. As a minimum requirement it should contain participants (nodes), direction of the flow, content and way. For example, Figure 2 shows how knowledge is disseminated for making a leave request (content of the flow): Merry personally tells by phone (the way) to John and he publishes the knowledge on the corporate portal (the way), so as the knowledge is available to George, Alan and Sid.
Some steps could be followed for solving the problem:

- In order to create a knowledge flowchart, conduct a survey to examine the knowledge flows in the organization.
- First, identify the knowledge areas which should be analyzed by using the results of CRITICAL KNOWLEDGE FUNCTION ANALYSIS.
- Create a questionnaire for the study, including all knowledge areas identified. Ask employees about people they contact for additional expertise and knowledge which they need for implementing their business tasks.
- Test the questionnaire in a small group of employees and validate the analysis you made with follow-up interviews. During the interviews make sure that employees provide correct answers and are not influenced by organizational rules and policies.
- If needed, after the pilot test, change the questionnaire and then conduct the survey in the whole organization.
- Draw a knowledge flowchart for each knowledge area. Analyze each flowchart by using social network analysis (SNA) and calculating degree centrality, betweenness and closeness for each node.
- On base of the calculated values, identify the employees who are the widely-used knowledge sources in each knowledge area. Identify isolated employees with few connections. Analyze all knowledge flowcharts to understand the overall picture of the organization.
- After drawing the knowledge flowcharts, group the employees in departments, and analyze the intra organizational network.
- During examining the intra organizational social network follow strictly the knowledge audit plan, and the knowledge audit methodology in order to reach the expected results and to ensure consistency of Knowledge Management initiatives.
- Keep the knowledge flowcharts up-to-date by regular investigating them.
- Apply KNOWLEDGE AUDIT TEAM in order to ensure a knowledgeable staff to conduct the survey and the analysis of the knowledge flowchart.
- Knowledge Audit success depends on involving staff in knowledge audit process. Thus, ensure leadership commitment and strong staff involvement in the Knowledge Audit process, as well as champions as a model to be followed by ordinary employees.

3.3. Consequences

The knowledge flowchart provides an insight into the informal social network within the organization and helps for identification of knowledge sources. The result of the analysis can be used for creation or improvement of the Knowledge Management strategy in terms of faster knowledge dissemination and more efficient knowledge sharing.

The identified knowledge flows show paths for dissemination of new concepts and ideas, as well as for awareness raising in the organization. That could be used for facilitating any change management initiatives.

The outcomes of the analysis could become an important component of a well targeted human resources strategy.
The lack of knowledge about informal social network in the organization and its knowledgeable people could lead to lower efficiency of knowledge sharing, as well as not well targeted training and human resources management.

Keeping the knowledge flowchart up-to-date will need further time and efforts, rising proportionally with the size of the company. Tools such as NetMiner, UCINet, GUESS, Pajek, etc. could facilitate the analysis of the organization’s informal network.

3.4. Example

The SNA for creating a knowledge flowchart is used in a foundation in Washington DC [8]. The methodology includes surveys and follow-up interviews of all members of the organization. Important questions were directed to identifying the top 3 persons who possess knowledge and share it in different knowledge areas. The analysis of the results is focused on departments and people. Weight is associated to the edges according to the priority of asking for the area. The SNA tool used for analyzing answers is NetMiner.

By analysis of degree centrality interesting facts are discovered for the Foundation [8]:
- ‘General Management’ is the source of knowledge for general advices; while ‘Knowledge Access and Technology Strategy’ is the less contacted department.
- For management and leadership advices people in the organization rely on outsiders.
- Central connectors are identified between foundation members. These people form informal networks and people can be determined as knowledge holders but they could be also bottlenecks.
- Potential communication issues are identified.

By analysis of betweenness centrality people are identified who are important for knowledge flows. Low level of closeness identifies a department which is hard to be reached by others.

In this case, SNA provides information about knowledge flows in an organization which should be a basis for changes in Knowledge Management strategy.

4. CRITICAL KNOWLEDGE FUNCTION ANALYSIS

As a manager of a middle-sized organization you face the need to improve the performance and efficiency of your organization. You are aware that the Knowledge Management effectiveness is closely linked to the support ensured to critical business processes (operational, managerial or professional). Therefore, you need to assess which processes are the most critical and knowledge-intensive ones for implementing your business strategy goals, and ensure the necessary knowledge flows and actions to support them.

3.5. Problem: Which are the most important knowledge-intensive business processes of your organization?

The business strategy determines the long-term goals for competitive development of the organization. Its implementation depends on some core business processes. Subsequently, the KM strategy should be aligned with the business strategy and ensure appropriate knowledge supply to those business processes.

KM requires additional investments of money, time and efforts. Its efficient implementation requires carefully determining priorities in order to ensure the expected efficiency and organizational benefits without spending unneeded resources. It should be decided if it is worth targeting all business processes or part of them, and in the second case – which.
- In principle, all business processes in an organization are designed to support achieving its business goals. However, some of them comprise fundamental activities that bring the highest value to the organization and are critical to its success and competitiveness.
- Managers consider that their own business processes are essential for meeting business goals. However, it should be understood from business point of view which business processes are the most important for the organization.
- Some business processes have an essential impact on organizational competitiveness. However, they are less knowledge-intensive and do not critically depend on KM initiatives, the latter adding small value to them.
3.6. Solution: Critically assess business processes and determine their dependence on proper Knowledge Management

Some essential steps could be followed for overcoming the problem:

- Start with acquiring information on the business strategy of the organization, its goals and activities for meeting these goals. You could use questionnaires, interviews and work monitoring at this stage. Check also the results of the KNOWLEDGE LANDSCAPE MAPPING.
- Determine the critical factors for the organizational success and competitiveness, e.g. which have an essential impact for reaching its business strategy goals (new products development, customers’ satisfaction, quality assurance, higher revenues, etc.).
- Identify and list all business processes in the organization, e.g. linked to company management, marketing, production, sales, resources management, etc.
- Prioritize the business processes according to their impact on organizational performance on bases of the critical factors determined before.
- Determine the knowledge intensiveness of the selected core processes and the knowledge processes linked to each of them using quantitative and qualitative methods for the analysis.

3.7. Consequences

The consequences after applying this pattern could be summarized as follows:

- By identifying knowledge-intensive business processes that are critical for the organizational competitiveness, it is possible to better target the KM strategy and ensure that the subsequent actions add higher value for the organization and its overall performance.
- The impact of KM for the organization could be higher if you overcome the knowledge gaps in its critical business processes.
- If critical knowledge-intensive business processes are not identified, the Knowledge Management priorities and actions will be not well targeted, and thus, less efficient.
- Applying a KM strategy to support all business processes, could lead to unnecessary spending and minimize the expected impact and return of investments.

3.8. Example

A Knowledge Audit methodology based on identification of core processes is proposed in [6]. The authors consider as especially important to “understand the processes which constitute the activities of a knowledge worker and see how well they address the “knowledge goals” of the organization”. Therefore, they suggest to carry out an evaluation of all processes of the organization in order to select its core processes that fulfill the following characteristics:

- “It has a direct impact with mission and vision.
- It generates revenues or is the most critical to overall success of the organization.
- It has impact and it gives an added value to the organization.
- It allows satisfying customer requirements.
- It has valuable human, technological and information resources.”

The Knowledge Audit methodology proposed in [6] contains 10 stages (Figure 3), whereas the identification of core processes and their prioritization are implemented at the very beginning of the whole process (stages 1 to 3). As tools which support stages 2 and 3, the authors propose using questionnaires, explore the general documentation of the organization and quantitative documentation (income, sales, and customers’ information), and consider documents which allow valuing the impact of the processes with respect to the organizational mission and strategic business objectives. Qualitative methods (face-to-face interviews, focus groups) can be used for deepening the understanding for business processes and the knowledge needed and generated by them.
The effective KM strategy uses the KA for building a competitive position using its knowledge resources in the best possible way. Therefore, it is essential for an organisation to be aware of its core knowledge, including operational knowledge (helping it to do its work), and strategic knowledge needed for its future development [9]. Generally, core knowledge [10] ensures the smooth implementation of the critical business functions, has long-term value for the organizational activities, ensures higher productivity, and its gathering and management pays off.

As a manager of a middle-sized organization you face the need to improve the performance and the efficiency of your organization. You completed an analysis of the competitors’ knowledge positions, and of your critical business processes. Now you need to be aware if you possess the knowledge necessary for your business processes implementation and for being more competitive in your market. At the same time, you need to be aware of knowledge mechanisms and processes that support the critical business processes of the organization, as well as to investigate how the available knowledge resources are used and how they support reaching the business goals of your organization. A further need is to evaluate the knowledge gaps linked to the business strategy goals.

3.9. Problem: Do your knowledge resources effectively support your business processes and implementing your strategy goals?

The KM strategy is linked to the business strategy and supports its implementation by filling in the knowledge gaps in order to reach the business goals of the organization. Some essential factors need to be taken into account before making the choices for overcoming the problem:

- It is not sufficient to identify and localize existing knowledge, but also to analyze knowledge processes and in particular its use in business processes and for support of the business strategy.
- The business processes in each organization depend on the available knowledge in the organization, in its units or held by employees. However, if managers don’t know which knowledge is needed for that purpose, they can not ensure it and overcome the knowledge gaps.
- The organization can possess a large variety of knowledge resources. However, if employees don’t know that this knowledge exists and where it is located, they can not use it and it can not add value for meeting the business goals of the organization.
- Organizations design their structures to support the smooth implementation of business processes. However, knowledge flows take different paths and sometimes do not reach the employees who need that knowledge.
- Business processes are the core of business strategy. However, the success of business processes critically depends on ensuring that knowledge processes (e.g. acquiring, storing, using, transferring knowledge) support them and supply the appropriate knowledge where and when it is needed.
- Information and Communication Technologies (ICT) provide a common environment for knowledge processes. However, they should be properly designed in order to link knowledge and business processes in a most efficient way.

3.10. Solution: Critically assess the knowledge needs for your business processes and ensure their knowledge supply.

Some essential steps could be followed for overcoming the problem:
- Start with CRITICAL KNOWLEDGE FUNCTION ANALYSIS and identify the core knowledge needed for your business processes.
- Take into account the KNOWLEDGE ASSETS MAPPING to understand to what extent the core knowledge is available in the organization.
- Analyze, first, the availability of knowledge needed in your critical business processes (identified in CRITICAL KNOWLEDGE FUNCTION ANALYSIS), and then continue with analysis of other business processes.
- Take into account the KNOWLEDGE FLOWS ANALYSIS for assessment of possible internal and external sources to fill in the knowledge gaps, and to understand if knowledge reaches the employees who need it for implementing their business tasks.
- On bases of KNOWLEDGE LANDSCAPE MAPPING and KNOWLEDGE ASSETS MAPPING check how knowledge is structured, and organised.
- Identify the available knowledge processes and analyze how they are managed and how they support the business processes (Figure 4).
- Assess the available technology support for knowledge processes in your organization, and how it is linked to business processes and accessible by employees.
3.11. Consequences:

As a follow-up of KNOWLEDGE DIAGNOSTICS:

- The organization could take appropriate decisions for filling-in the knowledge gaps and ensure efficient usage of the available knowledge resources for support of its business processes.
- The organization will be aware how knowledge processes support its business processes. It could ensure proper knowledge flows and could consider possible changes in its organizational structure and culture, as well as technology environment.
- Linking critical business processes to knowledge processes that support them could help to increase organizational performance by reusing knowledge, avoiding duplication of work, and using better the available knowledge assets.
- KNOWLEDGE DIAGNOSTICS is an essential part of the KNOWLEDGE AUDIT METHODOLOGY. If not properly implemented, it could lead to wrong decisions for the Knowledge Management strategy.
- The organization could face difficulties for implementing its business strategy, and reaching its goals if the knowledge resources are not well used and timely ensured.
- The organization could loose its market position and competitiveness if the knowledge gaps for critical business processes are not timely filled-in.

3.12. Example

Universities are knowledge-intensive organizations with the mission to increase the knowledge of their students and supply to the economy the knowledge needed for the labor market and development of new products, systems and services. The main business processes of a university comprise education, research and administration of students and staff, educational and research processes. Normally, universities consist of different faculties, departments and/or institutes who have their own administration of all business processes and available knowledge resources. With the support of ICTs these units develop their own knowledge systems to support educational processes and ensure knowledge supply to their clients – the students. ICTs support students’ administration, administration of learning programs and courses, provide e-learning platforms, knowledge sharing, etc.

In many Bulgarian universities research processes are not sufficiently supported by ICTs and a central depository of knowledge about research projects, individual research achievements, etc. is not in place. Researchers from different units could hardly find the expertise and knowledge they need. The university administration does not know where and what kind of knowledge is available. The process of regular evaluation and ranking of universities, however, requires measuring not only educational outcomes, but also the research and development results of individuals, units and the university as a whole. This requires development of knowledge system:

- to gather the knowledge available in the organization and by a single employee
- to organize the knowledge in a way to support educational, administrative and research processes
- to ensure regular knowledge update and transfer to employees.

As a first step, a central depository of research papers was designed. Second, a system was developed for collecting individuals’ information – research papers, participation at conferences and in projects, students’ MSc and PhD Thesis guidance, etc.

6. CONCLUSION

The paper presents three different patterns corresponding to the main tools used in the Knowledge Audit methodology - KNOWLEDGE FLOWS ANALYSIS, CRITICAL KNOWLEDGE FUNCTION ANALYSIS and KNOWLEDGE DIAGNOSTICS. This ensures that the Knowledge Management strategy and the related Action Plan will be better focused and facilitate organizational performance and competitiveness. The patterns can be applied in knowledge-intensive organizations as they propose guidelines for solving specific problems of linking knowledge and business processes, better usage and provision of knowledge when and where it is needed and taking advantage of existing knowledge flows.

ACKNOWLEDGEMENTS

The authors express their deepest appreciation to their shepherd Bettina Biel for the thorough guidance for preparing this paper. The authors gratefully acknowledge the results obtained within FP7 project SISTER.
REFERENCES
The Adoption of Software Patterns in the Dutch Software Industry

G. NIJBOER, J. KABBEDIJK and S. JANSEN, Utrecht University

There is no clear insight into the adoption of software patterns within the Dutch software industry. This leads to the fact software pattern researchers experience difficulties communicating pattern research with industry and struggle to gather pattern usages. This study explores the use of patterns within the Dutch software industry. Six different software companies were interviewed on how they used software patterns in their daily practise. The research questions are aimed at identifying what type of patterns are used and whether patterns are used by developers in their own or more as a company wide policy. The research concludes that software patterns are often implicitly or even unconsciously used by software companies, instead of explicitly in communication. Patterns identified within academic research are indeed known and used within the software industry, but still more on a personal level. The companies see a lot of value in the use of patterns, but the actual use currently lacks behind because of missing support from within the company.

Categories and Subject Descriptors: D.2.11 [Software Engineering]: Software Architectures—Patterns

1. INTRODUCTION

In software development, developers often face the same problems over and over again, gaining experience in designing and implementing solutions, resulting in faster delivery of the same solution over time. But why would you re-invent the wheel if others have already done so, and documented it for reuse? Patterns are developed as solutions for common problems in software development [1]. One of the characteristics of all software patterns (descriptions of sets of successful solutions for recurring problems within a certain context [2]) is that they have several levels of abstraction, spreading from detailed code-implementations to abstract architectural concepts. The concept of patterns is that they are generic, and can therefore be used in many processes of software design, and even in designing physical objects, such as buildings.

Since new software development methodologies have been introduced in the past 20 years, one might expect the concept of related concepts, including software patterns, to change along as well. The concept of patterns could even disappear, due to the lack of support for such topics in modern, agile software development methodologies. The generic characteristic of patterns, however, leads to the fact that they are currently still used and researched, also causing the rise of new patterns, for instance in research by Kabbedijk [3].

This research is part of the 'Product as a Service' project (www.productasaservice.org).
Corresponding author: J. Kabbedijk (j.kabbedijk@uu.nl)

©Copyright granted to Hillside Europe for use at/with the EuroPLoP 2012 conference.
Through this research, we have created an indication of how software patterns are currently used within software developing companies in the Netherlands. This research, performed at six software developing organisations, is conducted through interviews with key actors in the organisations, playing a central role in the development process. The result of this research is aimed at providing insight in the current use of software patterns in the software industry. This creates a better understanding of the way patterns should be communicated with the software industry. This paper will first report on the research approach, containing our interview guidelines and structure in section 2. In section 3 the characteristics of the case companies are presented, followed by the results of the conducted interviews in section 4. These results are analysed in section 5. A conclusion summarises the analysis in section 6 and answers each of the sub-questions introduced in the next section of this paper.

2. RESEARCH APPROACH

We have approached this research by stating one primary research question, and then dividing this main topic into four sub-questions, specifying the scope of the research. The research question and its four sub-questions are formulated as follows:

- How are software patterns currently used within software developing companies in the Netherlands?
- Are software patterns intentionally used in software development?
- How are software patterns consulted by developers and architects?
- Are software patterns mostly used at architectural, design or idiom level?
- Are software patterns primarily applied in implicit problem-solving, or are they also explicitly referred to in software development documentation and communication in-house?

Six of the invited companies were willing to participate in our research. We attempted to cover the full spectrum of software companies in the Netherlands as good as possible by selecting case companies from different domains. This prevents that our results are based on only companies from, for example, the ERP-domain. Though we have only spoken with one interviewee per company, a variety of responsibilities amongst the interviewees was gathered. Each of the participating organisations has been given an alias, in order for them to remain anonymous. The names of the interviewees can be found in the Acknowledgements in section 7.

The interviews conducted at the participating companies have focused at gathering qualitative results, rather than quantitative results. Each case has been conducted on a one-to-one basis, in a quiet space, to prevent interference from others during the interview. These interviews have primarily taken place at the location of the companies itself, though there are exceptions for the interviews with GenerateComp1 and AllroundComp. The case of GenerateComp1 was performed at the Utrecht University. The case of AllroundComp was performed through an online video call with Skype, because the interviewee was located in Belgium.

The interview protocol set up and respected as a guideline for conducting the interviews was semi-structured, flexible and dynamic. Due to the openness of the questions in the document, we have implemented some freedom in the thread of the interviews, leading to usable, qualitative data. The interview protocol did not focus only on the adoption of software patterns, as we also left an adequate amount of space for information about other aspects. Included was information on the participating company, covering its size, locations, focused industries and products. Data about the interviewees themselves was also gathered, to state more about their background and education. We also gathered data about the approach to starting new software projects, since we believe the way the software is developed plays a strong role in the adoption of software patterns by the same team.
The section of the interview focusing on software patterns covered both basic questions such as: "Could you give us a definition of what you believe software patterns are?" - and more in-depth questions about the way software patterns are used and referred to.

Audio was recorded during every interview, so that it could be used as a reference in later stages of the research. Apart from the recordings, notes on paper were also made during the interview, to capture the main topics and to produce an outline for the written reports. These reports consisted of approximately two pages of text about the topics discussed in the interview, which could be used as results for the analysis. The outcomes of the interviews were summarised carefully and with detail within 24 hours.

3. CASE COMPANIES

As mentioned in the previous chapter, we have gathered the data from the interview cases into separate reports, which will be discussed and analysed in this research. We will go into more detail about each of the companies, accompanied with data from Table I, which lists the participating organisations and interviewees and their characteristics.

<table>
<thead>
<tr>
<th>Company</th>
<th>Location</th>
<th>FTEs</th>
<th>Product</th>
<th>Type</th>
<th>Industry</th>
<th>Interviewee</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResearchComp</td>
<td>Enschede</td>
<td>60-70</td>
<td>Process modeller</td>
<td>On premises</td>
<td>Generic</td>
<td>Researcher</td>
</tr>
<tr>
<td>CRMComp</td>
<td>Utrecht</td>
<td>25</td>
<td>CRM-suite</td>
<td>On premises / hosted</td>
<td>IT / Telecom</td>
<td>Developer</td>
</tr>
<tr>
<td>GenerateComp1</td>
<td>Leiden</td>
<td>5</td>
<td>Software generator</td>
<td>SaaS to on premises</td>
<td>Generic</td>
<td>Lead Developer</td>
</tr>
<tr>
<td>GenerateComp2</td>
<td>Rotterdam</td>
<td>95</td>
<td>Software generator</td>
<td>On premises to SaaS</td>
<td>Generic</td>
<td>CTO</td>
</tr>
<tr>
<td>AllroundComp</td>
<td>Gouda</td>
<td>5,300</td>
<td>SCM, ERP, CRM</td>
<td>On premises</td>
<td>Generic</td>
<td>SCRUM master</td>
</tr>
<tr>
<td>SCMComp</td>
<td>Rotterdam</td>
<td>20</td>
<td>SCM-suite</td>
<td>SaaS</td>
<td>Generic</td>
<td>Lead Developer</td>
</tr>
</tbody>
</table>

The first company, ResearchComp, is a research-company in Enschede with 60 to 70 employees, focusing on research and innovation in ICT. One of their current topics of research is enterprise architecture. The company produced one software product in the past, which became an independent and successful spin-off. The product is a process-modelling software product, which is installed on-premises. The development of this software product has been our main focus during the interview, because our interviewee has participated as a principal researcher in the development process. He has worked in the development of the software product we spoke about in the previous paragraph. He currently researches topics on enterprise architecture for the organisation and its clients.

The second company, CRMComp, is located in Utrecht and builds a CRM-suite which focuses on IT and telecom companies. They currently have 25 people employed, including 5 developers. The reason they develop the product especially for these industries is that such organisations are distinguished by project-based planning and less logistic planning. The company currently exists for 15 years, and they have transformed their product into a .NET-product five years ago. The product is currently deployed on-premises, but a SaaS-solution is scheduled for development. A developer responsible for the architecture of the software framework was interviewed for our research. Their team is formed by a group of 5 developers. There is no specific software architect, all software developers are responsible for the software architecture.

GenerateComp1 is located in Leiden and builds a software generator, based on model-based development. It is a small business of 5 FTEs which is still evolving, but the lead developer has given us a good idea of the use of software patterns within their development process. The software generating product can be applied to generally any industry, though the applications often focus at CRM and ERP. The generator is a SaaS-solution, and the generator products are installed on-premises or hosted.
At GenerateComp2, also a software generator is developed, but then at a much larger and much more completed stage than at GenerateComp1. Their Chief Technology Officer has participated in our interview, giving not only details about the use of software patterns, but also about how to organise a modern, dynamic business that is purely active in developing software. Their development follows the SCRUM-methodology, the teams participate actively in this concept. Since the organisation is also located in the USA and in South-Africa, other sales-channels are opened for the software to be sold. Added to this is a large network of partners, who use the software generator to deliver software products to their clients. The responsibilities of the interviewee included product development and defining the corporate strategy.

The AllroundComp organisation does not focus on one information system, neither does it only develop software. But due to its size, it gave us insight in how software is developed in a multinational company that develops software in teams that could cross national borders. Our interviewee was one of the SCRUM-masters, being responsible for the planning and realisation of the planning of his team, by preventing any external factors of interfering with the daily workflow. He therefore delivers coordination and commitment to his team.

The last company listed in the table is SCMComp. The company, settled in Rotterdam, builds a hosted Supply Chain Management suite. It is also located in New York and India, making 24/7 development and support possible. Since the suite can be customised to fit any type of industry, the hosted deployments are often reused by customers with the same characteristics and industrial focus. The interviewee participating in our research has taken a more management-central task, and is also responsible for management in engineering. The SCMComp maintains a strict policy against software-faults, where the fault has to be acknowledged within 15 minutes, and solved within 4 hours, delivering great added value to its customers. Because the software focuses mainly on large organisations with complex logistics, it is also a required and requested service, as one may lose large amounts of money per minute if the entire logistics software suite has crashed.

4. INTERVIEW RESULTS

Table I gives an overview of the characteristics of the participating interviewees, the organisations they are employed for, and the software product that is developed. According to data from the Dutch Chamber of Commerce, the total amount of organisations in the industry of software development in the Netherlands was stated at 47,712 for January 2012 [4]. 495 organisations have entered the industry and 974 have stopped their business, leaving a negative net score of -479 for that month. We believe that the results of the six organisations that have participated in our research apply to the larger part of all software developing companies, because there is diversity in size, domain, product and industry amongst the participating companies.

The size of the organisations differs a lot amongst the participants. We see a small-sized company at GenerateComp1, employing only five FTEs. But nevertheless, the software product that is built is a software generator, which is not a common software product. On the other hand, we have visited AllroundComp, which employs 5,300 people and works on multiple software development processes at the same time, in multiple countries. This diversity gives us a good indication of how software patterns are used within both small and large companies.

Though all interviews were conducted from a location in the Netherlands, we have visited companies that are pure national players, but also some that are multinational software vendors. The GenerateComp2 company uses a broad network of partners to create new sales-channels, and the SCMComp has locations in New York and India, making 24/7 service delivery possible. We have requested each interviewee to give us a definition of the term software patterns. Though using different words, as expected, each interviewee managed to give a definition that meets the required elements that we
The Adoption of Software Patterns in the Dutch Software Industry

set before starting the interviews. We have focused on the general definition of "software patterns are descriptions of sets of successful solutions for recurring problems within a certain context" [2].

Each of the interviewees, excluding the participant of GenerateComp1, has had an education that included one or multiple courses on software patterns. This implicit knowledge was once taught, but is either consciously or unconsciously applied during implementation of code later, too.

In the case of ResearchComp, we were not able to gather results on the approach to new software development, since we could only identify and discuss one event where this actually took place. This did not give us the opportunity to speak of a standardised approach to setting up a new software development project. During the development of their previous software product, design patterns were actively used. Not only were they implicitly used in the minds of the software developers, they were also explicitly used by means of communication and books. They were not only used for problem-solving, but also as a guideline in designing and developing software. The use of software patterns has been at a higher level of abstraction, not at the lower, more detailed level with for instance the Singleton-pattern.

By being told that ResearchComp did not only implicitly use design patterns, the interviewee referred to a book case full of literature on topics such as patterns, architecture and methodologies. Not only printed literature was used, digital references on the internet were also used to gain information from. At ResearchComp, research has been done on multichannel-architectures, and patterns are being created to deliver common solutions on this topic, gathered in a catalogue by a group of thirty patterns.

CRMComp lays focus on implementing features from previous products when starting the development of a new software product. This requires little customer participation, because those existing features are already stabilised and accepted through customer feedback. The functional requirements of the new software product are formed by features from previous systems and new requirements requested by management, sales, and customers. The interviewee confirmed that software patterns are used at CRMComp. An example was raised when several interfaces used the same switch statement over and over again. They tackled the issue by implementing the Abstract Factory pattern, making the switch statement reusable throughout multiple classes. The interviewee indicated that factories are commonly used as a solution to tackle development issues. Our interviewee was educated to use patterns during software development, which created a basic knowledge for him to rely on during the process of development. This leads to the fact that our contact owns books like that of the "Gang of Four" [1].

Though the contact used the book during education, he no longer uses the book at his job. He does use resources on the internet when a quick recap on any of the concepts is required. We were told there is no such thing as communication in terms of patterns between colleagues, so patterns are only implicitly used by developers. Interesting to note is that the interviewee mentioned that "design patterns should not be used because one believes that they must be applied, but they should be used to solve problems during development". By this, he means that you should not confuse the means and the ends, since design patterns are those means to get to the ends. There is no management push to use software patterns, the decision to use them has been taken and executed by the developers themselves.

During our interview with GenerateComp1, it became clear that there is a strong presence of agile software delivery when using the generator to create a software product. Strong involvement of the customer or end-user is required and this makes it possible for them to deliver software while having completely no knowledge of the domain it is designed for.

Software patterns are "sometimes" used by the interviewee, and they are often applied at a lower level of abstraction, which means the more detailed, creational patterns. The use of patterns is actually implicit, so developers are not explicitly and consciously studying and implementing patterns.
during development. Added to this, the interviewee admits that he, as a graduated information scientist, did not have the proper education to be an expert in software patterns. Though, his understanding of software patterns was sufficient to contribute to the research results. There is no management-push to implement software patterns, and neither is there any training - internally or externally provided - in software patterns. Whenever necessary, it does happen that more information on patterns is researched through sources on the internet.

At GenerateComp2, new software products are explicitly guided by a product roadmap. Input for a new version of the software generator comes from over seven sources, and therefore strong considerations between product marketing and development have to be made. The flexibility in the development at GenerateComp2 comes from their SCRUM and Agile software development methodology.

Our interviewee was very aware of software patterns and the way they can be implemented. He was able to name some patterns and authors on books that describe software patterns. He indicates that design patterns are used as a source of inspiration, and are therefore implicitly used, but there is no top-down push to force the developers to use design patterns during implementation. The abstraction at which software patterns are used within GenerateComp2 is at a lower level, enabling “pattern based engineering”. The model-based engineering done through the software generator of GenerateComp2 is therefore purely the use of patterns, as is implementing a template of such a generated software product.

At AllroundComp, many software products are developed at the same time, and since each might have a completely different approach to software development, we decided to focus on the business unit that our interviewee is currently active in. There’s a great respect to their methodology, which is SCRUM.

The interviewee mentioned that “it is easy to write code, but it is hard to write good code. Code has to be reusable, flexible and not too rigid, so loosely coupled”. Those solid design principals assist in applying solutions to often recurring problems. He says: “and when you give such a solution a name, it becomes a pattern”. He states that it is a skill to be able to identify and implement patterns, but you should never apply software patterns just because you think it makes your code “cooler”. A developer should only implement patterns when it can aid in solving issues during design or development. The term “pattern-fever” is related to the overactive use of patterns and was mentioned to describe the previously statement.

The interviewee believes that one can become an expert in software patterns by studying the solid patterns, and by implicitly implementing these patterns in self-written code. After reviewing the code and learning from what you did, you can get highly experienced in patterns. In the organisation, there are no specific external trainings provided for (new) employees which focus on patterns, but it might be a subject during a more generic training. The most valuable thing in learning patterns and learning to write good code is transferring implicit knowledge amongst colleagues. There is no specific management-push to use software patterns, so whenever this happens, it is often the choice of the developers themselves.

Sometimes patterns may be explicitly communicated, though this does not take the form of code, but more in principles of implementation. Books on the topic are present in the office of AllroundComp, and even better, a complete book-exchange has taken place there. Patterns are often implemented at a lower level of abstraction, because the interviewee believes it is also easier for developers to think about them at a lower, concrete level, rather than at a higher, more abstract level. He also mentions to think of higher level patterns more as being software architecture.

We conducted the final interview at SCMComp. The plan of approach for new software products from SCMComp is based on the PRINCE2 five-phases model, existing of divine, design, develop, deliver and deploy. It is mainly the cost-technical driver in their software development. Their software is
developed according to the SCRUM and Agile methodology. There is no internal education on software patterns, but all employees have studied at a university, and software patterns were part of their education. New personnel is not sent to an outsourced training on software development and the use of patterns, but at SCMComp they would rather choose to hand the new employee some books on design patterns and make use of internal knowledge exchange. There is no sincere push from management for implementing software patterns, though new colleagues may be informed about changing their code to meet the patterns’ aspects during code reviews. This is rarely needed for the older colleagues, since their code is often developed conform the company’s patterns.

Software patterns can be found both at the higher and lower level of abstraction at SCMComp, both during the design and implementation phases of the software product. One may find patterns in terms of communication, for instance during code reviews, but they are mostly used implicitly. Currently, the use of patterns is becoming less, because they are already implemented in the product and so they require little attention for further development. Patterns are sometimes researched through books or the internet, but the "less exotic" software patterns are in the developers’ minds.

5. ANALYSIS

In order to analyse the results of our research at six software developing organisations, we will cover each of the four sub-questions in a separate section.

5.1 Are software patterns consciously consulted when requested for problem-solving in software development?

We have seen several cases in which it was explicitly mentioned that software patterns were referred to in software development, in order to solve problems or issues in code implementation. We have seen such cases at ResearchComp, CRMComp, AllroundComp and SCMComp, where they were truly, actively and explicitly used to tackle common software development problems.

Added to this, we saw very little management push. There was mostly bottom-up influence on the use of software patterns in problem-solving. It seems that in our six cases, software developers have decided for themselves that implementing software patterns into their code will help them in creating efficient, reusable, maintainable and sustainable code.

Five of the interviewees participating in our research have also been taught about software patterns during their education. This leads to the fact that the knowledge and skills on using and implementing software patterns have become implicit knowledge. Therefore, they are often implicitly implemented throughout code, not necessarily consciously, too.

5.2 How are software patterns consulted by developers and architects who are using them to solve problems?

When explicitly and consciously using software patterns to solve common issues and problems in software development, we have mostly seen books, literature and the internet as a source to conduct research. Researched topics did not only include recaps on how a pattern should be implemented, but also which software pattern is best to solve a specific problem. This second issue is often much more complex and could take up much more time than just doing a quick recap.

We have even heard about an internal library of related literature at AllroundComp, where the contents of the collection can be used to recap implicit knowledge or to gain new knowledge, including topics such as software patterns.

Searching the internet with Google for the terms "software patterns" returns approximately 366,000,000 results. Although the internet contains plenty of information on the topic, we saw that popular books such as that of the Gang of Four [1] still remain popular as a source for reference. This is a good indi-
cation that these books - and their contents, the software patterns - have survived the test of time, and are currently still of great value in software development.

5.3 Are software patterns mostly used at architectural, design or idiom level?

We have seen software patterns to be mostly adopted at the idiom level, meaning at code-implementation. This comes down to patterns such as the Singleton and Abstract Factory, patterns that are close to the implementation.

The analysis that the software patterns with least abstraction are most often used comes close with the analysis for the previous sub-question. Since the decision to use software patterns is often made by the developers themselves, they are only adopted during implementation, so when writing the actual code, and not during design or requirements analysis. However, if patterns were adopted by - for instance - software architects, we could expect software patterns at a higher level of abstraction to be used, too. Unfortunately, we did not speak with any software architect for this research.

An occurrence of usage of patterns with a higher level of abstraction was observed at the companies that have built a software generator, which are GenerateComp1 and GenerateComp2. They are using patterns with a higher level of abstraction to go along with their models, which enables the model-based engineering of the software generator.

5.4 Are software patterns primarily applied in implicit problem-solving, or are they also explicitly referred to in software development documentation and communication in-house?

We have seen cases of implicit problem-solving guided by patterns at each of the six participating organisations. During their education, the developers have been taught how to use software patterns, and reuse that implicit knowledge to apply software patterns to their implementation of code. This can be either consciously or not.

We did see cases at which explicit references to software patterns are used in software development, for instance at ResearchComp, GenerateComp1, GenerateComp2, Allround-Comp and SCMComp. At the companies building a software generator, explicit use of architectural patterns is found in the models that drive their software generators. AllroundComp and SCMComp seem to use explicit communication by means of patterns during code reviews and to transfer implicit knowledge.

6. CONCLUSION

We have started this research with the question how software patterns are currently used within software developing companies in the Netherlands, trying to find out whether or not there is still a good synchronisation between software patterns that have been documented in the past, and software products that are developed in the present.

We have visited six software developing organisations, conducting a qualitative, semi-structured interview with one of their employees involved in the software development process. Each organisation, product and interviewee had unique characteristics, in order to reach a broad spectrum of factors.

Software patterns are mostly consulted with focus on solving a common software problem, rather than being consulted to gain new knowledge. They are searched through means of literature or digital media, such as the internet. Software patterns are often used at a lower level of abstraction, because it is often the developer who decides to implement patterns in the code. In most cases, patterns are implicitly applied, instead of being used by means of communication amongst colleagues.

This means that software patterns and their accompanied knowledge are of added value in the process of software development, even though they might not be explicitly and consciously implemented in software code. Since software patterns are best practises, and a good developer aims at writing qualitative code, they can assist in reaching the previously named goal.
We may conclude that software patterns are actively used within the Dutch software industry. Thanks to the attention paid to patterns during education, the knowledge becomes implicit and can even be unconsciously applied later during software development. As a consequence, the description of patterns gathered by academia and the way they are communicated with the industry delivers added value in software development.

7. ACKNOWLEDGEMENTS

We would like to express our special thanks to the participating interviewees and their organisations for their time and knowledge, which made it possible to conduct this research. We would like to thank Marc Lankhorst, Sander Weerderburg, Sander Kouwenhoven, Johan den Haan, Arne Wauters and Pascal Durr for their participation.

We would like to express a special thanks to Dr. Eduardo B. Fernandez for reviewing and commenting on the paper in its several stages.

REFERENCES

E. Gamma, R. Helm, R. Johnson, and J. Vissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.


Kamer van Koophandel. Kvk: Software ontwikkeling, January 2012.
A Case Study of the Variability Consequences of the CQRS Pattern in Online Business Software

JAAP KABBEDIJK, SLINGER JANSEN and SJAAK BRINKKEMPER, Utrecht University

In order to maximize their customer base, business software vendors are trying to offer software products that support the business needs of as many customers as possible. The more standardized a software product is, the easier it will be to serve large numbers of uniform customers. However, if customers are not homogeneous, a trade-off must be made between flexibility and complexity. A case study is presented showing the implementation of the Command Query Responsibility Pattern (CQRS), a pattern dictating the strict separation between commands and queries. The study was performed at a large software product vendor currently designing a software product based on CQRS. Seven sub patterns related to CQRS are identified and discussed. The research results show the CQRS pattern is implemented and how its different sub patterns can result in a high level of variability within a software product and how the different sub patterns can interact to achieve this.

Categories and Subject Descriptors: D.2.11 [Software Engineering]: Software Architectures—Patterns

General Terms: Software Patterns

Additional Key Words and Phrases: Case Study, CQRS, Software Pattern, Software Architecture, Variability

ACM Reference Format:


1. INTRODUCTION

It is highly relevant in business software to offer a product to customers that fits their business processes, especially in ERP and related bookkeeping software. This can be problematic, since different customers have different business processes and because of this different, or even contradictory requirements to a software product. The architecture of a software product has to support the variability needed to offer a software product flexible enough to match all different customer requirements, while not introducing unwanted side effects, such as complexity, scalability or security challenges. In Software Product Lines (SPL), variability is known as the ability to change or customize a software product [11]. This definition is sufficient for software products that are manufactured in a software product line style and deployed on premises at customers, but does not hold true any more when it comes to online software products. Online software products have to be able to offer variable solutions at the same time from a single customizable instance, a concept known as runtime variability [16]. The principle of serving multiple customers from one online software product, giving each the idea they are the only customer using the product in terms of flexibility is known as multi-tenancy [12].

In order to create a software product, capable of offering a certain level of variability, most current software products separate logic into different layers. Each tier within this architectural principle is responsible for a different part of the architecture [14]. An often implemented solution to this multi-tier
architecture is the three-tiered application in which there is a separate data, logic and presentation tier. Within this solution, the database in the data tier is often seen as one CRUD (Create, Read, Update and Delete data) data store in which all commands and queries are performed on the same database. This can lead to locking, performance and scalability problems, especially with larger commands or queries, since all things have to be taken care of sequentially. Distributing parts of the system in combination with selective locking of data provides a partial solution, but leads to a high probability of data inconsistency.

Since the CAP theorem [8] states that it is impossible for a distributed system to have Consistency, Availability and Partition Tolerance at the same time, it is an option to split parts of the system that have an emphasis on consistency from parts that should have an emphasis on availability or partition tolerance. Following this line of thought, Greg Young and Udi Dahan came up with the CQRS (Command Query Responsibility Separation) pattern [23; 5] in which all logic of a software product is separated based on whether it changes the application state (commands) or only queries it (queries). This means executing commands is done by different components than the one responsibly for executing the querying tasks, all of which can be done distributed and in parallel.

Besides helping to solve the scalability problem of multi-tiered software products by enabling architects to distribute tasks of the system among an unlimited amount number of systems, CQRS also helps to implement a higher level of variability in online software products. The high level of variability is caused by the fact the main pattern keeps commands strictly separated from queries and has a large collection of sub patterns using the distributed nature of the pattern to enable, among others, all sort of different tenant dependant configurations, work flows, and business rules. This concept will be further explained in section 6.

This paper will first report on related research in section 2, after which the research approach will be discussed in section 3. After this, an example of the CQRS pattern will be shown we observed in the case company in section 4. Different sub patterns playing a role in CQRS will be dicussed in section 5, followed by the consequences CQRS has in section 6. The paper ends with a discussion and some future research in section 7, followed by a conclusion in section 8.

2. RELATED WORKS: CQRS AND VARIABILITY

The ground principle of CQRS, stating the strict separation of command and queries, is introduced by Betrand Meyer in his book Object-Oriented Software Construction [15]. He called it Command Query Separation (CQS), a pattern in which each method is either a command performing a certain action or a query returning data to the caller. Both commands and queries are performed independently from each other. In his own words, “asking a question should not change the answer”. This concept was picked up later on by Greg Young and Udi Dahan, who merged it with ideas out of Domain Driven Design (DDD) by Eric Evans [6] and combined this to create the CQRS pattern [23].

In a nutshell, the CQRS pattern is only about creating two subsystems, as can be seen in figure 1. From the user interfaces commands can be sent to the command manager or queries can be send to or received from the query manager. Commands are actions that will be performed on the data, while queries are requests for data to be shown. The CQRS pattern itself does not prescribe anything about communication between the command manager and the query manager, but there is a collection of patterns often used in combination with CQRS that take care of communication. An often applied pattern within CQRS for communication is communication through events, which will be elaborated on in section 5.1.

Currently there is an active community of developers, architects and enthusiasts working with the CQRS pattern, but it has not yet penetrated the broadly applied and widely known collection of software patterns described in the work of Gamma et al. [7] and the Pattern-Oriented Software Archi-
A Case Study of the Variability Consequences of the CQRS Pattern in Online Business Software

Several frameworks like NCQRS (github.com/ncqrs), Axon (github.com/axonframework) and Lokad (github.com/Lokad/lokad-cqrs) however, helping developers to implement the CQRS pattern in several languages are released in the last two years, making the CQRS pattern increasingly popular. The documentation and community around these frameworks are also an important source of knowledge related to the CQRS pattern.

3. RESEARCH APPROACH

In order to gather all data related to this research, a case study was performed at large ERP software vendor from the Netherlands having approximately 10,000 small and medium enterprises using their current online bookkeeping software product on a day to day basis. From now on we will refer to the case company as ERPSOFT. During this case study, (1) several CQRS information sessions for employees at ERPSOFT in which the principles of CQRS and are explained, emphasising on the particular implementation within ERPSOFT were attended. We (2) actively participated in the architecture team for a total of 40 hours in which we also (2a) conducted five interviews with architects working at ERPSOFT on the implementation of CQRS within their software product and its consequences on scalability and variability. Within these interviews, architecture design artifacts were discussed and shared with the author. Finally, (2b) all results of the interviews were analysed within the architecture team to have a constant feedback loop for interpreting the architecture and consequences of implementing the CQRS pattern. This constant feedback of key figures within the research area is common practice within cooperative inquiry [18] and the design science cycle of Hevner [10]. Besides gathering data within ERPSOFT, (3) all research findings are also discussed with an external expert panel consisting out of three leading CQRS experts from outside the case company. All experts are either authors of a CQRS framework or provide courses in applying the CQRS framework. The panel also actively participated in reviewing the research paper. The overview of the research approach used can be found in figure 2.

The case study is a single case case study according to the classification of Yin [22]. A case study database containing recordings of the interviews and all notes taken during the interviews was kept in order to improve the traceability and rigour of the case study research. The internal and construct validity of the research is ensured by using experts within the case company to check all artifacts and conclusions and by matching the results of the case study with expected results. A clear case study protocol was used as advised by Runesson and Höst [19] in which the planning and structure of the interviews was described.
3.1 Research Questions

The main research goal "How can the CQRS pattern influence the variability of a software product?" will be answered by answering three related research questions (RQs). These questions are:

1. How is the CQRS pattern designed within the case company?
2. What sub patterns can be identified within the CQRS pattern?
3. How do the different sub patterns influence the variability of a software product?

RQ1 is answered by using interview data and architectural design artifacts gathered at the case company during CQRS information sessions, interviews and a constant feedback during architecture team participation. The answers to RQ2 are primarily answered by using design artifacts from the case company, combined with literature and expert reviews from an external expert panel consisting of three CQRS experts. RQ3 again is answered using the interview data resulting from interviews held with architects at the case company and a review on our conclusions by an external expert panel.

4. CQRS IMPLEMENTATION

This section will report on the design of the software architecture of the main product created by ERPS. This implementation report is aimed at giving an impression of the possibilities of the CQRS pattern and related sub patterns. Currently ERPS is redesigning their software product from scratch, keeping a strict separation between all queries and commands, as indicated by the CQRS pattern. This section reports on the new software architecture designed at ERPS, so no legacy code or systems are in place.

EuroPLoP ’12, Article A6, Publication date: July 2012.
Figure 3 shows the CQRS-based design of the software architecture at ERPSoft. On the left side of the figure the User Interface is modelled from which Commands can be sent to the Command Bus (top of the figure) and Queries can be sent and received to and from the Query Bus (bottom of the figure). All arrows in the figure represent communication within the system. Whenever the communication is explicitly implemented as a command, query or event, this is indicated in the figure. All other arrows, including the double headed arrows only represent a certain form of communication, but nothing specific is specified. From the Command Bus, commands are sent to Command Handlers. The handlers perform the action indicated by the command, after which the action is stored in the Stream Store. From here all events are sent to the Event Bus, who can distribute the event among different Query Model Builders (QMBs) or route events back to the Command Bus, through Event Routers. Add the query side, different Query Model Builders listen to the events broadcasted through the Event Bus and act on certain events. The events they respond to depend on the goal of the specific builders. All built queries are stored in a Query Store, for easy access by both the Query Model Builders and Query Handlers. The Query Handlers can publish query results to the Query Bus, which can be used by the User Interface to display certain information.

A further analysis of sub patterns that can be observed within the CQRS-based software architecture design can be found in section 5.
5. CQRS SUB PATTERNS
The CQRS pattern can be extended and complemented by applying several additional patterns. The number of sub patterns that can be applied within the CQRS pattern are numerous, but this research focusses on the selection of patterns we observed within the case company.

5.1 Event Sourcing
One of the possibilities within the CQRS pattern that can play a big role in terms of scalability is the sourcing of the events created by the command manager. These events can be sent to an event bus to which the query model builders in the query manager listen. A query model builder is a different sub pattern that is able of translating events to appropriate data views, which is discussed in more depth in section 5.5. The different query builders can be on the same system, but also on different physical or virtual machines. Query model builders can be on different geographical locations or even at clients. Because of this, the system becomes scalable and all sub parts are specially geared towards the task they have to do (ie. read or write) [23]. Please see figure 4a for a representation of the Event Sourcing pattern. The most important aspect of the event sourcing pattern is the fact different events are broadcasted by the command manager to be processed by different components.

5.2 Event Store
Events in the CQRS pattern do not necessarily need to be stored in any way. They could be sent to the query manager immediately after being processed and never be stored (as can be seen in figure 4a). The query manager then can do with the event whatever is necessary to get the data in an appropriate form. Because the storage by the query manager could be anything from stored in cache to stored at the client or in some database you can not rely on the availability and recovery of data if the system crashes. Because of this, an often implemented pattern within CQRS is the use of an event store. In this store all events can be stored sequentially, so the all data can be reconstructed based on the events.
in case of a system crash [17]. Figure 4b shows the representation of the event store pattern. From the User Interface commands are sent to a handler (see section 5.4 for more information on command or query handlers) who sends it to the event store as an event.

5.3 Aggregate Root
Because data in the CQRS pattern is created by different query model builders of which you do not necessarily know what or where they are, and because of the asynchronous way these listeners work you can not say anything about the correctness of data at the time of querying. As an example, think about a large web shop selling laptops. Whenever someone wants to order a laptop, the system needs to know whether the inventory is sufficient to approve the order. In other words, the system needs to be sure there is at least one laptop available before the order can be processed. In the core CQRS pattern, there is no way to know for sure the laptop is in stock, because all events are processed asynchronous. The only way to know for sure the laptop is in stock, is to store the number of laptops available together with with the laptop itself and also process this as one. If not, it is possible that the system checks whether a laptop is in stock, sees one laptop in stock, starts processing the order and ends up with an erroneous order since the laptop is sold just before through another process. The concept of storing and processing all properties and entities that are dependent on each other together is know as aggregation. The main entity is called the entity root. An order, for example should always be processed together with its order lines, since the lines make no sense without the order. In the previously mentioned example, the order and order lines are an aggregate and the order is the aggregate root, since deleting the root would indicate deleting the other entities as well.

5.4 Command Handler
Commands coming in from the user interface have to be passed through to something that will perform the action dictated by the command. As discussed in section 5.3, these actions can be adequately performed by Aggregate roots, as observed within the design of ERPSoft. The command coming from the command bus has to be interpreted and translated somehow before it can be performed. A command handler is capable of catching one or more commands and passing it through to an object capable of performing the command [1].

Figure 5 shows an overview of the command handler pattern. The action performer in the figure should some how make sure an action is actually performed. One way of doing this is using a two phase commit [9] in which a request is sent in two phases, but since this adds significant load to the
system, other methods like delaying the sourcing of events until an aggregate root is totally finished are recommendable.

5.5 Query Model Builder

All events that are sent to the query manager can be caught by a query model builder, as discussed in section 4a. These query model builders can be everywhere from the clients cache to on all kind of different physical servers. The QMBs listen to events coming in through the event bus, and create a view of the data needed by the query manager. This view totally depends on the domain the QMB is in and the goal the data has. A QMB in a system responsible for generating inventories, for example, will build entirely different query models than a QMB in a system responsible a displaying the contact details of one person. Figure 6 shows a representation in combination with the query handler pattern discussed in section 5.6.

5.6 Query Handler

Queries sent by the user interface should be translated somehow in order to know what should be send back. The QMB only creates views of the data, but does not know how to relate this to the user interface. The use of a query handler can solve this problem by implementing a component able of receiving all queries and checking the query store for views created by the QMB [21].

![Fig. 6: QMB and Query Handler Sub Pattern](image)

Figure 6 shows a combination between the QMB pattern (section 5.5) and the query handler pattern. The concept of a query store is introduced to store queries build by the QMB. This store is not obligatory, but can improve the response time of the system.

5.7 Snapshotting

It is common practice in the CQRS pattern to only store changes (events) and no states. This is because states can always be determined based on all the changes happened in the system so far. Rerunning all events will bring the system back in its last state after a possible system crash. States only occur in aggregate roots (see section 5.3), but recovering the state of an aggregate root after a system crash can be quite intensive, since aggregate roots often stay active in the system for a long time. A solution to this problem is the use of snapshotting. In the snapshotting pattern, the state of the aggregate root is stored together with the events every $n^{th}$ event. The exact value of $n$ depends on the processing
load storing and monitoring the state of the aggregate root gives. When the system crashes, the latest stored state is recovered and only the events happened after this state storage have to be rerun. The snapshotting pattern is often used in combination with the memento pattern [7] that provides the ability to restore objects to their previous state.

6. VARIABILITY INFLUENCES

Applying the CQRS pattern in a software product does not immediately influence the level of variability of a software product. Applying CQRS in combination with sub patterns identified in this case study however, does have a positive effect on the variability level of a software product. On a functional level, it becomes possible to comply to specific customer groups or branches of industry having their own specific requirements.

![Diagram showing variability due to CQRS](image)

Figure 7 shows how applying the Event Sourcing and Event Store patterns (section 5.1) can help in offering specific functionality to different industrial sectors. The example is an adapted version of a design observed at ERPSoft. The figure shows three different (A, B and C) sectors, but this can differ per implementation. In the system, one core system is created containing the functionality that is shared by all sub systems. For example, a CRM system, having specific sub systems for sectors like retail, furniture and bakeries. All domains would share names and addresses for customers, so commands related to this would be performed by Aggregate Roots in the core CRM system. All branches would listen to events broadcasted by the core CRM system and build query models based on events that are relevant for them. Operations on attributes or entities that only exist on one of the sub systems (for example a membership card number for retail) will only be processed within the specific sub system. The core system should only receive commands and does not have to be able to process queries, since the sector specific sub systems handle the representation of specific data. By identifying the different requirements of customers and grouping them in different sub systems, the level of variability possible within a software product will be high.

The possibility of running Query Model Builders (section 5.5) at the client side, also opens possibilities for customer specific requirements that are not shared with other customers. Custom QMBs and Query Handlers can be developed and implemented at customers, allowing them to perform the specific task needed for their business process. The customer specific listeners listen to events broadcasted and can react in a way specific for the wishes of one customer. The location of deployment does not play a role, making it possible to run QMBs at the customer, but also at third parties. Overall, as observed
within ERPSoft, the CQRS pattern enables software vendors to create a software product better capable of complying to all kind of different customer requirements and by this achieve a high level of variability. This is primarily caused by the possibility to distribute events to specific event listeners and the ability to handle those events in a way that can be customer or customer type specific.

7. DISCUSSION AND FUTURE RESEARCH

Software products designed according to CQRS and sub patterns identified in this research can profit from an optimal configuration of data stores in such a way that it is geared towards a specific task (i.e. storing or reading data). By this, the CAP theorem can be less of a problem than it would have been if one data store had to do all tasks. The distributed asynchronous way in which events are handled, is primarily useful for business software product having a high concurrent load or a high need for variability. Business products, as analyzed within our case study, have both of the characteristics described above and will benefit from applying the CQRS pattern, including identified sub patterns, in terms of scalability, performance during load peaks and the level of variability.

The sub patterns reported on in the paper are all based on the patterns applied within the case company, cross checked with patterns currently described in CQRS related literature. The selection of patterns described is not a complete set of patterns related to CQRS. More and different patterns exist, but the patterns identified in this research are those used within ERPSoft. Further more extensive research at additional case companies can extend the collection of CQRS sub patterns and create an even more complete overview of CQRS related patterns.

Future research should also make clear when architects should choose certain patterns and how these different sub patterns work together to achieve some common goal. The characteristics of all patterns should be more extensively evaluated by domain expert to create a complete catalogue of all CQRS related patterns.

8. CONCLUSION

As the case study and variability example illustrates, the CQRS pattern can help in achieving a high level of variability in a software product. Different sub patterns are identified related to CQRS to solve specific problems within a CQRS based architecture design. This paper helps software architects by explaining the different sub patterns and showing how they can influence the variability off a software product. We showed an implementation of the CQRS pattern, including seven sub patterns that are observed at ERPSoft. This example shows how implementing a CQRS based architecture instead of a multi-tier architecture can help in creating a software product capable of serving thousands of customers with variable product requirements. All identified sub patterns can be implemented together or individually to create, but none of them are obligatory for implementing the CQRS pattern. Some sub patterns, like the Event Sourcing pattern and the use of distributed query model builders, can contribute directly to the variability of a software product in a significant way. Other sub patterns however have a supporting role for the architecture, dealing with scalability, performance or consistency of the system. There is no perfect combination of sub patterns when it comes to CQRS since everything specific situation differs, but the pattern descriptions in this paper help in making a weighed decision for software architects.

Acknowledgement

Many thanks goes to our case company, their helpfulness and their lead software architect. We also want to thank Mark Nijhof, Udi Dahan and Greg Young for their input and Uwe van Heesch for shepherding our paper.
REFERENCES


Highly Scalable, Ultra-Fast and Lots of Choices

A Pattern Approach to NoSQL

Tim Wellhausen
kontakt@tim-wellhausen.de
http://www.tim-wellhausen.de

November 07, 2012

Introduction

The main motivation to evaluate a non-relational database system often comes from non-functional requirements on performance and scalability that your favorite relational database system cannot support, at least not at reasonable cost.

NoSQL products promise to scale structured data storage beyond the limits of relational database systems. While this has already proven true for quite a number of applications, the ability to serve a huge number of concurrent users and great volumes of data comes at a price.

First of all, the term “NoSQL” encompasses a variety of approaches to store and retrieve data. Coming from the world of relational database systems, you will quickly realize that standards such as SQL do not exist (at least not yet). In fact, many NoSQL products have little in common other than that they are different from relational databases.

Fortunately, there are some commonalities among groups of NoSQL databases. The goal of this paper is to clearly distinguish the different types of NoSQL data stores. In reality, there are products that combine multiple approaches. For an introduction into the world of non-relational database systems, this paper emphasizes the individual advantages and drawbacks of each approach.

This paper does not go into details to explain advanced features. Neither does it contain an overview of the currently available NoSQL products since these products will change quickly. This paper rather aims at general insights into NoSQL data stores, which will hopefully remain valid for a longer period.

The data store types are described as patterns. The pattern approach has been taken to put emphasis on the question why you would choose a specific data store type. In other words, you will get to know what problem you should have so that a specific data store type is an appropriate solution.

If you want to quickly skim over the paper, just read the highlighted paragraphs. The paragraphs in italic explain the context in which the patterns can be applied. The paragraphs in bold face contain the main problem and solution statements. The paper ends with a couple of things to be aware of when introducing a NoSQL database for the first time and some final remarks.

The most basic concept of non-relational data storage is the Key/Value Store. Such a database stores little-structured data that is accessible by their keys only. Several variants exist that form groups of their own: Blob Stores are specialized to hold large binary data. Column Family Stores persist values of the same type closely together for better analyzing capabilities. Document Stores provide means to easily create rich domain models. Graph Databases rely on a different concept, focusing on the relations between data entries.
Key/Value Store

*Your domain model or an important part of it is rather simple, i.e. there are few interdependencies and constraints. But the application’s data changes often or your application generates a huge amount of it.*

There are many cases when you need to store data that is not critical to the application, but you’ve got plenty of it. Think about statistical data, status information of an online game or similar scenarios where small chunks of data are written often and read more or less often. Using a relational database in such cases does not scale well because of transactional overhead or database locks.

**Speed is your main concern when reading and writing data. Your application must react quickly even at peak time. The ability to scale well is more important than data integrity and the ability to comfortably query data.**

Typical features of relational database systems (such as transactional support and referential integrity) are not important to you. At least, they are less important than throughput. But your data still needs to be stored safely and high availability is a key issue.

Your domain model is not highly complex and your application’s data has little structure, but you need at least support for some data types.

Your application’s data might change rapidly, triggered by many simultaneous users, but read operations also need to execute quickly.

Therefore:

Choose a **Key/Value Store**, which stores little-structured data associated with unique keys.

![Key/Value Store Diagram]

The general idea of a **Key/Value Store** is to dispose of the concept of highly structured tables, columns and values in favor of a very simple programming interface: You associate simple data values such as numbers or texts with unique keys, and the database very efficiently stores and retrieves your data. Some **Key/Value Stores** provide operations to manipulate structures such as lists and sets whereas other **Key/Value Stores** only support basic or no explicit data types.

This simplicity provides the basis for supreme scalability. Most **Key/Value Stores** keep all data in memory all the time to achieve high performance. Their strategies to write data to disk and to distribute data within a cluster differ significantly. Keeping all data in memory limits the amount of data that every node can hold. Unless you are able to set up large clusters, this may also limit the maximum size of the stored data.

**Key/Value Stores** typically transfer the responsibility for the creation of unique keys to the application. It can be difficult to find a good strategy to choose appropriate keys (i.e. both meaningful and unique).

If you need to search for values, **Key/Value Stores** typically provide little comfort. The programming interface to a **Key/Value Store** is much more limited than what you might be used to from object-relational mapping tools, for example.

There are several variants of **Key/Value Stores** that focus on specific application development use cases and provide more sophisticated functionality. If you need to store large binary data, consider using a **Blob Store**. If you need to analyze large amounts of data at the same time as it is gathered, a **Column Family Store** might be a better option. If your application consists of a rich domain model, a **Document Store** might be appropriate.

Examples of **Key/Value Stores** are Memcached ([6]), Couchbase ([7]), Redis ([8]) and Riak ([9]).
Blob Store

Your application relies on binary data that needs to be distributed to the application’s users.

Think about web sites with plenty of large images. If the content of your web site changes regularly, the images often do so, too. Statically bundling the images with your application is not an option because of their size and because every image change would cause a re-deployment.

You need to dynamically store and publish huge amounts of binary data such as images.

Relational databases provide the data type BLOB to store binary data in tables. But if you keep many and potentially large BLOB entries in a relational database, you quickly drive the database to its limits. In addition, relational databases typically cannot serve their content directly to end users (and you wouldn’t want them to do so). So you need to stream the BLOB contents from the database through an application server, which puts load on two critical system components.

You could store binary data in a file system and set up a web server to make these files accessible. But large numbers of files and big files push many file systems to their limits. Moreover, in case of many simultaneous requests, the underlying storage system may soon become the bottleneck. Caching might help, but web servers are not optimized for caching large numbers of large files. In addition, neither file systems nor web servers provide means to manage files, e.g. they don’t create unique keys.

To overcome the caching problem, you could employ a content delivery network, which acts as a global proxy. Requests for your files are handled by the content delivery network and not by your own system. But that would add another layer of complexity, which makes the system difficult to test. And you still need to manage the files in your file system first. Additionally, you must expect significant delays in distributing your files to all nodes of the content delivery network.

Therefore:

Choose a Blob Store, which is specialized to manage a large number of potentially large files.

A Blob Store keeps the complexity of actually storing the data away from you. At the same time, a Blob Store is optimized to provide the content to your client applications, typically via http URLs.

A Blob Store associates every file with a unique key, by which the file can be accessed. Keys can be defined by your application or they can be automatically generated. Blob Stores provide little search capabilities. You need to keep track of your binary data in your main application.

Setting up and running a Blob Store is not trivial. Therefore, Blob Stores are often used as a service. Some Blob Store products (or rather vendors) physically distribute your files around the globe. You still have a single point of access to manage your data from within your application. But accessing the files becomes a (geographically) local operation from many points in the world. A Blob Store (again: as a service from a vendor) may therefore avoid the need to use a dedicated content delivery network.

On the downside, Blob Stores are like an attic into which you store everything that does not fit elsewhere. After a while, you might lose track of what is inside and what is not. Blob Stores are typically bad at categorizing your data – you’ve only got an identifier. And because Blob Stores are specialized at binary data only, they do not provide a means to store all your data. Rather, they add complexity to your data storage architecture.

The main difference to Key/Value Stores is that Blob Stores are made to distribute binary data to users whereas Key/Value Stores are made to handle frequent but small application data changes.

Examples of Blob Stores are Amazon S3 ([10]) and the Blob service of Windows Azure ([11]).
Column Family Store

Your application collects masses of little-structured data that is read more often than written.

Your application may evaluate metrics about user operations or generate reports on business data. This kind of functionality typically includes analytical operations such as calculating aggregates and minimum or maximum values.

Your application needs quick response times when analyzing masses of data. After some data has been collected, the data must show quickly in the analysis.

The traditional solution to analyze data is to set up a data warehouse. The data is stored in a normalized schema in one relational database. For the analysis, it is transformed into a denormalized star schema and transferred into a separate database. Queries on a data warehouse are fast and don’t harm the main database’s performance. But extracting and transferring data from one database to another is complicated, inefficient and error-prone. The transfer itself is time-consuming so that it may take a long time until new or updated data shows up.

Without a data warehouse, you would need to analyze the data in the main database. But calculating aggregates in a relational database leads to full table scans. Such operations are not only slow themselves but also harm the performance of parallel requests, in particular write requests.

Apart from that, you could pre-calculate the requested data and store the results separately. But refreshing the calculations is costly, in particular if masses of data are involved, and it might take a while until new data shows up.

In addition, you might need to change the set of collected data regularly. But changing the schema of a relational database can lock a relational database for hours if a table is already very large.

Therefore:

Choose a Column Family Store, which can store masses of structured data by partitioning the data at a column-level. All read operations and calculations on individual columns are very fast. Adding and removing columns are cheap operations.

A Column Family Store keeps data in columns rather than rows. The values of a column are kept in a contiguous space on the physical storage. The database can quickly retrieve all values of a column and perform operations on them. The less the values in a column vary, the better the content of a column can be compressed. The compression alone can lead to an improvement in an order of magnitude to row-based data stores.

New columns can easily be added and existing columns can be easily deleted even if the database already holds large amounts of data. On the downside, the values of a data set are spread among several columns. Inserting, updating or reading the complete data set can be more time-consuming than in a relational database as the data columns may even be distributed even among multiple machines.

A Column Family Store can be seen as a Key/Value Store where data is stored in a format ready to be consumed and analyzed very efficiently.

Examples of Column Family Stores are HBase ([12]) and Cassandra ([13]).
Document Store

Your application has a rich domain model and does not collect huge masses of data.

When data is stored in a relational database, a normalized entity-relationship schema is the typical way to go. A normalized data structure has many merits, e.g. it guarantees the consistency of your business data. The more complex the domain model of an applications becomes, the more tables are needed to store the data.

Given a rich and complex domain model, you need to efficiently search for and modify your domain entities. But mapping a complex domain model to a relational structure often leads to slow performance.

When a complex domain model is mapped to a normalized data structure in a relational database, you typically get a large number of tables and foreign key constraints in between. Querying such a data structure causes many join operations. But with concurrent read and write access, too many database joins quickly demolish the overall retrieval performance. Still, you want to keep your rich domain model instead of relying on a simple domain model as provided by a Key/Value Store.

The ability to query data by searching for occurrences of business values is important to your application but following relations among entities that are stored in separate tables of a relational database is an expensive operation.

An expressive, preferably object-oriented domain model improves the understandability of the business domain but mapping such a model on a fine-granular data structure may cause performance degradation. The results are often a large number of round-trips to the data store and therefore inefficient load operations.

Therefore:

Choose a Document Store, which manages hierarchically structured data records. Modifying and retrieving such records are a cheap operations.

A Document Store keeps all information related to a single entity in one document. The complete data of a document can be stored and retrieved atomically. Whereas in a relational database, you need to create and fill multiple tables to model one-to-many relationships, you can easily store such data in single documents.

Typically, data records in Document Stores are saved in a JSON-like data structure. JSON, which is short for JavaScript Object Notation ([19]), can hold any kind of hierarchically structured data. Developers who have developed Rich Internet Applications are typically quite familiar with this data format.

Mapping an object-graph to a JSON notation and back is easy and fast, both done manually and by libraries. On the downside, a JSON-based structure cannot model many-to-many relationships. In addition, some details such as non-standardized data types (e.g. the format of dates) may cause problems in creating a rich data model based on JSON.

Document Stores work best if the entities of your domain model are mostly Aggregates, as propagated by Domain-Driven Design ([20]). An aggregate entity is a complex data structure to which there is only a single point of reference.
The ability to express queries for entities based on their internal data structure is an essential feature of all Document Stores. Still, not all such products provide a dynamic query language to execute search requests. Some products mandate that you develop queries outside your application code in a different programming language (mostly JavaScript) as Map-Reduce ([21]) functions.

As another downside, few products provide the capability to span queries over several data records, i.e. joining them. Also, you typically cannot easily prefetch data that is stored in multiple data records in a single call.

Document Stores can be seen as Key/Value Stores with extra capabilities to structure and query the data. In reality, there is a smooth transition between both. Some products that started as basic Key/Value Stores slowly grow to become more like Document Stores.

Examples of Document Stores are MongoDB ([14]) and CouchDB ([15]).
Graph Database

Your application needs to dynamically manage and evaluate relationships between your entities.

For example, your application keeps track of who of your users knows whom and you would like to show common friends of any two users. Or, a mobile application needs to show how a specific location can be reached via public transport from the current location, etc.

Your application focuses on relationships between entities more than on the actual entities and their properties. The application needs to traverse the relationships of a large number of entities efficiently.

Storing relationships between entities is exactly what relational databases are made for. But querying recursive or transitive relationships in large entity sets is difficult due to limitations in SQL and the way relational databases actually store entity data.

Efficient traversal algorithms are the key to a high query performance, but implementing and optimizing such algorithms for large sets of data is a difficult task.

Therefore:

Choose a Graph Database, which make it possible to explicitly model the relationships between entities as a graph and provides means to efficiently traverse the graph.

Graph Databases offer search capabilities among entity relations better than any other database type. Traversal along the edges of large graphs can be done very fast. Graph Databases typically provide a set of standard graph search algorithms to aid the development of efficient queries.

The performance of searching within large graphs is very high if the database can execute a search request in a single call. If your application needs to traverse the graph explicitly, every traversal step may lead to an inter-process round-trip with the potential to harm performance significantly.

If you need to search nodes by their properties, Graph Databases typically provide some kind of support, for example in form of indexes. Be aware, however, that Graph Databases are not made to search for sets of nodes by their properties. Their support for such kinds of searches often is very limited. For such use cases, you may need to employ additional technologies such as full text search engines on top of the Graph Database.

Every domain model that is based on a relational or a key/value structure can be transformed into a graph. In theory, it is therefore possible to migrate any domain model to a graph database and to benefit from the search capabilities of graph databases. Even if a relational database or Key/Value Store remains master of the data, you could periodically load a snapshot of the data into a Graph Database to perform graph-related search requests there (although the creation and transfer of such a snapshot may be a costly operation by itself).

It may be difficult to partition a graph of data among several servers to support clustering in case a single server does not support the amount of data or number of requests.

Examples of Graph Databases are Neo4j ([16]), Sones ([17]) and OrientDB ([18]).

B1-7
A couple of things to be aware of

Although NoSQL databases are in fashion nowadays, the decision how to store an application’s data is difficult. And there are more obstacles to be aware of.

A NoSQL database enforces you to learn a technology that often differs significantly from a relational database. You need to quickly build up knowledge or spend money on support or for training sessions. As most of the NoSQL product are quite new and few experienced developers exist, expect difficulties in finding appropriate assistance.

Even if the developers in your team are eager to learn new technologies, errors and wrong decisions that hurt the project’s progress will happen. Do not assume that such a technological change will happen smoothly. Developers need to learn new APIs, administrators need to learn new tools and analysts need to understand the implications on domain modeling.

While you gain many powerful new features, you probably also lose features that you relied on so far. The implications of the loss of these features (such as referential integrity or transactional support) might not be fully understood at the beginning.

You may also lose the power of integration libraries that, for example, simplify the creation of an object-oriented domain layer on top of a non-object-oriented data store. The available libraries to integrate a specific product into your development environment (i.e. programming language, IDE, etc.) may lack all but the most essential support.

If you are part of a company where the development of an application is strictly separated from the operation of the application, you need to approach the operations team early. These people are responsible to keep the application running and they also need to learn and employ new tools to setup, monitor and backup the data store. They may even veto the introduction of a technology that is yet unknown to them.

You might be used to set up and manage a production environment where server applications are run on powerful, non-clustered machines. Do not underestimate the complexity of handling and managing server clusters – both in terms of software processes and hardware. For example, if you create a cluster based on commodity hardware, you have to accept and cope with failures of all core components.

Check out early how support is given for your chosen product. Many NoSQL products are supported by their developers on mailing lists or in forums. If severe problems happen (both at development time and in production), you may find out that no one feels responsible to help you solve your problem. Commercial support may not be available as you may be used to.

Final remarks

In general, you have little choice but to employ a NoSQL data store if the expected size of persistent data or the number of simultaneous requests exceed the capabilities of your relational database system. However, even if a relational database might still work for your project, NoSQL data stores provide value. So you could still decide on taking a NoSQL product; you just need to be aware of the consequences.

NoSQL databases provide a wide range of solutions when you need highly scalable data storage ([1], [2]). But there is no all-in-one solution available that promises to replace the currently dominating relational databases systems. Rather, most NoSQL databases are best suited to address specific problems and use cases ([3]).

While the data storage type of a NoSQL product is an important decision criterion, there are more criteria to consider. The CAP theorem ([22]) allows different tradeoffs to make, i.e. whether a product restricts the consistency, availability or partition tolerance of the system. Furthermore, different products support different clustering and replication strategies that may affect an application’s general architecture. All of these criteria (and several more) are out of scope of this paper and deserve papers of their own.

B1-8
The term *polyglot persistence* ([4], [5]) expresses the fact that a single data storage may not suit all needs of an application. Instead, an application might encompass several different approaches to data persistence. To give an example: while you would probably keep financial records in a relational database, catalog data that describes a variety of products may be better kept in a Document Store. Session information of the users of a web site may be best stored in a Key/Value Store. Data that tracks the users’ behavior for latter analysis is a candidate to be kept in a Column Family Store. Add a Blob Store for binary data such as images and a Graph Database to model and analyze the relations between your users and you’ve got a very rich data persistence landscape.

**Acknowledgments**

Cyrille Martraire gave a lot of constructive and valuable feedback as shepherd for this paper previous to the conference EuroPLoP 2012. During the conference, the paper was workshopped by Veli-Pekka Eloranta, Ioanna Lytra, Christopher Preschern and Vallidevi Krishnamurthy who provided even more suggestions for improvement.

**References**

[1] A catalog of NoSQL databases  
http://nosql-database.org/

[2] Alex Popescu: myNoSQL – A Blog on a variety of NoSQL topics,  
http://nosql.mypopescu.com/

[3] Kristof Kovacs: Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs Membase vs Neo4j,  
http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis

[4] Scott Leberknight: Polyglot Persistence,  
http://www.nearinfinity.com/blogs/scott_leberknight/polyglot_persistence.html

http://martinfowler.com/bliki/PolyglotPersistence.html


[7] Couchbase:  
http://www.couchbase.com/membase


[16] Neo4j: http://neo4j.org/

[17] Sones: http://www.sones.de/home


Human Machine Interface Patterns For Distributed Machine Control Systems

Veli-Pekka Eloranta, Tampere University of Technology
Marko Leppänen, Tampere University of Technology

As the control systems are becoming complex and there is more and more information that is handled in the system, the need for graphical user interfaces providing good user experience increases. Human machine interface (HMI) in the control system has some unique characteristics which affect its design. In the control cabin there is physical control equipment such as joysticks and hand panels. In addition, there is usually a screen that is used to display the graphical user interface showing machine’s status information. As control equipment is critical in the sense of response time and the graphical user interface is a non-real-time application, new challenges for the system design arise. In this paper, we present five patterns that tackle the challenges in HMI design for control systems.

Categories and Subject Descriptors: D.2.11 [Software Engineering]: Software Architectures—Patterns; H.5.2 [Information interfaces and presentation]: User interfaces—Graphical user interfaces (GUI); C.3 [Computer Systems Organization]: Special-purpose and Application-based Systems—Real-time and embedded systems

Additional Key Words and Phrases: Architecture patterns, Distributed control systems, Human machine interface, User interface design, Pattern language

1. INTRODUCTION

The main goal when designing distributed control systems is to increase productivity of a machine. To accomplish this, the control system software has to offer enhanced functionality when compared to older non-software controlled systems. Human-machine interface (HMI) plays a crucial role in this. HMI must enable the machine operator to fluently control the machine by providing information about the system state. Systems are also becoming more complex and HMI must adapt to this changing environment. Graphical user interfaces must be able to provide more and more information to machine operator. Furthermore, different user groups must be supported. Maintenance personnel must have access to system parameters. On the other hand, the machine operator may not be interested in this kind of information. In some cases, access to information must be limited in order to make sure that the machine operator does not make changes that may void the guarantee or break the machine. On the other hand, production personnel at the factory should have access to all system features.

In this paper, we present five design patterns for designing HMIs for distributed control systems. These patterns are part of a pattern language for distributed control systems consisting of 72 patterns in total. Patterns have been collected during 2008-2011. Initially, the patterns were mined while carrying out architectural evaluations in Finnish machine industry. After the patterns were formulated, we took them back to the companies and interviewed the architects. In this way, we gained knowledge which patterns were really used and which not. Furthermore, we gained more information how patterns were typically implemented. Companies that participated in the pattern mining process are global manufactures of work machines (e.g. mining drills and forest harversers) and process automation systems.

Author’s address: Veli-Pekka Eloranta, Korkeakoulunkatu 1 (33720 Tampere), P.O. Box 553, 33101 Tampere, Finland; email: veli-pekka.eloranta@tut.fi;
2. THE BIG PICTURE

The patterns presented in this paper are a part of a larger body of literature. The five HMI patterns presented in paper belong to a pattern language for distributed machine control systems. The whole pattern language consists of approximately 89 patterns and builds the control system software architecture for work machines. Some of those patterns are available in [Eloranta et al. 2010]. HMI patterns form their own sublanguage which is targeted for the graphical user interface (GUI) of the machine. Graphical user interface in this context includes the display that shows user interface elements and control equipments in the cabin of the machine. Control equipment can comprise of joysticks, steering wheel, hand panel button, dashboard buttons, etc.

In general, HMI and user interface patterns is a large area of study and already widely documented, see e.g. [Tidwell 2005]. Thus this work is limited to the human-machine interface patterns that are typical to the domain of distributed control systems. Other user interface design patterns can be applied to these systems as well, but the patterns presented in this paper are specific for the domain. In other words, it might be hard to apply these for the design of user interface for traditional work station application.

Figure 1 illustrates the relationships of the HMI patterns. Grayed patterns are presented in this paper. Other patterns in Fig. 1 are shown to give a context for these patterns. CONTROL SYSTEM is a root pattern for the whole language. It basically describes the motivation, why to have a control system in the machine in the first place. HUMAN-MACHINE INTERFACE pattern is the root of sublanguage presented in this paper. It is also architecture pattern that affects the system's architecture. The rest of the patterns presented in this paper are more like patterns for UI design. ISOLATE FUNCTIONALITIES, SEPARATE REAL-TIME and CONTROL SYSTEM OPTIONS are illustrated to show the relationships of the presented patterns to the rest of the language. The arrow in the pattern language represents "refines" relationship between the patterns. For example, HUMAN-MACHINE INTERFACE is refined by FEEDBACK NEAR ACTION pattern. This means that some problems remain unsolved after applying

---

Fig. 1: HMI pattern language for distributed machine control systems. Grayed patterns are presented in this paper.
the Human-Machine Interface pattern and the next pattern can be used to solve some of these remaining problems.

Table I (Appendix A) shows the patlets of the patterns presented in this paper. Table II in Appendix A gives an overview on patterns which are referenced in this paper but not presented in full detail. Some of those patterns can be found online [Eloranta et al. 2010] There are many other user interface design patterns available, but in this paper we focus on patterns which are typical to the domain of distributed control systems.

3. PATTERNS

We present the HMI patterns in this section using pattern format which is a combination of Portland form [Portland Pattern Repository 2003] and Alexandrian form [Portland Pattern Repository 2011]. First we describe the context and then present the problem in **bold** font face. Next we explain the forces and give the essential part of the solution in **bold** font face. Followed by the discussion of the solution. Finally, we present the consequences and known usage separated with three stars. All patterns include a sketch of the solution and sketch icon for the pattern as well. Pattern names are written in SMALL CAPS throughout the paper.

3.1 Human-Machine Interface

...in control system the operator needs to know the current status of the machine and information about the process to efficiently operate the machine. Operator uses joysticks, handpanels, dashboard buttons, etc to control the machine. These controls are typically physical controls such as joysticks, but in some cases also touch displays are used. To save cabin space and to build the machine cost-effectively, some meters and gauges might be implemented with software. However, controlling the machine should have short response times and high reliability implying that HMI should work in real-time. Now, if status displays and graphical user interfaces are required, the high end node providing GUI (see separate real-time) may not be able to provide real-time functionality.

**The control system should provide the machine operator feedback and a way to operate the system.**

![Sketch of a control system with gauges and buttons.]

There should be a way to show information about system’s status, task descriptions, etc, to make the machine more user friendly. Furthermore, if the machine could guide its operator during the usage what to do next or how the operator should change the controlling style, the productivity and efficiency of the machine operator could be increased.

The machine should be operable even if non-real-time part described in separate real-time is malfunctioning. Graphical applications are typically run on non-real-time part of the machine, so graphical user interface should reside on non-real-time part. However, the machine should be operable even when the GUI is not available for the machine operator.

One display showing gauges on screen is usually cheaper than multiple physical gauges placed in the dashboard.
There should be a way to provide fine-grained information about the system state. In many cases, traditional
gauges can not show detailed enough information. For example, in passenger car dashboard light is typically
used to indicate a fault in the system. However, it can not show what the fault is. But when control system is used
there is more information on the fault available. This information should be presented to the driver in convenient
form. Furthermore, accuracy of traditional gauges may not be enough. There should be a way to present the
approximate magnitude of the value and the exact value.

It should be possible to use commonly used technologies in the implementation as the life cycle of the ma-
chine is long.

**Therefore:**

**Add a human-machine interface. It consists of ways of presenting information and controls to manip-
ulate the machine. These typically are displays with GUIs, buzzers, joysticks and buttons etc. Separate
the way of presenting the information from the controls of the machine with a separate HMI bus.**

If **Separate Real-Time** has been applied, the natural place for the GUI is in the high end node. Typically
this high end node has enough processing power to run the GUI. However, other control equipment should be
available even when the high end node is not. Therefore, it is advisable to create a HMI bus which is connected via
a HMI bus master to the main bus (see **MESSAGE BUS**) of the machine. All control equipments (including the node
running GUI) should be connected to this bus. In this way, control equipment and GUI has short latency between
them and the amount of the traffic on the main bus is not needlessly increased. Furthermore, if the node running
the GUI malfunctions, the other control equipment is still available. Fig. 2 illustrates the solution. The solution
also enables the usage of COTS in control equipment such as joysticks, hand panels, etc. These devices can be
directly connected to the HMI bus. For example, if CAN bus is used as a HMI bus, CAN joysticks can be utilized.

A graphical user interface is used to show machine status, e.g. oil pressure or RPM as gauges. The user
interface can also show **NOTIFICATIONS** and other alarms. Additionally, work plans (e.g. in the case of mining drill,
which blast holes should be drilled next) and production reports (e.g. how many meters have been drilled) can be
shown in the UI. This kind of information increases productivity of the machine operator as work plans can be
optimized.

Machine controlling must implemented so that even if the graphical user interface is malfunctioning, the ma-
chine can be controlled and the operator can even continue working. In practice this means that there should

![Fig. 2: Illustration of typical structure of the HMI.](image)
be a HMI bus in parallel with the backbone bus. Controls are then attached to this HMI bus while graphical user interface resides on the cabin PC which is typically used to run the non-real-time application. In general, the GUI should be implemented as non-real-time application. Fig. 2 shows the typical structure of HMI.

Now, when the PC is malfunctioning, the meters and gauges are not shown on the screen and controlling the machine needs more expertise from the operator, e.g. listening to the motor or manually supervising the operations. It might be the case, that some functionalities can not be used anymore, but the machine can still be driven to the service. Additionally, some gauges can be implemented as separate CAN devices so they are available even when the GUI is not functioning.

Having graphical user interface showing gauges can lead to significant cost reductions. As GUI can have multiple views different gauges can be shown in different views using the same display. If hardware gauges would be used, each of them would increase the cost of the machine. Furthermore, cabin space can be saved as one display is used instead of multiple hardware gauges.

If the machine has multiple different user groups such as machine operator and service personnel, one should consider providing ROLE-BASED UI. Furthermore, different views in the GUI should have COMMON LOOK AND FEEL and to make the views easy to use one should consider using UPRIGHT IS OK pattern. Unintentional activation of functionalities can be prevented by using DOUBLE CONFIRMATION and feedback for the machine operator should be provided using FEEDBACK NEAR ACTION.

⋆⋆⋆

Machine operator can steer the machine using controls such as joysticks, handpanels, etc and see machine status information such as motor RPM, speed, etc from the cabin screen. This offers enhanced user experience and increases productivity.

If the non-real-time part of the machine is malfunctioning, the machine can still be controlled using the control equipment directly connected to the bus. However, the benefits of the graphical user interface, e.g. increased productivity, are lost. Furthermore, this increases safety of the machine.

By providing the graphical user interface for the machine operator, more fine-grained information can be shown. For example, information about fault situations can be shown on the display. In addition, work plans, production reports and diagnostics can be shown.

HUMAN-MACHINE INTERFACE makes the system more complex as an additional bus might be required in the cabin to ensure the availability of control functionality. Adding an additional bus also adds costs.

⋆⋆⋆

In a mining drill there are controls such as joysticks and intelligent CAN devices, i.e. they are directly connected to this cabin CAN bus. Control signals are bridged from this bus to the main bus whenever controllers attached to the main bus are interested in the value that is produced in the HMI bus. In the cabin bus there is also a PC unit attached. It will receive control signals from the control units via HMI bus, and show appropriate changes in the graphical user interface. The PC is running Linux and the GUI is implemented with Qt to ensure portability. If the PC crashes or malfunctions, it does not disable the whole system as the nodes responsible for controlling the machine are still receiving the control signals from the control equipment.
3.2 Role-based UI

In distributed CONTROL SYSTEM there is a lot of information available that needs to be shown or visualized on the display. For example, status of CONTROL SYSTEM's different processes are shown in the user interface and information about the possible faults is shown as well. Furthermore, there can be PARAMETRIZABLE VALUES that should be changed only by the trained service personnel or even by assembly personnel in the production line. So, the machine operator only needs subset of this information to control the machine. On the other hand, the service personnel may not be interested in some information that is presented during the normal usage of the machine. Machine operator needs to see all elements related to controlling the machine, whereas service personnel or assembly personnel needs other kind of information to do their jobs.

The user of the machine only needs such functionality in the UI that is required by the her job description. All other functionality should be hidden from the UI in order to make the HMI more usable.

The machine users have different kind of roles, e.g. machine operators and service personnel. Each role's job should be made as easy and efficient as possible. Furthermore, some functionalities should not be accessible by a person who hasn’t received proper training to use the functionality.

The machine is easier to operate if only information that is relevant to the user, is shown on the display. For example, mining drill operator needs information regarding the drilling process, but does not need to see the error log of the control system. However, a service person, might find the error log interesting. Furthermore, once only relevant information is shown, the machine operator can easily find the information she is interested in from the display.

Maintenance personnel or assembly team on the product line should have access to system parameters and be able to adjust them from the user interface. However, a third party service personnel should not have access to these parameters.

Access to certain information in the user interface should be limited, e.g. machine operator should not be able to do such adjustments that could possibly damage the machine or void guarantee.

Therefore:

Create an own user interface for each user group, i.e. own UI for machine operator, maintenance personnel and production line personnel. User interface activation may require dongle, password, or additional equipment.

First identify different user roles that the machine have. Typically there are at least three forementioned roles. After identifying the roles create an own UI for each role and decide the way how to change between roles. In many cases, the basic view, i.e. machine operator’s view is the default and does not require any activation. Typically
through this UI another role’s UI can be activated, for example by entering a valid password. Then some of the
ingo
time operator view’s elements may be hidden and additional elements, required by the activated role, are
shown.

The operator user interface is the default user interface that is shown when the machine is started. However, in
some cases it might be advisable to identify the operator. Especially, if OPERATOR PROFILE pattern is used,
the user identification can be used to restore the operator specific settings for the machine. The operator user
interface consists of gauges, meters and controls which are required to control the machine and to communicate
with other systems such as fleet management.

Maintenance personnels user interface is activated separately, typically from the operator user interface by entering
a password. Other possible means of activation are to use dongle or some other external equipment that is
attached to the machine. Once the user is identified, the UI corresponding to her job description is shown. This
may mean that additional features are shown in navigation menus or completely different kind of UI is shown.
From the maintenance UI, system parameters can be accessed and modified. Additionally, the maintenance UI
provides more diagnostic information of the machine, e.g. fault logs can be accessed. If the machine supports
CONTROL SYSTEM OPTIONS, the different options can be enabled from the maintenance UI. Typically also system
self-diagnostic application, input forcing (see FORCED INPUT VALUE pattern) and SENSOR BYPASSING can be
used only through the maintenance UI.

User interface for production line personnel usually contains all functionalities that the maintenance UI, but
offers some extra functionality. Once the machine is assembled and tested in the factory, there are some system
parameters that are set. These are typically saved as “factory defaults” by the assembly team. UI for Production
line personnel allows modifying these factory defaults and setting parameters that can not be changed afterwards.
Production line personnel’s UI may also allow modifying the system configuration, e.g. minimum setup of controllers
and sensors which are required to safely operate the system. Additionally, control system software installation is
usually only possible through this interface. Updates can typically be applied through maintenance personnels UI
or sometimes even through operator's UI. User interface for the production line personnel is usually enabled with
external equipment that is available only in the factory.

Operator’s user interface shows only information relevant for controlling the machine. The user experience is
enhanced as extraneous information is not shown.

The machine operator does not have access to information she is not supposed to modify. Same applies for the
maintenance person.

New options can be enabled only by authorized maintenance person. Unauthorized “Cowboy” services can
not enable options or modify the system parameters.

An additional identification mechanism for maintenance personnel and for assembly team is required. This
prevents the usage of 3rd party spare parts and service to some extent. On the other hand, the machine operator
may not be able to carry out simple service tasks as she can not access the required information or GUI elements.

Additional functionalities required by other roles than machine operator, must be taken into account in the
user interface design. It should be possible to integrate additional functionality seamlessly to the user interface in order to make the implementation process fast. For example, service person’s elements which are used to parametrize system parameters can be located in their own tab. Normally this tab is hidden, but once the maintenance UI is activated, this tab is shown. Implementing this kind of integration might require additional work.

A forest harvester has three different UIs. When the system is started, the operator user interface is shown as the default view. The operator can see machine’s status information from the UI and adjust parameters that affect her work. For example, the machine operator can bind buttons for certain functionalities, e.g. sawing a log, feeding it, etc. The maintenance person has a dongle that she can plug into the machine. When the dongle is plugged in and identified, new user interface elements are shown on the screen. Through these elements, the maintenance person can calibrate sensors or access system diagnostics application. In the factory, the assembly team can connect a computer to the machine and in this way they are able to install new version of the control system software or adjust factory default values for parameters.

3.3 Upright Is OK

...there is a machine CONTROL SYSTEM with HUMAN-MACHINE INTERFACE. The user interface is used to monitor several processes that are happening in the system. Many of these processes have measurable quantities, e.g. RPM of the motor, which are important for machine operator to see and which should stay stable during the process. Under normal operation conditions, the operator is not concerned about the actual values of the metering, but when something anomalous happens the operator should quickly react to the situation. NOTIFICATIONS could be used to inform the operator about exceptional situations which the machine detects. However, all anomalous situations may not be identified by the machine and therefore cannot be notified to the machine operator. Furthermore, in normal operation conditions operator should be able to see at a glance that everything is functioning correctly.

**Usually the machine operator is not interested in the details of the machine’s status, but just needs a quick overview that everything is functioning normally.**

There is a multitude of processes that are metered and the operator should see at a glance that everything is functioning normally. In this way, the operator does not need to concentrate on looking at the gauges, but can focus on the work at hand.

In some countries, it can not be assumed that the machine operator can read or understand the language used in the graphical user interface. Also reading takes more time than a visual overview and in fast paced environment there is not much time to read the information machine is providing but the machine operator needs to have a quick overview of the machine’s status.

Interpretation of colors as indicators of danger may vary in different cultures. It can not be assumed that red is
always sign of an exceptional situation.

Traditional looking gauges are familiar to most users. Furthermore, in many cases the exact value of variable is not interesting but the magnitude of the variable is. For example, in a passenger car it might not be good option to show RPMs as a plain number as it is not so easy to interpret than a traditional looking gauge which points the approximate magnitude of RPM.

Therefore:

Use traditional looking meters and gauges in the interface, even if the UI is implemented with graphical components. Use these meters so, that values, e.g. for oil pressure, etc are in OK range when the meter is pointing upwards. This way it is easy to take an overview that everything is OK.

The gauge needle gives the essential information with just a quick glance (see Fig. 3). If the needle is pointing upwards, everything is functioning normally and it is easy to spot if anything is out of ordinary working limits. No action is needed if all gauges point upwards. Any gauge needle that is leaning left, is below the normal operating level and any needle that points to right, is indicating that the system is running above the normal conditions and needs some attention. In some cultures, interpretation of needle positions (left and right) might differ. The gauges should be used in a way that the needle can not go around full circle. In other words, this pattern should not be used such gauges.

There can be an additional value in the digital gauge that shows the actual value of the measured variable. This value can be ignored during the normal operation, but it may help to see how quickly the variable is decreasing or increasing. For example, the driver may conclude how fast the oil pressure is dropping and if she still has time to drive for better position or is an immediate shutdown necessary.

Even if the machine operator can not read, it can be taught to her that if the gauge needle is pointing upwards, everything is OK. Once the needle shifts from upright position, the operator can call someone or do other remedying actions such as shutting down the machine. Furthermore, colors are not needed to indicate that something is wrong as NOTIFICATIONS can be utilized to show fault information. However, if the machine operator can not read, the information provided by NOTIFICATIONS maybe useless. So, it is good to have a visual way to present when something is going wrong. Colors are not very great for this as different cultures may have different colors indicating danger. In addition, machine operator might be color blind, so it might be hard to distinct different colors if they are presented close together.
The operator does not need to focus on meters in normal situation as she can see at glance that everything is functioning normally. This enables the operator to focus on the productive work. If there is an anomaly, it can be detected easily and operator can start further investigations.

People who can not read can use the machine safely as they know that when the gauge needle is pointing upward, everything is OK.

Traditional looking gauges are familiar to many people so they are easy to read. Furthermore, it is easy to see the magnitude of the quantity just be a quick look at the gauge. Accurate values would require the operator to interpret what the value means and therefore slower her response time.

It might be hard to present all monitored processes as traditional looking gauges. Sometimes the data produced by the process might be too complicated to present as gauge. Additionally, there is limited space on the display so all gauges may not fit on the same display at the same time.

The pattern can be applied only to some extent for consumables, e.g. fuel. It might be hard to indicate in fuel level meter that the upright position is the optimal situation.

In a mining drill, there are diesel motor’s RPM value, drilling speed and oil pressure shown in drilling mode as gauges in the user interface. The machine operator focuses to monitor the boom position and the drill penetration speed while drilling. Once in a while she takes a glance at the meters. As all gauge needles are in upright position, the operator knows that the machine is functioning as expected. In each gauge, there is additional range drawn in gauges that shows the normal operating area (range overlapping the upright position from both sides).

3.4 Feedback Near Action

...there is HUMAN-MACHINE INTERFACE in the work machine. It differs from traditional user interface so that the system may have a GUI, multiple control panels, screens and the operator’s chair might be rotating, so the operator is not always facing the dashboard. Additionally, when controlling the machine, the machine operator is typically very focused on the action she is carrying out. For example, when sawing a log, the user is concentrated on the log getting fed through delimbing knives in the harvester head. The operator presses a button to initiate the sawing operation when the desired log length is reached. Now, if there is a fault occurring (e.g hydraulic pressure is getting low) at the same time, the operator needs to be alerted about the situation.

When something noteworthy happens in the system while the operator is concentrated on another task taking her attention, the control system should be able to catch the attention of the operator.

The machine operator should immediately notice if something noteworthy like a fault is occurring in the system. Otherwise, the machine may get damaged when continuing the work.

Some machines have a rotating operator chair in the cabin. It can not be assumed that the operator is facing the dashboard all the time and noticing the NOTIFICATIONS and error messages shown on the screen.

In some systems there might be more than one display. So, there is a need to show errors or notifications on different screens according to the direction the operator is facing. For example, if the operator is facing back-
wards when loading a log forwarder with logs, the operator might not notice warning lights in the dashboard behind her back, because the main display is placed so that it is visible when facing forward.

Therefore:

**Place controls for functionality and functionality’s feedback on the same screen or device or some place which is near the focus area of the operator when using the functionality. For example, if a warning light goes on, the light should be located near the focus area of the operator.**

The machine operator is always focused on the task at hand: cutting trees, driving the machine, etc. So it is advisable to place the feedback so that it is close to the focus area of the machine operator. For example, if the operator is drilling and the drill gets stuck, the light or user interface message informing about this situation should be located so that it is close to the drilling controls. Now the operator is likely to take a glimpse at the drilling controls, so she will be alerted about the situation.

Centralized warning lights in the dashboard facing the head of the machine are not recommended if the operator has a rotating chair. If there are hand panels which are used to control functionalities when facing e.g. backwards, the feedback and warnings should be given through these hand panels. Of course, if there is a screen that rotates as well, that can be used.

Additionally, sounds or vibrations can be used to give feedback of actions. If the environment is fast-paced, it might be a good option to use 3D sound effects. For example, if the machine operator is facing backwards in the cabin and there is an alert originating from a controller or sensor in the front of the machine, the sound should be directed so, that the operator knows the approximate location of the fault origin. Another example could be that the operator of forest harvester is cutting trees and for some reason the boom gets stuck. This could be indicated to machine operator by giving feedback using vibration of the joystick and then further information can be seen from the screen.

In the graphical user interface, also visual connecting can be used (see Fig. 4). If a warning is shown on the display, it might have dashed line to the source of the notification. E.g. if oil pressure is low and it is shown as a pop up, there could be dashed line leading to the oil pressure meter. This makes the informing process more efficient. This kind of approach could be very useful for example in large process control systems where there might be multiple screens.

Normally there is only one operating station and one control for each functionality, so it is rather easy to decide
where to place the feedback for the functionality. However, sometimes it might be the case that there are multiple operating stations and not all of them have warning lights or displays near the controls. In this case, one might consider adding lights and/or displays to each operating station. But if that is not reasonable from the cost effectiveness point of view, then one must consider other alternatives, like using sounds as a way of giving feedback to the user.

The user is more likely to notice notifications, alarms, faults, etc caused by her actions as the feedback is given near the focus area of the operator.

The feedback of actions are given in the same display where the functionality can be enabled. This makes it easier to notice the feedback the system is giving to the operator.

In some cases, it might be hard to decide where the feedback should be shown. E.g. if there are no displays or warning lights near the controls of the functionality. Then the designer must use sound or other means to inform the machine operator about the situation.

If a fault causes more than one notification, sound, error message, etc, it may create very stressful situation for the machine operator. So one should consider finding out the root cause of the problem and then showing the information related to that.

The mining drill operator is drilling a blast hole in a mine. She has initiated the drilling by selecting the correct mode from the touch screen and pressed the button on the hand panel to start the operation. The drill gets stuck in the hole for some reason. This situation needs urgent attention from the machine operator. The warning light is switched on, on the hand panel and an additional warning dialog is shown on the screen where the operator selected the action. Additionally, a warning sound is played as the situation requires immediate attention. This ensures that the operator is informed about the situation regardless of if she is watching the drilling process.

3.5 Double Confirmation

...distributed CONTROL SYSTEM with HUMAN-MACHINE INTERFACE has multiple functionalities that the machine operator can use. Some of the functionalities may require long time to start or take considerable amount of time to execute. Activation of such functionality may cause that some resources of the machine are unavailable for a long time. Therefore the machine operator should be particularly careful when activating them and avoid unintentional activation of such functionality. Sometimes in a control system also inactivating a functionality may cause long lasting consequences, so it should be made sure that the operator really wants to instantiate that.

Unintentional (de)activation of a functionality which may have long lasting consequences should be avoided as they may prevent the usage of the machine.

Starting, usage or deactivation of a functionality may take long time. If the functionality is unintentionally activated, it may take some time until the machine is operable again and thus unintentional activations should be avoided.

The activation of the functionality can be complicated. The machine operator will not be annoyed if she needs to do multiple steps to activate the functionality. In other words, the activation procedure can take some time itself.
For example, emergency stop must take place immediately, but activation of drilling functionality in mining drill can take 5 seconds.

Unintentional activation of a functionality may affect the environment where the machine is. For example, if a drill in mining drill is accidentally activated and it starts drilling in the wrong position. This may create an unoptimal blast hole to be drilled. Even if the drilling can be stopped, the half drilled blast hole may make the ore mining with explosives harder.

Activation of a functionality should not commence unnoticed. The machine operator should always be aware that she has activated a functionality. If the machine operator is not aware that a functionality is in use, she can not monitor its progress or she may try to initiate another possibly conflicting action, e.g. to drive the machine when the parking brake is on.

If the system is due to heavy processing load slowly reacting to the operator's requests, it might be the case that the operator presses the button (in UI or in hand panels) multiple times. This should not alone activate a functionality as it may have unexpected consequences.

Therefore:

**Design the graphical user interface in a way that the machine operator needs to enable the functionality twice changing the way each time. In this way, it is made sure that the operator is making a conscious decision to activate the functionality.**

Normally a functionality is triggered by pressing a button in the GUI or in hand panels. So design the activation of long lasting operations in a way that the operator needs to press two distinct buttons in order to activate the functionality. For example, the operator needs to select the functionality from the touch screen of the cabin and then activate the selected functionality using a physical button in the hand panel in the cabin. In this way, the operator can not unintentionally activate a functionality. Sometimes the functionality is not enabled immediately when the operator presses the button to activate the functionality. For example, it might take some time to generate the required hydraulic pressure. If this is the case, the presented approach makes it possible to avoid double presses of the button while the activation is still commencing. For example, the operator activates functionality by pressing a button. Required hydraulic pressure is generated. Once that process is finished the operator can start using the functionality by pressing another button.

There should be some guidance methods for the machine operator, so that she knows which button she should use to confirm the action. For example, once the operator has selected the functionality from the touch screen, the physical button in the hand panel can start to blink. The operator now detects the blinking and knows that the physical button activates the selected functionality. For further details, please see [FEEDBACK NEAR ACTION](#). Whatever guidance method is selected, it should be uniform for all actions (see [COMMON LOOK-AND-FEEL](#)).

Another advantage of double confirmation is that if pressing the same button twice would activate the functionality, a malfunction of a button could accidentally activate the functionality. For example, a paper clip dropped in the keyboard could generate key presses. This could cause accidental activation of a functionality. But if two different keys are used, the probability it would happen is smaller. Yet another case is accidental button presses that may activate a functionality unintentionally. For example, a machine operator might be resting her arm on the hand panel and hit the button by accident. As confirmation for initiation of the functionality is required, no harm is done unless the second button is pressed.
If the pattern is applied so that the second activation (confirmation) is implemented as a pop up, then the default action of a pop up should not be to activate the functionality (see Fig 5. In many user interfaces, an item is selected by pressing enter. Now if the confirmation is shown as a pop up with the button confirming the action selected by default, it may lead to situation where user accidentally presses the enter key two times and the second press activates the functionality. In this case the user may not be able to even read what the pop up window says. Therefore, the default option in the pop up should be to cancel the operation or the buttons of a pop up window should not have focus on them initially.

This approach is not recommended for functionalities requiring immediate action such as emergency kill switch. Furthermore, this approach should be applied only to functionalities that take long time to initiate or take considerable amount of time to execute. If this approach is overused, the usability of the machine decreases.

Unintentional activations or deactivations of a functionality can be avoided by having two phase activation of a functionality. The second phase of this two phase activation uses different buttons or keys to get the user input than the first phase. In this way, the user is making conscious choice when activating the functionality.

The user is informed what she is about to do, so the functionality will not execute without the operator noticing it. Additionally, unintentional activations can be avoided.

Designing good user experience may be a bit harder as double confirmation can be annoying as it requires more action from the operator. One must design the interface really carefully without overusing this pattern as it decreases the fluency of the work.

Double confirmation approach should not be used for emergency kill switch or similar functionalities where the response should be immediate. Otherwise, it could compromise the safety of the machine.

In a mining drill when starting drilling the operator needs to activate drilling mode from the user interface by pressing a corresponding button next to the screen. After the drilling mode is selected, the button that activates the drilling starts to blink. Once the user presses this button the drilling is started. This ensures that the drilling is not started unintentionally. The blinking is used to attract the attention of the machine operator, so she gets feedback of her actions and knows which button to press in order to start the drilling process.
4. CONCLUSIONS

We have presented five patterns for designing HMI for distributed control systems. These patterns form a cornerstone for creating HMIs for machine control systems. All five patterns are very typical for the domain and have characteristics that constrain the patterns for this domain. However, other user interface patterns can be utilized in the domain as well and when designing HMI for distributed control system other UI design patterns should be incorporated as well. The work on the pattern language is expected to continue.

5. ACKNOWLEDGEMENTS

We would like to thank our colleagues Ville Reijonen and Johannes Koskinen for their rigorous work on our pattern language. Especially we are grateful to our Shepherd Dirk Schnelle-Walka and workshop group B in EuroPLoP 2012 for valuable feedback on our patterns. We would also like thank guys at Sandvik, Metso Automation, John Deere Forestry, Bitwise, Kone, Rocla, SKS control, Space Systems Finland, ACGO power, Creanex and Tana. Without their support this work would not be possible. This work is supported by Finnish Cultural Foundation’s Pirkanmaa Fund.

REFERENCES

### Appendix A - Patlets

#### Table I. Patlets of the patterns presented in this paper

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HUMAN-MACHINE INTERFACE</strong></td>
<td>In order to enable the machine operator to control the system efficiently and safely, create a graphical user interface where the operator can see the machine status and functions that are currently in use. Let the user to interact with the system using joysticks, buttons, etc. Separate these two so that if the graphical interface is not functioning, the machine can still be operated using other controls.</td>
</tr>
<tr>
<td><strong>ROLE-BASED UI</strong></td>
<td>To show only such functionality in UI that is required by the user’s job description, create own user interface for each group. This means creating own UI for machine operator, maintenance personnel and production line personnel. Activation of certain UI may require dongle, password, or additional equipment.</td>
</tr>
<tr>
<td><strong>UPRIGHT IS OK</strong></td>
<td>To be able to offer the machine operator an easy way to notice from UI that machine is functioning normally, use traditional looking meters and gauges in the interface, even when the UI is implemented with traditional user interface components. Use these meters so, that values, e.g. for oil pressure, etc are in OK range when the meter is pointing upwards. This way it is easy to take an overview that everything is OK.</td>
</tr>
<tr>
<td><strong>FEEDBACK NEAR ACTION</strong></td>
<td>To catch the attention of the machine operator to something noteworthy caused by the action she is concentrated in, place controls for functionality and functionality’s feedback on the same screen or device or as close as possible to the functionality in the dashboard. For example, if warning light goes on, the light should be located near the control unit, e.g. joystick.</td>
</tr>
<tr>
<td><strong>DOUBLE CONFIRMATION</strong></td>
<td>In order to prevent unintentional (in)activation of a functionality, ask the operator to enable the functionality twice changing the way of confirming each time. In this way, it is made sure that the operator is making a conscious decision to activate the functionality.</td>
</tr>
</tbody>
</table>

#### Table II. Patlets of the patterns that are referenced in this paper, but that are not presented in full detail.

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CONTROL SYSTEM</strong></td>
<td>In order to implement a work machine that offers interoperability between systems and is highly operable with good performance, implement control system software that controls the machine and can communicate with other machines and systems. Root pattern for the language.</td>
</tr>
<tr>
<td><strong>CONTROL SYSTEM OPTIONS</strong></td>
<td>For enabling support for different options which the customer can buy for the control system, include all software components required by the available options to the control system during development time. Once the customer buys an option the corresponding code block is enabled without any software updates.</td>
</tr>
<tr>
<td><strong>COMMON LOOK-AND-FEEL</strong></td>
<td>To improve learnability and user experience provided by the user interface, make all user interface screens and notifications to have unified layout and colouring. Common UI elements are presented in the same way and place regardless of the view.</td>
</tr>
<tr>
<td><strong>FORCED INPUT VALUE</strong></td>
<td>To enable testing whether control unit on machine control level or its communication channel is malfunctioning, create a mechanism that can force I/O to certain value. In this way it can be seen that the output of the control unit receiving the forced I/O is desired. If it is not desired, the control unit is malfunctioning, otherwise the communication channel is broken.</td>
</tr>
<tr>
<td><strong>ISOLATE FUNCTIONALITIES</strong></td>
<td>To divide the system reasonably, distribute the system into subsystems according to their functionalities. Interconnect these subsystems with the bus. Use multiple interconnections between subsystems if necessary [Eloranta et al. 2010].</td>
</tr>
<tr>
<td><strong>NOTIFICATIONS</strong></td>
<td>To inform the machine operator or communicate to subsystems that something worth of noticing has occurred in the control system, communicate noteworthy or suspicious state changes in the system using notifications. Implement also Notification service that is used to create, handle and deliver notifications.</td>
</tr>
<tr>
<td><strong>OPERATOR PROFILE</strong></td>
<td>Allow the machine operator to use her personal settings for UI and controls regardless of the current machine in use and the adjustments other operators have made by packaging all operator’s settings in a single easily movable archive. The archive may reside on the operator PC hard disk or it might be saved on removable media, e.g USB memory stick. Provide an easy way to transfer the settings between machines and take them in use [Eloranta et al. 2010].</td>
</tr>
<tr>
<td><strong>SEPARATE REAL-TIME</strong></td>
<td>In order to offer high-end services without violating real-time requirements, divide the system into separate levels according to real-time requirements: e.g. machine control and machine operator level. Real-time functionalities are located on the machine control level and non real-time functionality on the machine operator level. Levels are not directly connected, they use bus or other medium to communicate with each other [Eloranta et al. 2010].</td>
</tr>
<tr>
<td><strong>SENSOR BYPASSING</strong></td>
<td>To be able to offer the machine operator a way continuing using advanced control mechanism even when some sensor of minor importance is faulty, implement a mechanism that the value provided by a sensor can be replaced with a default or simulated value.</td>
</tr>
</tbody>
</table>
Product Line Engineering (PLE) is an approach for optimizing economic benefits through the pro-active, constructive reuse of assets for customer-specific products/solutions. The main aim is to increase product quality and decrease development effort and cost by exploiting commonality among products/solutions.

PLE extends the development focused “platform” approaches in different dimensions. First, it turns the backward oriented ad-hoc discovery of similarities to a strategic forward-oriented elaboration of commonalities and variabilities of current and future products. Second, it widens the scope of reuse from the products themselves to by-products and to auxiliary tools that help creating concrete products, thereby opening additional ways to improve development efficiency. Both aspects are necessary to leverage the full benefits of reuse, and minimize risks.

This paper is intended to introduce and motivate the essential concepts of a PLE approach, targeting a management oriented audience. It refines some of the patterns touched on in the “big picture” of PLE [Schuetz2012].
An established development organisation (in product business or solution/project business, building software-intensive systems, optionally with mechanical/electrical components too), with ideally several similar successful projects or products, has identified the potential of reuse, and wants to benefit from it in an easy and effective way.

The similarities between these development projects are related to the functionality provided, but can also arise from using common development setup (parts, libraries, tools, processes).

Consider your company operates in solution business in the web applications domain, developing customer specific (software) solutions, which typically share a common set of features and solutions. For every customer/solution, a separate project is created, responsible to satisfy customer needs within the given budget, and generate reasonable profit.

In order to reduce their own development effort, the projects have applied a copy/paste approach on the code (and project) base: they are using the most similar project from the past, copy it as a starting point for their own work. Afterwards, the code base is adapted to the projects specific needs, by deleting, replacing or modifying mismatching parts. Over time, each project grows its unique code and development structure.

This ad-hoc reuse resulting from local needs has sped up initial development, but doesn’t contribute to maintenance. Every bug that is found in one of the projects has a high probability to affect other projects too, but is difficult to be located and fixed in the different branches.

From an outside view, all those development projects work independently, but on the same time seem to follow similar paths. This coherence indicates potential for improving efficiency.

Now, the management, influenced by success stories from other “platform” approaches, has identified this potential of reuse and beyond, and started a strategic initiative to raise development efficiency and business revenue.

The patterns described here are intended to address most important aspects in the PLE approach. They have to be considered early, and cope with popular misapprehension and pitfalls.

<table>
<thead>
<tr>
<th>PRODUCT LINE SCOPING</th>
<th>Activity: How to settle the “platform” in the context of the overall product portfolio.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSINESS DRIVES PLATFORM</td>
<td>Strategy: Align your approach with future needs, according to the current business strategy, which not necessarily match those from the past. (not covered in this paper)</td>
</tr>
<tr>
<td>MORE THAN PARTS</td>
<td>Strategy: How to define a platform (respectively its “content”), in order to increase its economical benefit.</td>
</tr>
</tbody>
</table>
### PRODUCT LINE SCOPING

**Summary**
Leverage the platform (respectively its support activities) by carefully scoping it according to your product portfolio.

**Problem**
How to align a platform with the portfolio of products?

**Forces:**

- **Quality requirements.**
  Platform components are required to be of high quality, since negative consequences of mistakes are multiplied with every instance derived from the platform. These quality requirements are causing increases of development effort.

- **Upfront investment.**
  Creating the reusable assets requires a solid amount of upfront investment, and a long time to pay back. Since it is difficult to find a “sponsor” that stays confident for such a long duration, the financial risk must be kept considerably low. At the same time, such a strategy should not jeopardise the return on investment at all.

- **Too Big.**
  A platform intended to support a huge variety of products faces a high combinational diversity. It runs risk to suffer from overly complex solutions that are needed to be combined excessively in order to cover the required constellations.

- **Too Small.**
  A platform intended to support a huge variety of products often is focused on the “common to all” components, and therefore tends to cover only infrastructural aspects. This limits the potential of reuse to a rather small and meagre area. The products starting from that base still have a long way to success.

- **Diverse products**
  Product managers for products with very specific USP’s might be reluctant to give up some of their “exotic” features.

<table>
<thead>
<tr>
<th>SEPARATE DOMAIN ENGINEERING</th>
<th>Organisation: How to map a platform approach into an organisational structure, its responsibility structure and processes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMONALITY VARIABILITY ANALYSIS</td>
<td>Activity: An initial step to explore the potential of a product line approach. (not covered in this paper)</td>
</tr>
</tbody>
</table>
Solution

Overcome the temptation to support every product in your portfolio with a “one for all” platform. Instead, focus on a coherent subset of products, and intentionally exclude at least one.

Scan through your portfolio to discover sections with a tighter cohesion than average. Use these as initial point for careful explorative expansion. List advantages and disadvantages of each extension of scope, and quantify its effects on the BUSINESS CASE of the platform.

Figure “Platform, scoping, and related products”

Continue with the extension as long as your “business case” shows significantly more pay-off than investment (e.g. factor two to three), following a “high impact first” strategy. Hesitate when adding features (market segments) that heavily interfere with other variability dimensions, hence creating cross-effects in your development.

But do not stop with the extension just because you are “running out of money”.

On the other extreme, strive to withstand the temptation to support every product in your portfolio with a “one for all” platform. Instead, focus on a coherent subset of products, and intentionally exclude at least one. Experience shows that such “big” platforms show the tendency to provide only infrastructure, hence miss a big potential on functional reuse.

After this process, your portfolio will be separated into two categories: products that belong to the family and will be supported by the platform core assets, and on the other side some that are scoped out. The latter might be subject to later restructuring of the overall portfolio.
Consequences  

**PLATFORM SCOPING** provides the following **benefits**: 

- **Level of reuse**
  A small set of products to be supported increases the rate of commonalities on the functional level, hence fostering reuse in this area. The specialisation enables reuse of key functionality, which is much more effective than reuse of infrastructure.

- **Business risks**
  By addressing only a subset of the product portfolio with the platform, the risk for the overall business in case of failure with the platform is considerably lower (but might still be high). The restriction also reduces the risk of a “Monoculture” [Marquardt2010] that dominates the business but hinders fast reaction to market trends.

- **Appropriate set of core assets**
  With the limited scope of the platform, it is much easier to define and develop a set of core assets that provides sufficient support for the product derivation. The success story of the platform becomes evident quite soon.

- **Freedom to say “No”**
  The fact that there are already things that are “scoped out” by the platform reduces the resistance against (further) de-scoping. By this, products with feature that would increase the complexity to much can be ruled out easily, for the sake of other products and their benefit from the platform.

- **Portfolio optimisation**
  Discovering the inner relationships of the different products in the portfolio, combined with the potential savings due to a platform approach, might generate hints how to optimize the product portfolio.

On the other hand, there might arise several **liabilities**: 

- **Delayed savings**
  The focus on a reduced set of products postpones benefits for those excluded from the platform.

- **Suboptimal design**
  Expanding the platform later to an extended set of products might lead to a less optimal design compared to an approach considering the larger set from the beginning.
MORE THAN PARTS

Summary
There are common parts that find their way into all (or at least many of the) products in your product line. That’s the obvious part. But in order to make the development of concrete products efficient, there is more that might benefit from support. First of all, every product has some less prominent sidelines that are handed over to the customer (e.g. user documentation, certificates), which should be supported to. Second, the development process for every of the products generates some similar or even identical artefacts (e.g. an architecture specification, or test cases), and requires similar tasks (such as configuration management, bug tracking), which should be provided or supported by corresponding core assets.

Problem
How can one maximise the benefit from reuse?

Forces:

- **Focus on the “Bill of Material”**
  Success stories like the Volkswagen and Audi platform or the Airbus suggest that everything is perfect “if we just had enough common parts”. Scaling effects from producing or buying common (more often used) parts in high numbers and therefore cheaper are perceived as the main factor for cost reduction. That’s the omnipresent platform hype.

- **Efficiency in product derivation**
  Research and development (R&D) with their small fraction regarding to sales\(^1\) are considered minor details, although they are main drivers regarding quality and time to market.

Solution
Consider at least the following categories of supportive artefacts when defining the set of core assets:

- Components that are built into the product/solutions
- Artefacts that are needed for or during development
- Components, artefacts, and services that are required before or after the development phase

Provide components that are common to (built into) all products/solutions. This basic set is extended with artefacts that contribute to predefined options, hence are expected to be used more than once, and therefore should be provided for reuse too.

\(^1\) 7-10% in product business incorporating production of physical goods.
The core assets is not restricted to “parts” or code fragments that are directly build into the desired product/solution or shipped with it. Add to the desired efficiency by providing assets that are actively supporting the development process: To this end, generate reusable development artefacts, such as requirement specifications or design documentation. Provide a common structure for such documents, with dedicated areas to add “modules” that cover a various feature or design element.

Analyze your typical development process regarding steps that might create overly effort, and address them with supportive measures or tools, and provide.

Furthermore, additional artefacts that increase the efficiency of the development or production process steps might be valuable core assets too. Examples are tools that automate tedious development tasks, such as transforming high level representations into the corresponding low level structures (model driven development, generating code out of descriptions).

Last but not least, investigate in the whole lifecycle of your development, from inception through development and engineering, production, commissioning, maintenance and service, to discover other useful core assets.

The set of core assets should be defined not only by the product line manager himself, but a joint initiative incorporating “customer representatives”, assembled from project leads and architects of the application project teams. The customers can state their needs, and contribute to the decision s regarding the platform.

Consequences **MORE THAN PARTS** provides the following benefits:

- **Efficiency in product derivation**
  The main use case of a platform is the derivation of concrete product, respectively the whole process for that, is supported with artefacts that provide optimal efficiency.

On the other hand, there might arise several liabilities:

- **Frequent changes**
  Some of the “side” artefacts, especially those supporting the derivation process itself, depend on the structure of the product itself. Hence, every significant change of the product might cause need to adapt these artefacts too.
**SEPARATE DOMAIN ENGINEERING**

**Summary**
Develop the core assets ideally within a separate project, decoupled from the time and budget restrictions of outward oriented development projects creating concrete products.

**Problem**
A single development project is first and foremost responsible to develop its desired result (a product or a solution) fast, with minimal risk and cost. Addressing “platform” topics is in conflict with these primary goals, and hence with high probability will be abandoned.

How to set up the concurrent and conflicting development for platform and products?

Forces:

- *Long term issues.*
  The release plans and roadmap for the platform might not fit to the needs of an ongoing product development project.

- *Quality requirements.*
  A single project might be able to fix its own bug later on, but for core assets that are reused multiple times, quality demands are much higher to prevent an economical disaster.

- *Context switches*
  Switching from on development context to another one with different requirements and quality characteristics might cause confusion and errors, and decreases efficiency

- *Credibility*
  Platform development in most organisations has a higher reputation and is considered more challenging/rewarding in most organisations.

**Solution**
Develop these supportive core assets within a separate project, adhering the “unusual” circumstances, decoupled from the time and budget restrictions of outward oriented development projects (*SEPARATE DOMAIN ENGINEERING*).

The activities around products/solutions and core assets are organized into two different groups [SPLE2006]:

- The **Application Engineering** is responsible to define and develop concrete products or solutions. This creation/derivation of products is supported by reusing and applying the core assets.

- The **Domain Engineering** is taking care of cross-product topics, hence subsumes the scoping and definition of the product line and its core assets, as well as developing and maintaining the core assets.
Figure “Domain Engineering and Application Engineering

Both groups internally follow a typical development process, with some adaptation to their specific needs. Each group is at least staffed disjoint and independently regarding its key roles (Product owner, project manager). Developers, architects and testers should also have a fixed (but maybe temporary) assignment to these segments.

**Consequences**

**SEPARATE DOMAIN ENGINEERING** provides the following **benefits**:

- **Well defined responsibility**
  All project tasks are clearly assigned to just one of the segments, hence have a clear focus and responsibility.

- **Balanced environment**
  Developers have a chance to switch occasionally to the other side, in order to experience and understand their specific needs. The “goodies” and loads can be distributed quite evenly.

- **Separated Goals**
  The core roles are not exposed to conflicting goals. They don’t run danger to grow implicit favourites. Arising conflicts become obvious and can be resolved in appropriate decision boards.

On the other hand, there might arise **liabilities**:

- **Conflict between teams.**
  There is still sufficient conflict potential between both segments to generate a crisis.
Further Patterns  Related patterns that still need to be refined cover the following topics:

- Core Asset Definition
- Product derivation

Known Uses  The approach described above has (at least partially) been applied successfully in platform developments across several Siemens domains:

**Airbus Industries**

Since its inception, Airbus has placed a high priority on maximising the commonality of its aircraft families. The goal is to make operations, training and maintenance easier and less expensive for customers—a value that contributes to the efficiency and operational flexibility of Airbus jetliners every day.

Fifteen of the company’s aircraft models, from the 100-seat A318 to the double-deck A380 feature nearly identical flight decks and similar handling characteristics. This commonality enables multi-qualification of pilots and reduced pilot training times, while bringing significant savings through reduced maintenance training and streamlined procedures. [Airbus]

**Volkswagen/Audi**

Volkswagen and Audi have received a lot of attention for developing different cars on a platform of common elements, ranging from small parts to complete motors, gears, and chassis groups.

**Siemens**

Within Siemens, different business segments are applying PLE successfully.

- Postal Automation provides a set of scalable letter sorting machines, based on many common parts and solutions, running common control software for more than hundred customer specific installations.

- Industrial Automation, VAtech strand casting:
  Different installations for steel production are engineered from a scalable product base, operated by specifically derived control software.

- Industrial Automation, sensor development: a set of gas and fluid mass flow sensor with a common measurement principle is developed on a common set of hardware parts, embedded processors, and software components.

Further success stories and case studies can be obtained from [LindenSchmidRommes2007].
Credits
Thanks to my shepherd Michael Weiss, for encouraging and really helpful feedback right to the point. I also want to thank my colleagues Horst Sauer, Anne Hoffmann, and Christa Schwanninger, for providing their thoughts and never getting out of discussion.

References

[Airbus]
http://www.airbus.com/aircraftfamilies/passengeraircraft/commonality

[LindenSchmidRommes2007]
Linden, Frank J. van der; Schmid, Klaus; Rommes, Eelco: Software Product Lines in Action / The Best Industrial Practice in Product Line Engineering; Springer, Berlin, 2007

[Marquardt2010]
Klaus Marquardt: Monoculture; in “Proceedings of the 15th European Conference on Pattern Languages of Programs 2010.”

[Schuetz2010]
Schuetz, D: Transition to PLE; in “Proceedings of the 15th European Conference on Pattern Languages of Programs 2010.”

[Schuetz2011]

[SPLE2006]
Klaus Pohl; Günther Böckle, Frank von der Linden: Software Product Line Engineering, Springer Berlin Heidelberg, 2006
A Pattern Language for Service-based Platform Integration and Adaptation

Ioanna Lytra\textsuperscript{1}, Stefan Sobernig\textsuperscript{2}, Huy Tran\textsuperscript{1}, Uwe Zdun\textsuperscript{1}

\textsuperscript{1}Faculty of Computer Science
University of Vienna, Austria
Email: firstname.lastname@univie.ac.at

\textsuperscript{2}Institute for IS and New Media
WU Vienna, Austria
Email: stefan.sobernig@wu.ac.at

1. INTRODUCTION

A software platform is a unified foundation on top of which applications can be developed and executed. Software platforms are an important organizational strategy to achieve software reuse in software development at a large scale. Platform-driven software reuse reaches out beyond systematic code reuse (e.g., through component orientation) and involves reusing use cases, software tests, documented design decisions (e.g., design documents), and development procedures (e.g., continuous application integration, release management, testing approaches; see [Ghanam et al. 2012]). As a software artifact, a software platform abstracts from details inside and underneath the platform and thereby eases application development and maintenance.

Many software systems that are developed today are based on one or more software platforms. For instance, an online shopping portal will rely on an enterprise resource planning (ERP) platform such as SAP R/3 for receiving and processing purchase orders and a warehouse system for managing large inventories of goods. Other examples of software platforms include platforms for steel plant control, telecommunication platforms such as Mobicents\textsuperscript{1}, data storage platforms such as Amazon's S3\textsuperscript{2}, social network platforms such as the Facebook Platform\textsuperscript{1}, and mobile platforms such as Google's Android\textsuperscript{4} or Apple’s iOS\textsuperscript{5}. This short and non-exhaustive list of platform examples illustrates that different application areas require different and specialized platforms. Often it is the case that heterogeneous platforms need to be integrated to provide unified services to be used by domain-specific applications. For example, inside a warehouse usually a warehouse management system, an enterprise resource planning, a yard management system and various communication systems need to provide services in an integrated way for building warehouse operator and control applications.

As in many other areas, one of the current trends of software platforms is to offer their services using standardized interface technologies such as Web Services [Christensen et al. 2001]. With this trend comes the growing requirement for the integration over these specialized service platforms. If the current trend towards offering platforms through service-based interfaces continues, we will see a large ecosystem of heterogeneous service-based platforms emerging in the near future. However, the increasing specialization in the applications domains of software platforms, as well as the diversity of technologies for platform development and of platform interfaces leads to open challenges regarding the integration of multiple platforms.

In the context of platform integration, the heterogeneity of service platforms with respect to their functional and non-functional properties often leads to several alternative ways of successfully tailoring, adapting, and integrating platforms. In other words, software architects and developers are usually confronted with numerous design decisions at different levels of abstraction and at different levels of granularity when designing a platform integration solution.

This paper aims at addressing challenges regarding offering platforms as services and integrating multiple heterogeneous platforms. In particular, we will address how to offer the functions of a (legacy) platform using a service-based interface and how to tailor as well as to combine multiple platforms into a unified service platform. For this, we introduce a pattern language targeting high-level and low-level design decisions for integrating and adapting platforms using services. This pattern language is addressed to software architects and designers as well as software developers that deal with the problem of designing unified service platforms over heterogeneous platforms –often owned by third-party vendors– to be used as the basis for developing new applications.

So far, a substantial amount of patterns have been described covering many aspects of service-based integration and adaptation. However, those patterns have been presented with a different focus, such as general software design [Gamma et al. 1994], software architecture [Buschmann et al. 2000; Averioul and Zdun 2005], distributed system design [Buschmann et al. 2007a], enterprise application architecture [Fowler 2003], messaging [Hohpe and Woolf 2004], remoting middleware [Völter et al. 2005], service-oriented systems [Hentrich and Zdun 2009], service design [Daigneau 2012] and process-driven SOA [Hentrich and Zdun 2012]. The contribution of this paper is to survey and

\textsuperscript{1}http://www.mobicents.org
\textsuperscript{2}http://aws.amazon.com/s3
\textsuperscript{3}https://developers.facebook.com
\textsuperscript{4}http://www.android.com
\textsuperscript{5}http://www.apple.com/ios
to organize the existing patterns and design decisions in a comprehensive pattern language for the service-based integration and adaptation of platforms. With this, this paper primarily addresses software architects who face the challenge to design, to realize, and to deploy a service-based integration architecture for software platforms. As a secondary audience, developers of client applications, which are built from the integrated platforms, can consult this pattern language to evaluate the impact of the underlying integration architecture (and the design decisions embodied therein) on the observed non-functional properties (e.g., QoS properties such as execution timings, distributed exception state) observed for their applications.

The remainder of the paper is structured as follows. Section 2 provides the problem statement and the background on service-based platform integration. In Section 3, we illustrate an industry case study as a motivating example. The actual pattern language is then presented in detail in Section 4. To demonstrate its applicability, the pattern language is applied to our motivating example in Section 5. We discuss the related work in Section 6 and conclude in Section 7.

2. SERVICE-BASED PLATFORM INTEGRATION: BACKGROUND AND PROBLEM STATEMENT

We consider a software platform as a collection of software sub-systems, like communication middleware and databases, and interfaces which together form a reusable infrastructure for developing a set of related software applications. The functional and non-functional properties of a service platform and its interfaces vary with the requirements of the application area in which the platform is deployed. To build a concrete application by reusing software artifacts in a platform, the platform lays out a customization and configuration process on top of its interfaces [Ghanam et al. 2012].

In a service-based software platform, software platforms expose their interfaces in terms of services which provide the programming models for developing platform-based applications. Platform customization and configuration usually involves adaptations of service interfaces (e.g., interface aggregation) and/or service implementations (e.g., service specialization and substitution) as well as forms of service composition (e.g., batched service execution, service chaining, or process-driven service orchestration). Platforms are then integrated by applications via their exported platform services.

When looking at multiple service-based platforms, the exported services and their interfaces are heterogeneous in terms of the middleware technologies, the transport protocols, the programming languages, and the programming models (e.g., remote procedure calls, document-centric services) used. In addition, platform service interfaces change over time and platform services are substituted for others (e.g., service specialization, service substitution [Ruokonen et al. 2008]). Applications using platforms must cope with the heterogeneity of the platforms they integrate, as well as with interface changes. At the same time, (groups of) applications exhibit different requirements on the same set of platform services; and may change in these requirements over time. For instance, applications might require functionally tailored interfaces (e.g., operation subsets, aggregated operations and aggregated operation data) and different interface capabilities for separate application groups based on their QoS requirements or on different authorization levels.

If this platform heterogeneity and the characteristic integration requirements were fully anticipated (for example, by analyzing the platform domains using a domain engineering approach), software platforms would be developed in terms of software product lines [Ghanam et al. 2012] and platform integration would turn into an issue of developing multi software product lines [Rosenmüller and Siegmund 2010]. In this paper, however, we are interested in integration scenarios involving software platforms which were not necessarily designed as product lines and, most importantly, which were not designed to be integrated with one another (e.g., by using product line engineering techniques such as code generation or component weaving).

A strategy to deal with previously unanticipated platform integration is service-based platform integration. In such an integration strategy, applications should strive for programming towards stable, service-based interfaces that hide technology, protocol, language, and programming model dependencies as much as possible. Some platforms already offer a suitable service-based platform integration interface to their platform-driven applications, but in most cases such interfaces are not available (consider, e.g., legacy platforms). Service-based platform integration addresses situations like the one depicted on the left-hand side of Figure 1: The client applications developed on top of a software platform have direct dependencies to the services offered by the platform. Service-based platform integration changes this situation to the one depicted on the right-hand side of the figure, with an intermediate abstraction layer hiding the details of the platform underneath. Thus, an application is developed on top of a service-based integration platform which integrates services from one or more platforms. With such an integration platform in place, client applications with changing requirements on the platform services, on the one hand, and platforms exposing changing platform services, on the other hand, can be effectively shielded from each other.

An intermediate platform has an additional benefit, illustrated in Figure 1: Applications from different domains can be programmed against different service-based integration platforms, each offering a domain-specific view on the platform abstractions. For instance, consider developing applications for Android. The integration platform could provide different views on the Android platform, with each view exposing selected Android API chunks for, e.g., business applications, Web applications, 2D and 3D games, and so on. The integration platform could provide a stable view over different versions of the Android SDK.
A Pattern Language for Service-based Platform Integration and Adaptation

Fig. 1: Service-based Platform Integration

Such a design also offers the advantage of rendering the underlying platforms exchangeable. For example, an abstraction layer can be provided in a way that similar script code can run both on Android and iOS. In this example, this is possible provided that a scripting engine running on both platforms is used.

In addition to the abstraction of platforms and their interfaces, this paper also considers the problem of integrating multiple platforms and their platform services into one and the same application. This problem is schematically illustrated in Figure 2. Consider an online shopping portal that relies on an enterprise resource planning (ERP) platform such as SAP R/3 for receiving and processing purchase orders and a warehouse system for managing large inventories of goods. In a service-based integration platform, we could select only those services of the two platforms that are relevant to the online shopping portal and present them in an integrated fashion. We could offer them through the different communication channels needed for the online shopping portal, but not offered by the legacy platforms. Internally, the service-based integration platform would perform all necessary tasks of integrating, adapting, and routing messages for the two backend platforms.

Fig. 2: Service-based Platform Integration for Multiple Platforms

3. MOTIVATING EXAMPLE

To illustrate the problem of service-based platform integration and adaptation, we present an example from a case study on industry automation performed in the context of the EU FP7 project INDENICA. In this case study, there are three heterogeneous platforms: a Warehouse Management System (WMS), a Yard Management System (YMS), and a Remote Maintenance System (RMS). An operator application utilizes the services provided by these three platforms. The YMS manages the scheduling and the coordination procedures for trucks which are needed for the loading and unloading of the goods. The WMS handles the storage of the goods (or storage bins) into racks via conveyor systems. The RMS system is connected to the warehouse to monitor every incident occurring in the warehouse and the yard. An operator application uses the services exposed by the domain-specific integrating virtual platform (VSP). This intermediate platform integrates the services of the three backend platforms. All the interactions between the integrated platforms as well as between the operator application and the three platforms are performed through the VSP.

http://www.indenica.eu
Figure 3 illustrates a single scenario covered by our case study, presented as a sequence diagram. This scenario addresses unloading storage bins onto the racks in the warehouse. When a loaded truck arrives at the yard, the operator gets notified (truck\text{Arrived}) and requests a free dock (get\text{FreeDock}) for the truck. After receiving a free dock (a dockID), the operator communicates with the personal (initiate\text{VoiceCall} and end\text{Call}) and coordinates the redirection of the truck to the assigned dock (move\text{TruckToDock}) for its unloading. When the operator gets notified that the truck is ready for unloading (truck\text{Ready}), she sends a command to the personal to start unloading (start\text{Unloading}). For each product a storage unit is registered (register\text{StorageUnit}) and a suitable bin location is reserved (search\text{AndReserveSuitableBinLocation}). The product is stored in the reserved bin location (transport\text{StorageToReservedBinLocation}). Upon finishing the unloading, a notification (unloading\text{Finished}) is sent to the operator. Finally, the operator gets notified when the truck leaves the dock (truck\text{Left}).

To enable the operator application to use the services of the three platforms through an integration platform, many architectural decisions regarding the adaptation and the integration of the heterogeneous interfaces as well as the routing of the information between the operator application and the platforms must be made. In this paper, we study the design alternatives that must be considered when software architects and developers are confronted with the platform integration problem. In the next section, we describe the pattern language for service-based platform integration and adaptation in detail.

![Fig. 3: Integration Scenario in a Warehouse](image-url)
4. PATTERN LANGUAGE FOR SERVICE-BASED PLATFORM INTEGRATION AND ADAPTATION

4.1 Pattern Language Overview

In this section, we describe a pattern language for service-based platform integration and adaptation. This pattern language documents interconnected design decisions by drawing from existing pattern material, such as patterns for general software design [Gamma et al. 1994], software architecture [Buschmann et al. 2000; Ageriou and Zdun 2005], distributed system design [Buschmann et al. 2007a], enterprise application architecture [Fowler 2003], messaging [Hohpe and Woolf 2004], remoting middleware [Völter et al. 2005], service-oriented systems [Hentrich and Zdun 2009], service design [Daigneau 2012], and process-driven SOA [Hentrich and Zdun 2012].

We introduce an overview of the main categories of design decisions documented in our pattern language in Figure 4. The direction of the arrows implies follow-on decisions. Arrows in both directions imply that the decisions can be made in parallel. In our pattern language, we consider the following architectural decision categories: Integration and Adaptation, Interface Design, Communication Style and Communication Flow.

The Integration and Adaptation category collects design decisions regarding the integration of platform services into a service-based integration platform and their interface and protocol adaptation, if required. The Interface Design category mainly covers design decisions regarding the design of the exported interface(s) of the service-based integration platform. Decisions in the categories Integration and Adaptation can be performed in parallel to decisions in the Interface Design category. These categories mainly concern developing components and interfaces for connecting applications, platforms, and the service-based integration platform.

The Communication Style category comprises design decisions that must be taken for each distributed component connection. These decisions relate to options for connecting two components (e.g., blocking and non-blocking component interaction styles). These decisions reside at a lower level of abstraction than the decisions of the two previous categories.

The Communication Flow category describes additional decisions that must be considered in case the service-based integration platform introduces more complex communication flows than simple forwarding from an exported interface to imported interfaces. For instance, such decisions relate to handling the aggregation or the splitting of the messages on their way through a service-based integration platform.

The patterns in each category are documented in the form Problem – Solution – Decision Drivers. These pattern sketches are based on the information extracted from the aforementioned design pattern material. The decision drivers reported have been selected based on the authors’ assessment of their relevance for service-based platform integration.

In the following sections, we discuss the four pattern-language categories in more detail and provide visualizations of the characteristic decision flows in each category. The decision flows document relevant patterns and their interconnections, as well as the follow-on pattern category or categories to be considered. The decision flows and the related pattern sketches will help software architects and developers to structure their own decision-making processes by highlighting characteristic decision steps and by presenting pattern alternatives, pattern variants and pattern compounds. With this, we aim at offering a guideline on how to lay out a concrete decision-making process rather than presenting a ready-made “recipe” for selecting patterns in service-based platform integration.

![Fig. 4: Overview of Pattern language for Platform Integration](image)

4.2 Integration and Adaptation

The simplest case of integrating a platform into an application is to directly invoke the platform services from the application code. However, often we would like to avoid direct invocations in order to support abstraction or stable interfaces as motivated in Section 2. In addition, often simple direct invocations are not enough, as the integration logic should introduce extra functionality, such as logging, monitoring, indirectioning, or adapting the platform access. Such situations are discussed in this section.
For the case that the interfaces offered by the platform are compatible to each other and that the extra functionality needed does not change the invocation flow, the PROXY pattern [Gamma et al. 1994] can be used to indirect the service invocations, to perform additional tasks on the invocation data, to select and to access the actual platform services. If extra functionality such as logging, monitoring or access control is needed and this does not change the invocation flow, the functionality can be handled using a PROXY between the platform services and the application.

**PROXY [Gamma et al. 1994]**

**Problem** There are situations in which a client does not or can not access a platform service directly, but wants still to interact with it. A surrogate or placeholder for an object to control the access to the service is needed.

**Solution** A PROXY acts as the intermediary between the client and the target. The PROXY has the same interface as the target. The PROXY holds a reference to the target and can forward requests to the target.

**Decision Drivers** A PROXY is used whenever there is a need for a more versatile or sophisticated reference to an object than a simple pointer. It introduces a level of indirection when accessing an object. It also supports creation of objects on demand. In the context of platform integration, it can be used to introduce extra functionality or control, but it does not change the interface of the invoked service or the invocation flow.

Direct invocations vs. proxy-based platform integration are illustrated in Figure 6. When using direct invocations the platform services are called directly from the application, thus the application is tightly coupled with the platform interfaces. In the second schematic example, the PROXIES introduce extra functionality for monitoring the invocation flow from the application to the platform. From the viewpoint of the application, they essentially introduce a new platform abstraction, in this paper called the service-based integration platform.

**REMOTE PROXY [Schmidt et al. 2000a; Buschmann et al. 2007a]**

**Problem** As in the PROXY pattern, we need to access an object through a placeholder for another object to control access to it. In addition, the object and its client are residing in different process or machine contexts.

**Solution** A REMOTE PROXY is a PROXY that connects two objects in different process or machine contexts. Usually, communication middleware is used to cross the process or machine boundary.

**Decision Drivers** The REMOTE PROXY has the same decision drivers as the PROXY pattern plus the need for integration of distributed applications and platforms.
In many cases, applications and platforms are residing in different process or machine contexts. Hence, invocations must cross the process or machine boundary. In such cases, we can apply the remote variant of the PROXY pattern, the REMOTE PROXY [Schmidt et al. 2000a; Buschmann et al. 2007a]. In the platform integration context, the REMOTE PROXY resides in the service-based integration platform and connects application and platform. The schematic illustration on the right hand side of Figure 6 also applies to REMOTE PROXIES, but the arrows depict remote invocations instead of local invocations.

In addition to simple integration, service-based platform integration requires coping with the diversity of the interfaces that these platforms expose. Calling a remote interface directly or through a PROXY is not always possible, for instance, because the interfaces offered by a platform may not offer exactly what the calling application expects. Using the original interface might be possible, but we need to take into account that usually the applications are tightly coupled with their interfaces and implementations. Changing the interfaces of a platform is a possible solution. But, firstly, an interface change is tedious and error-prone, and, secondly, most often it is not possible at all because many platforms that need to be integrated are provided by third parties. In addition, platforms are typically used by many applications and it is usually not possible to offer a different interface for each of them.

For these reasons, an ADAPTER [Gamma et al. 1994] can be inserted between the caller and the remote interface that converts the provided interface into the interface that the caller expects and vice versa. The adapter also transforms the data returned by the adaptee into the data structures expected by the caller. For distributed systems two variants of the ADAPTER pattern, the OBJECT ADAPTER [Gamma et al. 1994; Buschmann et al. 2007a] and the INTEGRATION ADAPTER [Hentrich and Zdun 2012], can be used to connect the interfaces and to perform the appropriate transformations.

**OBJECT ADAPTER** [Gamma et al. 1994; Buschmann et al. 2007a]

**Problem** A class or component offers an interface, but the interface does not match the one that is needed by a client. We need to resolve the interface incompatibility.

**Solution** An OBJECT ADAPTER converts the interface of a class or component into another interface clients expect. The ADAPTER lets classes or components work together that could not otherwise because of incompatible interfaces.

**Decision Drivers** Interface adaptation lets us incorporate our classes or components into existing systems that might expect different interfaces. The amount of work for creating an ADAPTER depends on the similarity between the adaptee and target interfaces.

From a high-level perspective, OBJECT ADAPTERS usually have a similar structure as the PROXY example depicted in Figure 6. The ADAPTERS would simply replace the PROXIES and introduce the additional interface adaptation behavior.
Very often new versions of platforms come with new versions of interfaces. This can be hidden from the applications using the interfaces by exchanging the OBJECT ADAPTER. However, the more complex the mapping between the interfaces is, the more expensive is the mapping in terms of performance and development effort.

A general problem of components like OBJECT ADAPTERS in platform integration scenarios is that invocations reaching the ADAPTER while it is being maintained (i.e., stopped and redeployed) would get lost. In many cases, this is highly undesirable. This problem is addressed by an extension of the ADAPTER pattern, the INTEGRATION ADAPTER [Hentrich and Zdun 2012] pattern.

**INTEGRATION ADAPTER [Hentrich and Zdun 2012]**

**Problem** Heterogeneous systems need to be connected and we need to shield the client from the impact of system and system interface changes. The calling and called interfaces might change over time and maintenance activities should not cause invocations or messages to get lost.

**Solution** The INTEGRATION ADAPTER contains two connectors: one for the client system’s import interface and one for the target system’s export interface. It plays the role of the translator between the heterogeneous systems and for their different interfaces, protocols, technologies and synchronization mechanisms. The adapter can be made configurable at runtime by using the COMPONENT CONFIGURATOR pattern, so that the adapter can be modified without affecting the requests to the adapter. A COMPONENT CONFIGURATOR offers a configuration interface for stopping, suspending and starting adapters. When new versions of the adapter must be deployed the adapter is stopped. When new versions of the target system are deployed or the adapter is configured at runtime the adapter is suspended. After the maintenance activities the adapter can process all requests that have arrived in the meantime.

**Decision Drivers** The INTEGRATION ADAPTER provides flexible integration for applications from external vendors. Generic adapters can be offered to support interconnectivity via common standards (e.g., Web Services). A drawback of INTEGRATION ADAPTERS is that if many adapters from different systems exist, they need to be managed in a centralized and controlled way.

An important part of the INTEGRATION ADAPTER pattern is its use of the COMPONENT CONFIGURATOR pattern [Schmidt et al. 2000c] to stop, suspend, and start the adapter component during the process lifetime of the integration platform. This pattern can also be used to make other integration solutions, like the PROXY based solutions discussed before, configurable at runtime. This form of runtime adaptability complements other configuration techniques available at the deployment time (e.g., deployment descriptors) and at the runtime of the integration platform (e.g., invocation interceptors for the middleware framework).

We illustrate in Figure 7 a potential INTEGRATION ADAPTER design. The INTEGRATION ADAPTER implements a configurable component interface to realize the COMPONENT CONFIGURATOR pattern. To avoid losing messages while the adapter is being maintained, the INTEGRATION ADAPTER has an asynchronous messaging interface to the client, which queues up messages until the maintenance actions are performed (see the discussion of MESSAGING in Section 4.4). The integrated platform is connected via a synchronous connector. The adapter also performs the translation from asynchronous calls to synchronous calls (see the discussion of CORRELATION IDENTIFIER in Section 4.5).

**COMPONENT CONFIGURATOR [Schmidt et al. 2000c]**

**Problem** The application must provide a mechanism to configure components at any time of the application lifecycle. The components should be initiated, suspended, resumed, terminated or exchanged dynamically at runtime without having any impact on the rest of the application.

**Solution** The component interfaces are decoupled from their implementation and used from the application to dynamically control the components. Concrete components implement these interfaces in an application-specific manner.

**Decision Drivers** COMPONENT CONFIGURATOR offers a common interface for the administration of components (initialize, suspend, resume and terminate). The implementation of the components is decoupled from their configuration, thus increasing modularity and reuse. Configuration and reconfiguration of components can be performed dynamically. This pattern increases also the range of configuration alternatives. However, it has the liability of a lack of determinism, since the behavior of an application is not determined until its components are configured at runtime and a potentially lowered reliability, since the dynamically configured components can affect the execution of other components. Also, the dynamic linking adds extra levels of indirection to invocations.

When the service-based integration platform must bridge between different communication protocols, PROTOCOL PLUG-INS [Völter et al. 2005] can be used to realize translation between the different protocols.
A Pattern Language for Service-based Platform Integration and Adaptation

**PROTOCOL PLUG-IN** [Völter et al. 2005]

**Problem** A distributed application must support multiple communication protocols at the same time. The protocols used for the scope of, e.g., single requests or different clients should be configurable at runtime.

**Solution** PROTOCOL PLUG-INS contain implementation details at the communication-protocol layer and provide common interfaces for different communication protocols. For the configuration of their parameters, the PROTOCOL PLUG-INS offer either an API or a configuration file.

**Decision Drivers** PROTOCOL PLUG-INS abstract from communication protocol details and allow flexible support for several communication protocols. They can also allow configuration and optimization of the communication protocols used.

4.3 Interface Design

**Fig. 7: Integration Adapter: Example Design**

**Fig. 8: Interface Design**
When developing a service-based integration platform, we need to expose interfaces to the application. Through these interfaces the applications built on top of the integration platform will be able to invoke the remote platform services. Common interfaces are also needed for monitoring, adaptation etc. In the simplest case, we can simply expose the PROXIES and ADAPTERS, as discussed in the previous section. However, often we are faced with additional interface design requirements, such as the unification of or the abstraction from interfaces, supporting different protocols or channels, optimizing invocation flows, avoiding redundancies in interfaces, or supporting multiple interface versions.

We might have the same requirements for one or more of the platforms to be integrated. For instance, many legacy applications do not expose an appropriate service-based interface. Sometimes it is more appropriate to first craft an appropriate service-based interface design for each of the platforms, and then to develop a service-based integration platform that offers a unified interface.

When designing interfaces for platforms or integration platforms, the design of the data transfer might be an important concern. Transferring data over the network between two distributed applications can be very expensive when the number of calls increases. Therefore, we can use DATA TRANSFER OBJECTS [Fowler 2003] which hold all the data to be sent. DATA TRANSFER OBJECT transfers the needed information within a single call. DATA TRANSFER OBJECTS may wrap primitive data types (e.g., integers, strings) or other DATA TRANSFER OBJECTS.

From the viewpoint of the client of a platform, interface unification is often important. Platforms expose multiple interfaces, often in multiple versions. The interfaces exposed by the platforms are often not the interfaces required by the applications using the platforms. The FACADE pattern [Gamma et al. 1994] describes a general way to unify interfaces.

A FACADE [Gamma et al. 1994] provides a coarse-grained interface on fine-grained components. In distributed systems, a REMOTE FACADE [Fowler 2003] can be used to specify a single point of access for a group of components which provide complex services in order to mediate client requests to the appropriate components. A REMOTE FACADE can also aggregate features of different components into new and/or higher-level services. It does not contain any domain logic and can use data from DATA TRANSFER OBJECTS. Using bulk accessors for the data ensures that invoking on the remote interface remains efficient.

A GATEWAY [Fowler 2003] is another variant of FACADE that represents an access point to an external system used by an application. The application thus becomes independent from the specific interfaces of the external system and also from its internal structure.
REMOTE FACADE [Fowler 2003]

**Problem** Interaction between objects is better understood when small objects have small methods, which leads to a fine-grained behavior. However, using fine-grained interactions when making calls between process or machine boundaries can be very expensive in terms of performance. Any object that is intended to be used as a remote object needs a coarse-grained interface to minimize the number of calls needed for a process.

**Solution** A REMOTE FACADE translates the coarse-grained methods onto the underlying fine-grained objects. Thus, it separates distinct responsibilities into different objects. A bulk accessor is used to replace a number of getters and setters of the underlying objects with one getter and setter.

**Decision Drivers** REMOTE FACADE provides access to a fine-grained object with a coarse-grained interface. Using a coarse-grained object model improves performance because of the reduced number of calls. It adds, however, additional programming effort, as the remote calls have to be translated into smaller internal calls.

GATEWAY [Fowler 2003]

**Problem** Complex interfaces lead to complicated applications. When there is a need to call an external API that is difficult to understand and use, this complexity is spread through the whole system.

**Solution** A GATEWAY is a wrapper that translates a specialized and complicated API into a simpler API. All applications that need to call this API call instead the API offered by the GATEWAY.

**Decision Drivers** The introduction of a GATEWAY makes a system easier to test and any possible changes in resources flexible. When the source API changes only the GATEWAY component needs to be modified. When implementing a GATEWAY an issue that has to be considered is the handling of exceptions and return values from the source API.

When a platform needs to support consuming and providing remote objects through multiple channels, a SERVICE ABSTRACTION LAYER [Vogel 2001] can be used. It introduces an extra layer which contains all the necessary logic to receive and delegate requests originating from the different channels. To create a SERVICE ABSTRACTION LAYER a FACADE can be used to offer an interface for creating and for sending service requests.

SERVICE ABSTRACTION LAYER [Vogel 2001]

**Problem** A system or platform must allow for providing and consuming remote objects through multiple channels, i.e., remoting technologies and transport protocols. This channel support should be independent from the core invocation handling for remote objects. New channels should be addable on demand.

**Solution** The SERVICE ABSTRACTION LAYER adds an extra layer which receives and mediates requests originating from different channels. Each channel contains a channel adapter which translates requests back and forth between the backend and frontend channel formats.

**Decision Drivers** SERVICE ABSTRACTION LAYER separates business from communication logic, thus clients become decoupled from the business services. Therefore, changes in the business logic do not affect the client implementations, as the clients use stable generic interfaces to interact with the remote system. The SERVICE ABSTRACTION LAYER increases, however, the level of indirection of requests. The introduction of this separate layer may reduce efficiency as all requests have to be processed at runtime.

We show in Figure 9 an example of interface design by implementing a FACADE which uses data from different DATA TRANSFER OBJECTS. The FACADE aggregates functionality from two application components and exposes an interface for integration with the remote platform. In this example, an ADAPTER inserts additional interface adaptation between the FACADE and the remote platform services. By providing a SERVICE ABSTRACTION LAYER, as illustrated in Figure 10, we support multiple remoting technologies through three different channels: a JMS, a SOAP, and a REST Interface. A FACADE unifies the different channels and exposes a common interface for the remote platform.

Another issue related to the design of interfaces is that the interfaces provided by platform applications are subject to adaptations and/or extensions due to changing requirements. To support different client-specific interfaces, related functionality can be grouped in separate EXTENSION INTERFACES [Schmidt et al. 2000b] and the common functionality can be included in a root interface. We illustrate in Figure 11 an example of an EXTENSION INTERFACE design. Clients access component functionality via interfaces. A component may provide multiple extension interfaces which implement the root interface functionality. Clients create new components and specify the initial extension interface using a factory associated with each component type.
Fig. 9: Interface Design with Facade and Integration with Adapter

Fig. 10: Interface Design with Service Abstraction Layer

Fig. 11: An exemplary Extension Interface Design
EXTENSION INTERFACE [Schmidt et al. 2000b]

**Problem** The interface provided and exposed by a component is subject to adaptations and/or extensions due to changing requirements of client components. Similarly, an anonymous number of clients requires alternative, client-specific interfaces for component interfaces. Being limited to a single and monolithic component interface means that changes propagate into existing client components in an uncontrolled manner.

**Solution** Related functionality is grouped and exported via separate EXTENSION INTERFACES. This grouping results from domain-specific (e.g., functional views) and/or temporal bindings (e.g., interface versioning). Common and/or administrative functionality (e.g., for selecting a particular view or version) is exposed by a root interface, to be included by each single EXTENSION INTERFACE.

**Decision Drivers** The use of EXTENSION INTERFACES decreases the coupling between the clients and components. The clients depend only on the interface roles they actually use, which ensures that they do not break when signatures of services change or new services are added to the components. To extend the functionality of a component only new EXTENSION INTERFACES need to be added. EXTENSION INTERFACES can also be aggregated to offer a new functionality of a component that aggregates other components. EXTENSION INTERFACES, however, may cause additional indirection and runtime overhead, as they are introduced between the components and the clients. It can also lead to increased complexity of client programming, as the clients must decide which EXTENSION INTERFACES are suitable for their use case.

4.4 Communication Style

For each connection between two components in the platform integration solution, follow-on decisions about the communication style must be made. For instance, once the design decisions for integration and adaptation, as well as interface design, have been taken at the component or service level, decisions on the communication style between the components must be tackled. In this section, we focus on the different options for connecting distributed components. That is, in the platform integration design space these design decisions are especially relevant for the connections between applications and the service-based integration platform, connections between the service-based integration platform and the platforms, distributed connections between the platforms, and connections among distributed components within the service-based integration platform.

A basic option is to use synchronous invocations for the connection between two distributed components. Often synchronous invocations are realized following the REMOTE PROCEDURE INVOCATION pattern [Hohpe and Woolf 2004]. The remote application may respond
either by sending a result value or a void result, unless an execution problem occurs and an exception is sent back. In a platform integration solution, this synchronous invocations option will rarely be used because synchronous invocations can lead to slow and unreliable systems, as the communication of the calling application must block until it receives the result.

Thus, in the following, we mainly focus on the asynchronous communication style and study the various options for implementing it.

<table>
<thead>
<tr>
<th>REMOTE PROCEDURE INVOCATION</th>
<th>[Hohpe and Woolf 2004]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem</strong></td>
<td>Applications in different programming languages that run on different platforms need to share data and processes.</td>
</tr>
<tr>
<td><strong>Solution</strong></td>
<td>Using REMOTE PROCEDURE INVOCATIONS means that each application offers a remote interface to interact with the other applications. Thus, one application can get or change data from another application by calling its remote interface.</td>
</tr>
<tr>
<td><strong>Decision Drivers</strong></td>
<td>Applying the pattern results in tightly coupled applications. It is difficult to deal with application downtimes, such as system crashes or downtimes for maintenance, as incoming invocations will get lost during the downtimes. Hence, REMOTE PROCEDURE INVOCATION based systems might be more unreliable than, e.g., MESSAGING based systems. Synchronous REMOTE PROCEDURE INVOCATION may lead to slow and blocking applications.</td>
</tr>
</tbody>
</table>

Applications that communicate with each other using asynchronous communication do not need to block their execution, but they can continue with other tasks while they are waiting for the results of their invocations. The asynchronous invocation patterns offer many alternatives for invoking a remote service asynchronously. They describe asynchronous variants of the REMOTE PROCEDURE INVOCATION pattern. In particular, when a result or application error needs to be delivered either a POLL OBJECT [Völter et al. 2005] or a RESULT CALLBACK [Völter et al. 2005] can be used. A FIRE AND FORGET interaction [Völter et al. 2005] does not return any result or acknowledgment to the application that invokes a remote object, but only offers best effort semantics. When a notification that the request arrived to the remote application is necessary, then SYNC WITH SERVER [Völter et al. 2005] can be used instead of FIRE AND FORGET.

<table>
<thead>
<tr>
<th>FIRE AND FORGET</th>
<th>[Völter et al. 2005]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem</strong></td>
<td>A client application wants to notify a remote object for an event. Neither a result is expected, nor does the delivery have to be guaranteed. A one-way exchange of a single MESSAGE is sufficient.</td>
</tr>
<tr>
<td><strong>Solution</strong></td>
<td>A FIRE AND FORGET operation is performed by the communication middleware without acknowledging the processing or delivery status to the client. The thread of control is yielded back to the client immediately.</td>
</tr>
<tr>
<td><strong>Decision Drivers</strong></td>
<td>The FIRE AND FORGET pattern provides non-blocking communication with unreliable transmission. That means that the client is not notified of errors in transmission or execution of the remote object. The remote object does not deliver any execution results to the client.</td>
</tr>
</tbody>
</table>

While FIRE AND FORGET offers one-way communication, SYNC WITH SERVER follows a REQUEST-ACKNOWLEDGMENT communication style [Hohpe and Woolf 2004].

<table>
<thead>
<tr>
<th>SYNC WITH SERVER</th>
<th>[Völter et al. 2005]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem</strong></td>
<td>A client application needs to ensure higher reliability of asynchronous invocations than FIRE AND FORGET, but does not require the transmission of a result.</td>
</tr>
<tr>
<td><strong>Solution</strong></td>
<td>The client sends the invocation as in FIRE AND FORGET but waits for a reply from the server about the successful transmission of the invocation. The communication middleware blocks only until the notification of the successful reception of the invocation arrives and then continues the execution.</td>
</tr>
<tr>
<td><strong>Decision Drivers</strong></td>
<td>SYNC WITH SERVER ensures successful transmission of requests and makes the remote invocations more reliable than FIRE AND FORGET. It introduces, however, additional latency, as the client must block until the notification of successful reception is received. Thus, there is a trade-off between higher reliability and worse performance in comparison with FIRE AND FORGET. Also the server application cannot inform the client for application errors as the execution of the remote invocation happens asynchronously.</td>
</tr>
</tbody>
</table>

RESULT CALLBACK and POLL OBJECT offer the REQUEST-REPLY [Hohpe and Woolf 2004] communication style. POLL OBJECT can be used in an imperative programming model.

In contrast to POLL OBJECT, RESULT CALLBACK requires an event-based programming style to consume the result. It has the benefit over POLL OBJECT to support immediate reaction upon the arrival of a result.
**POLL OBJECT** [Völter et al. 2005]

**Problem** Remote invocations of a client must be processed asynchronously but the client needs the result to continue its computations.

**Solution** A POLL OBJECT receives the results from a remote invocation on behalf of the client. The client periodically queries the POLL OBJECT for the results. The client can continue with other tasks and when the results are available to the POLL OBJECT the client can fetch them the next time it queries the POLL OBJECT.

**Decision Drivers** The server side stays oblivious to the client side POLL OBJECTS. The pattern offers more reliable communication compared to FIRE AND FORGET, as the result is an implicit acknowledgment, but it cannot immediately inform the client about an incoming result.

**RESULT CALLBACK** [Völter et al. 2005]

**Problem** The client needs to be informed about the results of its asynchronously invoked operations once the results become available to the communication middleware.

**Solution** A callback-based interface for remote invocations is provided on the client which passes a callback object to the communication middleware upon a remote invocation. After the invocation, the client can continue with other tasks. When the call completes and the results become available a callback is invoked on the client to process the result.

**Decision Drivers** The pattern has the same basic decision drivers as POLL OBJECT. RESULT CALLBACK is preferred over POLL OBJECT when an immediate reaction on the incoming result is needed. While POLL OBJECT works well with the imperative programming model of today’s OO application, RESULT CALLBACK requires an event-based programming model to handle the callbacks. The same or different callback objects can be used for different invocations of the same type.

In asynchronous remote invocations, ASYNCHRONOUS COMPLETION TOKENS [Schmidt et al. 2000a] are used to associate the callback with the original invocation. The pattern fulfills the same role as the CORRELATION IDENTIFIER pattern [Hohpe and Woolf 2004] discussed below.

To ensure reliability of communication and increase decoupling of the integrating platforms, MESSAGING [Hohpe and Woolf 2004] provides the most convenient solution. The integrating applications exchange MESSAGES [Hohpe and Woolf 2004] via a MESSAGE CHANNEL [Hohpe and Woolf 2004] which can be either a POINT-TO-POINT CHANNEL [Hohpe and Woolf 2004] or a PUBLISH-SUBSCRIBE CHANNEL [Hohpe and Woolf 2004]. The difference between them is that in the first case we have only one receiver of the requests and in the second case the messages are broadcasted, as there exist multiple receivers-subscribers of the messages.

The communication using MESSAGES can be either one-way or two-way. In a one-way communication, the sender directs a message towards a receiver using a one-way channel, without waiting for any notification or result of its request. A two-way communication requires a two-way channel to allow delivery of responses (void, result values, or exceptions). A REQUEST-REPLY communication can be implemented in different ways combining different asynchronous communication styles. For example, the client can first receive an acknowledgment of its request and then poll for the results (REQUEST-ACKNOWLEDGE-POLL [Daigneau 2012]) or get notified about the delivery of its request and receive the request results with a callback service (REQUEST-ACKNOWLEDGE-CALLBACK [Daigneau 2012]).

**MESSAGING** [Hohpe and Woolf 2004]

**Problem** Applications that are developed independently, are built in different languages, and run on different platforms need to share data and processes in a responsive way.

**Solution** MESSAGES are used to transfer packets of data frequently, immediately, reliably, and asynchronously using customized formats.

**Decision Drivers** MESSAGING offers high reliability, as sending a message does not require both systems to be up and running at the same time. Instead, message queues in the messaging system can temporarily store the messages when systems are down. Hence, systems become more decoupled from each other than in REMOTE PROCEDURE INVOCATION. However, complexity increases, as many decisions about the messaging system, such as the message formats, the message routing, the message transmission and the connection of the applications to the messaging system, etc. have to be made.

The PUBLISH-SUBSCRIBE CHANNEL is the version of the PUBLISH-SUBSCRIBER [Buschmann et al. 2007a] pattern that applies for messaging. Apart from messaging, the POINT-TO-POINT and PUBLISH-SUBSCRIBER styles can be also used in synchronous or asynchronous
remote invocations for unicasting and multicasting respectively. Hence, \textit{PUBLISH-SUBSCRIBER} is an alternative to \textit{POINT-TO-POINT} connections, mainly discussed so far, that can be applied for all communication styles discussed in this section.

\textbf{PUBLISH-SUBSCRIBER} [Buschmann et al. 2007a]

\textbf{Problem} Data changes in one place, but many components depend on this data and have to be updated. That means, multiple application components need to be notified about changes in one component.

\textbf{Solution} One dedicated component takes the role of the publisher and the other components are its subscribers who subscribe in order to get notified for state changes of the publisher. An object can be a subscriber to many publishers and can also play the role of both the subscriber and publisher.

\textbf{Decision Drivers} Publishers are loosely coupled to subscribers. \textit{PUBLISH-SUBSCRIBER} allows listening for events without disturbing the communication flow. Thus, it can also be used for debugging and logging purposes. However, using \textit{PUBLISH-SUBSCRIBER} may introduce security issues, as any subscriber is able to look at the events generated by the publisher. Although \textit{PUBLISH-SUBSCRIBER} offers high scalability, it does not guarantee the delivery of events to the subscribers.

As in synchronous \textit{REMOTE PROCEDURE INVOCA TIONS} or in the asynchronous \textit{POLL OBJECT} or \textit{RESULT CALLBACK} patterns, messages are also often used to deliver messages in \textit{REQUEST-REPLY} style.

\textbf{REQUEST-REPLY} [Hohpe and Woolf 2004]

\textbf{Problem} Two applications communicate through an exchange of \textit{MESSAGES}. Each \textit{MESSAGE} realizes a one-way conversation. The sending application requires a reply from the receiver of the initial \textit{MESSAGE}.

\textbf{Solution} To realize a two-way conversation, pairs of request and reply \textit{MESSAGES} are exchanged. Depending on the intended coupling between the sender and receiver, the reply \textit{MESSAGE} is sent either via the request’s back channel or, alternatively, via its own communication channel.

\textbf{Decision Drivers} The pattern provides a two-way message transmission on a two-way channel. The requestor is always notified about the successful or unsuccessful completion and/or of the result of its request. The two processes (request and reply) are decoupled. If the connection between requestor and receiver fails before the reply is sent, then the requestor must re-send its request unless messages are persistent.

As in the \textit{SYNC WITH SERVER} pattern, messages can be delivered in \textit{REQUEST-ACKNOWLEDGE} style.

\textbf{REQUEST-ACKNOWLEDGE} [Daigneau 2012]

\textbf{Problem} A client would like to notify a system about the fact that a request has arrived or about an interesting event after a request has arrived. Requests do not need to be processed right away, and the responses to the requests do not need to be delivered.

\textbf{Solution} When a service receives a request it forwards the request to a background process and then returns an acknowledgment containing a unique request identifier.

\textbf{Decision Drivers} \textit{REQUEST-ACKNOWLEDGE} communication alleviates the problem of unavailable resources or request spikes, as unlike \textit{ONE-WAY} communication it lets the client know that the requests have been received and will be processed. However, the client does not get informed about application errors that may happen during the process execution.

4.5 Communication Flow

Transferring distributed service invocation data from the client applications to the integrated platform services, mediated by the service-based integration platform, requires from the software designer to make design decisions related to the data transformations in the service-based integration platform. These decisions touch a variety of concerns, e.g., the routing of the invocations and their invocation data to the intended receivers, as well as all data transformations at different levels (e.g., data representation, marshalling, data transport).

The communication flow perspective considers the flow of requests and replies through the integration platform as a series of data transformations, performed by infrastructure components. The relevant data items are in-memory objects (e.g., \textit{DATA TRANSFER OBJECTS}) and \textit{MESSAGES}.

While many patterns described in this section have originally been described in the context of messaging, in variants they can also be applied in combination with the other (asynchronous) invocation patterns.
If sophisticated message or invocation routing is required, a MESSAGE ROUTER [Hohpe and Woolf 2004] offers an appropriate solution. The MESSAGE ROUTER listens at the incoming, or frontend, message channels and redirects the messages intercepted towards the necessary processing chains and towards the actual backend receivers, i.e., the platform services. With such a central routing component, there is a single point of responsibility for administering the routing rules and to configure the processing chains needed for preparing the messages for the individual platform services. The MESSAGE ROUTER can be made configurable following the COMPONENT CONFIGURATOR pattern (see also Section 4.2).
**MESSAGE ROUTER** [Hohpe and Woolf 2004]

**Problem** A message channel can be used to exchange messages of varying structure and content, thus requiring different processing steps. To decouple the processing steps, without introducing dedicated message channels, the MESSAGES have to be filtered depending on a set of conditions.

**Solution** A MESSAGE ROUTER component inserts a special filter that consumes a MESSAGE from one incoming MESSAGE CHANNEL and republishes it to a different outgoing MESSAGE CHANNEL, after having evaluated certain filter criteria. MESSAGE ROUTERS distribute MESSAGES either to a fixed destination or redirect the messages depending on the message content (see CONTENT-BASED ROUTER).

**Decision Drivers** A MESSAGE ROUTER centralizes message filters and the actual filter functionality (routing rules) in a single component which becomes the single point of maintenance and failure. Message routing however adds to the processing overhead of the integration platform.

In a service-based integration platform, routing is often performed by a CONTENT-BASED ROUTER [Hohpe and Woolf 2004]. As a variant of the MESSAGE ROUTER, this router accesses the message content, i.e., envelope and body elements, to evaluate the standing routing rules against the data extracted from the messages. This way, the routing conditions can be set and transmitted by the MESSAGES themselves (e.g., in their envelopes or by their type annotation), rather than by providing the routing-critical data through an external source.

**CONTENT-BASED ROUTER** [Hohpe and Woolf 2004]

**Problem** An integration solution deploys a MESSAGE ROUTER to have MESSAGES processed adequately. However, the message routing is not to be decided by external factors or by fixed routes, but rather by the messages’ content.

**Solution** A CONTENT-BASED ROUTER has the capacity to examine the message content and distributes the message to a different channel based on its content (e.g., routing data in the message envelope, its structure, or message values).

**Decision Drivers** As a kind of MESSAGE ROUTER, the use of a CONTENT-BASED ROUTER allows for centrally managing message routing; without the need to modify either the client applications or platform services. The routing functions, however, may change frequently causing extra maintenance effort. The recipients also have no control over the routing process.

Content-based routing is not applicable only for the exchange of MESSAGES representing service invocation data (e.g., implicit invocations on domain objects), but it can also be used to differentiate between invocation messages and messages carrying invocation-unrelated or opaque types of data. Imagine application scenarios which involve setting up audio/video streaming data between client applications and platform services (i.e., here, streaming services). Such data requires alternative processing steps when being mediated by the integration platform; for example as part of an optimization which bypasses routing and processing steps applicable to handshake and invocation messages only.

Besides acting as a matchmaker between messages and the available data transformation tasks, a MESSAGE ROUTER also allows for composing processing chains to be applied on selected messages. Message processing and filter components can be organized in a PIPES AND FILTERS [Buschmann et al. 2000; Avgeriou and Zdun 2005] style. Finally, the processing chains can be constructed in a way so that the delivery to the responsible platform service is performed by republishing the transformed message to a backend, or outgoing channel.

The data sent across the network will not always be used by the data receiver, i.e., the platform services, as it is; whatever the dominating communication styles or the communication flow approach is (MESSAGING vs. explicit component invocations). For example, for exposing FACADE interfaces using DATA TRANSFER OBJECTS, the backend invocations must be decomposed into a series of invocations upon one or more platforms and their input and output parameter types. The MESSAGING equivalent to FACADES and DATA TRANSFER OBJECTS are compound messages, with each of the part messages addressing a distinct platform service.

A SPLITTER [Hohpe and Woolf 2004] disassembles the compound messages into their constituents which are expected by the target platform(s). Sometimes multiple elements need to be collected and reassembled to be delivered to their final destination and to be accepted by the platform services as message endpoints. On the back channel, e.g., for asynchronous REQUEST-REPLY interactions, there is the need for re-assembling the resulting data elements into a composite reply message. This bears the risk of duplicates or an out-of-order reassembly. The SPLITTER can, for instance, split the messages in the integration platform that are sent to the different platform services.

Conversely, an AGGREGATOR [Hohpe and Woolf 2004] merges individual messages or element subsets thereof into compound messages to be delivered to the platform services. The AGGREGATOR detects the related elements as well as their right order according to their CORRELATION IDENTIFIERS. On the reply channel, an AGGREGATOR might require a SPLITTER. The AGGREGATOR can for instance aggregate messages in the integration platform that are sent to the different platform services.
SPLITTER [Hohpe and Woolf 2004]

**Problem** Messages passing through an integration solution consist of multiple elements, each of which must be processed separately. The incoming message appears as a composite message.

**Solution** A SPLITTER component is incorporated into the integration platform to break up the composite message into a series of individual elements or element subsets. Each element subset is then published as a distinct message. Common elements of the initial message are maintained in the resulting messages (e.g., identification and sequencing tokens) in order to allow for re-integrating reply messages later on.

**Decision Drivers** The primary driver for adopting a SPLITTER is that target platform services, which are grouped by a dedicated FACADE, must receive the respective data subsets for which they are responsible when answering invocations dispatched onto the facade interface. A SPLITTER can not only break a message into its repetitive data chunks, but also a large message into individual messages to simplify the further processing. On the negative side, there is data overhead in the resulting messages (e.g., CORRELATION IDENTIFIERS, timestamps). Also, extra processing effort is needed for aggregating reply messages afterwards.

AGGREGATOR [Hohpe and Woolf 2004]

**Problem** We need to combine the data of individual, but related messages so that the aggregated data can be processed as a whole by the target system.

**Solution** An AGGREGATOR component observes the message stream, collects and stores individual messages based on filter criteria and identification tokens (CORRELATION IDENTIFIERS) until it has received a complete set of related messages. After having assembled a single message out of these parts, based on selected aggregation strategies, the resulting message is published for delivery to the target system.

**Decision Drivers** In order to be able to aggregate incoming messages to an AGGREGATOR the messages need to have a correlation that indicates which messages belong together. The aggregation decisions described by the aggregation algorithm bear the risk of introducing extra development effort and additional processing complexity (depending on the aggregation strategy). At the same time, message aggregation can realize an important optimization strategy in service-based platform integration: the batch processing of service invocations. For batching, multiple content-wise unrelated messages are packed into a single composite message to be delivered to a platform service.

Apart from this whole-part mismatch between senders and receivers at the level of messages, the data contained in the messages might simply be too excessive or incomplete to be (efficiently) processable by the receivers. There are many possible reasons for this problem. For example, the domain model of the target system might only correspond to a subset of the source domain model. Or certain auxiliary invocation data contained in a message might not be relevant; for instance, the data might only be required for add-on services or constitute metadata relevant only for the underlying middleware technologies. Sometimes, security requirements demand the removal of message parts (e.g., identity tokens). In such cases, a CONTENT FILTER [Hohpe and Woolf 2004] is included in the processing chain of a message to extract and drop excessive data.

CONTENT FILTER can be applied in the integrated platform to filter the messages that pass through it.

CONTENT FILTER [Hohpe and Woolf 2004]

**Problem** When sending messages from one system to another, it is common that situations occur in which the target system is not interested in all data included in the forwarded messages.

**Solution** A CONTENT FILTER component is provided to remove unneeded, obsolete, or protected data items from a message.

**Decision Drivers** Data filtering for messages is a critical adaptation mechanism to be supported by an integration platform. It can also simplify the structure of a message (e.g., convert a tree into a flat structure), or remove redundancy or ambiguity in a data structure. CONTENT FILTERS can act as pre-processors before handling messages using MESSAGE TRANSLATORS.

Requirements for additional data can result from domain model mismatches, different underlying middleware, or security requirements. In such cases, a CONTENT ENRICHER [Hohpe and Woolf 2004] augments the message with the missing information by accessing external data sources or the message context. CONTENT ENRICHER can be used in the message processing of the integration platform.
CONTENT ENRICHER [Hohpe and Woolf 2004]

**Problem** When sending messages from one system to another, it is common that situations occur, in which the target system requires more data than included in the original message.

**Solution** A CONTENT ENRICHER is a specialized message transformer which accesses data sources external to the message processing system to add the missing data.

**Decision Drivers** A CONTENT ENRICHER may need to consult external resources to find the required data, based on references contained in a message. This may not only affect the message processing throughput negatively, but also the synchronization decoupling in the communication flow. External resources might require synchronous communication, which requires special treatment for the enricher component (e.g., introducing a special message consumer internal to the integration platform). If the external source is an integrated platform service, the CONTENT ENRICHER acts as a variant of AGGREGATOR.

A frequent source of mismatch between client applications and platform services are incompatibilities between the data formats supported. Such format mismatches involve differences in data models, data types, data representation, and data transport techniques. When using explicit component invocations and in-memory object representations of the invocation data, a DATA MAPPER [Fowler 2003] can be used to deal with the unaligned or non-canonical data formats between integrating platforms. A DATA MAPPER transforms, e.g., the data from one object type to another. For dealing with marshalling and transport protocol mismatches, the DATA MAPPER can use the services offered by MARSHALLER [Völter et al. 2005] and PROTOCOL PLUG-IN [Völter et al. 2005] components as offered by the underlying middleware framework.

DATA MAPPER [Fowler 2003]

**Problem** Two or more components exchange data in terms of in-memory objects. However, the receiving interfaces require an incoming data format which is incompatible with the object structures exchanged. A format mismatch is the consequence, covering inconsistencies at the level of two incompatible data models, and in-memory representation styles.

**Solution** Incorporate an auxiliary transformation layer in the component architecture which hosts a group of mapper components. These DATA MAPPERS provide model and representation transformations for data objects. Certain model and representation strategies are so captured as dedicated modules and can be applied to data objects exchanged between different pairings of client applications and platform services.

**Decision Drivers** The processing of service data in terms of in-memory objects requires collocation of the two components for which the data is mapped. PROXIES can be used when process or machine boundaries need to be crossed. When crossing process and machine boundaries, MESSAGE TRANSLATORS take the role of format filters.

MESSAGE TRANSLATOR [Hohpe and Woolf 2004] can be incorporated into the processing chains of MESSAGES for transposing them from one data format into another. In the processing chains, the MESSAGE TRANSLATORS usually come last; as they operate on the already filtered messages (see Figure 15). The MESSAGE TRANSLATOR can reside, for instance, in the integration platform and translate between the client application message formats and the message formats of the platform services.

MESSAGE TRANSLATOR [Hohpe and Woolf 2004]

**Problem** Two or more interacting applications operate on different message formats and there is a message format mismatch.

**Solution** A dedicated filter component is used by the MESSAGE ROUTER to transform an incoming message format into an outgoing message format. A single MESSAGE TRANSLATOR can cover any, or even all, levels of transformation (model, type, representation, and transport).

**Decision Drivers** A key driver for providing a MESSAGE TRANSLATOR component in the integration platform is to avoid enforcing a uniform message format throughout all (existing and future) client applications and platform services; if possible at all. A MESSAGE TRANSLATOR preserves the independence of the integrated clients and services in terms of message formats; the adoption of different standard formats or the modification of proprietary ones remain a localized event without propagating to the other participants. However, depending on the degree of heterogeneity within the distributed systems in terms of message formats to be mated, there is the risk of high complexity (i.e., combinatorial explosion of message format transformations). This is particularly valid for each MESSAGE TRANSLATOR realizing all transformation steps, potentially in redundancy to other translators. This risk can be reduced by limiting a single translator’s responsibility to a single transformation step (e.g., marshalling) and combining them to message processing chains for a given integration scenario (e.g., a pair of client application and platform service).
A particular source of complexity in the communication flow design of a service-based integration platform is the repeated dis- and reassembly of data items; and bridging between process synchronization styles. Both the content and the synchronization decoupling require the identification of decoupled parts. Important examples are message parts of disassembled compound messages (see SPLITTER pattern) or non-blocking backend replies to blocking frontend requests. Also, the permanent interleaving of related messages in the integration platform requires a message tracking mechanism.

Adopting CORRELATION IDENTIFIERS [Hohpe and Woolf 2004] is an adequate design decision to address such tracking requirements. For asynchronous communication styles, where one has to (implicitly or explicitly) identify exactly a corresponding pair among multiple communication parties, these identifiers are also referred to as ASYNCHRONOUS COMPLETION TOKENS [Buschmann et al. 2007a].

As for designing the frontend interfaces, for instance, CORRELATION IDENTIFIERS can be employed and stored in the FACADE to track the resulting backend invocations at a per-request level. One option is to maintain the identifier in the service descriptions, such that every communication with the service needs to refer to a specific CORRELATION IDENTIFIER. Alternatively, a FACADE could also store the CORRELATION IDENTIFIERS in the DATA TRANSFER OBJECTS, if available (see Section 4.3).

In a MESSAGING infrastructure, the CORRELATION IDENTIFIER is extensively used to realize conversational interactions, i.e., for exchanging and processing messages such as in REQUEST-REPLY interactions and MESSAGE SEQUENCE interactions [Hohpe and Woolf 2004], to name but a few.

In some particular cases, one might need to integrate two or more software platforms that do not support compatible CORRELATION IDENTIFIER mechanisms. The reason can be that either one of the platforms does not support CORRELATION IDENTIFIERS or both support CORRELATION IDENTIFIERS but their CORRELATION IDENTIFIERS are not simply interchangeable. In such cases, components, such as the PROXIES or ADAPTERS in this pattern language, are often introduced for mediating the communication and data exchange between these platforms, i.e., translate and temporarily store the CORRELATION IDENTIFIERS. This can be realized, e.g., by letting the mediators maintain an additional table to map the CORRELATION IDENTIFIERS from one communication partner to the CORRELATION IDENTIFIERS of the other communication partner, and vice versa.

**CORRELATION IDENTIFIER** [Hohpe and Woolf 2004]

**Problem** Using asynchronous remote invocations or messages, the requesting component does not block, even if a reply is expected as part of an asynchronous REQUEST-REPLY conversation. However, upon incoming an asynchronous reply, the receiving components must align the incoming reply to the corresponding request.

**Solution** To correlate two messages, such as a request and a reply processed at different times, both messages embed a unique identity token. In the request message, the CORRELATION IDENTIFIER is referred to as the request ID. The reply message then includes a token, the CORRELATION IDENTIFIER, which matches or refers to the initial request ID.

**Design Drivers** For general MESSAGING and asynchronous REMOTE PROCEDURE INVOCATION scenarios, it is sufficient to generate and maintain unique IDs for the messages. In service-based platform integration, it is also required to preserve a reference to the client application having submitted the original request, along with the unique identifier for the message as such. This is needed to resume serving a service invocation for a particular client across the asynchronous frontend connector of an INTEGRATION ADAPTER. Assuming that the client applications should not and cannot be altered to emit an additional identifier token, to be used as the CORRELATION IDENTIFIER or as a part of it, the integration platform has to maintain a mapping table which aligns the requesting clients and the CORRELATION IDENTIFIERS issued.

The design decisions become embodied in the way the service-based integration platform lays out the communication flow in terms of component interactions as depicted in Figure 15. Depending on the decisions taken on the communication styles (see Section 4.4), there are various possibilities to laying out the data transformation infrastructure in the service-based integration platform. For example, the integration platform can be built using basic MESSAGING principles. Alternatively, an explicit invocation style between transformer components can be applied. Both variants are sketched out as exemplary setups in Figure 15.

The initial drivers for opting for either approach are the communication styles supported by the components to integrate (i.e., the client applications and the platform services), as well as the decoupling strategies to be implemented by the integration platform. For example, while a straightforward OBJECT ADAPTER can be easily constructed using explicit invocations, an INTEGRATION ADAPTER with an asynchronous frontend connector which attaches to the client applications can leverage an underlying MESSAGING infrastructure. Both approaches allow for minimizing, or ideally turning obsolete the need for modifying either the client applications and/or platform services to assist in the data transformations required. Client applications or platform services not enabled for MESSAGING can be integrated using bridging PROXY/ADAPTER components which act as the sending or receiving message endpoints to a frontend and backend channel, respectively. This way, client applications and the platform services do not have to be manipulated even for overcoming such a mismatch in communication style.
5. MOTIVATING EXAMPLE REVISITED

In this section, we apply our pattern language to the industry case study introduced in Section 3, with special emphasis on a single integration scenario as illustrated in Figure 3. We show, thus, how the patterns presented in this paper can be used individually or in combination for building a service-based platform integration solution. In order to make informed decisions, we consult the pattern descriptions. For selecting follow-on patterns, we review the pattern interconnections as documented for the pattern language throughout Section 4.

In Figure 16, we present an excerpt from the integration architecture containing the three backend platforms (i.e., the Yard Management System YMS, the Warehouse Management System WMS, and the Remote Maintenance System RMS), the Virtual Service Platform (VSP), and the operator application. We select appropriate patterns from the pattern categories Integration and Adaptation and Interface Design. The services introduced for the integration scenario are grouped into components; for example, the services initiateVoiceCall and endCall are enclosed by the component CallHandling. A FACADE component (OperatorAppFacade) provides a common application interface for invoking the different platform services. The component CommunicationFlowManager embodies the communication flow between the operator application and the integrated platforms. In order to invoke the remote platform services, ADAPTER and PROXY components are introduced into an integration layer below the CommunicationFlowManager component. In case the access to the remote services does not require any interface adaptations, a PROXY component is used (e.g., TruckManagementProxy, PositionReportingProxy, etc.). Otherwise, an ADAPTER component is deployed to resolve interface incompatibilities, i.e., changes to the parameter structure (e.g., CallHandlingAdapter and VideoHandlingAdapter) or to operation names.

In Figure 17, we introduce two exemplary designs of the communication flows between the operator application and the three platforms as provided by the CommunicationFlowManager component in Figure 16. For this, we select appropriate patterns from the pattern category Communication Flow. In the first communication flow diagram, the platforms send notifications. These notifications are assigned CORRELATION IDENTIFIERS before they are enriched with platform details required by the operator (WMSNotificationEnricher, YMSNotificationEnricher, RMSNotificationEnricher). The atomic notifications are aggregated into one notification (in the PlatformNotificationAggregator) which is then delivered to the operator application. To receive the notifications, the operator is expected to subscribe to the appropriate notification channel (following the PUBLISH-SUBSCRIBER pattern). In the second communication flow diagram, the operator invokes the operation moveTruckToDock and the request receives a CORRELATION ID. Afterwards the request is logged using a PUBLISH-SUBSCRIBER interaction style. Finally, the request is added to the TruckRequestsQueue message queue at which the YMS is listening for incoming tasks.
A Pattern Language for Service-based Platform Integration and Adaptation

Fig. 16: Excerpt from the Integration Architecture

Fig. 17: Examples of Communication Flows
6. RELATED WORK

Our pattern language describes a family of service-oriented architectures [Hentrich and Zdun 2012] for platform-integration purposes. From this viewpoint, our pattern language complements architectural patterns emphasizing application, service, and business process integration in the large [Hohpe and Woolf 2004] by adding the architectural description of another integration style, service-based integration platforms, which has not been explicitly discussed in pattern language form before.

However, our work is closely related to many other pattern languages. The works most closely related are existing pattern languages that are integrated into ours. In particular, we used selected patterns from those other pattern languages in the specific context of service-based platform integration and adaptation. With this, we documented missing links and decision drivers (i.e., forces and consequences). For decisions relating to adapting and integrating platform services, patterns on SOA integration [Hentrich and Zdun 2012; Vogel 2001; Buschmann et al. 2007b] and software architecture integration [Avgeriou and Zdun 2005; Gamma et al. 1994] have been consulted. Regarding decisions on service and component interface design, we identified relevant patterns in two pattern languages on general software and service-oriented architectures [Buschmann et al. 2000; Avgeriou and Zdun 2005], as well as in the pattern language on enterprise application integration [Fowler 2003]. For decisions regarding the communication style and communication flow in the platform integration solution, patterns on object remoting middleware [Völter et al. 2005] and messaging systems [Hohpe and Woolf 2004] have been integrated.

Zdun and Hentrich [Hentrich and Zdun 2009; 2012] describe a pattern language for process integration architectures. This pattern language can be combined with the patterns described in this paper, by using either a workflow or a business process engine (called macroflow engine in [Hentrich and Zdun 2009; 2012]). To the same effect, a microflow engine for handling the integration logic can be incorporated into the service-based integration platform. That is, the process-based integration patterns can be used to replace the communication style and communication flow patterns, described in our pattern language, to support the process-based integration style in a service-based integration platform.

Closely related are contributions about managing the variability in a SOA, in particular the pattern-based approaches by Khan et al. [Khan et al. 2011]. By managing variability, we mean the separated specification of differences (i.e., variation points and the binding options) between the variants to be instantiated from a SOA (see also SEPARATE DESCRIPTION OF VARIABILITY in [Voelter 2009]) and the corresponding implementation of the so-specified variants using different variability techniques (e.g., static and dynamic parametrization, injection). SOA variability [Ruokonen et al. 2008] must be dealt with or must even be realized in a service-based integration platform. Prominent examples for SOA variations stressed in our paper are forms of runtime re-configuration of platform internals (e.g., interface adapters) by applying the COMPONENT CONFIGURATOR pattern. Varying service interfaces (e.g., interface evolution over time, client-specific interfaces) and substituting services (e.g., for different client bases) are addressed by the various patterns touching service interface design (e.g., FACADE; see Section 4.3).

While there have been numerous contributions for describing and managing variability in SOAs (see more recently, e.g., [Chang and Kim 2007; Narendra and Ponnalagu 2010; ai Sun et al. 2010; Nguyen et al. 2011]), to the best of our knowledge, the approach by Khan et al. [Khan et al. 2011] is the only other pattern-based approach existing so far. Khan et al. [Khan et al. 2011] present a catalog of six patterns for describing recurring variability problems and variability solutions in SOAs. The patterns touch on service parametrization (e.g., distinguishing between invocation parametrization and extrinsic configuration through various configuration descriptors), on conditional routing of service invocations (e.g., based on business rules), on providing signature compatibility for services, on client-driven service adaptation (e.g., by injecting behavior into existing services through configuration interfaces), and on service cloning (to serve client applications independently from each other). While the pattern collection sketches the considerable range of variability in a SOA, it is severely limited. Some patterns relate to very specific SOA variants, in particular SaaS systems; others address general issues of SOA adaptation and SOA integration without referring to the extensive body of existing pattern works on these subject matters. For example, the PARAMETER and SERVICE WRAPPER patterns closely relate to patterns on component configuration and different variants of the ADAPTER pattern [Gamma et al. 1994]. Finally, the relationship among the six patterns and between the six patterns and related patterns are not described.

In recent years, software architecture is often seen as the principal design decisions governing a system [Taylor and van der Hoek 2007; Jansen and Bosch 2005]. An important idea is to document the design rationale of the architecture using means such as architectural design decisions (ADDs). ADD approaches propose prescriptive architectural design decisions meta-models for structuring, relating, and navigating the actual templates created for a given architecture [Tyree and Akerman 2005; Kruchten et al. 2006; de Boer et al. 2008; Zimmermann et al. 2009; Capilla et al. 2011]. As conceptual meta-models, ADDs are decomposed into compounds of decision descriptions, decision alternatives, decision groups, artifacts and activities related to individual decisions; and the relations between these building blocks.

By reusing and linking ADDs to patterns as design artifacts [Zimmermann et al. 2008; van Heesch and Avgeriou 2009], documenting architecture decisions can be substantially facilitated [Harrison et al. 2007]. The work by Zimmermann et al. has integrated various SOA pattern languages in a reusable ADD model. For Zimmermann et al. [Zimmermann et al. 2009] patterns take the role of architectural decision alternatives, i.e., they represent the solution space of an ADD. The authors further discriminate between four different levels
of decisions (executive, conceptual etc.), and different kinds of patterns as decision output are proposed for each level: At the executive decision level, process and requirement analysis patterns enter as decision options. Then, at the so-called conceptual decision level, high-level architectural patterns (e.g., BROKER vs. SHARED REPOSITORY) and critical technology choices follow. At the third and technological decision level, design and remoting patterns apply as decision alternatives [Völter et al. 2005]. For asset-level decisions, implementation-level patterns and concrete technology options apply.

Drawing upon their experiences using a structured Wiki as an ADD documentation tool and a SOA-centric industry case study, Capilla et al. [Capilla et al. 2011] consider architectural patterns as concrete decision alternatives. Regarding process-driven SOA patterns [Hentrich and Zdun 2012], patterns enter the concrete solution catalog maintained by the Wiki-based documentation tooling. In terms of their ADD meta-model, SOA patterns predominantly represent decision alternatives (ADAlternatives), but also decision issues and outcomes.

7. CONCLUDING REMARKS

A typical (service-based) application often relies on functions provided by different (service) platforms specialized for different domains. As a consequence, many applications are faced with the requirement for integration of services from one or even multiple heterogeneous platforms. However, platform integration is a rather challenging task as the software architects and developers are confronted with several decision designs at different levels of abstractions and different levels of granularity. There is a considerable amount of patterns targeting various aspects of service-based integration and adaptation [Gamma et al. 1994; Buschmann et al. 2000; Buschmann et al. 2007a; Fowler 2003; Hohpe and Woolf 2004; Völter et al. 2005; Hentrich and Zdun 2009; Daigneau 2012; Hentrich and Zdun 2012]. Unfortunately, these patterns, on the one hand, have been documented with a different focus and, on the other hand, walking through several patterns scattered in different literature in order to arrive at a design solution for service platform integration is tedious and time-consuming.

The major contribution of this paper is to revisit the existing patterns and design decisions regarding service-based integration and adaptation of platforms and organize them in a comprehensive pattern language such that software architects and developers can systematically reference and follow the pattern language to build up an appropriate platform integration and adaptation solution. The pattern language presented in this paper considers four essential high-level architectural decision categories in the context of service platform integration, which are Integration and Adaptation, Interface Design, Communication Style, and Communication Flow. Each category constitutes a number of architectural design decisions described in terms of relevant patterns and their relationships, along with their variations or alternatives and the decisive reasons leading to choosing these patterns. Based on the descriptions of this pattern language, the functional and non-functional properties of the service platforms, and particular requirements of the service-based applications built on top of the platforms, one might develop not only a platform integration solution but also a number of alternative configurations of the solution.

While our pattern language covers the core design space of service-based platform integration, there are many open issues relevant for the design of platform integration solutions, but not covered yet in our pattern language. Our future endeavors will consider categories such as monitoring, QoS and SLAs for platform integration solutions, more sophisticated adaptation options, and further patterns at lower levels of abstraction and finer levels of granularity. In addition, follow-on tool support would also be useful to enable software architects in better devising and utilizing an adequate set of questions for developing and documenting a certain architectural design in service platform integration and adaptation. The pattern language described in this paper will provide the basis for such tools supporting architectural design.

8. ACKNOWLEDGEMENTS

Thanks are due to Michael Weiss as our EuroPLoP shepherd and to all EuroPLoP 2012 workshop participants who contributed to improving our paper with their comments. This work was partially supported by the European Union FP7 project INDENICA (http://www.indenica.eu), grant no. 257483.

REFERENCES


Applying Patterns to Model-Driven Development of Automation Systems: An Industrial Case Study

CHRISTOPHER PRESCHERN, NERMIN KAJTAZOVIC, and CHRISTIAN KREINER, Institute for Technical Informatics, Graz University of Technology

Model-driven approaches enable more efficient system development, but are yet rather uncommon for the automation domain due to limited practical examples and guidelines on how to apply them. We present an industrial case study using code generators attached to a Domain-Specific Language for model-driven automation system development. The case study applies the Pipes&Filters architectural pattern and the Template&Metamodel and Code Attributes code generation patterns. We present the results of the pattern application for the case study and discuss the general usage of the presented patterns for model-driven automation system development.

1. INTRODUCTION

Automation systems (AS) were in history often developed in a traditional way, where each project is individually programmed. This is still state-of-practice for many small and medium enterprises (SME) [Leitner et al. 2012]. The traditionally used methods just allow limited software reuse and require a lot of maintenance effort. Individually developed automation systems often contain many software bugs and the product has to mature quite some time and has to be maintained intensively in order to reach an acceptable quality level [Preschern 2011]. Therefore, the implementation of automation systems following the traditional approach can be time consuming and rather costly. Alternative approaches like model-driven development can decrease product development and maintenance time in order to decrease the overall costs, but are rarely applied by automation SME due to lack of practical examples on their effective development [Wang and Tan 2006].

This paper presents the application of such a model-driven approach to an industrial case study. To achieve structured software reuse for a given automation domain, a Domain-Specific Language (DSL) is developed. DSL models can be constructed and provide information to code generators for generating automation software. We discuss the development of the DSL, its corresponding code generators, and the automation software with focus on the usage of architectural and code generation patterns. We use the Pipes&Filters, the Template&Metamodel, and the Code Attributes pattern to improve the system flexibility and maintainability. We discuss the pattern application and consequences for the specific case study and for automation systems in general.

Section 2 of this paper covers related work on automation systems, model-driven approaches, and patterns in this context. Section 3 presents the automation system case study with focus on the application of patterns and Section 4 discusses the pattern application for automation systems in general. Section 5 concludes this work.
2. RELATED WORK

Several examples for model-driven automation system development approaches can be found in literature. Compared to our work, most of the following papers cover model-driven approaches for automation systems in general. We focus on a specific automation domain and describe basic architectural decisions for the construction of a model-driven framework. However, there are several parallels to our work, especially for the mapping of models to the automation system source code.

Fantuzzi [Fantuzzi et al. 2009], for example, maps automation programs to UML diagrams and proposes the application of design patterns to automation software. Function blocks of programs represented with state charts are mapped to UML classes. In another paper he describes patterns addressing the mapping of the automation software to the UML representations [Fantuzzi et al. 2011]. The process of generating automation source code from UML models is also described in [Vogel-Heuser et al. 2005] where the mapping of the UML blocks to IEC61131-3 automation program code is covered in detail. The close relationship between high abstraction UML automation models and the function block diagram automation language is discussed in [Thramboulidis 2011]. Other papers on model-driven automation system development present alternative platform independent model representations with petri nets [Thieme and Hanisch 2002] or XML [Estévez et al. 2007] including their mapping to IEC 61131-3 automation languages and to vendor-specific automation products.

Just very few attempts can be found in literature where patterns are applied to develop DSLs. A catalog of design patterns for DSLs is presented in [Spinellis 2001] where patterns describing general domain model structures and code generating approaches related to DSLs are discussed. Furthermore, patterns for model-driven software development are presented by Völter et. al. [Völter 2004]. This catalog also includes several patterns for DSLs. Guidelines for the development of DSLs for automation systems in particular are presented in [Maga et al. 2011]. Kleppe [Kleppe 2008] covers the development of DSLs in general and presents guidelines on how to construct DSLs.

3. CODE GENERATION BASED AUTOMATION SYSTEM: CASE STUDY PISCAS

This section presents the PISCAS (pisciculture automation system) project, an industrial case study on a model-driven approach to develop fish farm automation systems. We focus on the DSL, in particular its mapping to the automation domain, and on the code generators. We discuss the application of code generation patterns (Template&Metamodel, Code Attributes) and architectural patterns (Pipes&Filters) to develop a flexible and easily maintainable model-driven framework for PISCAS.

The intention of the model-driven approach is to make the development of several automation systems in the same domain more efficient. This is achieved by providing a framework which allows to develop new fish farm automation systems easily by generating the actual automation system software out of a system model which is constructed with the help of a DSL (see Figure 1). For the PISCAS project this framework consists of a DSL specifically developed for fish farm systems and code generators attached to this DSL. This enables systematic reuse of domain-specific program components and allows future systems in the fish farm domain to be developed more effectively. Constructing the DSL and the code generators, of course, requires additional effort which usually pays off in terms of decreased overall development and maintenance costs if several systems in the same domain are being developed [Leitner et al. 2012].

The PISCAS projects was carried out by the author as a master’s thesis at Graz University of Technology. Currently, two PISCAS system are in operation which allows us to present results on the consequences of the applied patterns. The two systems in operation are of about the same size and contain about 20 ponds each equipped with feeding automates, oxygen sensors and pond aerators. Further information on the project can be found in [Preschern 2011] or at the PISCAS website [pis].
3.1 System Description

PISCAS automation systems maintain feeding and water oxygen level control functionality of fish farms. Several other functionalities such as timed actuator switching or water level supervision including an alarm system are also part of PISCAS. Usually, all fish farm automation systems maintain these core functionalities and just differ in the number of ponds and their feeding and oxygen control equipment. This equipment is, for the automation domain, changed rather often why a flexible automation software is required. Still, it has to be ensured that such changes are safe, because an oxygen control failure, for example, can mean the death of all fish in a pond. Applying a model-driven approach enables one to build reliable fish farm automation systems by developing and modifying systems in a more structured way.

Fish farm automation components such as feeders or oxygen sensors and the ponds they are allocated to, are modeled graphically with a DSL developed with the MetaEdit+ [met] DSL tool. Out of the information in the DSL model, PISCAS generates the automation software source code, the automation hardware I/O-mapping, the full automation system visualization, network component configuration files, and the project documentation including an installation plan.

PISCAS uses Bernecker + Rainer (B&R) [br] automation hardware, because B&R systems bring the advantage that their source files including vendor-specific project files such as hardware structure files, hardware mapping files, and visualization files are stored in XML format. Therefore these files can easily be parsed and changed by a code generator.

3.2 PISCAS Architecture

In this section we describe the PISCAS domain model structure, the meta-model, the automation software structure, and the mapping of DSL elements to the automation software. Furthermore, we show how the Pipes&Filters pattern can be applied to the PISCAS architecture to make it more flexible.

3.2.1 Domain Model Structure. We choose a DSL domain model instead of a feature oriented domain model for PISCAS in order to keep the domain model complexity low. Usually, automation domain models contain a low number of element types which strongly vary in amount and connections for different systems within a domain [Leitner et al. 2012]. Such systems can very well be described by DSLs. Basic DSL elements refined by properties build a hierarchical, nested domain model. Such domain models are very effective for automation systems, because later domain refinements, which are quite common in the automation domain, can easily be integrated into the DSL [Maga et al. 2011].
3.2.2 Meta-Model. The meta-model representing the fish farm domain uses the GOPPRR (Graph-Object-Property-Port-Role-Relationship) meta-meta-model which is used by MetaEdit+. In Graphs, Objects containing Ports are arranged and can be connected by Relationships which can be refined by Roles. All these entities can be further specified by Properties. PISCAS maps groups of physical devices (e.g., oxygen sensors, feeding units) just varying in amount and behavior for similar fish farm systems, to Objects. The amount of devices is defined by the number of Objects arranged in a PISCAS Graph and the behavior is further specified by Properties. The direct mapping of PISCAS Objects to physical automation devices and the usage of a graphical DSL enables implicit modeling of the position of physical automation devices. This information is used for the generation of a system visualization and a documentation including an overview of the automation plant. In the meta-model, Objects can be connected through Relationships which represent physical wire connections between automation devices. This straight mapping makes the language very intuitive for domain experts, because the graphical model can represent the automation system in a way that is familiar to them. Ports attached to Objects are docking points for their Relationships and add additional semantics to wire connections. The PISCAS meta-model requires each Object at least to have an ‘input’ and an ‘output’ Port as well as a ‘name’ Property. This convention defines a standardized interface between Objects and makes the code generators attached to the DSL simple and less dependent on Object types. This is very important for the PISCAS DSL, because during system development Objects had to be modified or new Objects had to be added rather often. Figure 2 shows an example model of the PISCAS DSL adhering to the described meta-model. A more detailed description of the meta-model can be found in [Preschern et al. 2012].

3.2.3 Mapping of DSL Elements to the Automation Software. In the fish farm automation software, DSL Objects are mapped to function blocks. These function blocks contain a regular program file describing the function block behavior and a variable declaration file defining internal, interface, and configuration variables. Interface variables represent DSL Ports and are connected according to Relationships between Objects in a separate global mapping file. Configuration variables are handled in a separate initialization file, where constants are assigned to the variables based on the information in the DSL Properties. The amount of Objects in the fish farm model determines the number of function block instantiations in the automation software. This number is configured for each Object type in a global variable declaration file as the bound of a function block array. Table I gives a complete overview of the DSL mapping to the automation software and to physical automation devices.

Applying Patterns to Model-Driven Development of Automation Systems: An Industrial Case Study — B5-4
For the deployment of a specific fish farm automation system, a generic fish farm automation system is configured by code generators according to the DSL model. The configuration of the generic fish farm automation software is strictly separated from regular automation programs which are developed as function blocks for the fish farm domain. This allows the code generators to be kept simple, because the code generator developer does not need detailed knowledge of the function block program functionality as long as the function blocks provide the specified interface and configuration variables. Figure 3 gives an overview of PISCAS automation software files and shows which parts of the software are hard-coded by the developer of the generic fish farm automation system and which parts are generated for specific fish farm automation systems.

Table I. PISCAS: Mapping of the physical system to MetaEdit+ concepts and to the automation software

<table>
<thead>
<tr>
<th>Physical system</th>
<th>MetaEdit+ concepts</th>
<th>Automation software</th>
</tr>
</thead>
<tbody>
<tr>
<td>automation plant</td>
<td>Graph</td>
<td>automation software</td>
</tr>
<tr>
<td>device</td>
<td>Object</td>
<td>function block instantiation</td>
</tr>
<tr>
<td>wire</td>
<td>Relationship</td>
<td>interface mapping</td>
</tr>
<tr>
<td>-</td>
<td>Hole</td>
<td>-</td>
</tr>
<tr>
<td>wire connection</td>
<td>Port</td>
<td>interface variables</td>
</tr>
<tr>
<td>device attribute</td>
<td>Property</td>
<td>function block parameter</td>
</tr>
</tbody>
</table>

Applying Patterns to Model-Driven Development of Automation Systems: An Industrial Case Study — B5-5
3.2.4 Application of the Pipes&Filters Pattern. The fish farm domain is very dynamic and changed a lot during the beginning of the project due to limited domain knowledge and incomplete customer requirements. Therefore, a mechanism to allow flexible domain changes was required.

We applied the Pipes&Filters pattern which allows to flexibly combine system components by defining an input and output format for them. The pattern allows connecting any component’s input to any other’s output and defines the data flow between the components. Further information on the pattern can be found in [Trowbridge 2004]. The Pipes&Filters pattern can be found on different levels of the PISCAS DSL architecture. On a physical level it is implicitly present. Automation devices represent filters which are connected by wires, representing pipes. Through the straight mapping of this concept to the DSL, the Pipes&Filters pattern can also be found here, where DSL Objects are filters and their Relationships are pipes. The DSL interfaces are kept very simple and all connections between any types of objects are allowed in the meta-model. At the automation software level, Objects are mapped to function blocks, which now represent the filter concept. The connection of these function blocks can be seen as pipes. Figure 4 gives an overview of the Pipes&Filters pattern in the PISCAS architecture.

For PISCAS, several automation device types had to be added later on to the DSL. Such modifications are now easily possible because of the applied Pipes&Filters pattern where new Objects do not require new DSL interfaces. Also the basic interface definition of the function blocks in the automation software stays the same and allows any possible arrangement of function blocks. A liability of the application of Pipes&Filters is that the PISCAS DSL models has to be carefully checked for the validity of connections, because any assembly of elements is allowed by the pattern.

Applying Patterns to Model-Driven Development of Automation Systems: An Industrial Case Study — B5-6
3.3 PISCAS Code Generation

In this section we describe the PISCAS code generators, which generate software out of fish farm DSL models. We present pattern-based solutions for the code generators addressing automation software maintenance issues.

3.3.1 Application of the Template&Metamodel Pattern. PISCAS contains many automation software and network configuration files which have to be generated. These files have to be maintained, if possible, independently from the model-driven framework. For the PISCAS project it is very important to have a compiling piece of automation software which allows maintenance without the code generation step necessary in order to check if the automation software still compiles and works as intended.

We applied the Template&Metamodel pattern presented by Völter [Völter 2003] which uses a template and a system model for code generation. A generic configurable system template is developed for a domain and a specific system is modeled following the rules of a corresponding meta-model. The final software is generated by adding the information in the model to the template. We implemented a generic B&R fish farm automation template which is configured by the code generators during program generation. The generators do not include the actual automation programs, but just set parameters for the fish farm software. The Template&Metamodel pattern is also used for code generation of network device configuration files. For example, network router configuration files were exported and taken as templates. Modifications of these files (e.g. IP address) are then added to the configuration file during the code generation.

With this pattern, the template compiles and can be maintained independently from the code generators. Thus, the automation template developer can maintain the automation software without the need to work with MetaEdit+. During the maintenance of PISCAS, most of the necessary changes had to be made to the automation template (e.g. bug fixes, additional device types). Due to the application of the Template&Metamodel pattern these changes did not affect the DSL or the code generators. Additionally, the automation software contains many files which do not at all or just partially have to be changed by the code generator. The usage of templates allows easy code generation of these files without the need to explicitly implement them into the code generators. For the network device configuration files, the Template&Metamodel approach has the advantage that when changing non domain-related settings in the router configuration, just the template file has to be replaced and the code generator does not have to be touched.

3.3.2 Application of the Code Attributes Pattern. PISCAS required a mechanism allowing the automation developer to make changes to the generated code without making the code inconsistent with the DSL model.

We applied the Code Attributes Pattern presented by Völter [Völter 2003] which parses information added to the generated code and uses this information for program generation. When generating a PISCAS project for the first time, special comments defining user code sections are added to all automation program files (see Figure 5). User code sections allow modifications to the generated code which are preserved when the code is updated by the generators. This is sometimes necessary due to bug fixes or changes in the model. The DSL allows to model interfaces (global variables with defined name) which can be accessed from the user code section.

For the PISCAS project two actual systems were generated and both of them had non domain-related requirements. By implementing solutions to these requirements in the user code section, they were easily satisfied without the need to change the DSL. Some of the manual changes had to be implemented quickly during the initial operation of the fish farm system. The user code sections allowed making these quick changes directly to the generated automation software. However, the drawback of implementing functionality in the user code section is that this functionality is not visible in in the DSL model.
4. FORCES AND CONSEQUENCES OF THE PATTERNS’ APPLICATION

In this section we discuss the patterns applied to the PISCAS case study in general. We show forces specific for the automation domain in the context of model-driven development and discuss the pattern application and its consequences.

4.1 Pattern Application: Pipes&Filters for Automation DSLs

**DOMAIN-SPECIFIC FORCES.** - Often the full domain requirements are not known and automation domain changes often require to add or modify automation device types.

**APPLICATION.** - The Pipes&Filters structure affects the domain model, the code generators and especially the automation software. When looking at automation devices as filters and looking at wires as pipes, basically the Pipes&Filters pattern is already present on a physical level. On the domain model level, physical devices are modeled as basic model elements representing the filters. Connections between the elements represent physical wire connections and, therefore, are the pipes of the pattern. On the automation software level, primitives for filters are realized as function blocks with standardized interfaces. Interface connections represent pipes in the automation software.

**CONSEQUENCES.** - The system becomes more flexible and the code generators become more simple. Any kind of automation device can easily be connected by just setting appropriate restrictions in the domain model. Adding new components brings less changes for the code generators and for the domain model, because they can be kept independent from component types. A drawback is that DSL interfaces do not carry a lot of semantic information which might be necessary for complex automation domains. Another drawback is, that due to the direct mapping of DSL elements to function blocks, the pattern limits the DSL abstraction to the abstraction level of the function block diagram representation.

**KNOWN USES.** - The Pipes&Filters pattern is applied to the automation domain without the context of code generation in [Narzt 2000]. Component-like automation blocks connected by a DSL are also applied to the automation domain in [Maurmaier and Göhner 2009].

4.2 Pattern Application: Template&Metamodel for Automation Code Generation

**DOMAIN-SPECIFIC FORCES.** - Maintenance of the automation software should not be more complicated, but still the automation software has to be integrated into the model-driven development process.
APPLICATION. - A domain-specific generic automation system template is developed. This template is a regular automation project which is developed and maintained independently from the code generators. The code generators use information from the system model to configure the automation template.

CONSEQUENCES. - An advantage of the pattern is that it isolates the template from the code generators. This allows the code generator developer to use the automation template without having detailed insight into its functionality. The automation system developer can maintain and develop the template independently from the code generators by using normal automation development tools. This is especially helpful for using functionality such as simulation or debugging which is usually provided by automation development suites. A drawback of the Template&Metamodel pattern is that the separation of the automation template and the code generators requires keeping the two parts in a consistent state. This is particularly important if the template and the code generators are developed by different people.

KNOWN USES. - The Template&Metamodel pattern is applied to the home automation domain in [Voelter and Groher 2007].

4.3 Pattern Application: Code Attributes for Automation Code Generation

DOMAIN-SPECIFIC FORCES. - Automation systems often require handling of project-specific requirements which are not related to the domain and therefore are not covered by the domain model.

APPLICATION. - During the code generation, special sections (user-code sections) are compiled into each code file. These sections are marked by tokens which are hidden in comments. Automation developers are allowed to add code to these sections. Such manual changes of the generated system are parsed from the code before it is rewritten when updating the generated source code. The domain model has to provide an interface to link domain elements to the user code. This is realized by mapping domain related variables to global variables which can be used in the user code section.

CONSEQUENCES. - Every kind of non domain-related changes to the software can easily be made by the automation developer by adding user code. This has the advantage that software can be added by the automation developer without having to consider the code generation process. Changing the generated source in such a way does not imply that the generated artifacts become inconsistent with the model. It just means that non domain-related information is not added to the model but to the generated source code. However, in case of necessary changes which affect the whole domain, the automation developer can just add the relevant code in the user code section. This is easier at first hand, but it works against the basic idea of structured code generation. Another drawback is that if the automation developer wants to influence generated programs, he has to change the model by introducing interface variables for the user code which reduces the abstraction of the system model and makes it less intuitive.

KNOWN USES. - The Code Attributes pattern is applied to the home automation domain in [Voelter and Groher 2007].

5. CONCLUSION

This paper shows how the development of domain-specific model-driven automation systems can benefit from the application of patterns. The presented case study benefits from the use of code generation patterns in terms of automation code maintainability and it benefits from architectural patterns in terms of usability and flexibility. Model-driven approaches bring many problems regarding domain engineering which are new to people who usually develop automation systems in a traditional way by implementing each system in a family individually. The patterns applied to the PISCAS project provide an example on how to easily overcome some of these problems. The generalization of the applied patterns can be used to aid model-driven development in the automation domain and encourages using model-driven approaches for the automation domain where it is currently just rarely applied.
We think that automation system developers using model-driven development can benefit from the information given about the forces and consequences of the the application of the Pipes&Filters, Template&Metamodel, and Code Attributes patterns. The discussion of the pattern application can serve as a guidance on the choice whether the application of the pattern is suitable. Perhaps this paper even motivates a collection of patterns and their application for model-driven automation system development.

ACKNOWLEDGMENTS

Thanks to Ernst Oberortner for significantly improving this paper by giving helpful hints and comments during the shepherding process.

REFERENCES

PISCAS website. www.piscas.at.
Dynamically Reconfiguring Services in SOA Applications: A Pattern-Based Approach

Vallidevi Krishnamurthy and Chitra Babu

SSN College of Engineering,
Department of Computer Science and Engineering,
Anna University,
Chennai, India.
{vallidevik,chitra}@ssn.edu.in
http://www.ssn.edu.in

Abstract. Service Oriented Architecture (SOA) facilitates developing applications that are inherently dynamic in nature since the service binding happens at runtime by matching the functional as well as Quality of Service (QoS) requirements of the user. The running application can be dynamically reconfigured by monitoring the application for possible violations in the agreed QoS requirements. This paper proposes suitable patterns to facilitate such dynamic reconfiguration in any service-oriented application to achieve dynamic switching between different versions of the same application without manual intervention. A proposed pattern named the multiple level bridge pattern is also discussed in this paper which helps structure service interface and implementations for reuse across different application versions. This pattern based approach has been tested by applying it to a sample SOA based e-Shopping application.

1 Introduction

Service-oriented applications are inherently dynamic in nature since the service binding happens at runtime by matching the functional as well as Quality of Service (QoS) requirements of the user. The service oriented applications can be monitored to pro-actively detect the possible violations in the agreed QoS requirements which helps in dynamically reconfiguring the application. The monitoring[12] can be done either by the client himself or by a third party service. Assume that you are the service provider for a complete service-oriented application which has many versions(see [1] for an explanation on versions) such as basic, intermediate and advanced. Further, the service provider should be capable of providing new versions for the same application, whenever there is a need for an advanced version. In this context, you want to provide a dynamically reconfigurable application to the consumers where the dynamic reconfiguration takes place basically in two different ways.

- The first is to provide service substitution by another service at runtime that has the same service interface and implementation. This level of reconfiguration happens when a particular service which is currently active is not
meeting its QoS requirements or when a particular active service becomes unavailable.

- The second dimension is to switch among several versions of the same application. The need for this level of reconfiguration arises whenever the pre-defined rules for reconfiguration are satisfied.

This paper introduces a pattern language for the dynamic reconfiguration of services which is comprised of existing patterns from [10], [5] and [15]. In addition, the paper also deals with the reuse of services in different versions of an SOA application which is achieved through a proposed pattern named the multiple level bridge pattern. This pattern based approach has been tested by applying it to a sample SOA based e-Shopping application.

The proposed pattern is intended to aid the SOA application developers. When studied and applied in the evolution of SOA application development, the proposed pattern description helps in reusing the services in a well defined and structured manner.

The remainder of the paper is structured as follows: A motivating example is explained in Section 2. In Section 3 the pattern language for the dynamic reconfiguration is described. A thumbnail is also given for the existing patterns in the pattern language in Appendix A. In Section 4, the proposed pattern is explained. The motivating example is revisited in Section 5. Section 6 explains the implementation of the proposed pattern description and also the role of the pattern language to achieve dynamic reconfiguration. In Section 7, the related work is discussed and Section 8 summarizes and concludes.

2 Motivating Example: e-Shopping Application

Consider a service provider providing an entire SOA based e-Shopping application as a service. This application has two versions (refer Fig. 1). The first version $V_1.I_1$ provides a multimedia view of the application. There is second implementation for this version $V_1$ which is a view without any animation and it is named as version $V_1.I_2$. There is a second version, $V_2.I_1$ which is inherited from the second implementation of $V_1$. This second version has the features of the version $V_1.I_2$ with some additional features, such as notifying the administrator with an alarm when the product availability goes below a certain pre-defined threshold. This second version will be useful to the service consumers who would like to proactively refill their stocks so that they will not lose potential customers because of the lack of availability of their products. This feature is useful to the service consumers during special offer period. In addition, this version has an added feature of selecting an alternative payment gateway which has the capability of handling heavy loads which will be suitable during the special offer period.

The e-Shopping application is represented in the SOA Reference Architecture [2] as shown in the Fig. 1. The lower layer is the operational layer in which all the databases are represented. Above this layer, are the components or the implementation of the services, that constitutes the service component layer.
Services layer comprises of the interfaces that map to the implementations provided by the service component layer. Business process layer provides the flow of the service implementation. The consumer layer captures the corresponding user interface design. There are two different cases considered in which the dynamic reconfiguration can be achieved through version switching:

Case 1:
The calendar period corresponding to special offers specified by the service consumer is checked periodically. If the service consumer, is in need of $V_2.I_1$ or $V_1.I_2$, for certain calendar period alone, switching to the appropriate version happens automatically from $V_1.I_1$. After the specified calendar period elapses, the version $V_1.I_1$ replaces the exchanged version.

Case 2:
Certain service consumers may not be interested in specifying the calendar period at the beginning itself. In that case, if the load reaches a critical threshold, automatic switching from the version $V_1.I_1$ to any one of the alternate versions specified in the SLA, has to be enabled.

3 Pattern Language for Dynamic Reconfiguration

This section explains the context in which the pattern language is used, and the pattern language itself for the dynamic reconfiguration in SOA. Finally, this
section also provides an idea of how the pattern language and the proposed pattern binds with the service composition in SOA.

3.1 Context for the Pattern Language

![Diagram for Context for the Pattern Language]

Fig. 2. Context for the Pattern Language

The basic architecture or the context in which the pattern language can be applied is shown in Fig. 2. This architecture is different from the standard SOA as it has a monitoring agent and a reconfiguration engine in it which helps in dynamic reconfiguration. While developing any SOA application, there are many atomic services available. Those services are composed by the service composition engine. The service composition engine has the service composition logic, monitoring agent and the reconfiguration engine within it.

The service composition logic makes use of the service configuration to know about the nature of the various services. The services with the same configuration such as services with the same life cycle or the services which provides the same functionality can be grouped together. The services that are grouped are then composed together which forms the service composition logic.

The monitoring agent monitors the service composition logic of a particular version of the SOA application which will be active with its respective contracts established between the service providers and the consumers. This monitoring agent triggers the reconfiguration engine whenever there is any need for service substitution or there is a need to switch among different versions.

The reconfiguration engine which contains the component repository helps in the co-ordination of dynamic reconfiguration. A component repository is the one which contains the list of components arranged together that has the same common properties.
3.2 Pattern Language for Dynamic Reconfiguration in SOA

A thumbnail is given for the existing patterns that are discussed in the pattern language which is available in Appendix A. CONFIGURATION GROUPS\[15\] pattern will group remote objects with common properties, for example objects with the same QoS properties or that belong to the same life cycle. A server application can have multiple CONFIGURATION GROUPS at the same time. Using this pattern, the developer can have different versions of the same service oriented application as the services in the same version will have the same life cycle and can be composed to form a group. The service compositions are done by the CAPABILITY COMPOSITION pattern. This CAPABILITY COMPOSITION\[5\] pattern is used to solve a problem that requires logic outside of the service boundary.

The QOS OBSERVER\[15\] pattern is used to monitor the QoS of services for which it is registered. Once the QOS OBSERVER is triggered by the INVOKER the QOS OBSERVER starts intercepting the service and collects the QoS data. At the end of the service execution, INVOKER stops the QOS OBSERVER. Subsequently, the QOS OBSERVER analyzes the collected data and decides whether to trigger any reconfiguration or not. If the QoS parameters specified in the SLA, are expected to be violated it triggers the reconfiguration engine.

In order to switch between different versions of the SOA application, the reconfiguration engine can be implemented by the SERVICE CONFIGURATOR\[10\] pattern. The reconfiguration engine interprets and executes the code specific
The pattern language for dynamic reconfiguration is given in Fig. 3. The dynamic reconfiguration in the business process layer of the SOA reference architecture can be obtained by using those patterns.

As the user requirements may change at any time, there may be situations where a single service in the application needs to be substituted by some other service. In that case, an aspect oriented programming based class exchange pattern can be used as an alternative for the service configurator pattern and that is also shown in Fig. 3.

The implementations with the same configuration are grouped together with the help of configuration groups pattern. The service facade pattern is used to decouple the contract generated between the service consumers and the service provider. The communication between the service facade, the capability composition and the implementation are like a cascaded bridge pattern[14] which is explained in Fig. 4. Initially, the request from the consumer is processed through the service facade where it acts as an interface for the service consumers. The request is then delegated to the capability composition acts as the implementation for the service facade. Again the capability composition behaves as an interface for the original implementation which are grouped together by the configuration groups pattern. Thus, there are two bridges which are arranged in the form of cascaded bridge pattern. The capability composition plays the role of cascading the interface and the implementation part.

Depending on the contract generated with the different service consumers, the provider can come up with different service facades without disturbing the original service composition and also the functionalities of the individual services. The contracts generated between the service provider and the service consumer should be physically decoupled from their implementations. This can be achieved by generating the contract which could be very different from the underlying technology. Decoupled contract pattern helps in generating contracts this way.
The **multiple level bridge** pattern (see Section 5) which is the proposed candidate pattern helps in the reuse of services available in the previous version. With this pattern, the developer is capable of developing an advanced version of the same service with some additional features.

The proposed pattern and the pattern language are mapped to the context in Fig. 5. The patterns used in the various components of a SOA application with many versions, which can be exposed to multiple clients with different contracts are shown in this.
4 Multiple Level Bridge Pattern

4.1 Context
You are the provider for the service oriented application that comprises multiple services. First you are in a position to satisfy the service consumers for whom you have agreed the SLA. Secondly, you would like to provide additional features of the services in a service oriented application when it is going to evolve.

4.2 Problem
How to design the services when the change in the service implementation would and would not have any impact on the service contract?

4.3 Forces

– Ability to extend from any level.
  For example, version \( V_n \) of a SOA application need not inherit all the features from version \( V_{n-1} \). It should be possible to either retain or extend some of the features of the services in any of its previous versions. It should be backward compatible.

– Ability to evolve from its previous version without disturbing the already existing contract.
  Whenever there is any major change in the implementation, then it may lead to the change in the contract. In such a case, the new contract should not affect the already existing contract which is agreed with the existing customers.

4.4 Solution
If the new version of the application replaces the existing one, then the already existing contracts may be disturbed. (Here only the new version of the application will be existing and not the older one.) Whenever there are interface changes such as addition and removal of any operation in a service and policy changes such as change in quality of service, then there will be a change in the contract[1]. If there is no major change that induces change in the contract, then the service extension can be designed in the same namespace and the same version number exists for that. Otherwise, the service extension need to be saved in a new namespace and a new version number is given for such cases.

Apply a variant of the BRIDGE pattern which separates an internal service interface and service implementation. The internal service interface is to be referenced in the service compositions. Apply a hierarchy of bridge pattern which helps us to vary the interface and the implementation to any level separately.

The structure of the multiple level BRIDGE pattern in the view of service provider is shown in the Fig. 6. This is a variant of the BRIDGE pattern in which both the interface and the implementations are being extended to multiple levels.
For the same interface, many implementations can be mapped using the BRIDGE pattern. However, the interfaces and the implementations can be extended to multiple levels which is not possible using the BRIDGE pattern alone.

4.5 Consequences

The benefits of using this pattern are explained below.

**MULTIPLE LEVEL BRIDGE pattern,** exhibits the inherent characteristics of Service Oriented Architecture such as,

*Decoupling interface and implementation* This pattern decouples the interface and the implementation. Hence, there is no tight coupling between the interface and the implementation. At runtime, the interface and the implementation are bound which is a very much essential need for the Service Oriented Architecture.

*Hiding the implementation details from the client* As the abstraction and the implementation are decoupled from each other, any changes made in the implementation will be hidden from the client.

*Improved Extensibility* This pattern is helpful in extending the features of version $V_n$ to version $V_{n+1}$ of any service in the SOA environment i.e., the abstraction and the implementor hierarchies can be extended independently
in the same level (where there is no change in the contract) or it can be extended to subsequent levels (where there is a change in the contract).

**Extension Guideline** When the developers start extending for every small change, then it may lead to many versions unnecessarily. Whenever there is a change in the contract of the SOA application, then the extension to the next level can be done. Else it can be extended in the same level. This guidance to the extension is provided in the multiple level bridge pattern.

### 4.6 Known Uses

- CORBA supports the following Interface inheritance which is also called as diamond inheritance. It does not support the implementation inheritance and the multiple level bridge pattern also supports the interface inheritance to a great extend and supports implementation inheritance only when it is needed.

```java
interface A{ ... }
interface B:A{ ... }
interface C:A{ ... }
interface D:B,C{ ... }
interface E:A,B{ ... };
```

With this above CORBA inheritance code snippet, it is clear that, a part of multiple level bridge pattern is supported by the applications built in CORBA which uses the interface inheritance as explained above.

### 4.7 Motivating Example

For the previously discussed e-Shopping application in Section 2, the class diagram is shown in Fig. 7. The AbstractInterface, AbstractImplementation classes and their subclasses together represent the multiple level bridge pattern. The **Customer** class handles the responsibility of obtaining the personal details of the customer and registering them. The **Account** class, is responsible for authenticating the customers using username and password. Both the Customer and the Account details are stored in the database. There is an Administrator class with which the items in the e-Shop are managed by adding and removing the items and their details. There is a MonitorAgent class which monitors the number of people currently online. Whenever there is a need for change in interface or implementation, then the MonitorAgent triggers that corresponding change. There is a StockManager which checks for the stock availability in each and every Warehouse and refills the items if it goes below a pre-defined critical limit.

The multiple level bridge implementation is shown in the Fig. 8.
4.8 Application Of multiple level bridge Pattern in the e-Shopping Example

The mapping of the application scenario to the proposed pattern is shown in Fig. 9. At run time, object corresponding to any one of the versions is created.

Assume that version $V_1$ and the implementation $I_1$ of the application is composed of services, $s_1$, $s_2$, $s_3$, and $s_4$; Version $V_1$ and the implementation $I_2$ of the application is composed of services $s_{1.1}$, $s_{2.1}$, $s_3$ and $s_4$. $s_{1.1}$ has the features of $s_1$ with different implementation. Similarly, $s_{2.1}$ has the features of $s_2$ with different implementation. Such services of $V_1.I_2$ are created in the same namespace of the services of $V_1.I_1$. This second implementation helps to maintain the agreed SLA with minor changes in the existing implementation.

Assume that, version $V_2.I_1$ is composed of services, $s_{1.1}$, $s_{2.1}$, $s_3$, $s_4$ and $s_5$. Then at run time, if the application is suddenly switching from version $V_1.I_1$ to version $V_2.I_2$, then there are 3 changes totally.

- $s_1$ is exchanged with the service $s_{1.1}$
- $s_2$ is exchanged with the service $s_{2.1}$
- a new service $s_5$ which is not available in version $V_1.I_1$ but in version $V_2.I_1$ is added now in that application.

Thus, the services layer of the SOA reference architecture is dynamically reconfigured with this proposed pattern.
This pattern when used with the QoS observer [15], service configurator[10] and configuration groups [15] helps to switch between different service compositions in the business process layer or service composition layer of the SOA reference architecture.

5 Related Work

There are several existing research works which have addressed the evolution of services with design patterns in the context of SOA.

- Kaminski[11] has focused on the compatibility issues that arise in dealing with different versions and has proposed a pattern named chain of adapters. On the other hand, the focus of the proposed pattern based solution is towards dynamically reconfiguring the whole application by switching between different versions.
- Cascaded Bridge [14] is a series of linked bridges, in which intermediate implementors perform some of the implementations and then delegate the rest to the downstream. Whereas, the proposed pattern talks about having different instances of the various versions of the same application. The creation of those instances is carried out by multilevel inheritance. This pattern is not designed in a way that, if version $V_n$ is to be executed, then it is not
downstreamed from the top. Rather, it is connected directly to the version $V_n$ of the implementation which is designed by inheriting from version $V_{n-1}$.

- **Extension Interface** pattern [4] allows multiple interfaces to be exported by a component. This avoids bloating of interfaces when the developers extend or modify the functionality of the component.

- For using **Extension Objects** pattern [7] we need to support the addition of new or unforeseen interfaces to existing classes and you don’t want to impact clients that don’t need this new interface. Extension Objects lets you keep related operations together by defining them in a separate class.

Middleware based dynamic reconfiguration has also been discussed in the existing literature by many authors already.

- Xiaoying Bai [3] talks about DRESR which discusses the necessity of dynamic routing during dynamic selection and composition in SOA middleware.

- Geebelen [8] had proposed a technique for dynamic service composition in SOA middleware by using templates. Whereas we have suggested some collection of patterns for the same.
Manel Fredj[6] has proposed the SIROCO middleware platform, enabling the runtime, semantic-based service substitution. Here, service substitution is indirectly taking place when there is a version change.

Ibrahim[9] has proposed MySIM which is a spontaneous middleware that integrates services in a transparent way without disturbing users and applications of the environment. Lionel[13] has discussed about the reconfiguration in the middleware platform which is a component based one in the SOA.

6 Conclusion

A pattern language from the existing patterns is introduced for the dynamic reconfiguration of SOA based application. This paper also proposes a new pattern namely, the multiple level bridge pattern. As the known uses for this proposed pattern is missing, this pattern has been treated as a candidate pattern. With the proposed pattern, the service provider is capable of deciding how to extend the services when the change in the service implementation would and would not have any impact on the service contract. Thus, the service provider is capable of providing a flexible and dynamically extensible service-oriented application. Additionally, the service consumer obtains a whole SOA application as a service with an improved cost structure. Usually, most of the service providers providing the entire application as a service will be interested in providing all the advanced features in their application to their customers. In that case, the consumer will be paying even for what he is not using. The service facade pattern can be used to expose the services to different clients with varying contracts. Thus the provider gains his business both by using the multiple level bridge pattern and the service facade pattern. By using the service configurator pattern, the basic version alone is exposed to the service consumer. The advanced features are exposed to their consumers, only when there is a need for them. Hence, the consumers are can reap the benefit of paying for only what they are using.

7 Acknowledgement

The authors would like to thank sincerely, Stefan Sobernig and Uwe van Heesch for their continuous encouragement, support and tireless shepherding.

References


## Appendix

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVOKER [15]</td>
<td>There are many services which are registered for some monitoring purpose.</td>
<td>The INVOKER will invoke the QOS OBSERVER when a particular service which is registered for monitoring is to be executed.</td>
</tr>
<tr>
<td>CONFIGURATOR GROUPS pattern [15]</td>
<td>You architect and design a SOA application where there are many versions maintained for the same.</td>
<td>The atomic services (with the advanced features) of the same life cycle are available in a pool they are to be selected and configured into groups. Thus you have many versions of the same service-oriented application with the help of CONFIGURATOR GROUPS pattern.</td>
</tr>
<tr>
<td>SERVICE CONFIGURATOR [10]</td>
<td>As the QOS OBSERVER has suggested, you are ready with the selected version now. This version of the SOA application should replace the existing version at run-time.</td>
<td>For the new version to be replaced instead of the existing version, many configurations need to be done. Those configurations can be carried out by the SERVICE CONFIGURATOR pattern.</td>
</tr>
<tr>
<td>QOS OBSERVER [15]</td>
<td>You have many different versions of the same SOA application. You need to follow the SLA agreement to switch among those versions.</td>
<td>Some monitoring agent needs to check whether the system meets the SLA. Whenever there is a need for switching, then, the QOS OBSERVER which is acting as a monitoring agent triggers the reconfiguration engine.</td>
</tr>
<tr>
<td>SERVICE FACADE [5]</td>
<td>Different versions of the SOA application are available. The service provider needs to optimally market his business using these versions. The service consumer also need to be benefited by suitably using these different versions.</td>
<td>Different service consumers can come up with different contracts for those versions of the SOA application. Thus the consumer and the provider both are benefited.</td>
</tr>
<tr>
<td>DECOUPLED CONTRACT [5]</td>
<td>How can a service express its capabilities independently of its implementation?</td>
<td>The service contract should be physically decoupled from its implementation. This can be achieved by making the contract independent of the underlying technology.</td>
</tr>
<tr>
<td>CASCaded BRIDGE pattern [14]</td>
<td>Separate implementation dimensions into independent components so that they can be independently interchangeable, evolvable, and testable.</td>
<td>Use a CASCADING BRIDGE pattern. A CASCADING BRIDGE is a series of linked bridges, in which intermediate implementers perform some of the implementation chores and then delegate the rest to the downstream.</td>
</tr>
<tr>
<td>CAPABILITY COMPOSITION [5]</td>
<td>How to use the capability which is available out of the scope of the service?</td>
<td>This pattern composes the capabilities available in various services and achieves the goal of solving the larger problem.</td>
</tr>
</tbody>
</table>
Lessons from EuroPLoP: About Workshop Moderation
(EuroPLoP 2012)
Allan Kelly - allan@allankelly.net

1 Introduction
EuroPLoP 2012 will mark this writer's tenth EuroPLoP. During this time the author has learned.
This paper set out to capture some of the author's observations and learning about writers workshops, and to record the author's preferred way of workshop moderation.
The author does not claim significant originality in the workshop moderation format; rather, the process described is a fusion of observations about workshops and the author's one learning.

2 Audience
This paper is written for those participating in a pattern conference, e.g EuroPLoP or PLoP, and in particular those who will moderate writers workshops.

3 The workshop
For several years EuroPLoP workshop moderators have been encouraged to experiment with the writers workshop format. The author's observations and experiments lead to the creation of the Figure 1 flow chart - often referred to as a crib sheet.

This sheet has undergone several modifications over the years and has been used by multiple workshop moderators. The intention of this paper is to a) put the diagram on record and b) to add some descriptive text.
The crib aims to both help workshop moderators steer the workshop and to tweak the traditional workshop style (Schmidt, 2006, Coplien, 2001, Gabriel, 2002) to one this author feels is more effective. Changes to the style are centred on four forces: The Moderator, Questions about the paper, Time and Loudmouths.

Copyright 2012, Allan Kelly – allan@allankelly.net, www.allankelly.net
Permission granted to EuroPLoP 2012 / Hillside Europe to use and reproduce.

1 Oxford Dictionary of English defined crib as: informal, translation of text for use by students, especially in a surreptitious way.
2 Schmidt provides the shortest description of the three; Gabriel’s work is a 300 tour de force in workshop format and style.
Figure 1 - Writers workshop crib sheet

Pacing a traditional workshop to allow enough time for surface level comments to be aired and for deeper issues to be explored is hard, even for experienced moderators. In particular, experienced authors may find too much time in the traditional workshop style is eaten up with compliments -
"what we liked about the paper" - and deeper issues once exposed cannot be adequately addressed in the time remaining.

The format presented aims to flush out as many surface level, uncontentious, comments quickly in order to allow more time for in-depth discussion.

Workshop moderators are free to adopt this crib sheet as is, ignore it completely or make any modifications they deem desirable. Less experienced moderators may well prefer to stay closer to the crib while experienced moderators may feel it unnecessary.

3.1 Moderator role

In each workshop there is a nominated moderator –sometimes referred to as the moderator or workshop leader (Gabriel, 2002). Whatever the role is called someone takes the lead, at the request of the conference chairs, in ensuring the workshop operates.

While the workshop moderator will normally run the workshop for the first paper it is traditional that other members of the review group will act as moderators for other papers. In small groups each reviewer will have the opportunity to moderate one paper, in larger groups not everyone will have the opportunity. Typically no one person will moderate the discussion of more than one paper.

One of the first acts of the overall moderator therefore is to set out the order in which the papers will be reviewed and decide who will moderate each review session.

The duties of this leader typically cover:

• Agreeing the order the papers will be reviewed in
• Moderating at least one review session
• Soliciting volunteers to moderate the other review sessions
• Guiding first time workshop moderators as needed
• Keeping the workshop within time constraints
• Key holding for the workshop room
• Defend the absent author from over zealous reviewers
• Educate reviewers and authors about the workshop format and possibly about patterns in general if useful

Additionally workshop moderators are asked to participate in the final review of papers in the on-site shepherd group (previously writing group) which are assigned to the group so they may have a voice in accept the revised paper into the workshop (typically on the last day.)

In recent years some workshop moderators have taken to sending a personal e-mail to participants by way of introduction and to provide a friendly reminder to read papers before the conference.
3.2 Voice of the author

The traditional workshop style limits the voice of the author under review to constrained elements at the beginning and end of the workshop: the author is asked to read a section from the paper as introduction and, only at the end, to ask questions for clarification.

In recent years this author has been privy to several discussions in which it has been suggested that authors be allowed a greater voice. Indeed, some have suggested a more active role for the workshop moderator, one closer to a television interviewer asking the author questions about the paper.

Such a style calls for the moderator to take on a more active role and for a more engaged author during review. In so doing the moderator needs to devote preparation time and attention to reading a paper in depth and preparing for the review. Because dedicated preparation is required only those who can, and have, prepared for the paper can take on the role. Consequently the ad hoc assignment of workshop moderators also becomes more problematic.

The format presented in this paper allows the author’s voice to be heard more fully and framing questions to be resolved at the start without asking for significant extra work from the moderator.

3.3 Questions about the paper

This workshop format opens in the traditional way with the moderator asking the author to read a section from the paper under review. However, rather than immediately asking the author to become a "fly on the wall" the next step is to allow questions to the author from the gathered reviewers. This time is deliberately limited so it does not dominate the workshop – which is itself time limited.

Allowing questions allows reviewers to clarify their understanding of the nature of the paper, why it was written, who the audience was and similar questions. The answers to these questions frequently changes the comments made in the review.

Where pre-questioning is not provided for workshop participants sometimes spend time reflecting on the nature of the paper, the audience, or some other matter, which adds little by way of useful feedback to the author.

Normally the workshop chair starts this process with a question like:

"What motivated you to write this paper?"

"What did you hope to achieve by writing this paper?"

Or by asking a question they have from their own reading of the paper.

Like the questions illustrated in Figure 1 these are suggestions only. The moderator, and other group members, is free to formulate and ask their own questions.

Next the moderator asks the wider group is they have any immediate questions to the author about the paper.
The intention is not to hear from every reviewer, or to subject the author to integration. Rather the intention of these questions is to understand the context the paper was written in and surface anything the workshop reviewers find confusing about the paper that would prevent them from reviewing the paper.

For example, a paper without an audience statement frequently leaves the reviews wondering about the tone of writing, the level of detail in solutions and even the referencing style used. On occasions the review group can spend half the time debating who the audience was and what is the correct approach. Surfacing this early and hearing from the author directly helps the group situate their comments.

Authors usually appreciate the opportunity to answer direct questions and help direct the reviewers. Reviewers may then be able to point out tension between what the author states verbally and what is written in the paper. Caution is required to limit the time spent on opening questions. Every minute spent quizzing the author reduces the time for review. While there is no hard or fast limit to the number of questions which can be posed experience shows that five or six is the common maximum. Beyond this, or when the author and a reviewers start an open debate, the moderator should step in and move the workshop on.

Occasionally during the workshop review itself a question, or point of questionable understanding arises. Traditionally the reviewers would debate this among themselves, sometimes at considerable length, while the author remained silent as a “fly on the wall.” On such occasions the moderator may choose to temporarily recall the author, ask the question and receive the answer before having the author again absent themselves.

On these occasions the moderator themselves should ask the question to the author themselves. They should not allow debate and should seek to keep the authors presence short.

3.4 Loudmouths and silent ones

Traditional workshop style relies on reviewers to initiate comments, e.g. one reviewer decides to speak up on a point; they may be interrupted – gushed or ungushed3 – during their comments. At the end their comments someone else will continue with the theme or start another comment.

EuroPLoP regulars know that there are some workshop participants who like to talk, and there are others who remain silent. Neither position is right or wrong. However, powerful speakers can easily drown out quieter less forthright participants.

The following sections outline several techniques that can be used to encourage feedback from silent ones and put loudmouths to the background.

---

3 By tradition EuroPLoP and PLoP workshop participants say “gush” when they wish to quickly agree with a comment being made by another reviewer, and, “ungush” when they wish to disagree. Usually “ungush” is used as a marker for a comment which will be made in future.
at least momentarily. Sometimes it is only a matter of knowing how to politely pause a loudmouth and allow space for a silent one.

3.4.1 Once around the circle

Feedback to the author tends to fall into two categories: straight observations about the paper, whether something to keep or a suggestion for improvement, and discussion between reviewers which should lead to a concrete suggestion for improvement. However, too much of either can prevent the other from being discussed.

Therefore, in this style, the moderator is encouraged to go once around the circle: each participant is, in turn, given an opportunity to make their comments while other participants stay silent. Comments in response to these comments are held until the end of the circle – with the exception of “gush” which is permitted.

For example, when moderating this author frequently goes once around the circle for the initial round of “what we liked about the paper comments”. Starting to his left, or right, each participant is given an opportunity to state what they liked about the paper. When they have finished the person next to them takes their turn.

This procedure generates a great deal of feedback for the writer relatively quickly, most of these comments are general uncontentious and only need stating once. Consequently the review can quickly to move onto the more contentious issues and items requiring discussion.

An additional benefit is that every reviewer gets an opportunity to speak and can have their voice heard. When this occurs at the start of a workshop, as with “what we liked” this can be a powerful way of encouraging people to speak. Once a reviewer has spoken they are more likely to speak again.

It is common to also use this format for a final “what we liked about the paper” or “closing remark” comments when closing the review.

3.4.2 “Anyone who hasn’t spoken?”

Another way of soliciting comments from quieter members of the group is to ask:

“Would anyone who hasn’t spoken on this topic like to add anything?”

This simple question tells the loud mouths to give space while encouraging quiet ones to speak up. The moderator should not expect an immediate response; indeed several seconds of silence may elapse before someone speaks up. The moderator may follow up with a specific invitation,

“Peter, do you have anything to add here?”

Moderators should restrain their use of directed questions to named individuals. While this can be a useful technique for drawing a participant in to conversation some participants will find it awkward and really do want to stay silent.
3.4.3 Multiple summaries

At the opening of a pattern review the moderator normally asks for a reviewer to summarise the paper. The moderator is at liberty to ask for multiple summaries form different individuals. This can be another opportunity to encourage a silent one to speak up and express their thinking. Again the moderator might pose the question in such a way to encourage quieter review to speak up, e.g.

“Would someone who hasn’t given a summary before like to offer one?”

“Peter, what are the main points you see in the paper?”

3.4.4 Parking comments

At times loudmouths in the group can introduce a esoteric or metaphysical topic to the review group. When there are multiple, experienced, pattern reviewers in the group this can lead the discussion far from the paper and intimidate first timers.

Moderators might address this by deliberately “parking” a topic, for example:

“That is an interesting line of thought Andreas, maybe we could park it for now and address it later”

“Right now I’d like to stick to suggestions for improvement, maybe we can park that line of thinking and talk about it later”

“I suggest we hold that thought and continue it that conversation over lunch or in the bar”

In this context the term “to park” alludes to the practice of parking a car and leaving it unattended for a period of time. The car is still there when the driver returns after doing something else. Metaphorically questions are park and can be revisited later, as with a parked car.

Parking a question involves no loss of face but can serve as a useful filter for topics. Participants who think a topic is significant will ensure it is returned to, less significant topics might not be revisited – they remain parked, perhaps for discussion over lunch or beer.

4 Acknowledgements

Many thanks to Claudius Link for shepherding this paper – one of my more difficult papers to shepherd I may say. Thanks too to the participants in Workshop C at EuroPLoP 2012 for taking time to read and comment on this paper, and in particular Andreas Fiesser, Andreas Rüping and Klaus Marquardt.

The author, Allan Kelly, welcomes further feedback and comments on this paper and the issues discussed. Please contact Allan at allan@allankelly.net, www.allankelly.net or @allankellynet.
5 History

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2011</td>
<td>First draft</td>
</tr>
<tr>
<td>April 2012</td>
<td>Intermediate EuroPLoP review updates</td>
</tr>
<tr>
<td>June 2012</td>
<td>EuroPLoP conference draft</td>
</tr>
<tr>
<td>July 2012</td>
<td>Post conference revisions: paper split into two</td>
</tr>
<tr>
<td>November 2012</td>
<td>Formatting changes for conference proceedings</td>
</tr>
</tbody>
</table>

References


Lessons from EuroPLoP:
Two patterns for Shepherding (EuroPLoP 2012)

Allan Kelly - allan@allankelly.net

Figure 1 – Shepherding can be tiring

1 Introduction

EuroPLoP 2012 will mark this writer's tenth EuroPLoP. During this time the author has learnt. This paper set out to capture some of the author's observations and learning on shepherding. Two patterns about shepherding are presented: ONE PATTERN and GROW YOUR AUTHOR.

2 Audience

The patterns in this paper are written for pattern shepherds and others attending pattern conference, e.g EuroPLoP or PLoP.

3 The Patterns

The patterns presented here build on, and extend the patterns of Neil Harrison (Harrison, 1999, Harrison, 2006). Figure 2 shows a simple sequence for combining these patterns and Table 1 provides thumbnails of the Harrison patterns.

Copyright 2012, Allan Kelly – allan@allankelly.net, www.allankelly.net
Permission granted to EuroPLoP 2012 / Hillside Europe to use and reproduce.
Figure 2 - Sequence combining Harrison & Kelly patterns
<table>
<thead>
<tr>
<th>Pattern name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THREE ITERATIONS</td>
<td>How to budget your time and effort to make shepherding effective.</td>
</tr>
<tr>
<td>THE SHEPHERD KNOWS THE SHEEP</td>
<td>How to establish a productive relationship between you and the author.</td>
</tr>
<tr>
<td>HALF A LOAF</td>
<td>How to make sure that shepherding continues to move forward.</td>
</tr>
<tr>
<td>BIG PICTURE</td>
<td>How to grasp the gist of the pattern right off the bat.</td>
</tr>
<tr>
<td>AUTHOR AS OWNER</td>
<td>How to keep from writing the pattern for the author.</td>
</tr>
<tr>
<td>MATCHING PROBLEM AND SOLUTION</td>
<td>How to ensure that the pattern really is pattern-ish.</td>
</tr>
<tr>
<td>CONVINCING SOLUTION</td>
<td>How to make the pattern believable.</td>
</tr>
<tr>
<td>FORCES DEFINE PROBLEM</td>
<td>How to understand the problem at a deeper level.</td>
</tr>
<tr>
<td>BALANCED CONTEXT</td>
<td>How to help get the pattern at the right scope.</td>
</tr>
<tr>
<td>WAR STORIES</td>
<td>How to help the pattern flow.</td>
</tr>
<tr>
<td>FORM FOLLOWS FUNCTION</td>
<td>How to put a new form into a pattern.</td>
</tr>
<tr>
<td>SMALL PATTERNS</td>
<td>How to keep patterns easily digestable.</td>
</tr>
</tbody>
</table>

Table 1 - Thumbnails of Harrison patterns
3.1 Grow Your Author

You are a shepherd, you know about patterns. The author is the sheep, they know what they want to say and they own the patterns – see Author As Owner. Your role is to help the author express themselves, to share their story and insights, and help the author become a better communicator in the process.

Shepherds need to avoid imposing their sense of what the paper should look like, or what it would say; to do this they need to know what the author is trying to achieve and where the author feels the need for help.

Authors are most motivated when they are self-directed but it is a mistake for a shepherd to assume the author’s motivations. Shepherds should remember that authors have varying motivations and reasons for writing.

Therefore, structure the shepherding process using the GROW model - Goal, Reality, Options, What - used by business coaches - described in (Whitmore, 2002) - to the pattern and the author. This model has four stages:

- Goal - what does the author want to achieve with this paper?
- Reality - where does the author think the pattern(s) needs most attention?
- Options - what can be done about the paper?
- What - what will the author do?

While true business coach will jettison any specialist knowledge they may have when using the model pattern Shepherds often need to help the author by explaining some elements of pattern form, e.g. forces. This imposes limitations on how far the GROW model can be applied.

A typical shepherding session starts with the shepherd introducing himself - as per Shepherd Knows the Sheep and asking the author to send him the
latest version of the pattern paper - this avoids any confusion about online systems and provides a first exchange.

Before reading the paper the shepherd begins the GROW by asking the author about his or her aims, the goal:

*What is the author’s objective in writing this pattern or pattern language? Is it to better their own personal understanding? - about the pattern content or patterns in general. Is it to obtain a publication? Or part of some other work? Maybe it is to share knowledge with colleagues.*

Next comes reality:

*Where does the author feel the pattern needs the most attention? Where would they like help most of all?*

Usually these questions are asked by e-mail but once in a while the shepherd is treated to a face-to-face experience.

Sometimes the author is unaware of where the patterns need work, for example, when the author is new to patterns. The shepherd might then ask the author to read a few well-written patterns, or tell a story (like the intermission below), and then repeat the question. And sometimes the shepherd might just suggest where work is needed.

With an experienced author the shepherd advances to the options stage, asking the author what they might change. The idea here is for the author to propose changes they think would benefit the paper. Used for coaching the model has the coachee list options then select one or more to undertake.

When used for shepherding patterns this stage is more likely to produce a list of changes most of which will be adopted. As such the “what” (i.e. selection between multiple options) stage tends to be a minor step.

The GROW model is easier to describe than apply. Application requires practice and more practice. Particularly in the later stages it can be difficult to stay true to the model.

While the GROW approach can help all authors it is probably most applicable when working with experienced authors, or at least authors who have written one or two patterns previously. Even with experienced authors the shepherd might want to break from this model to provide an outsiders perspective on the paper.

With an inexperienced author shepherds frequently need to switch soon to teaching mode - typically explaining some aspect of pattern lore. When this happens the shepherd is often the one who offers the options and lets the author decide which to accept and which to pass over.

With this information the shepherd can set about applying ONE PATTERN. With an experienced author the exchange may continue to follow the GROW model. More questions can help draw out of the author what they could do with the pattern.

Ultimately shepherding is a coaching exercise which, using the material of the patterns in hand, seeks to make a better writer and better person. The exercise is a growth opportunity for the shepherd too: giving feedback and coaching an author helps the shepherd grow and improve too. For an
individual these skills are not confined to shepherding, they are core management skills in any domain.

When applied in conjunction with **ONE PATTERN** the two patterns support one another. Because the GROW model encourages the author to internalise the lessons they are better able to apply them to other patterns in the language without the shepherd.

Thus it is not necessary to apply GROW to every pattern in the language explicitly. Rather, GROW is applied to the overall paper (*What is the author trying to achieve? Which pattern(s) does the author want to focus on first?*), then to the first pattern. As shepherding proceeds through the paper there is usually less need to apply the model explicitly to each pattern. The more experienced the author the more this non-directive model holds; with a less experienced author shepherding reverts to more directive approach sooner or later.
Think of a pattern as a detective story. There is an opening context….  

*The house stood on a bleek moor, the guests came from far and wide and few knew one another.*

There is then problem….  

*The Master of the House was found dead in the wine cellar. A murder was on the loose.*

There are forces which make this problem hard, forces are often written in “the but form” which juxtaposes two forces….  

*As Detective Smith continued his enquires it became clear everyone in the room had reason to want the Master dead but everyone had an alibi – although some alibis were better than others.*

*The Master had been shot at close range but there was no trace of a gun.*

The problem and forces build tension, the reader is drawn in and wants to know the answer. If there were no forces complicating the problem then solving it would be easy and perhaps obvious, there would be no tension and no story worth telling.

Then, the solution….  

*It was the butler*  

A ha! The reader is enlightened.

The solution section then expands on how to build the apparatus of the solution. In some patterns this is included in the solution section, in others there is an explicit implementation section….  

*Charles the butler had set up a concealed hand-gun with a trip wire. He knew the Master would go to the wine cellar shortly before the meal was served. He knew that in the dark light of the cellar he would be able to remove the equipment before the police arrived. Still, a forensic examination had shown where the trip wire had cut into a wooden pillar.*

Every solution has consequences, some good, some not so good, indeed, a pattern without any negative consequences usually means the author needs to think a bit harder.

*The gun had been cunningly concealed under the floorboards.*

*Charles was sent to prison for life.*

Ideally consequences should link back to the forces. Each force should be resolved. This might mean that for every force there is a corresponding consequence statement. But life isn’t always that tidy.

*The Masters niece, Mary, inherited the estate and all his assets.*

Of course, some other consequences occur which the forces did not hint at.
The Masters former business partner Peter claimed he had been promised a share in the will and vowed to fight Mary in the courts.

Patterns don’t always end with everything just right, indeed they shouldn’t. Some consequences will be negative; some things will still be problematic.

Consequences are forces of a kind and in a pattern sequence one pattern’s consequences form the basis for the next pattern’s forces. The solution to one problem itself creates a problem, or problems, which following patterns can address. It is not one story but a series of stories.
3.2 One Pattern

You are starting to shepherd a patterns paper. The author has submitted a pattern language of several patterns. **Within a fixed schedule time you need to help the author improve their language but you don’t know how much time, and focus, both the author and yourself can give to the work.**

Applying 3-rounds of shepherding would result in touching each pattern a little bit three times. As a result the patterns at the start of the paper get more attention than those at the end of the paper. Since time is limited this may result in surface level changes to many patterns.

Authors tend to write each pattern in a similar style. As a shepherd you may want to make similar comments about each pattern. The short term aim of shepherding is to improve the given paper, the long term aim is to improve the author’s writing skills.

**Therefore go deep rather than broad, work on one pattern at a time in sequence rather than work on the whole paper. Use the first pattern as an example of shepherding advice. Ignore any introduction or conclusion, treat each pattern as a stand alone entity. Do several short iterations on each pattern before moving onto the next.**

Tell the author you will proceed one pattern at a time - they may have experienced other approaches. Also, at first contact tell them that they should think about how the comments about each pattern can be applied to the other patterns in the language.

Ask the author which order they would like to take the patterns in. They will probably suggest the order they are presented in but not always. When the author is new to pattern-writing working through one pattern, no matter how
short, in detail will create valuable learning that will can be applied to the rest of the collection.

Hopefully, with a little encouragement, authors will apply this learning without the shepherd needing to review the rest of the collection. Indeed, the shepherd may choose to ignore other patterns – save their time - until they become the focus of shepherding.

As authors improve one pattern they will get insights into how the other patterns could be improved - while each pattern has its own specifics the general points often apply to more than pattern. By making these connections themselves authors understand the concepts better, become better authors and use the shepherd’s time more effectively.

Experience shows that as much as half the time available for shepherding may be used on the first pattern alone. If the author is taking the shepherding points to heart the other patterns will also be improving, albeit out of sight. The second, third and subsequent patterns to be shepherded will take a lot less time.

Individual patterns are brought to a "shepherding complete" state far earlier in the process. Shepherd and author see the end result in miniature sooner.

Success with this pattern depends on the willingness of the author to apply the lessons from one pattern to subsequent patterns. When an author is unwilling, or unable, to make this connection then similar comments are made about each pattern. This increases the workload for both shepherd and author. In these circumstances it may be unrealistic to expect the whole paper to be shepherded entirely.

Some conferences conduct half-way reviews of paper before final acceptance. This approach allows reviewers to see the progress, author willingness and quality more clearly.

Time allowing the shepherd will return to review any non-pattern text – introduction, abstract, etc. – and provide comments on the paper and language as a whole.

When time for shepherding is used up the author has the option to reduce the number of patterns in the language for final submission, or submit patterns which have not been explicitly shepherded.

When the author has submitted a pattern language there is usually not enough time to give each pattern THREE ITERATIONS (Harrison, 2006). Using ONE PATTERN each pattern receives its one iteration to itself; so the ultimate number of iterations may well be more than three.

This pattern continues the idea of withholding comments on a pattern as described in HALF A LOAF (Harrison, 2006) however this pattern applies to a pattern languages rather than an individual pattern.

Pattern authors are invited to adopt this approach when writing patterns and pattern languages. This author frequently focuses on one pattern in a language, as the pattern developers and other patterns are identified these are noted with the barest details – perhaps name and an example. With the first
pattern completed the author reviews the candidates and repeats the process with the next pattern.

4 Acknowledgements

Many thanks to Claudius Link for shepherding this paper – one of my more difficult papers to shepherd I may say. Thanks too to the participants in Workshop C at EuroPLoP 2012 for taking time to read and comment on this paper, and in particular Andreas Fiesser, Andreas Rüping and Klaus Marquardt.

The author, Allan Kelly, welcomes further feedback and comments on this paper and the issues discussed. Please contact Allan at allan@allankelly.net, www.allankelly.net or @allankellynet.

Figure 1: A shepherd and his sheep on the way to Hampta Pass in the Indian Himalayas of Himachal Pradesh by Raja Selvaraj under Creative Commons Attribution 2.0 Generic license from http://commons.wikimedia.org/wiki/File:Shepherd_on_the_way_to_Hampta_Pass.jpg

Figure 2: Authors own creation

Figure 3: Image from Mick Lobb under Creative Commons Attribution-Share Alike 2.0 Generic license; source Wiki commons, http://commons.wikimedia.org/wiki/File:%22It%27s_OK_son,_it%27s_just_a_human%22_-_geograph.org.uk_-_1204270.jpg

Figure 4: Court roof of the British Mueseum by Andrew Dunn under Creative Commons Attribution-Share Alike 2.0 Generic from http://commons.wikimedia.org/wiki/File:British_Museum_Great_Court_roof.jpg

5 History

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2011</td>
<td>First draft</td>
</tr>
<tr>
<td>April 2012</td>
<td>Intermediate EuroPLoP review updates</td>
</tr>
<tr>
<td>June 2012</td>
<td>EuroPLoP conference draft</td>
</tr>
<tr>
<td>July 2012</td>
<td>Post conference revisions: paper split into two, one workshops and one of shepherding patterns.</td>
</tr>
<tr>
<td>November 2012</td>
<td>Formatting changes for conference proceedings</td>
</tr>
</tbody>
</table>

References


Presentation Patterns
A Pattern Language for Creative Presentations

Takashi Iba
Aya Matsumoto
Kaori Harasawa
Faculty of Policy Management, Keio University

Abstract
In this paper we present a pattern language for creative presentations, which we named “Presentation Patterns.” The Presentation Patterns consists of 34 patterns describing practical knowledge for problem finding and problem solving in designing presentations. Although we use the word “presentation,” it can be applied to all kinds of activities of representation, including public speaking, performance of music, drama, and dance. In this paper, we show the following seven patterns from the Presentation Patterns: Creative Presentation (0), Main Message (1), Touching Gift (2), Image of Success (3), Dramatic Modulation (7), Unexpected Evolution (8), and Doors of Mystery (9).

1. Introduction
In the present day, in any kind of area, it is getting more important to convey your idea for promoting social changes and realizing innovations. In such a situation, what is required for the presentation is not only attractive but also creative, where it intrigues the audiences to make their own discoveries [1] during or after the presentation. However, it is not easy to design such creative presentations. In this paper, we will show the art of designing such a creative presentation as a pattern language. This pattern language is a pattern language of communication, which we call “pattern language 3.0” [2], and can be considered as a sister volume to the Learning Patterns, which has been presented in past PLoP conferences [4,5,6,7].

2. Overview of the Presentation Patterns
The Presentation Patterns consists of 34 patterns. In the center of the patterns sits pattern Creative Presentation (0). Three of the main patterns: Main Message (1), Touching Gift (2), and Image of Success (3) surround this. Subsequent patterns are grouped into three categories. The first group, patterns No. 4 - No.12, deals with the contents and the expressions of the presentation. Patterns No.12 - No.21 consider how to make your presentation appealing to
the audience. How you should act once you are on stage is discussed in the final set of patterns, No.22 - No.30.

Figure 1: The Whole Structure of the Presentation Patterns

3. Pattern Form of the Presentation Patterns

Each pattern in the Presentation Patterns is written in the same format: pattern number, pattern name, introductory sentence, illustration, quotations, context, problem, forces, solution, actions, and consequences: Pattern Number is added to every pattern for convenience sake, helping readers to find patterns; Pattern Name is the attractive and memorable words that can be used as a building block for thinking and a vocabulary for communication about the way of designing presentations; Introductory Sentences and Illustration are introductory parts that impress the meaning of this pattern lively; Quotations rephrase the essence of this pattern with notable sayings; Context is the condition for applying this pattern. Problem describes a difficulty that often occurred in the context but is not easy to overcome, which is shown as a bold-typed sentence just after the heading “▼ In this context”; Forces are unavoidable laws that make the problem hard to solve; Solution describes the way to solve the problem, which is shown as a bold-typed sentence just after the heading “▼ Therefore”; Actions offer concrete approaches to put the solution into practice; Consequences describes the result of applying this
patterns and also connection with other patterns, which is shown as a bold-typed sentence just after the heading “▼ Consequently.”

In the catalog booklet of the Presentation Patterns, each pattern is printed in a double page spread, as shown in Figure 2. In the first half of pattern, which is printed at the left page in the catalog, the overview of the pattern is described: Pattern Number, Pattern Name, Introductory Sentences, Illustration, and Quotations. In the last half of pattern, which is printed at the right page in the catalog booklet, the detail of the pattern is described: Context, Problem, Forces, Solution, Actions, and Consequences.

![Figure 2: Page Layout in the catalog booklet of the Presentation Patterns](image)

4. Dialogue Workshop with the Presentation Patterns

We held the workshop that provides an opportunity for participants to reflect on their experience of designing presentations in the undergraduate class at Keio University, Japan (Figure 3). In the workshop, the participants recalled their experience of designing presentations using the Presentation Patterns and made a list of patterns they had already experienced. In addition, they made a list of patterns they wanted to master in the near future. Then, the participants sought others who experienced patterns they wanted to master, and listened to their experience.

It is worthwhile to emphasize that the Presentation Patterns encouraged participants to talk about their experiences, although people have few opportunity to talk about their experiences in their daily life. This allows us to add the example to each pattern by driving personal experience. Presentation Patterns dare to be described in short sentence and abstract expression in order to dwell the simplicity within each pattern because a detailed description can restrict the range of target which recall that I had experienced. It enables participants to adapt own personal
experience to each pattern and to drive the dialogue lively each other. The number of participants who have experience of each pattern is shown in Figure 4.

Figure 3: Experience mining and dialogues workshop with the Presentation Patterns

Figure 4: The number of participants who have experience of each pattern
5. Patterns presented in this paper

We present, in this paper, the following seven patterns from the Presentation Patterns: *Creative Presentation* (0), *Main Message* (1), *Touching Gift* (2), *Image of Success* (3), *Dramatic Modulation* (7), *Unexpected Evolution* (8) and *Doors of Mystery* (9). Although the rest of the Presentation Patterns will be submitted to the future PLoP conferences, the summary of all patterns are shown in Appendix A in this paper for reference. In addition, the making process of the Presentation Patterns is summarized in Appendix B.
Creative Presentation

Not just an explanation, but a creation.

“We cannot teach people anything; we can only help them discover it within themselves.” --- Galileo Galilei

“Discovery consists of seeing what everybody has seen, and thinking what nobody has thought.” --- Albert Szent-Gyorgyi
You have an idea that you want to spread and share.

▼ In this context

Plain old explanations of your idea won’t motivate the audience to take further actions. Information is first understood when it relates to preexisting knowledge.
- Understanding and Believing are two different things.
- It is doubtful for a person to take action without believing in it.

▼ Therefore

Treat your presentation not as just a chance to explain your idea, but as a chance for creation. Work towards the audience to trigger new findings in them,
To design a creative presentation, carefully compose the Main Message (1) that drives the audience, and suppose the needs of the audience in it to make the presentation a Touching Gift (2) for them. Throughout the process, all decisions have to be made according to your Image of Success (3).

▼ Consequently

By performing a creative presentation that inspires and motivates the audience, innovation is possible. In addition, through the process of preparing, performing, and receiving feedbacks, the presenter will also experience new findings. Thus, a creative presentation must be inspiring for both the audience and the presenter.
Main Message

What do I want to tell the audience the most?
You are designing a presentation, and thinking of what to tell to the audience.

▼ In this context

You have too many things you want to say.
- It takes a lot of time and effort to organize a presentation.
- If you can’t organize the information, neither will the audience.
- What you want to say to the audience doesn’t necessarily correspond with what you want for them to understand.

▼ Therefore

Extract the one most important message, and create your presentation around that idea.
Consider what the audience should gain most from your presentation. Use this as a measure to prioritize the multiple messages you may have. Once you have chosen your main message, start to build your presentation so it best conveys that message.

▼ Consequently

The audience can easily understand the most important message in the presentation. In addition, having a message in the presentation would allow the presenter to do the Storytelling (4) easier.
No.2

Touching Gift

A presentation is a present for the audience.

“I think that every novelist has a single ideal reader; that at various points during the composition of a story, the writers is thinking, ‘I wonder what he/she will think when he/she reads this part?’”

---Stephen King
You have decided on your *Main Message (1)*, and you are now thinking how you can convey it to the audience.

▼ In this contexts

**The audience may understand your presentation, but wouldn’t be moved and share the feelings of your message.**
- What you want to tell is not always what the audience wants to know.
- You don’t have the audience at the preparation stage of the presentation.

▼ Therefore

**Focus on who the audience is and think how to specifically make them impressed with your message.**
If you don’t care much about the audience, your presentation would end in self-satisfaction. You need to research about the audience in many respects before the presentation—How familiar or interested are they about the subject? Are there any common characteristics among them? What kind of stories do they tend to favor?

▼ Consequently

You can gain the sympathy of the audience, and they will become interested in the contents and the presenter. Subsequently, it creates a warm atmosphere for you to speak in front of the audience to get your *Main Message (1)* across.
Image of Success

Set a clear goal that you want the audience to achieve.

“Vision is the art of seeing the invisible.”---Jonathan Swift

“The ones who are crazy enough to think that they can change the world, are the ones who do.”---Steve Jobs

“If one advances confidently in the direction of his dreams, and endeavors to live the life which he has imagined, he will meet with success unexpected in common hours.”---Henry David Thoreau
You made a plan on how to give your *Touching Gift* (2) based on the *Main Message* (1), and now you are thinking of what you want to accomplish through the presentation.

▼ In this context

**With unclear goals, you can’t decide on the details of your presentation.**
- The same information can be presented in multiple ways.
- If the purpose isn’t clear, your presentation can easily head toward an unnecessary direction.

▼ Therefore

**Have an image of success for your presentation.**
Think at what point your presentation would become a success, and prepare for that goal. Imagine both yourself presenting and the audience’s reactions to it. Never forget about this image throughout the preparation and performance of the presentation, giving yourself *Reminders of Success* (23).

▼ Consequently

By having a clear *Image of Success*, you can work towards it without losing track of your goal.
Dramatic Modulation

Attract the audience with an accentual rhythm.

“Sweet music will also become unpleasant, if tempo collapses or the balance of sound is out of order. A life of people is also the same.”---William Shakespeare
You want to make your Storytelling (4) that conveys the Main Message (1) more appealing.

▼ In this context

The audience may become confused what the Main Message (1) is in the presentation.
- It is difficult to distinguish the important facts from the less important ones.
- A monotonous presentation is boring.

▼ Therefore

Make a modulation in your presentation by creating a difference in your tone when telling the Main Message (1) against the other parts.
Effectively change your inflection and the speed of your voice to interest the audience in the presentation. Visual modulation is also effective by using different fonts or animations in the presentation.

▼ Consequently

The modulation helps the listener to follow the Storytelling (4) of the presentation; subsequently enabling them to better understand the Main Message (1). Furthermore, you can keep attracting the attention of the audience, making it easier for you to speak.
Unexpected Evolution

Beyond the audience’s anticipations.

“When you lose interest in anything, you also lose the memory for it.” ---Johann Wolfgang von Goethe
You want to make an attractive Storytelling (4) that conveys the Main Message (1).

▼ In this context

Your Storytelling (4) is boring and wouldn’t give any impression of the Main Message (1) to the audience.
- Predictable stories are boring to hear.
- The audience won’t remember uninteresting things.

▼ Therefore

Intentionally swerve the Storytelling (4) from the audience’s expectations to add an interesting and unpredictable turn.
First, consider where to add the unpredictable turn in the presentation. The introductory line where you attract the attention of the audience, or before the main message where you start to expand the contents, are good examples. Imagine what the audience anticipates, and purposely present differently from that anticipation.

▼ Consequently

The audience can discover completely new aspects or different points of view from the surprise. At the same time, you can expand your own point of view in the attempt to go beyond the audience’s anticipations.
Doors of Mystery

Solving mysteries one after another.

“A rainbow which lasts for a quarter of an hour is looked at no longer.”---Johann Wolfgang von Goethe
You want to make an attractive *Storytelling (4)* that conveys the *Main Message (1)*.

▼ In this context

**You can’t keep the audience’s attention.**
- Interests towards a single subject don’t last long
- The audience will stop paying attention unless they feel impressed or interested.

▼ Therefore

**Design the structure of the presentation so that it appeals to the curiosity of the audience and continuously drives their interest.**
Consider what the audience already knows and what they don’t know. From this assumption, consider what kind of things the audience would show interest in. Then construct the presentation so that the audience will continuously be surprised and learn something new.

▼ Consequently

The audience will become interactive to the presentation since they are curious and intrigued. In addition, you can plan your *Storytelling (4)* in elaboration with the *Doors of Mystery (9)*.
Acknowledgement
We would like to thank to the member of Presentation Patterns Project: Mami Sakamoto, Kana Matsumura, Rinko Arao, Yosuke Yano, Masahiro Hamada, Daiki Muramatsu, Eri Shimomukai, Emiri Nakano, Satoshi Nishina, Ai Nomura, Saya Yasuura, and Yuka Yamazaki. We want to thank to Ko Matsuzuka and Ken Fujiyoshi and Taichi Isaku for helping brush-up of this paper, Prof. Peter Sommerlad for shepherding our patterns and paper, and the participants of writer’s workshop for giving advices on our paper.

References
Appendix A: Brief Summaries of the Presentation Patterns

**The Center Pattern**

**No.0 Creative Presentation**

*Context* You have an idea that you want to spread and share.

*Problem* Plain old explanations of your idea won’t motivate the audience to take further actions.

*Solution* Treat your presentation not as just a chance to explain your idea, but as a chance for creation. Work towards the audience to trigger new findings in them.

**The Core Patterns**

**No.1 Main Message**

*Context* You are designing a presentation, and thinking of what to tell to the audience.

*Problem* You have too many things you want to say.

*Solution* Extract the one most important message, and create your presentation around that idea.

**No.2 Touching Gift**

*Context* You have decided on your *Main Message* (1), and you are now thinking how you can convey it to the audience.

*Problem* The audiences may understand what you said, but wouldn’t be moved and share the feelings of your message.

*Solution* Focus on who the audience is and think how to specifically make them impressed with your message.

**No.3 Image of Success**

*Context* You made a plan on how to give your *Touching Gift* (2) based on the *Main Message* (1), and now you are thinking of what you want to accomplish through the presentation.

*Problem* With unclear goals, you can’t decide on the details of your presentation.

*Solution* Have an image of success for your presentation.
Patterns for Expressions and Representations

No.4 Storytelling

[Context] You have chosen your Main Message (1) and are thinking of in what order you should tell things.

[Problem] A presentation with just the information laid out is boring and unmemorable.

[Solution] Create an attractive story with the information you have.

No.5 Exploration of Words

[Context] Storytelling (4) that tells the Main Message (1) is made and you are writing out your script for the presentation.

[Problem] To avoid using dull or overused expressions you are tempted to use unfamiliar words, sometimes hard for the audience to understand.

[Solution] Search for words and expressions that both the presenter and the audience feel attractive.

No.6 Visual Power

[Context] Storytelling (4) that tells the Main Message (1) is made and you are writing out your script for the presentation.

[Problem] You feel your presentation is too wordy and hard to understand.

[Solution] Take advantage of a visual figure that expresses the information.

No.7 Dramatic Modulation

[Context] You want to make your Storytelling (4) that conveys the Main Message (1).

[Problem] The audience may become confused what the Main Message (1) is in the presentation.

[Solution] Make a modulation in your presentation by creating a difference in your tone when telling the Main Message (1) against the other parts.
No.8 Unexpected Evolution

[Context] You want to make your Storytelling (4) that conveys the Main Message (1).

[Problem] The audience may become confused what the Main Message (1) is in the presentation.

[Solution] Make a modulation in your presentation by creating a difference in your tone when telling the Main Message (1) against the other parts.

No.9 Doors of Mystery

[Context] You want to make your Storytelling (4) that conveys the Main Message (1).

[Problem] You can’t keep the audiences’ attention.

[Solution] Design the structure of the presentation that continuously drives the audiences’ interest.

No.10 Beautiful Clarity

[Context] You are brushing up expressions in your presentation to better convey the Main Message (1).

[Problem] Your presentation is either attractive but difficult to understand, or easily understood but lacks attractiveness.

[Solution] Brush up your presentations so it has a good balance between clarity and allure.

No.11 Perfect Portion

[Context] You are brushing up expressions in your presentation to better convey the Main Message (1).

[Problem] Your presentation has too much information, or too little information.

[Solution] Make sure your presentation has just the right amount of information at various levels of the presentation.
No.12 Cherry on Top

CONTEXT] You are brushing up expressions in your presentation to better convey the Main Message (1).

[Problem] Your presentation doesn’t have any problems, but you still feel it lacks something.

[Solution] Make extra improvements in not the contents of the presentation, but in its expressions.

Patterns for Preparation and Performance

No.13 Mind Bridge

[Context] You are making your presentation into a Touching Gift (2) that gives the audience a memorable experience.

[Problem] You notice yourself repeating the same words trying to explain an idea.

[Solution] Use metaphors and specific examples to help explain to the audience.

No.14 Reality Sharing

[Context] You are making your presentation into a Touching Gift (2) that gives the audience a memorable experience.

[Problem] There is a limit to what you can get across to the audience with just words and pictures.

[Solution] Make an opportunity within the presentation for the audience to actually experience first-hand the sensation you want to share.

No.15 Participation Driver

[Context] You are making your presentation into a Touching Gift (2) that gives the audience a memorable experience.

[Problem] The audience tends to get passive just listening to the presenter.

[Solution] Make an opportunity for the audience to participate in your presentation.
No.16 Quality in Details

[Context] You are working to make your Touching Gift (2) even better.

[Problem] You tend to stop giving your best effort when your presentation starts to take shape.

[Solution] Taking into account the presentation as a whole, work on fixing the small details.

No.17 Expression Coordinator

[Context] You are working to make your Touching Gift (2) even better.

[Problem] You feel you are always giving the same sort of presentations due to a lack of techniques.

[Solution] Watch other people present, and take in any techniques that you think would help your presentation skills.

No.18 Discomfort Removing

[Context] You are working to make your Touching Gift (2) even better.

[Problem] Your unconscious habits are bothering the audience.

[Solution] Make an opportunity for yourself to notice your habits.

No.19 Triggering Blanks

[Context] You want your audience to take active responses to your Touching Gift (2).

[Problem] The audience becomes too busy trying to understand the material.

[Solution] Purposely leave out some information so the listeners can use their imagination to fill in the missing parts.

No.20 Activation Switch

[Context] You want your audience to take active responses to your Touching Gift (2).

[Problem] Your audience understood the material of your presentation, but they think the subject is unrelated to their lives.

[Solution] Include in your presentation a chance for the listeners to think about their own opinions, and then show them the path for the next step.
No.21 Take-Home Gift

[Context] You want your audience to take active responses to your Touching Gift (2).

[Problem] Your ideas struggle to spread even after the presentation.

[Solution] Hand out a gift for the audience to take home that would remind them of the details of the presentation.

The Patterns for Creative Performance

No.22 Stage Building

[Context] You are imagining your Image of Success (3) while going through final adjustments for your presentation.

[Problem] Your presentation may be ruined due to technical difficulties.

[Solution] Treat facilities and equipment as part of your presentation, and check thoroughly for any problems and adjust them beforehand.

No.23 Reminders of Success

[Context] You are imagining your Image of Success (3) while going through final adjustments for your presentation.

[Problem] You notice you are heading towards a different direction than your original goal.

[Solution] Frequently remind yourself with your Image of Success (3), whether verbally or visually, to refresh your mind as needed.

No.24 Construction of Confidence

[Context] You are imagining your Image of Success (3) while going through final adjustments for your presentation.

[Problem] Despite the effort you’ve put in, you don’t have much confidence in your speech.

[Solution] Remind yourself of all the time and effort you have put into this presentation, stack them up and build your self-confidence.

No.25 Presentership

[Context] You are minutes away from your stage and your Image of Success (3) is about to come true.

[Problem] You become too busy trying to speak and forget that an audience is watching you.

[Solution] Act as if you are part of the presentation.
No.26 Best Effort

[Context] You are minutes away from your stage and your Image of Success (3) is about to come true.

[Problem] You tend to make excuses for the parts you don’t have as much confidence in, only to make the presentation even worse.

[Solution] Make no excuses and give the best presentation you have at the point.

No.27 Personally for You

[Context] You are minutes away from your stage and your Image of Success (3) is about to come true.

[Problem] Your attitude towards the audience becomes careless from the rush to get your speech across.

[Solution] Intentionally make eye contact with the audience in effort to give the speech to each and every one of them.

No.28 Invitation to the World

[Context] You want to give a presentation with a high level Image of Success (3).

[Problem] You are working on brushing up your expressions, but you feel your presentation isn’t improving as much.

[Solution] Have an alluring “world” of your presentation, let the audience have a glimpse of that “world”, and then guide them into it.

No.29 Improvised Presentation

[Context] You want to give a presentation with a high level Image of Success (3).

[Problem] You find yourself having trouble trying to progress your speech according to the reactions of the audience.

[Solution] Have a repertoire of speeches that you can put together and improvise based on the reactions of the audience.

No.30 Reflecting Forwards

[Context] You want to give a presentation with a high level Image of Success (3).

[Problem] You feel satisfied when you finish speaking, and take no actions afterwards.

[Solution] Evaluate your performance on the presentation that you gave through self-reflection and the reaction of others.
The Extreme Patterns

No.31 Pursuit of Uniqueness

[Context] You want to give a truly Creative Presentation (0).

[Problem] Your presentations are somewhat similar to other people’s presentations.

[Solution] Be aware of the differences you have from other presenters, and pursue your originality.

No.32 Aesthetics of Presenting

[Context] You want to give a truly Creative Presentation (0).

[Problem] Your presentations all seem the same.

[Solution] Continue your pursuit for beauty in your presentations, and build your own values on aesthetic.

No.33 Be Authentic!

[Context] You want to give a truly Creative Presentation (0).

[Problem] You feel your presentations lack some depth and quality.

[Solution] Be authentic! Live your way of Life.
Appendix B: The Making Process of the Presentation Patterns

The Presentation Patterns were made by 15 project members in 2011. In what follows, the making process of the Presentation Patterns is shown according the procedure explained above: Patterns Mining, Patterns Writing, Patterns Improving, Language Organizing, and Catalogue Editing.

B.1. Patterns Mining
What we did in the first place for mining patterns is to explore our own experience. We started brainstorming about the way of designing presentations. In the brainstorming, we conceive “presentation” in broad sense, including all kinds of activities of representation, including public speaking, performance of music, drama, and dance. After collecting the ideas, the next step is to organize them by compiling similar ideas, dividing into groups of ideas, giving names to the groups, and connecting each other according to their meaning. Note that these groups must be organized by emergence through iterating the operation to bring an idea near another and to keep one away from others, not by guidance of existing category. This process is well known as KJ technique in Japan. Thus, the pieces of idea about presentation design were organized over the discussion, and consequently we obtain about 40 candidates of patterns.

B.2. Patterns Writing
After that, we describe the core parts of patterns, namely pattern name, problem and solution. In this phase, we decided the concrete form of our pattern language. Every member took charge of some patterns that they have experienced. This policy of assignment is important, simply because writing patterns require the tacit knowledge of what is written in the pattern.
B.3. Patterns Improving

After finishing writing the core parts of patterns, we hold writers’ workshops and rewrite the patterns. Drawing illustrations of the patterns were preceded in parallel, and here we realized that illustrations help to shape our understanding of the patterns. In this phase, writer’s workshops are held several times for each pattern in order to improve the contents and expression of patterns.

B.4. Language Organizing

The fourth phase was to organize the patterns into a language as a whole. In completed pattern language, each pattern does not exist alone; every pattern has relations to other patterns. Thinking the whole structure and patterns’ relations led to the reflection and reconsideration for the meaning of each pattern. When organizing the patterns as a language, we make a final decision that Presentation Patterns consists of 34 patterns. We decided the layer structure of the Presentation Patterns according to the abstract level.

B.5. Catalog Editing
The fifth phase was to design the catalogue in which the pattern language is contained. In this phase, we thought not only the pattern language but also other object contained in the catalogue; for example, cover, imprint, preface, table of contents, explanation how to read, overview of the pattern language, navigation to each pattern, and so on.
A Pattern Language for Costumes in Films

David Schumm¹, Johanna Barzen², Frank Leymann¹, and Lutz Ellrich²

¹ Institute of Architecture of Application Systems, University of Stuttgart,
Universitätstr. 38, 70569 Stuttgart, Germany.
{schumm, leymann}@iaas.uni-stuttgart.de

² Institute of Media Culture and Theater, University of Cologne,
Meister-Ekkehard-Str. 11, 50937 Köln, Germany.
{jbarzen, lutz.ellrich}@uni-koeln.de

Abstract. A closer look behind the scenes of film making and media science reveals that the costumes used in film productions are products of a complex construction process. The costume designer has to put a lot of creative and investigative effort into the creation of costumes to provide the right clothes for a particular role, which means the costume reflects the place and time of play as well as it shows understanding of the characteristics of the role, actor and screenplay overall. Consequently, the creation of a costume is a difficult problem that is more or less occurring often, whereas the frequency of problem occurrence strongly depends on the genre and specifics of the film. For the genre of Western films, for example, the costume of a Wild West Sheriff is qualified for identification and description as a pattern because it appears frequently. In this paper, we propose a pattern language for composing costume patterns through a rich set of composition operators, more fine-grained costume patterns, and costume primitives. The pattern language aims at supporting media science, costume design, and costume management through providing a basis for the development of advanced information systems assisting the management of costumes considering their inherent structure and relations between their constituent parts. Our pattern approach is exemplified through deep-dive modeling of two costume patterns.

Keywords: Costume Language, Vestimentary Communication, Empirical Film Analysis, Pattern Language.

1 Introduction

Clothes are a visual communication medium that is used to transport particular information. Whereas the use of this so-called ‘vestimentary communication’ in the real world remains most of the time unconscious [8], the costumes in a film are carefully elaborated to give information about a certain role (e.g. a King), character (e.g. King Henry VIII), and associates stereotypes (e.g. charismatic like: rich person who cares little about the poor) as well as a particular place (e.g. England) and time (e.g. ~1530). The recipient of the information that is transported by the costume interprets associated messages trained through prior media socialization. In other words, when growing up surrounded by media such as books, films, television shows, and advertisements one knows that the role of a king typically wears a crown – without ever having seen a real king. Each costume is a solution to a design problem, which is defined through place and time of a film, the screenplay, a character, an actor it is tailored for, as well as expectations of the film director and intended film audience alike. As a consequence, the process of costume design requires significant effort in a creative, but also in an investigative manner.

Regarding the description of a costume solution and its elements there is however currently no widespread or standardized format or, speaking in terms of computer science, no formal model.
Therefore, designed costumes are often described textually with additional sketches or photos. This circumstance makes the investigation for already existing costume solutions quite difficult as multiple different formats are used, each providing a different set of information artifacts and a different level of detail. Furthermore, good solutions, which have been created once, are hard to communicate due to the lack of a knowledge management methodology for this problem domain. Therefore, we propose a pattern language for costumes to (i) overcome the shortcomings in the current state of costume descriptions and (ii) enable advanced usage scenarios and applications through information system technologies.

The main target audience of this work is the group of costume designers in order to enable structured reuse of knowledge. Further, media scientists are targeted as they can increase the quality of research through a catalog of precise and formal costume descriptions. Also costume rentals and shops are a target audience, because a structured costume format would allow them to offer advanced accessibility to their offerings. This brings us to the last group – information system developers are also a target group because IT solutions like catalog systems, modeling tools, and digital marketplaces for structured data exchange are needed to support the approach from a technical perspective. In our case the audience of the approach is quite heterogeneous in its knowledge in the field of costume design, media science, and information technology. Therefore, we explain special terms from any of these fields when used, see also ‘terminology tailored to the audience’ by Harrison [3].

Our understanding of the terms around ‘pattern’ is explained in the following:

1. We understand a pattern as a good solution to a frequently occurring problem.
2. We understand a costume primitive as a simple piece of clothing, accessory, or costume-related props.
3. We understand a costume pattern as a good solution to a frequently occurring costume design problem. A costume pattern may be composed of costume primitives and other costume patterns. The composition description is the main solution aspect of a costume pattern.
4. We understand a costume primitive format as the formal model that each costume primitive conforms to.
5. We understand a costume pattern format as the formal model that each costume pattern conforms to. A costume primitive has only a subset of the attributes of a costume pattern. If required, a costume primitive can be converted to a costume pattern.
6. We understand composition operators as a grammar to compose costume primitives and costume patterns to more complex structures. Composition operators are crucial for the technical realization.
7. A costume pattern language is combining three sets: A set of costume primitives conforming to a particular costume primitive format, a set of costume patterns conforming to a particular costume pattern format, and a set composition operators.

Our understanding of a costume pattern is based on the needs of vestimentary communication. One could argue that costume patterns show some similarities with the term ‘idiom’ in the terminology of pattern languages of programs. However, costume patterns capture complex design problems and respective design solutions. Furthermore, applying and concretizing a costume pattern requires creative effort and expertise of a costume designer. Therefore, we argue that the term ‘pattern’ is more applicable.
Basically, costume patterns aim at providing a fast role understanding through efficient vestimentary communication building on established conventions of clothing in films, which are trained through media socialization. Whereas the costumes of the main cast need a high degree of individuality to be authentic, the stock characters have to be immediately understood when they appear. As we sketched in Figure 1, costume patterns focus on a high degree of stereotype in clothing while applying a low degree of individuality to ease perception of the vestimentary code. The more important a role in a film is, the more authenticity is required, which needs to be achieved through creative costume design. At the starting point of creative costume design there is always investigation needed on how other films have solved the design problem to dress a certain role. We argue that this investigation can be done a lot more efficient by applying and concretizing costume patterns. When working on the costumes for the stock characters efficiency is likely to increase even more. As highlighted in Figure 1 the pattern focus is located close to the stock characters, which are related to a high degree of stereotype clothing. Hardly any costume designer would dress a character without applying a few individual elements. Therefore the pattern focus does not cover an extreme degree of stereotype in clothing. The approach we present in this paper has many potential application areas, as we describe in more detail in Section 2. An application of the approach in the field of fashion suggests itself, but cannot be given at this point.

Figure 1: The costume pattern approach aims at fast role understanding through usage of established conventions of clothing in films.

Figure 2 depicts an overview of the interrelations between concrete costumes and costume patterns. The interrelations are originating from a set of concrete costumes which have been used in films which had significant impact on the establishment of clothing conventions for a particular genre. In other words, these are costumes which have been used for famous roles, like ‘Sheriff John Chance’ in the film ‘Rio Bravo’ (1959). In this paper we show how a set of concrete costumes can be abstracted into a costume pattern by identifying commonalities, similarities, and multiple occurrences of particular costume primitives. Through abstraction, the concrete costumes become ‘known uses’ of the costume pattern. Going one step further, the identified costume pattern can be applied as conceptual basis for the design of a new Sheriff costume. The creative aspect of applying a pattern is to create a costume design which reflects requirements derived from the characteristics of the role, actor, screenplay and desired authenticity. Many decisions need to be made when applying a costume pattern. Concretization refers to these decisions, which includes for example deciding on concrete fabrics of costume primitives.
Concrete Costumes
(Sheriffs in Western Films)

Costume Pattern
(Wild West Sheriff)

Pattern-based
Costume Design
(New Sheriff)

Figure 2: A costume pattern originates from abstraction of concrete costumes; vice versa, the concrete costumes become known uses of the costume pattern.

The paper’s further structure is the following. In Section 2 we illustrate different scenarios which show the benefits of the approach in different application areas. In Section 3 we present the literature survey we conducted and the research design that we applied to design the overall costume pattern language. In Section 4 we discuss the format of costume primitives. Afterwards, in Section 5 we describe the costume pattern format that we use as formal model to describe costume patterns. Then, in Section 6 we elaborate on the composition operators we propose as grammar for describing the composition of complex costume solutions out of costume primitives and more simple costume patterns. In Section 7, three costume primitives are presented. In Section 8 we perform the deep-dive modeling of one complex costume pattern in detail, the Wild West Sheriff. As a contrast we present another pattern in Section 9, the Wild West Outlaw. Differences and commonalities of patterns are discussed in Section 10. The paper concludes with a summary and outlook in Section 11.

2 Potential Application Areas

In this section we discuss different scenarios and list multiple stakeholders that benefit from the approach of capturing knowledge about complex costumes in a structured way. The major aim is to provide new insight into unrevealed secrets of vestimentary communication which can be utilized for practice of costume design. The vision we present is beyond of what is currently possible, as we describe in more detail in the literature survey in Section 3. The overall vision and the interrelation of scenarios are depicted in Figure 3.
Figure 3: Different scenarios to illustrate the benefits of the approach, ranging from costume design (Scenario 1), media science (Scenario 2), costume rental and shop (Scenario 3), and film production (Scenario 4).

**Scenario 1: Reusing and sharing knowledge of costume design**

The costume designer is not only concerned with the design of costumes, but also with their organization. This means, when a costume design is ready, the pieces of clothing needed to realize the design have to be found on the market for being lent, bought, or otherwise they need to be tailored or manufactured on demand. The key question is how to make complex solutions to costume design problems reusable. The core solution aspect that a costume constitutes is how it is composed out of more simple parts and how they relate to each other. A modeling tool could be a useful support to describe such compositions in an easy-to-use and consistent manner. To make this knowledge easier accessible and more structured it needs to be refined with information beyond its structure, for example indicating the usage context and a description containing important keywords. The costume pattern language supports this scenario through an extensible format to capture that knowledge. Furthermore, distilling costume design knowledge into a pattern format allows easily sharing it with others, and also easily consuming costume design knowledge shared by others.

**Scenario 2: Analysis of vestimentary communication in media science**

It is hard to provide evidence for theories in media science related to costumes in films. For example, how could one validate the assumption that there are ‘stereotyped costumes’ that allow a particular audience to quickly understand the role of a supporting actor just by seeing him for a short time? With a structured format for costumes we would like to provide a new
way for validating such assumptions. By focusing only on a particular film genre and time and through the use of abstraction we aim at identifying patterns of costumes and structurally describing them. For this, a costume pattern language is required, of which we present an initial sketch in this paper. However, this language is just the start for research in vestimentary communication that is based on information technology for the humanities. We envision a large catalog of costumes and costume patterns that can be analyzed by media scientists to create new knowledge and insight by applying techniques like data mining and statistical reporting, for example to analyze the frequency of a particular combination of pieces of clothing in a particular genre. A question that can possibly be answered through information technology is, for instance, how established conventions of clothing in films changed over time. The answer a system might provide would be rather structural and mathematical, which is a new way of media science research in this field.

**Scenario 3:** Costume rental and shop with advanced accessibility

From the perspective of a costume rental or costume shop the aim is to ease the access to the pool of available costumes and constituent elements in order to increase profit. Current state is to provide potential customers with the access to a database where descriptions of available costumes are virtually stored. Search functionality, e.g. based on keywords or text contained in the costume description, is state of the art in such systems. However, when modeling costumes on a level of detail that is more fine-granular than treating costumes as monolithic entities, such a system can provide much more functionality. For ‘virtualizing’ a real costume we propose modeling all of its constituent parts as ‘costume primitives’ and to link them to structured costume descriptions. Then a user of the database, e.g. a costume designer, could submit a search query formulated as complex costume. This query could be broken down into a set of pieces of clothing which are contained in the costume. The database can then create an according offer based on the parts available on stock. The benefits of such a system become even clearer when applying the dimensions of film productions. For instance, if Wild West clothing for thirty extras (‘Statisten’ in German) representing a gang of outlaws is needed, the required clothing could be expressed as a costume pattern used to query the database which then compiles thirty concrete costumes. This way, the costume pattern language would increase accessibility of costumes and clothing.

**Scenario 4:** Efficient work with costumes in a film production

The dressers in a film production are working for the wardrobe department. They are responsible for the availability of the right costumes at the right time and that the actors get dressed properly. They are also responsible for continuity regarding costumes, which basically means that, for example, from one shooting day to another a shirt should look exactly the same and should be worn the same way. For dressing an actor properly, the ordering of applying the different parts of the costume and how they are related to each other are relevant, representing an operational point of view. A well-defined format to describe a costume and additional information that instructs a dresser how to use it properly are therefore quite handy and make the work with costumes in a film production more efficient.

**IT Perspective:** Development of information systems for costume management

The information systems perspective is cross-cutting all four scenarios described before: a tool is needed for graphically modeling costumes and costume patterns, a catalog system is needed for storage, versioning, and search, and also analysis methods to operate on such a catalog are needed. Figure 4 depicts the information system development perspective which reveals
different challenges that need to be addressed. Development starts with managing the requirements from the different stakeholders of such information system. These requirements describe the functions needed and the features desired to obtain a system that provides significant advances compared to the systems currently in use. The requirements of the stakeholders are also represented in form of the costume pattern language. Having a clear view on these requirements the first step can be made, which is to formalize the costume pattern language to provide the basis for its support in a costume catalog and modeling tool.

**Figure 4:** The IT Perspective, which considers information systems development, is cross-cutting the other scenarios.

Regarding the costume catalog, a data format for structured data exchange needs to be designed for building up a digital marketplace where costumes can be lent, bought, or manufactured on demand (catalog extensibility: export modules). A major challenge in the architecture and development of the costume catalog and modeling tool is to support creative work, which is very much ad-hoc and flexible, thus the requirements regarding functions tend to change over time. Different types of search are required to support the different stakeholders’ needs, for instance a costume designer may specify a search query as a composition of costume primitives (catalog extensibility: search modules). There is also a need for different wizards that assist a user in instantiating a costume design, i.e. choosing real pieces of clothing (extensibility: instantiation wizards). Also different analysis modules are needed for specific types of film analysis. For example, the usage of particular data mining techniques like segmentation or profiling could be used to analyze the ‘fingerprint’ of a costume designer. A costume shop owner might be more interested in an analysis module that provides recommendations, which parts ‘fit’ to other pieces of clothing (catalog extensibility: analysis modules). Such information can also be used in web shops for clothes providing product recommendations for customers based on products in the shopping basket. Regarding the modeling tool, which may also provide catalog browsing functionality for a good user experience, there are similar requirements for extensibility. The instantiation of designs also needs to be supported in the modeling tool,
demanding for integration of the modeling tool with the costume catalog. Furthermore, the results of the analysis modules operating on a catalog need to be made visible to the user, therefore extensions are required to support visual analysis.

3 Literature Survey and Research Design

Film making is a challenging domain in many different aspects. Many organizational units are involved in film making, forming highly complex business processes. Many different stakeholders are participating in the different phases of film making, which are namely development, pre-production, production, and post-production [27]. As described in [26], established organizational aspects of film making can be abstracted to patterns of roles and patterns of products, which these roles use and create. Such knowledge is very useful to get an overview of this very broad and complex problem domain. However, the different organizational units and their business processes require more investigation. The focus in our work within the broad scope of film making and film analysis is set on the wardrobe department, which is responsible for the creation and management of costumes. Here, several problem domains exist: The domain of costume design and the research on vestimentary communication in media science that is associated with it, the domain of costume management for selling and lending costumes, the domain of tailoring costumes and pieces of clothing, and the domain of practical usage of costumes at a film set. Given these particular problem domains and narrowed scope, we conducted a literature survey to get a thorough understanding of current practice and science of costume design, available tool offerings, and related pattern approaches. The main results of this survey are described in the following. In order to increase transparency of our research and for traceability of the results we achieved, we subsequently describe our research design.

3.1 Literature Survey

Regarding the domain of costume design and related research, the main problem is to design a costume that reflects the place and time of play as well as it has to show understanding of the characteristics of the role, actor and screenplay overall. Complex aspects of vestimentary communication need to be considered. For example, physical requirements imposed on the costume like protection from high or low temperature need to be arranged with social expectations of the film audience and stereotypes trained through media socialization. Although costumes are a very important aspect in a film and although fashion and clothing have been more intensively discussed since the 1980s [2] [8] [18], the discussion of costumes in media science is still limited. Research was made in the course of the Mise-en-scène analysis [19], focusing on the communicative aspect [20], investigating the sign character of a costume [8], and discussing the relation between costumes and femaleness [21]. The term of a ‘costume language’ is used in many of these works, however metaphorically. Without neglecting the communicative aspect of clothing, the current state in research is that a common costume language is hard to develop due to the complex relationships of cultural, social, and situational aspects [8] [22] [23]. Barthes [2] developed an inventory system to describe the most significant aspects of clothes and to decompose them into their constituent parts. Burger [8] made this system accessible to film costumes. Though, to the best of our knowledge, there is still no formal approach in terms of a well-defined syntax and semantics to make costumes ready for processing in information systems.

The lack of an established format is also reflected in the next domain, costume management. Current tools are providing support for documentation, administration and inventory for costumes as a monolithic entity. Although the pieces of clothing a costume is composed of are known, this level of detail is not yet reached in practice. Prominent examples for such tools are
go_disco [15] and Pro Fundus [16]. In current practice, either the format that a tool is using is applied or an individually defined format is used. Due to the lack of an established format, there is no integrated electronic marketplace where structured costumes can be ordered for buying or lending.

Regarding the domain of tailoring, the main problems are how to cut particular fabrics and how to sew them to obtain a piece of clothing. In fact, this domain makes use of so-called ‘sewing patterns’. The approach of sewing patterns evolved in the second half of the 19th century giving also non-industrials the opportunity to represent and reuse a good composition of pieces of fabrics which can be sewed up to a piece of fashionable clothing [11]. Basically, sewing patterns are cutting and sewing templates which describe the shapes of the different elements of a piece of clothing [12]. Nowadays, sewing patterns still play an important role in the domain of tailoring, both in industrial practice and in the daily business of tailors working in film productions and theatres. There is a number of sewing pattern catalogs available, for example the Burda catalog [13], the Vogue patterns [14], or the COPA archive [17]. Recently, simulation systems have emerged in which sewing patterns can be tested virtually. For example, in [25] sewing patterns can be placed around a three-dimensional body. An integrated simulation system then computes the result of the sewing process to provide a so-called ‘virtual try-on’ to increase the efficiency of the design process. In the following, we refer to sewing patterns as sewing templates to make the discrimination towards costume patterns more clear. These templates can be seen as additional information as guidance for a tailor.

Regarding the domain of practically working with costumes on a film set, the problems are rather operational. For example, a problem is how to dress a particular actor with a given costume. For many costumes the operational knowledge is already available from daily life. For instance, the order of applying a shirt, trousers, socks, and shoes to an actor is quite clear. However, what about the order of dressing a complex uniform costume? For example, the uniform of a ‘Stormtrooper’ costume from the movie series ‘Star Wars’ consists of 25 parts which are either black or white [24]. To name another example, dressing a carnival costume can also be quite challenging if the ideal order of applying the parts is not known.

In conclusion, the main findings of the literature survey are:

- Currently, there is no established format and no formal language for complex costume descriptions.
- Costumes are usually described rather informally.
- Due to the lack of formal approaches there is also a lack of empirical research on the conventions of vestimentary communication.
- Tools supporting the administration and management of costumes typically treat costumes as monolithic entities.
- A pattern approach has, to the best of our knowledge, not yet been proposed to describe complex costumes to account for reusable solutions to costume design problems and knowledge capturing.

Given these findings, the major aim of the project has been set on designing and building an information system that is based on a pattern language for costumes to support the different stakeholders of the problem domain with an efficient and advanced solution to analyze, design, and manage costumes. To account for the different problem domains that benefit from the approach, the pattern language makes relations to available knowledge, for example the relation to sewing templates serves the tailoring domain.
3.2 Research Design

In order to establish a common vocabulary for information system developers, media scientists, and costume designers, we modeled and described the problem domain in terms of processes, documents structures, and roles involved in costume management in film making. The results of these efforts have been compiled to an article [4]. This article contains a refinement of what ‘management of costumes’ and patterns respectively comprises, namely costume modeling using a graphical editor, costume archival using a versioning system, costume queries (for search in a catalog, lending, and manufacturing), as well as managing costumes at a film set. However, an answer on how to design a pattern language in order to be used with that system was not provided. We created a preliminary pattern format, based on the attributes of prominent pattern formats. The resulting paper [1] illustrated the pattern approach, though, the pattern format and sample pattern was simply sketched.

We used this format as a starting point for usage, revision, and improvement. The attributes of the original format as well as the new or revised aspects are described in Section 5. Based on further literature study on costumes, fashion, and clothing, an initial set of costume primitives has then been identified, described in Section 4. These pieces of clothing provide the basic constructs for costume patterns. The set of costume primitives we discuss in Section 4 and Section 7 is far from being complete, as the field of clothing, fashion, and costumes in particular, is incredibly large. The costume primitives are described in a simple format to reduce the effort of applying the approach in practice and to support a quick start. This approach has also proven successful to reduce the complexity in the field of patterns of software architecture, as described by Zdun and Avgeriou [9].

To exemplify our approach, we then conducted a film study with a small set of selected films (called film corpus) to obtain two complex costume patterns that are specified in all details including the required operators to describe their composition from costume primitives and more simple patterns. The concrete composition operators used in these complex patterns have been abstracted to a composition grammar. To the ease of understanding, we designed a graphical notation to describe compositions. The resulting complex costume patterns are described in Section 8 (Wild West Sheriff) and Section 9 (Wild West Outlaw).

In the following, we describe the procedure we performed to identify the patterns step-by-step, see also Figure 5 for an overview of the procedure.

![Figure 5: Step-by-step identification of costume patterns.](image)

1. **Genre selection:** We decided for the genre of ‘Classical American Western’ as it (i) has a very limited, almost constant inventory of roles, e.g. a Sheriff, Cowboy, Outlaw, Indian, Saloon Lady, Bartender, or Piano Player occur frequently. Further, (ii) as it is the basis for multiple sub-genres like Italo Western or Space Western, it manifests the stereotypes and their costume conventions people obtain through media socialization. Also, (iii) the roles, especially the Sheriff, change their costumes seldomly, which increases the importance of related costumes for the vestimentary communication. Next to the ‘Classical American Western’ there are a lot of other genres, which would be appropriate for costume pattern identification. For example the Mafia-Genre or the High-School-Comedies are also defined by a collection of frequently occurring stereotypes. However, the definition and scope of genres is rather heterogeneous and
there are some genres which are quite broad and versatile like the Thriller-Genre or the Science-Fiction-Genre. Further investigation is required to evaluate if the costume pattern approach works within such genres.

2. **Film corpus selection**: This step is quite important for the outcome of the pattern identification as it provides the basis of material that is abstracted into one or more patterns. We selected films, which are either highly ranked in critics or frequently discussed, ranging from early evolvement of the genre (Stagecoach) up to late films (Ride The High Country). The final film corpus is as follows:

   1. 1939 Stagecoach (German: Ringo)
   2. 1946 My Darling Clementine (German: Faustrecht der Prärie)
   3. 1952 High Noon (German: 12 Uhr Mittags)
   4. 1956 The Searchers (German: Der schwarze Falke)
   5. 1958 The Big Country (German: Weites Land)
   6. 1959 Warlock (German: Der Mann mit den goldenen Colts)
   7. 1959 Rio Bravo (German: Rio Bravo)
   8. 1960 The Magnificent Seven (German: Die glorreichen Sieben)
   9. 1962 Ride The High Country (German: Sacramento)

3. **Character selection**: The films of the selected corpus were analyzed and screenshots of the characters associated with the role ‘Sheriff’ and the role ‘Outlaw’ have been identified. In most of the films, the Sheriff and the antagonist ‘Outlaw’ were very important for the film, thus they are also very important from a costume design perspective. In six out of the nine films 13 different Sheriff characters and 22 different Outlaw characters were contained, varying in age, attitude, behavior, and authority (Sheriff, Deputy Sheriff, Marshal // Head of the Outlaws, Accomplice).

   My Darling Clementine: Marshall Wyatt Earp, Luke the old Marshal // Old Man Clanton, Billy Clanton, three other Clanton brothers
   High Noon: Sheriff Will Kane, Deputy Harvey Pell // Frank Miller, Ben Miller, two of Frank Miller’s accomplices
   The Searchers: –
   The Big Country: –
   Warlock: Deputy Roy Thomson, Deputy Johnny Gannon, Sheriff Eduard Calhoun // Abe McQuown, Curley Burne, Billy
   Rio Bravo: Sheriff John Chance, Deputy Stumpy, Deputy Dude, Deputy Colorado // Nathan Burdette, Joe Burdette
   The Magnificent Seven: –
   Ride The High Country: Ex-Marshall Steve Judd // Billy Hammond, Henry Hammond, three other Hammond brothers

4. **Costume description**: The costumes of the selected characters have then been described in detail from the perspective of the actor (regarding left and right), including the clothing elements of the costume (e.g. trousers) and the respective clothing specifics (e.g. material: jeans, color: blue stone-washed, form: five-pocket, aging: old and worn for several days). Modeling clothing elements contributed to the set of costume primitives.
5. **Composition description:** For the described costumes the operators have been identified that need to be available to specify a costume structure as a composition of costume primitives and, if applicable, of other costume patterns. This step contributed a basic vocabulary for the composition of costume primitives and more simple costume patterns.

6. **Abstraction to costume pattern:** In order to obtain the costume pattern, similarities and multiple occurrences of particular aspects have been identified. The ‘Sheriff’s Star’ has been revealed as the most significant property in order to recognize a role functionally as a Sheriff which qualified it to be described as a costume pattern with a rich amount of attributes that characterize its aspects. For the ‘Wild West’, clothing like a hat, vest, or holster define place and time in a vestimentary manner. Thus, for the ‘Wild West Sheriff’ pattern both components are required, the Sheriff’s Star pattern and the Wild West clothing.

4 **Costume Primitive Format**

In this section and the following section (Costume Pattern Format) we describe the results of our requirements analysis, mainly conducted from the IT perspective. For formalizing costumes and their abstraction to costume patterns we distinguish between costume primitives and costume patterns. Costume primitives are simple clothing elements which constitute the basics of our pattern language, whereas costume patterns include advanced concepts like composition graph and dressing sequences. These concepts are of vital importance for the costume patterns, but they can also be applied for different approaches which are not concerned with patterns, e.g. for modeling a particular outfit.

In the semiotic approach to clothing (semiotic means study of signs) we found an initial approximation on structuring and formalizing scientific findings, which we could build on to obtain a ‘starter kit’ of costume primitives. When we refer to costume primitives in the following, we do not break them down to the level of granularity as made by Barthes [2]. For example, we see a shirt as a costume primitive, although it is made up of buttons, a collar etc. Furthermore, we see a pair of boots as costume primitive, although it consists of a left boot and a right boot, which are possibly not even symmetric. However, as these boots are typically applied together and they are strongly related to each other, they are considered as one primitive.

A set of costume primitives can be derived following Barthes [2] description of meaningful elements of clothing, which have significant impact on the vestimentary communication: Accessory, Apron, Bag, Blouse, Bracelet, Cape, Clip, Coat, Dress, Glove, Handbag, Headdress, Jacket, Necklace, Shawl, Shirt, Shoe, Sweatshirt, Tie, Trousers, and Veil. To this set, manifold extensions can be made, by (i) studying literature, such as encyclopedia like the dictionary of costumes and fashion [12] or fashion magazines like Vogue [7], and (ii) by analysis of costumes and clothes frequently used in films. For example, further costume primitives that could be added based on fashion magazines are Boxers, Bra, Briefs, Cardigan, Glasses, Flip Flop, Hoodie, Pantie, Polo, Sweater, Socks, and Undershirt. Regarding film analysis, the identification of the Wild West Sheriff pattern revealed further costume primitives not contained in the initial set. These are Ammunition Belt, Belt, Boots, Holster, Neckercchief, Pocket Watch, Revolver, Spurs, Union Suit, Wild West Vest, Wild West Hat, and Wild West Tie. The identification of the Wild West Outlaw additionally contributed Chaps and Suspenders. The set of costume primitives can be further extended, given that the distinction between costume primitives and props is not always clear. Props can become a costume primitive when they are related to a costume in a strong way. For example the Ammunition Belt is an important part of
the Sheriff costume, but it can also be seen as a prop because the actor often takes it on and off and uses it on screen, which also classifies it as a prop.

However, completeness of the set of costume primitives will likely be never achieved. The main reason for this is the incredibly large base of information that would need to be analyzed. Moreover, it makes a significant difference on which literature, fashion and lifestyle magazines, or films the identification of costume primitives is based. Thus, our approach is to specify new costume primitives only when they are needed to describe a costume pattern. Another aspect of the costume primitives is the degree of detail in which they are described. For example, Barthes [2] provides a lot of details about properties of the clothes, like size, softness, and flexibility of certain material. Due to the level of abstraction that we chose for the description of costume primitives we do not go into further detail here. It is up to future research to find the right degree of granularity and abstraction for a costume primitive.

Figure 6 shows the format of costume primitives, containing the attributes Name, Icon, Context, Description, and Additional Information to serve the needs of different stakeholders. The time that is needed to document a costume primitive depends on the details provided in the description. We use the same definitions of attributes in costume primitives and costume patterns. Details about the attributes are given in Section 5. Using a subset of attributes allows converting a costume primitive that turned out to be a solution to a significant costume design problem into a costume pattern. For example, a Sheriff’s Star could be seen and modeled as a costume primitive. However, thorough analysis of Western films revealed that the Sheriff’s Star is the most significant property to recognize a role functionally as a law enforcement officer. This qualifies the Sheriff’s Star as solution to a frequent costume design problem. It can be used in other compositions though. For example, the Wild West Sheriff composes the Sheriff’s Star pattern with a particular constellation of Wild West clothing. We argue that the discrimination of the simple types in our approach, i.e. the costume primitives, and the complex types, i.e. the costume patterns, needs to be performed by a domain expert.

Figure 6: Conceptual model of costume primitives.

We further propose to use a clothing taxonomy to classify each piece of clothing. A simple sketch to illustrate such clothing taxonomy is provided in Figure 7. The purpose of the classification is to ease search in a catalog of costume primitives and to easily find related primitives. For example, if a particular hat in a costume rental catalog is classified as Cowboy Hat in the clothing taxonomy, a search for Wild West Hats will show it as result.
5 Costume Pattern Format

In our former work [1] we proposed a preliminary pattern format to illustrate the pattern approach in costume management. The pattern format oriented itself on the architectural patterns by Alexander et al. [5] and the Enterprise Integration Patterns by Hohpe et al. [6]. In the course of our further work with the costume pattern approach, we had to perform several changes to this format to account for insight in applying this format in practice. The basic attributes of the format were name, icon, context, sub-patterns, application, and related patterns. We replaced the sub-patterns list through composition. The composition is the main solution aspect of a pattern, revealing the inner structure of the constituent parts of a costume solution and the way the parts are composed. A composition graph, that can be used to describe the composition, is based on the composition operators we identified so far. Further, we have added the attributes purpose (which resembles the attribute problem that is used in most pattern languages), description, examples, application, sewing templates and aging instructions. As this format is rather large, we distinguish aspects of a pattern which we consider ‘timeless’ in terms of Alexander [10] and aspects which focus only on information needs of particular stakeholders of the pattern (‘additional information’), see Figure 8.

Figure 8: Conceptual model of costume patterns and their main constituents.
The pattern format now comprises the following core attributes:

- **Name**: For readability each costume pattern is given one or more names that specify the concept that the pattern represents. To ease readability technical identifiers are omitted in this paper.

- **Icon**: Each costume pattern contains one or more graphical icons that sketch the costume pattern. An icon may either be a drawing, a computer-created figure, or a photo.

- **Purpose**: The vestimentary message that the costume pattern transports.

- **Composition**: Often the structure of a costume pattern is a complex composition out of costume primitives and other costume patterns. The composition contains (i) a set of all elements, (ii) the operators used to compose the parts, either described textually or as composition graph, and (iii) optionally a recommended composition order. The elements are identified by name, id, or icon. The operators can be described textually or through a more formal description, for instance through process fragments [28]. The composition is a very important aspect of a costume pattern as it explains how the different aspects are working together, i.e. how single pieces of clothing are set into a complex relation.

- **Forces**: Forces reflect interpretations and reasons for the composition design of a costume pattern. Forces of a costume pattern are defined by the suspense of the diegetic world. That means each character and each costume is related to several terms and conditions that are part of the cinematic world they occur in. Thereby they represent the main results of media-scientific research.

- **Context**: For each costume pattern the context states ‘where’ (area), ‘when’ (epoch/era/period), and in ‘which genre’ it is usually applied. Multiple entries can be stated here.

- **Description**: The description characterizes the costume pattern itself and the stereotypes it comprises. We consider usage of ontology as quite useful here, with respect to usage in information systems.

In addition to the ‘timeless’ attributes described above additional information is useful for some scenarios:

- **Related Patterns**: This attribute provides references to costume patterns that are related, e.g. through the genre, epoch, stereotype etc.

- **Known Uses**: Examples of the application of the costume pattern can be provided through photos and references to films, film scenes, theatre plays etc.

- **Application**: This attribute describes different procedures associated with the costume pattern for its application in practice. This may contain (i) dressing instructions, i.e. a description how and in which order to dress an actor with a complex costume following this pattern. Further, (ii) preparation instructions, i.e. procedures related to preparation for a scene like aging and cleaning of the costume are covered here, informally as plain text or formally in form of process fragments.

- **Sewing Templates**: To support tailoring of costumes, sewing instructions in form of sewing templates can be added. This attribute is more important to costume primitives, rather than to costume patterns.
6 Composition Operators

As noted in the introduction, we understand the composition operators as a grammar to compose costume primitives and costume patterns to more complex structures. In the process of modeling the Wild West Sheriff costume, we identified several operators, which are required to describe how its constituent parts are connected to each other.

An operator can technically be described by a 5-tuple \( O = (Source, Target, Type, Multiplicity, Details) \), where:

- *Source* and *Target* identifies a costume primitive, a costume pattern, or an ad-hoc grouping of clothing elements.
- *Type* identifies the operator that connects the *Source* to the *Target* (written \( <<\text{verb-preposition>>} \)).
- *Multiplicity* states the cardinality relation between the *Source* and the *Target* (written \( c_1:c_2 \), where \( c_1 \) and \( c_2 \) may be one of these cardinalities: 1, n, 0..1, 1..n, n..m, where \( n \) and \( m \) are natural numbers).
- *Details* contains description and annotations beyond the abstract description of the operator type, for instance text, process fragments which explain how to apply them, sketches or photos.

The operators we identified during modeling the composition structure of the Wild West Sheriff costume pattern are listed in Table 1.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attached-to</td>
<td>The source element is attached to the target element, so that it is visible on its surface.</td>
<td>A Sheriff’s Star is attached to a vest.</td>
</tr>
<tr>
<td>Belted-on</td>
<td>The source element is belted on the target element.</td>
<td>A leather belt is belted on a pair of trousers</td>
</tr>
<tr>
<td>Hooked-in</td>
<td>The source element is hooked in the target element.</td>
<td>A pocket watch is hooked in a vest</td>
</tr>
<tr>
<td>Tied-around</td>
<td>The source element is tied around the target element.</td>
<td>A neckerchief is tied around a shirt.</td>
</tr>
<tr>
<td>Tucked-in</td>
<td>The source element is tucked in the target element.</td>
<td>A shirt is tucked in trousers.</td>
</tr>
<tr>
<td>Worn-above</td>
<td>The source element is worn above the target element, thus the target element has to be applied first.</td>
<td>A vest is worn above a shirt.</td>
</tr>
<tr>
<td>Worn-on</td>
<td>This operator is required to express which elements are worn with direct physical contact.</td>
<td>A hat is worn on a Sheriff’s costume.</td>
</tr>
</tbody>
</table>

Table 1: Overview of composition operators.
When the film corpus is extended or further complex costumes are investigated, there is most likely the need to add further operator types. Therefore, we do not see the presented set of composition operators as final, but extensible. Even more types of operators can be identified in real life when dressing oneself, or in the working practice of costume designers and dressers. Multiple operators can be used at the same time to connect two elements. Besides these composition operators, there is also a need to express logical connectives between elements in a grammatically valid way. Through such connectives, variants of a composition can be captured in the composition graph. For the Wild West Sheriff pattern however, the ‘or’ operator was already sufficient, see Table 2. Further connectives to extend the costume pattern language are likely to be required in further modeling of complex costume patterns, like ‘only-if’.

<table>
<thead>
<tr>
<th>Connective</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>The ‘or’ connective is required to express composition variants. It represents a logical operator (here: either... or... / XOR).</td>
<td>Either a neckerchief or a tie, but not both of them at the same time.</td>
</tr>
</tbody>
</table>

Table 2: Overview of composition connectives.

For a more tangible and easy-to-read description, we created a simple graphical notation to describe compositions. An operator is represented by an arrow, labeled with the operator type that is used. It is drawn from one costume pattern icon to another to indicate the elements of the connection. The arrow-head represent the target, its origin the source. Figure 9 illustrates how the graphical notation can be used, attaching a Sheriff’s Star to a vest.

Figure 9: Example for graphical composition.

In our graphical notation the ‘or’ connective does not have any arrow-head, because a subject or object cannot be distinguished anyway, see for example the operation on choosing either a neckerchief or tie depicted in Figure 10.

Figure 10: Example for usage of a logical composition operator to express variants.

The use of multiplicity allows specifying in a composition, how many pieces of an element are required. Normally, the multiplicity is 1:1, meaning for instance there is one Sheriff’s Star required to be attached to one vest. This default multiplicity is not shown on the operator arrows. If the multiplicity however differs from that default, it is shown explicitly, as exemplified in Figure 11: A pair of boots consists of a left and a right boot. One spur is belted on each of them. Thus, the multiplicity is specified as 2:1 (it reads from the source to the target). Through this concept, it can also be specified, that particular elements are optional using the style common in information modeling: 0:1 stands for cardinality zero or one.
In order to account for more fine-grained elements than on the level of granularity we proposed, one could use ‘composed-of’ as an additional operator type. A vest, for instance, actually consists of parts like a (left and right) front part, back part, front collar, back collar, pocket and pocket braiding. We considered this level of granularity in the annotations part, in which corresponding sewing templates are placed. When using such an operator, compositions could be described much more fine-grained. This, however, increases the degree complexity of the composition graphs.

7 Examples of Costume Primitives

In the following, we provide the description of three costume primitives we found in the context of Western films, the ‘Wild West Vest’, the ‘Wild West Boots’, and the ‘Cowboy Hat’.

7.1 Wild West Vest

Name: Wild West Vest

Icon:

Figure 12: Icon for the Wild West Vest.

Context: The Wild West Vest is mainly used for Sheriffs, Cowboys, Ranchers, and Bandits around the Pioneer era (mid to late 19th century) in Western North America; Main genre: Western.

Description: A Wild West Vest is an upper part of clothing without sleeves. It mainly consists of a front and back part, collar, bound pockets and some buttons. When the vest is worn as a part of a suit then it should be made out of the same fabric as the suit. Most recommended materials for a Wild West Vest are leather, jeans or other robust fabrics available in the mid to late 19th century.
Additional Information

Templates for sewing:

Figure 13: Templates for sewing, which are annotated to the Wild West Vest.

7.2 Wild West Boots

Name: Wild West Boots

Icon:

Figure 14: Icon for the Wild West Boots.

Context: The Wild West Boots are mainly used for Sheriffs, Cowboys, Outlaws, and Ranchers around the Pioneer era (mid to late 19th century) in Western North America. Next to the original use as riding boots the Wild West Boots became a sign for American culture and are found in different actual styles; Main genre: Western

Description: Wild West Boots are a specific type of footwear. They are a special sort of riding boots, which cover foot and ankle, and extend up the lower leg. Wild West Boots can be distinguished by their high heel compared to other male shoes, their pointed toe, and their lack of lacing. Traditionally made of leather, the Wild West Boot is characterized by typical embellishment.

7.3 Cowboy Hat

Name: Cowboy Hat

Icon:

Figure 15: Icon for the Cowboy Hat.
**Context:** The Cowboy Hat is mainly used for Sheriffs, Cowboys, Outlaws, and Ranchers around the Pioneer era (mid to late 19th century) in Western North America. Nowadays it is still worn by many people like ranch workers in the USA, Canada or Mexico, participants at the rodeo, or country music singers. Since its date of origin the Cowboy Hat is related to associations like freedom, adventure, virility, and loneliness. It has a confirmed place in western culture. Main genre: Western

**Description:** The Cowboy Hat is a type of hat that originates in the Pioneer era in Western North America. The Cowboy Hat can be identified by his high crown, a wide brim that is bent up, and a leather or rip ribbon around the crown. The material an original Cowboy Hat is made of is the beaver fur, which gives the hat its special shiny soft-felt look. It was developed to protect the wearer of weather circumstances while working outside or riding. Because the Cowboy Hat is worn outside and typically for a very long time it often gets modified by the special needs of the wearer like bending the brim in a way that water will not run into the face of the wearer during rain.

8 Costume Pattern: The Wild West Sheriff

**Name:** Wild West Sheriff

**Icon:**

![Icon of the Wild West Sheriff pattern.](image)

**Purpose:** The Wild West Sheriff costume stands for a man of law working in a town in the Wild West.
Composition:

**Figure 17:** Composition graph of the Wild West Sheriff pattern.

**Composition elements:** The costume consists of the following costume patterns: Union Suit, Shirt, (Wild West Tie or Neckerchief, specified as ‘ad-hoc’ group in the composition graph), Wild West Vest, Sheriff’s Star, Pocket Watch, Pair of Trousers, Belt, Ammunition belt, Socks, Wild West Boots (spurs belted on boots), Coat, and Hat.


**Forces:** The role of the Sheriff stands for the man of law in his town. Therefore he is wearing a Sheriff’s Star, which is the symbol of law and order and the official sign of being a person of law who takes care of the police duties of the town. In being a respected person in his town he is considered to be related with garments that match a character that is behaving controlled, grown up and sorted. For example the Wild West Tie, the Watch and the Coat demonstrate this. Especially the cut of the Coat is associated with a military coat and relates the inherent values like neat, fair and lawful to its wearer.

**Context:** The Wild West Sheriff occurs around the Pioneer era (mid to late 19th century) in Western North America; Genre: Western.

**Description:** The role of a Wild West Sheriff represents a law enforcement officer. The Wild West Sheriff is responsible for a particular county.
Additional Information

Related Patterns: Modern Sheriff, Fantasy Sheriff, Cowboy, Indian, Outlaw, and Rancher.

Known Uses: John Wayne as John Chance in Rio Bravo (1959); Gary Cooper as Will Kane in High Noon (1952); George Bancroft as Marshal Wilcox in Stagecoach (1939).

Application:

Dressing instructions (textual): Dressing instructions follow the recommended composition order. The buttons of the shirt have to be closed, the sleeves of the shirt are mostly worn long, but there are some characters with willowed sleeves. A Wild West Sheriff wears his shirt tucked in the trousers, while the trousers are worn either above the boots (usually) or tucked inside.

Dressing instructions (graphical):

Figure 18: Graphical dressing instructions to support costume usage in practice.

Preparation instructions: For preparation the costume needs to be checked for completeness. Then, cleanliness and aging can be checked. Aging, which means giving character or ‘patina’ to a costume element, refers to preparing dirt or dust on boots, trousers, coat and hat; the shirt can be aged with water for simulating sweat. Then, applying the belt on the trousers; setting time of the pocket watch to time in the film scene and hooking it on the vest; preparing the knot of the tie; belting spurs on the boots, and laying socks on the boots.

Variants: The vest is most of the time worn open, only the very neat characters are seen with a closed vest. Characters may vary in some details, e.g. some wear a neckerchief instead of the tie (usually), a long sleeve t-shirt instead of a shirt, or a shirt with an application at the front. Also the ammunition belt can vary in having one or two revolvers attached to it. Sometimes the trousers are tucked in the shoes, worn above, or even rolled up.
9 Costume Pattern: The Wild West Outlaw

Name: Wild West Outlaw

Icon:

![Icon of the Wild West Outlaw Pattern.](image)

**Figure 19:** Icon of the Wild West Outlaw Pattern.

**Purpose:** The Wild West Outlaw costume states that the role who is wearing it is the troublemaker of a town in the Wild West. Not being part of the community life of the town, the Wild West Outlaw invades the town and causes danger.

**Composition:**

![Composition graph of the Wild West Outlaw pattern.](image)

**Figure 20:** Composition graph of the Wild West Outlaw pattern.
Composition elements: The costume consists of the following costume primitives: Union Suit, Shirt, Neckercchief, Wild West Vest, Pair of Trousers, Chaps, Belt or Suspenders, Ammunition Belt, Socks, Wild West Boots (optionally with Spurs), Jacket, and Hat.


Forces: Because the Wild West Outlaw is coming as a troublemaker from the plains to the town, his clothes are usually already worn for a long time and quite dirty. He is expelled from the community life of the town and lives on a horseback most of the time. This can be communicated through the Chaps. Other specific costume primitives like the Jacket and the Suspenders are related to a person who seems less distinguished and less grown up. In this spirit the cut of the Jacket in the Wild West Outlaw pattern is less formal, less accurate, and associated with youth.

Context: The Wild West Outlaw occurs around the Pioneer era (mid to late 19th century) in Western North America; Genre: Western.

Description: The role of a Wild West Outlaw represents the antagonist to the Wild West Sheriff. He resides outside the law and terrorizes townspeople by stealing, raping and killing.

Additional Information

Related Patterns: Wild West Sheriff, Barman, Saloon Girl, Cowboy, Modern Outlaw, and Biker.

Known Uses: Tom Tyler as Luke Plummer in Stagecoach (1939); Sheb Wooley as Ben Miller in High Noon (1952); John Russell as Nathan Burdette in Rio Bravo (1959).

Application:

Dressing instructions (textual): The topmost buttons of the shirt are often worn open and the sleeves of the shirt are mostly worn long. A Wild West Outlaw wears his shirt most of the time tuckted in the trousers, while the trousers are worn either above the boots or tucked in. To secure the trousers a belt or suspenders are applied, sometimes both. The west is worn open while the neckercchief gets tied around the neck or the nose to cover the face.

Dressing instructions (graphical):

Figure 21: Graphical dressing instructions to support costume usage in practice.
Preparation instructions: The costume needs to be checked for completeness and especially its state of aging (usually: quite dirty, worn for a long time). Then belt or suspenders can be applied to the trousers, as well as the knot of the neckerchief can be prepared and the socks can be laid on the boots before the actor gets dressed.

Variants: There are different occurrences of Wild West Outlaws like the head of the outlaws or one of his accomplices. They also vary in the costume. Most of the time the Wild West Outlaw in higher position is presented without the chaps, wearing the tie instead of the neckerchief, being more tidy than the others. Also, there can be found a wide range of weapons including knives, revolvers, and rifles carried by the Wild West Outlaw. When looking at the concrete costumes from which the pattern originates, there are several differences in the cut of clothes and in their specific way of wearing. For example, several types of shirts can be found and several different ways of wearing them, like buttoned up or leaving a few buttons open, tucking it in the trousers or wearing it on top of them etc. The pattern captures the most common way.

10 On Differences and Commonalities of Patterns

Having a set of costume patterns ready, further analysis can be made based on the differences and commonalities in the different pattern attributes and composition structures. For example, a relationship diagram can be created that shows the relations between different patterns and how they fit into a particular context. Figure 22 shows how such a diagram might look like. Starting from a general category, Wild West from mid to late 19th century, we can distinguish between male and female roles. Within the category of the male role we identified the Wild West Sheriff and the Wild West Outlaw. They relate to each other as ‘Antagonist’. Many further patterns can be identified in the genre and the relations between them can be used to create a graph with lots of information to reason about.

![Figure 22: Relations between costume patterns](image)

Furthermore, through an in-depth analysis and comparison of composition structures we are able to generate new knowledge about costumes. In the following we show one exemplary analysis, which we call ‘confrontation’. For this analysis we merge the composition graphs of the Wild West Sheriff and the Wild West Outlaw. Based on this merged graph, we apply color-coding to highlight the differences of the Wild West Sheriff (in blue) and the Outlaw (in red). Commonalities are depicted in grey. Figure 23 shows the outcome of the merge and color-coding which can be interpreted in terms of costume primitives and forces.

The specific costume primitives of each role, as highlighted in blue and red in Figure 23, are closely related to the characteristics of the stereotype role they are part of. As described in the pattern forces the Sheriff’s Star is the symbol of law and order and thus it could never be found in the Wild West Outlaw pattern. On the other hand, the Sheriff is a citizen of the town, whereas the Outlaw is expelled and lives most of the time on a horseback and therefore needs the Chaps.
Figure 23: Differences and commonalities of the Wild West Sheriff and the Wild West Outlaw.
When taking a closer look at the costume primitives which are only found in the Wild West Sheriff pattern like the Wild West Tie, the Watch, and the Coat, one can conclude that these garments are considered to be related with a character that is behaving controlled, grown up and sorted. The Wild West Outlaw pattern with specific costume primitives like the Chaps, the Jacket and the Suspenders seem less distinguished and less grown up. This vestimentary communication provides the recipient already with a lot of information about the characters, without the need to transport this information in spoken language. In showing the differences and commonalities of the two patterns as depicted in Figure 23, some parts of a ‘vestimentary code’ become obvious.

We can apply abstraction and generalization to obtain a more general costume, the Wild West Role (male), see Figure 24. Such a ‘generalized’ costume can be used in many different ways. One way is to use it as a template for costume design, to accelerate the design process. Another way is to use it as query to be executed against a costume catalog offered by a costume rental and shop to get a first offer that can be refined in a next step. In addition, such generalized costume structures can be used to design characters which fit into a particular film context without indicating their function.

Figure 24: By abstracting from the details of a Wild West Sheriff and a Wild West Outlaw we obtain a generalized Wild West costume: the Wild West Role (male).

By reducing all the variations and different possible ways a certain character can be dressed, abstraction into a pattern allows thorough comparison to enable making general statements about the vestimentary conventions in films. However, this abstraction also means reduction. The constituent parts of the patterns are supposed to be linked to a clothing taxonomy to ease the selection of real pieces of clothing when creating a concrete costume in order to not lose the different variations of costumes. This does not mean that there is only one concrete costume possible to realize the solution sketched by the pattern. In fact, the pattern provides the user with a suggestion that needs to be interpreted and applied in a creative and innovative manner for every film and every character. In designing a costume, the costume designer oscillates between stereotype elements on the one hand and individual elements, changes, and refinement on the other hand. The stereotype elements are important for the understanding of a character while the individual elements make a character authentic. The patterns show the pure stereotype elements that everyone understands. Variations in color, material, design, and size can show individualization related to a certain figure.
11 Conclusion and Outlook

In this paper we presented a pattern language to capture knowledge of costume design for films. We followed a clearly described research design to obtain the different components of the language – a format for costume primitives and costume patterns, a set of costume primitives and costume patterns, and composition operators for modeling complex costumes. We exemplified the components along different examples to illustrate the key concepts and findings. Based on the developed pattern approach some media-scientific statements can be made. In having an initial format to capture the vestimentary code of films it is possible to show the conventions that have been developed between the use of costumes in films and their understanding by the recipient. As exemplified in the previous section, the comparison of the identified patterns makes the specifics of a certain role clearer. This formal approach based on a pattern language allows an empirical analysis of costume usage. Through this pattern-based approach we would like to address the problem of finding a costume language that allows overall statements on the use of costumes in films.

However, from a practical point of view the question remains if and how well identified costume patterns are applicable in practice, i.e. in other films. Although the costume patterns describe the core of a solution to a costume design problem, their application in a particular film for a particular character and actor, etc. demand a lot of decisions. For instance, concrete fabrics have to be chosen for the costume primitives used in a costume pattern. In addition, depending on many different influencing factors, a costume designer may need to make changes and extensions when applying a costume pattern in costume design. Further investigation is required to find out how the pattern language needs to be modified or extended to efficiently support this decision making and to ease and fasten the regular way of creative working with costume designs in practice. Evaluation through domain experts and subsequent improvement of the pattern format and the graphical composition notation is another important task in our future work.

From a research point of view, some aspects require further exploration before building such a supporting information system. Two of them are shortly described in the following: (i) so far we only modeled operators that compose the different elements of a costume. However, further connections beyond the composition operators are conceivable to be defined, for instance ‘same-color’, ‘different-color’, or ‘same-fabric’. Such connections do also have an impact on decision making when applying the costume pattern in practice. Because the used film corpus contained a few black-and-white films the color aspect has not been considered yet and requires further investigation; (ii) most challenging from a research point of view are advanced usage scenarios exploiting the information that is contained in the pattern description and in the additional information provided for it. For example, the dressing instructions have been modeled explicitly, based on best-practice in dressing at a film set. Possibly, such dressing instructions can also be generated automatically. If there is no best-practice information available for a costume yet, a possible solution could be proposed on the fly. However, not all operations in our current composition grammar imply dressing dependencies and thus further investigation is required for generation of dressing instructions from a composition graph.

To conclude, we aim at enabling a new branch of research, which opens many new questions, but also provides a lot of new answers to questions never asked before. We change the setting of patterns well-known in architecture and programming to a new area, crossing stakeholders and disciplines by applying patterns and modeling technology to film costumes. Our approach focusses on film productions, but is not limited to this scope. Possibly, it can even be used as a basis for research on formalizing fashion and clothing thinking in order to enable analysis of day-to-day vestimentary communication.
About the Authors

The author team of this work is composed of researchers and practitioners from different disciplines. On one side, David and Frank are working in computer science research. Frank has been working in software industry for over two decades; David is currently finishing his Ph.D in computer science, focusing on business process technology. Both contribute to the paper as experts from the domain of computer science, information technology, and language design. On the other side, Johanna and Lutz are coming from the media science domain. Lutz is an expert in the domain of media science research, with focus on the field of media and communication theory and the social impact of information technology. Johanna studied costume design at an international film school and worked in several film productions in the costume department in different roles. At the film school she worked with several renowned costume designers. Johanna contributes to the paper as a domain expert in costume design, film making, and media science. This interdisciplinary team provides the domain knowledge to leverage the pattern approach for costume design considering the possibilities of information technology.

Acknowledgments. Many thanks go to Uwe Zdun for his constructive feedback and helpful comments in the shepherding process and to Laura Roberts for proofreading of the paper. Further thanks go to EuroPLoP’12 participants for their valuable feedback: Allan Kelly, Andreas Fießer, Andreas Rüping, Aya Matsumoto, Damien Gouteux, Kaori Harasawa, Klaus Marquardt, Michael Weiss, and Uwe van Heesch.

References


   Online available at http://voguepatterns.mccall.com/

15. go_disco*: actorsBreakdown, 2011.
   Online available at http://www.go-disco.com/products/index/2

   Online available at http://www.frt.at/produkte/profundus/

17. University of Rhode Island: Commercial Pattern Archive (CoPA),
   Online available at http://www.uri.edu/library/special_collections/COPA/sample.php


   Online available at http://costumebot.com/archives/8


A process for the validated integration of patterns

D. Gouteux, B. Hamid and C. Percebois
IRIT, University of Toulouse
118 Route de Narbonne, 31062 Toulouse Cedex 9, France
gouteux.hamid.percebois@irit.fr

Abstract. Design patterns are now widely used as a meaning to store and retrieve valuable expert information in order to solve a given problem or provide a selected property. During the software development life cycle, software engineers choose and integrate patterns depending on the specifications. We are proposing a proven pattern integration process. The proposed solution highlights a set of artefacts and practices, based on widely accepted standards such as OCL and UML, a pattern modeling language and using SPEM for the process definition. While ensuring the correctness of the integration, our process is flexible enough to let the designer adapt the solution to his/her particular needs. To illustrate our process, we will apply it to the GoF Mediator pattern and an UML class diagram representing the application of an air traffic controller.

Keywords: Design Pattern, Class Diagram, Process, Pattern Integration, Proofs, OCL, UML, SPEM.

1 Introduction

Model-Driven Engineering (MDE) [6] provides a very useful contribution for the design of embedded systems, since it bridges the gap between design issues and implementation concerns. It helps the designer to specify in a separate way functional and non-functional requirement issues at an even greater level which is very important to guide the implementation process. Of course, a MDE approach is not sufficient only by itself but offers an ideal development context. While using the MDE conceptual framework, it is possible to help software engineers specialists in their task. Indeed, it would be interesting to suggest solutions and to guide them according to their specific requirements using patterns. We leverage on this idea to propose a new pattern integration process.

Object-oriented design patterns [2] are medium-scale patterns comparing to architectural patterns but they are still at a higher level than the bare programming language. Design patterns help to implement application functionalities and can be used for example to specify communication schemes of a particular application domain. The wanted role of pattern use is to ease, systematize and standardize the approach to the construction of software. However, the problem consists in identifying them explicitly for reuse and adding information to ease their integration. This leads to the necessity to propose integration models.
The key point which appeals to combine MDE and patterns, is to use the patterns expressed as models. The GoF [2] presented a first step by giving a class diagram and some free text for each design pattern, but by modeling more accurately the pattern and dropping the free text, we can consider the pattern as a model. In that case, it fits perfectly into an MDE approach. Viewing the pattern as another model, we will be able to handle it and to have the same transformation possibilities than the application model. Furthermore, we could then see the integration process as a purely MDE transformation, an operation taking the application model and the pattern model as arguments and resulting into a new application model. Constraints implied by the formalization are counteracted by the automation provided by MDE.

Still, there is no clear, well-documented and accepted process dealing with the integration of a pattern into the system under development, and that is what we are aiming for. Although some progress has been made on modeling this problem [4], no process has emerged during the past years to offer a clear and concrete method for integrating a pattern correctly by construction, and ensuring that its properties are correctly absorbed by the application. Our integration is aimed to add a best practice, provided by the pattern, to an application. We can integrate at any moment of the conception, whether at the start of it, or to enhance an existing system. The main point is that the integration doesn’t repair faulty design or unfinished one, but adapts the pattern to the given context of the application. Our approach is focused on the design phase and on design patterns for object architecture. We will discuss the different artefacts implied by our process, then we will present the process itself and finally an example to demonstrate it.

2 The artefacts

This section describes the different artefacts consumed and produced during each activity of our process. First we have to formalize the properties provided by the pattern, and its constraints. Then we need to link the application diagram with the pattern. To achieve this, we introduce specific artefacts such as the role diagram and the casting diagram. Correctness of the integration is assumed by checking the properties and proving the final result when the pattern is fully integrated into the resulting application diagram.

2.1 Pattern modeling and formalization

Our process requires the modeling and formalization of a pattern. This formalization captures the properties provided by the pattern and its constraints. In this paper, we use the TERESA pattern modeling language [3] for this modeling.

Properties are specification constraints which capture the intent of the pattern and formulate the solution of the problem. On the opposite, implementation constraints are requisites of the pattern and concern the solution itself in terms of classes and links, or specific artefacts needed by the implementation language
or the framework. The pattern is defined with its properties and constraints validated. Both will be compiled as a set of assumptions which will have to be satisfied by the domain application. If the assumptions are not met, the pattern will not be able to deliver its properties. Patterns described with TERESA offer an external interface to allow interaction with it such as our integration or composing patterns. Internal structure is represented by a class diagram or a sequence diagram or both, showing how the pattern resolves the problem. In this paper, we use OCL [7] as the formalization language for both external properties and internal constraints.

2.2 The role diagram

The role diagram [1] represents the solution offered by the pattern. The role diagram is more abstract than the class diagram and does not represent specific implementation details of the pattern like the abstract classes but focuses on the solution provided by the pattern. It can be represented by an UML class diagram using the role profile, which extends the UML class diagram metamodel, as in Fig. 1. The stereotype Role of the metaclass Class is defined, along with the stereotypes Implies and Excludes of the metaclass Association, excluding or forcing the combination of roles together.

Two roles excluding each other means that no class can play both roles at the same time. This kind of information was stored into the role matrix of Riehle [1], aside the role diagram itself, but we merged both into one diagram. An example of role extracted from GoF patterns is for example the Mediator role in the Mediator Pattern or the Colleague role in the same pattern, which exclude each other.

The role diagram will be used as the external interface of the pattern, presenting its solution to the designer in order to be linked with. During our process, we will link the role diagram to the class diagram of the application using the casting diagram.
2.3 The casting diagram

The casting diagram is an UML class diagram using the casting profile which defines the stereotype Play of the Association metaclass which links the class to its role. Using the Play stereotype means that the class undertakes the same responsibility as the one defined by the corresponding class in the pattern. If a role is not casted, a default class will be generated by our tool. Fig. 2 shows the profile.

![Diagram showing casting profile](image)

**Fig. 2.** The casting profile.

For example, a class in the application could be chosen to play the role of Mediator and an other one could play the role of a Colleague. Because of the freedom left to the user to produce his/her casting diagram, this artefact could be rejected by our process, for example he could choose the same class to play the Mediator and a Colleague. In order to check this, we use the check file artefact.

2.4 The check file

The check file contains a set of preconditions, issued from the properties of the pattern. The application must meet these preconditions in order to be a valid target for integration. In its form, the check file is a simple text file which has the preconditions written as a set of conditional expressions in OCL. The check preconditions are using the roles as formal parameters. During our process, we will replace these formal parameters by classes of the application following the casting. Chosen classes in the application class diagram will replace the roles in the OCL expression and produce new conditional expressions. If all expressions pass, the casting will be validated.

There are two kinds of preconditions: preconditions on the application and preconditions on the casting. A class playing a role must meet some constraints, but also, some roles are exclusives: a unique class can not plays two roles which are exclusives to each other. The former is an example of application preconditions and the latter of casting preconditions.
2.5 The proof file

A first result of our process is the simple merging of both class diagram in one, following the links made by the casting. The designer can exit the integration process at this step with a correct new application model, but he can also choose to manually edit the model, using Papyrus [15] or any other UML modeling tools producing XML. After each change, we must be sure that the model is still correct and provides the functionality the pattern provided. In order to do so, we are confronting it to the proof file.

The proof file is, as the check file, a simple text file with OCL conditional expressions. The proof file embeds all the conditional expressions that the pattern must met in order to provide its properties and be proven. It acts as the postconditions of our integration. Whereas the check file only enumerates the preconditions necessary to have a correct casting. Our tool will again replace the formal parameters by the real ones and attempt to make the model pass the condition generated.

2.6 Artefacts consumption and production

Fig. 3 shows the four different phases of our process and their activities and where the different artefacts are consumed and produced. The phases, that we will explain in the next section, are Preparation, Elicitation, Consolidation and Adaptation.

3 The Integration Process

Our process is divided into four phases, each consisting of one or two activities. It aims to provide the correct-by-construction integration of an object-oriented design pattern into an application while offering a certain degree of liberty to the designer using it. The four phases are: Preparation, Elicitation, Consolidation and Adaptation. Preparation extracts from the pattern its role diagram, which focuses on the solution provided by the pattern, and the properties needed for checking. Elicitation builds a bridge between the application diagram and the pattern owing to the role diagram. Consolidation does the merging between them. Finally, adaptation is optional, and offers the possibility to make tailor-made integration by letting the user refines the new application obtained. We will now see in details every phase and explain each of them. In order to be able to validate the integration, we must have a formal specification of the pattern, i.e. its properties and constraints, modeled following a pattern modeling language.

For this paper, we choose the TERESA pattern modeling language and expressed its defining characteristics using the OCL language. In the Mediator pattern, we identified the following set of properties: (1) every colleague object communicates to a fellow object only by the means of a mediator object; (2) the mediator object must not be a colleague; (3) no association, direct or indirect, must exist between colleagues. Moreover, we have an implementation constraint: the concept of abstract class must be available.
3.1 Preparation

In this phase, there is only one activity named preparing. We verify the presence of the pattern, expressed in a pattern modeling language. We extract from this representation its check and its proof, both expressed in a formal language. The check are preconditions on the application or the casting to integrate rightfully. The proof is a formal description of the pattern properties and constraints, which can be asserted on its structure. The structural information of the pattern is also extracted, to perform the integration itself. To simplify this paper, we will be considering the structural information of both the pattern and the application as an UML class diagram, and we will reduce the application to its core structural information.

<table>
<thead>
<tr>
<th>Preparing Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: Pattern</td>
</tr>
<tr>
<td>Output: Pattern Class Diagram, Role Diagram, Proof File, Check File</td>
</tr>
</tbody>
</table>
The output of this phase are the pattern class diagram, the role diagram, the check file and the proof file. The role diagram will act as a blueprint to link the pattern and the application during the next phase. The check and proof files are used during our process to validate different steps of the integration.

The role diagram for the Mediator pattern is composed of two roles: one is the mediator role, and the other is the colleague role. There can be multiple colleagues but only one mediator. We link the Mediator role and the ConcreteMediator class and the Colleague role and all of the ConcreteColleague classes.

### 3.2 Elicitation

Fig. 4 gives an overview of SPEM [14] of the elicitation phase.

![Diagram](image)

**Fig. 4.** Elicitation overview.

**Binding** The first activity of elicitation is the binding. The designer links the roles of the pattern role diagram to the different classes of its application model. The output is a casting, linking classes to the role they play. Because of the freedom let to the designer to do its casting, an automatic checking of it will be done in the next activity.

<table>
<thead>
<tr>
<th>Binding Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
</tr>
<tr>
<td><strong>Output</strong></td>
</tr>
</tbody>
</table>

For the Mediator pattern, the designer binds the mediator and colleague roles to classes of his/her application model. The result of this binding, that we call a *casting*, is a set of pairs, role-class, indicating which role the classes of the application are going to play. If a role has no class attributed, a default class will be generated later.
Checking Using the check file obtained during the preparation, an automatically check is done of the casting. The check file is the collection of all the assertions relating to the different roles. In order to be allowed to play a given role, a class of the application must met all its related assertions. Each expression is evaluated on the model application. If all are true, the casting is correct, and we go the next phase. Else the designer must redo its binding. The binding activity can always be aborted and the process cancelled.

<table>
<thead>
<tr>
<th>Checking Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
</tr>
<tr>
<td>Output</td>
</tr>
</tbody>
</table>

The checking activity will automatically verify if these links expressed by the casting are correct. Starting from the constraints, we can check if the chosen classes met the requirements and if there is no incoherency in the casting itself. In the Mediator example, it will be checked that the classes chosen as colleagues don’t have already association defined between them.

3.3 Consolidation

Fig. 5 gives an overview in SPEM of the consolidation phase.

![Consolidation overview](image)

Adding It should be noted that both pattern and application are expressed into the same metamodel, an UML class diagram. Therefore the adding activity consists of simply merging the two diagrams, using the merge points as defined in the casting diagram. The merge points are the classes of the application playing a given role. Each role references a class of the pattern. A simple substitution is done and all associations and classes of the pattern are pulled out and put into the application model. The checked casting ensures the correctness of the construction. At this step, the new produced model of the application is now providing the properties of the pattern, but the designer can choose to go further and edit manually the model, it is the adaptation phase.

The consolidation phase realizes the integration itself by following the links created and checked during the previous phase. The adding activity merges the
two models into one, producing a resulting model which includes the properties of the Mediator pattern: the adding ensure that for a set of colleagues, all communications between them pass through a mediator.

<table>
<thead>
<tr>
<th>Adding Activity</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pattern Class Diagram, Application Class Diagram, Casting Diagram</td>
<td>New Application Class Diagram</td>
</tr>
</tbody>
</table>

3.4 Adaptation

Fig. 6 gives an overview in SPEM of the adaptation phase.

**Fig. 6.** Adaptation overview.

**Editing** In this activity, the designer is free to edit the model. However, this edition is controlled by the process. Each change is confronted to the proof file, extracted during the preparation phase of the pattern. The proof file is similar to the check file but contains all the proof of the pattern. All properties related to the impacts that the change made will be processed again, and if one fails, the change will be rejected. This way, at any step, the edited model is still validated. This incremental proof lasts as long as the designed is not satisfied. Once done, the process is terminated, and the integration is done.

<table>
<thead>
<tr>
<th>Editing Activity</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>New Application Class Diagram, Proof file</td>
<td>Modified New Application Class Diagram</td>
</tr>
</tbody>
</table>

For the Mediator example, the designer could merge one of his/her abstract classes with the AbstractColleague class without undermining the properties of the mediator communication process.
In the next section, we will see in more details the enforcement of our process on an traffic airway simulator.

4 The Mediator pattern example

We will now apply our process to an user story. A designer wants to create an application handling the control of a runway. He starts by defining its requirements, that every airliner should communicate through the control tower, in order to be sure that the control tower handles all communications and traffic in its area. The designer obtains the class diagram of Fig. 7 representing its application.

![Class Diagram](image)

**Fig. 7.** The application class diagram.

4.1 Preparation

Browsing through a pattern repository, he chooses the Mediator design pattern which will ensure that all communications of colleagues will pass through a mediator. Taking the Mediator pattern as described in TERESA, we obtain the following role diagram of Fig. 8.

![Role Diagram](image)

**Fig. 8.** The Mediator role diagram.
4.2 Elicitation

Colleagues and mediator are abstractions defined at the role diagram level, and the designer wants to specify that the two subclasses of Airliner will play the role of colleagues, and the control tower the one of mediator. To materialize these links, the designer draws the casting diagram of Fig. 9.

![Casting Diagram](image)

Fig. 9. Casting diagram: linking role and class diagrams.

Once the casting has been made, the designer tries to validated it through the checking activity. The checking activity will look at the assumptions extracted from the check file and verify each of them. For example, if the conditional expression "no associations between colleagues" has been defined, the casting will tell us which classes in the application are going to play the role of colleagues. Therefore it will be easy to check the class diagram in order to examine if such an association doesn’t exist. Once all the checks have been validated, the casting is then declared as checked and consolidation phase can be started.

An example of improper casting is to consider the existence of a link between PassengerAirliner and CargoAirliner in the design of the application, previously made before the process of integration. At checking, we are taking the check file, reflecting the conditions on each role, which allow the pattern to provide its properties. The check file created during the preparation phase uses the roles as its parameters, like Colleague. Now, following the casting diagram, we are substituting the chosen class CargoAirliner and PassengerAirliner for their role, and producing the request of Fig. 10. This request tries to verify if any link already connects the chosen classes for the colleague role. Finding the link, the assertion that no link exists between colleague fails and the casting is rejected.

4.3 Consolidation

After the checking of the casting, the consolidation phase merges both class diagrams to obtain the new application model of Fig. 11.

The designer can now accept this result and terminate the process, or choose to manually edit this diagram during the adaptation phase.
A proven pattern integration process Cs-12

context mmUML2_2_1_0::application::uml::Association

def : Car : mmUML2_2_1_0::uml::Class =
    mmUML2_2_1_0::application::uml::Class.
    allInstances->select(name='CargoAirliner?')->asSequence->first

def : Pas : mmUML2_2_1_0::uml::Class =
    mmUML2_2_1_0::application::uml::Class.
    allInstances->select(name='PassengerAirliner?')->asSequence->first

query : self cầmEnd.type=self::excludesAll('BagCar, Pas')

Fig. 10. Query to check that there is no communications between colleagues.

Fig. 11. New application model after adding.

4.4 Adaptation

The designer can edit the resulting model with his/her favorite tool. Each of his change will be tested by our tool against the proof file extracted of the pattern properties. If the properties still hold, the change will be accepted and commited. Else, the change will be dismissed. The designer can abandon the editing activity at any moment. The delivered model will be at the state of the last validated change if he chooses to do so.

For example, if the designer tries to create a new link between PassengerAirliner and CargoAirliner. This time, it's the property "all communications between colleagues must pass through a mediator" which is challenged. A new request will be made, extracted from the proof file, in order to check this. Finding a connection failing to be compliant, the change will be rejected and the class diagram restored to its previous state.

A modification could be to merge Airliner and Abstract Colleague. In this case, the change will be validated because all the properties still hold. In the end, our process deliver a new application model meeting the defined requirements.

5 Towards an integration toolchain

In order to able to experiment our process, we made a first tool to assist us, using Java. Fig. 12 shows the main idea of our tool. On the left, the pattern whose properties are needed, and on the right, the target application for integration. The pattern and the application are simple XMI files. In our experimentation,
we are using the Papyrus tool to produce our class diagram. We will be able later to open and parse them easily like any other XML file. It should be noted that any tools producing standard compliant XMI could be used too. Once both loaded, a click on a button will launch the Elicitation phase.

The binding activity will prompt a window to assign the different roles to their classes, as shown at the left in Fig. 13, to define the casting used, which will be displayed as shown in the window at the right. After that, the tool launches the checking activity. The requests extracted from the check file are then passed to Neptune [5] in order to check the casting. Neptune is an OCL interpreter developed at IRIT and supports all the major metamodels such as UML and Ecore and can be used as an internal library of an application or as a stand-alone application.

Fig. 12. The main screen of our tool.

Fig. 13. The window to assign roles to classes and to display the resulting casting.
If the casting is accepted, our tool goes up to the consolidation phase, where
the resulting application is produced as a new XMI file. The designer can decide
to stop the process there, or to edit further more the model during the adaptation
phase.

The adaptation phase is not automatic. The designer must edit the new
model resulting of the consolidation, which can be done with an simple text or
XML editor. Then, he must launch the verification manually inside our tool. It
will check the edited model against the proof file, which consist in OCL rules,
through Neptune and finally state if it is correct or not. If it is correct, the model
will be saved in order to offer to the designer to rollback at this previous step in
case of incorrectness of the model during a later step.

6 Related work

Since the work of Gamma et al. [2] (or GoF), design patterns have attracted much
interest. However, defined at first as textual information with UML diagrams,
they lacked a proper formal definition of their structure and proven properties.
We will make a brief tour of the attempts that have been made, and why our
solution is relevant compared to them.

Pattern modelization The first attempt to model pattern is given by the
GoF itself, where each pattern is described by UML diagrams. But there is
only natural text and some examples to link the diagrams together and explain
the integration. This is not enough for what we aimed for. Extending this way,
UMLAUT was proposed in [10] as an approach that aims to formally model de-
sign patterns by proposing extensions to the UML meta-model. They used OCL
language to describe structural and behavioral constraints. Mechanisms of asso-
ciation of these meta-level diagrams to their instances level (instances of design
patterns) are then defined. This allows to model design patterns accurately in
UML language.

By specifying design patterns as metamodels and defining a set of features
dealing with models themselves, RBML (Role-Based Meta-modeling Language)
proposed in [11] tries to bridge the gap between the pattern and its use. The
RBML formalism, based on UML, is able to capture precisely various design
perspectives of patterns such as static structure, interactions, and state-based
behavior. Each perspective used its own metamodel and language, in which the
pattern can be defined precisely. Then, a clear relationship is defined between
the pattern and its application: the application is viewed as a model which must
conform to the pattern. But still, the integration by itself is not clearly defined.

The LePUS [13] and DPML (Design Pattern Modeling Language) [8] lan-
guages are both using modeling and meta-modeling together, dealing also with
the issue of visualization. In [13] is presented a formal and visual language for
specifying design patterns called LePUS. It defines a pattern in an accurate and
complete form of formula in Z, with a graphical representation. The framework
promoted by LePUS is interesting but the degree of expressiveness proposed to
A proven pattern integration process C5-15

design a pattern is too restrictive. But here too, there is a lack of relationship between the pattern and its instantiation. By using the role diagram and the properties, we are able to express it inside our pattern definition. Therefore, our MDE approach delivers a complete solution, from the modelization of the pattern, to the last stage of its integration, by insuring at all stages that the pattern properties are respected.

**Pattern integration** With regard to the integration of patterns in software systems, the DPML allows the incorporation of patterns in UML class models. However, this kind of techniques does not allow to achieve the high degree of pattern structure flexibility which is required to reach our target, and offered by the pattern modeling language TERESA. But if DPML and the work of Bayley [12] were interested by pattern composition, it is not what we are aiming for. Our primary target is an application being developed in which we would like to integrate a pattern.

Recently, [9] explained how pattern integration can be achieved by using a library of precisely described and formally verified solutions. In concept, our modeling framework is similar to the one proposed in [9]. Nevertheless they used a rigid structure (a pattern is defined as a quadruplet) and consequently their approach is not usable to capture specific characteristics of patterns for several domains. Another attempt has been made in [4] which creates a metamodel for both the problem and the design pattern. Then, by using a mapping between the two models, it is able to create an integrated model using model transformations. Although we found similarities between this approach and ours, we want to go further than the transformation by defining a full process for a proven integration, and be able, inside this defined process, to let the user free to alter the automatic result, while always checking the correctness at the end.

Finally, to the best of our knowledge, none of the existing approaches is able to deal with the combination of patterns and proving their integration into an MDE refinement process of an application. Our solution is based on considering the integration as a transformation, based on the metamodel for patterns, and the metamodel of both the application model and resulting model.

7 Conclusion

As viewed, an integration-ready pattern consists in four artifacts: a class diagram, a role diagram, a check file and a proof file. These artefacts can be deduced from the language used for modeling the patterns. During our process, the casting diagram will be generated by the designer in order to link the role diagram to its application diagram.

The role diagram is the interface of integration. Besides the role diagram, the check file displays the mandatory constraints that the chosen classes must met, providing a solid guide in order to achieve a correct casting. Finally the proof file displays the entire proof of the properties of the pattern, which will
be checked in case of manual editing of the resulting application model after the consolidation phase.

The four phases of the process, preparation, elicitation, consolidation and adaptation offers a conceptual framework to the designer, in which at every step checks are made, but at same time offering to the designer enough freedom to adapt the pattern to his/her particular context.

Our process is not tailored for an unique pattern modeling language, as long as the pattern language deals with proven patterns. However, an adapter must be written for each, in order to be able to extract information from the pattern, like its properties and constraints. Later, the information extracted will be used to let our process ensures that the properties and constraints of the pattern are still be met at any time of our process.

We are beginning to experiment our toolchain and using it in different user stories, aiming to evaluate the relevance of the activities of the process and possible ways of amelioration. We could try to integrate the same pattern in different contexts in order to measure the flexibility of our process and the interactions with the designer. Later on, we aim to encompass behavioral object-oriented patterns.
References


2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994).


Annex

context mmUML2_2_1_0::uml::NamedElement

static def: colleaguesConcretes:
  Set(mmUML2_2_1_0::uml::Class) =
  mmUML2_2_1_0::uml::Class.allInstances
  ->select(c: mmUML2_2_1_0::uml::Class |
  c.generalization.general.name->includes('Colleague'))

static def: classesConcretes:
  Set(mmUML2_2_1_0::uml::Class) =
  mmUML2_2_1_0::uml::Class.allInstances
  ->reject(c: mmUML2_2_1_0::uml::Class |
  c.isAbstract)

static def: concreteMediator:
  mmUML2_2_1_0::uml::Class =
  mmUML2_2_1_0::uml::Class.allInstances
  ->select(c: mmUML2_2_1_0::uml::Class |
  c.name = 'Concrete Mediator')->asSequence->first

static def: mediators:
  Collection(mmUML2_2_1_0::uml::Class) =
  concreteMediator->union(concreteMediator.generalization.general)

static def: associationMediator:
  Set(mmUML2_2_1_0::uml::Association) =
  mmUML2_2_1_0::uml::Association.allInstances
  ->select(c: mmUML2_2_1_0::uml::Association |
  c.memberEnd.type->includes(self.concreteMediator))

static def: classesConnecteesSansMediator:
  Collection(mmUML2_2_1_0::uml::Association) =
  mmUML2_2_1_0::uml::Association.allInstances
  ->select(a: Association |
  a.navigableOwnedEnd.type->excludesAll(mediators))

Fig. 14. Query to check that all communications between colleagues pass through a mediator - Part 1.
context mmUML2_2_1_0::uml::Class

def : ClasseEnAssociationDeLaClasseSansMediator :
    Set(mmUML2_2_1_0::uml::Class) =
        classesConnecteesSansMediator
        ->select(a : mmUML2_2_1_0::uml::Association |
            a.memberEnd.type->includes(self)).navigableOwnedEnd.type->asSet

query : Tuple {nomDepart = self.name, succ =
    self->closure(self.ClasseEnAssociationDeLaClasseSansMediator).name}

query : colleaguesConcrets->includes(self) implies
    self->closure(self.ClasseEnAssociationDeLaClasseSansMediator)
    ->excluding(self)->excludesAll(colleaguesConcrets)

**Fig. 15.** Query to check that all communications between colleagues pass through a mediator - Part 2.
A Real-Time Design Pattern for Advanced Driver Assistance Systems

Hela MAROUANE, Achraf MAKNI and Rafik BOUAZIZ, MIRACL-ISIMS, University of Sfax
Claude DUVALLET and Bruno SADEG, LITIS, University of Le Havre

Advanced Driver Assistance Systems (ADAS) are hard real-time control systems in the automotive domain which use a lot of data reported from several embedded sensors. These data must be updated regularly to reflect the current environment state. Thus, these systems need to be managed by real-time database systems in order to store and manipulate real-time data efficiently. However, the design of ADAS is highly complex; it is difficult to model the time constraints related to both data and transactions. To tackle this problem, the design patterns present a reuse solution that improves the quality of the development process and reduces the complexity of ADAS design. In this paper, we focus on defining a real-time domain specific design pattern named ADAS-DA (ADAS Data Acquisition). This pattern describes system’s inputs/outputs, their behavior interrelationships, non functional requirements, its real-time aspects and temporal evolution. To make the pattern more flexible, we add some semantics to the UML concepts. Then, we propose a new UML-Profile which expresses the real-time elements of the pattern and its variability.

Categories and Subject Descriptors: I.5.2 [Pattern recognition]: Design Methodology—Pattern analysis

Additional Key Words and Phrases: Real-time, ADAS, Design patterns, UML-Profile, Data acquisition, variability

1. INTRODUCTION

The number of vehicles on the road has greatly increased in recent years. Consequently, the number of accidents grows. To alleviate the road traffic and reduce the risk of accidents, new technologies in vehicles have been appeared, called Advanced Driver Assistance Systems (ADAS), such as the Adaptive Cruise Control (ACC) and the Lane Departure Warning (LDW) systems. In fact, ADAS are intelligent systems that aim at avoiding the risk of accidents and improving road safety by assisting the drivers in their driving tasks.

An ADAS is a complex real-time (RT) embedded system which consists of three layers [Rezaei and Sabzevari 2009] [Amditis et al. 2010] [Amditis et al. 2008] (Figure 1):

(1) The perception layer, that includes a series of sensors (e.g., radar and vehicle sensors) and a sensor data fusion unit allowing the computation of appropriate sensors data to estimate a consistent state of a vehicle and its environment.
(2) The decision layer, that uses the data fusion unit outputs to analyse the current situation and to decide the appropriate actions to be transmitted to actuators.
(3) The action layer, that receives the actions from the decision layer, and either it delivers visual, acoustic and/or haptic warning information to the driver, or it provides automatic actions such as braking.

The design of ADAS must specify the functional and non functional requirements knowledge, the system’s inputs/outputs, and their behavioral interrelationships, in order to respond to the road safety needs. In addition, the design of ADAS becomes highly complex due to new technologies. Therefore, it is still a challenging problem to model ADAS by taking into account all their requirements. A way to design these systems may be to exploit reusable design elements that improve the quality of development process, like design patterns [Gamma et al. 1994].
In this paper, we are interested to define a RT domain specific design pattern that models the data acquisition of ADAS. In fact, our solution shows how to model the static view (e.g., objects, attributes and methods) and the dynamic view (e.g., lifelines and sequence of messages exchanged between the lifelines) using a new UML profile extending class and sequence diagrams. To be able to define this pattern, we study and model several ADAS systems in order to highlight their similarities and differences. Among these systems, we are interested, in this paper, in modeling the Adaptive Cruise Control (ACC) system and Lane Departure Warning (LDW) system.

2. EXAMPLES OF TWO ADAS

In this section, we focus in the design of the data acquisition of the two following ADAS systems: the Adaptive Cruise Control (ACC) system and the Lane Departure Warning (LDW) system, using the class diagram in order to highlight their similarities and differences. These systems are emerged on the market in modern cars (e.g., BMW, Audi and Mercedes-Benz) [Prestl et al. 2000] [Johansson et al. 2012].

2.1 Adaptive Cruise Control system

ACC system is an automotive application that reduces the risk of accidents and provides safety and comfort to drivers and vehicles by adapting the vehicle’s speed to the traffic environment. The goal of this system is to maintain safe distance between the ACC-vehicle and the forward vehicle. ACC system uses exteroceptive sensors (e.g., radar) that observe the external environment, and proprioceptive sensors (e.g., vehicle sensors) that observe the vehicle’s state. The radar or laser sensors are used to detect the presence of a lead vehicle and to measure its speed and the distance between ACC-vehicle and forward vehicle.

Vehicle sensors (e.g., accelerometer and wheels sensors) are used to measure sensor data [Ramamirtham et al. 2004] (e.g., vehicle speed, acceleration and throttle/brake positions). These data are periodically updated to maintain consistency with the current state of the environment. The controller reads these data and calculates the desired acceleration or deceleration to maintain the safe distance. Thus, these data constitute the derived data [Amirijoo et al. 2006].

We model the class diagram which represents the data acquisition of ACC system as shown in Figure...
2. This class diagram has been resulted from the study several documents provided by the automotive companies such as Citroën and BMW [Prestl et al. 2000]. This model represents the following classes:
(a) \textit{Sensor} class that contains the main properties of sensors incorporated in the controlled vehicle;
(b) \textit{ExteroceptiveSensor} and \textit{ProprioceptiveSensor} that represent subclasses of \textit{Sensor} generic class;
(c) \textit{Laser} and \textit{Radar} that represent subclasses of \textit{ExteroceptiveSensor} generic class;
(d) \textit{Accelerometer} and \textit{WheelSpeedSensor} are subclasses of \textit{ProprioceptiveSensor} class;
(e) \textit{ObservedElement} class that represents the physical elements which are observed by the sensors; it can be instantiated by the controlled vehicle or by a tracked vehicle;
(f) \textit{ControlledVehicle} and \textit{TrackedVehicle} classes that are associated with \textit{ProprioceptiveSensor} and \textit{ExteroceptiveSensor} classes, respectively; this association indicates that the proprioceptive and exteroceptive sensors observe the controlled vehicle and the tracked vehicles, respectively;
(g) \textit{Measurement} class that encapsulates the measurements related to the controlled vehicle or one of the tracked vehicles;
(h) \textit{SensorData} and \textit{DerivedData} that are subclasses of \textit{Measurement} class. \textit{SensorData} class concerns the data acquired from sensors. \textit{DerivedData} class stores the data calculated by the controller.

2.2 Lane Departure Warning system

LDW system provides lateral control by warning the driver when the vehicle is unintentionally drifting out of its lane. This system maintains the vehicle position by detecting lane markings using a video camera.

The data acquisition of LDW system consists of a video camera and vehicle sensors (vehicle speed sensor, steering wheel angle sensor and yaw rate sensor). The camera is used to observe the road. It acquires images, while computer vision algorithms extract in real-time the desired information such as lane width, curve radius and vehicle’s lateral deviation. The vehicle sensors are used to determine the vehicle’s state (e.g., vehicle’s speed and steering angle). Figure 3 illustrates the class diagram of the data acquisition functionality of LDW system. This diagram represents the following classes: \textit{Sensor}, \textit{ProprioceptiveSensor}, \textit{ExteroceptiveSensor}, \textit{SpeedSensor}, \textit{YawRateSensor}, \textit{SteeringWheelSensor}, \textit{CCDCamera}, \textit{ObservedElement}, \textit{ControlledVehicle}, \textit{Road}, \textit{Measurement}, \textit{SensorData} and \textit{DerivedData}.

2.3 Comparison of these systems

We note the similarities and the differences between ACC system and LDW system: the controlled vehicle, the measurements (sensor and derived data), the sensor and the observed elements constitute the common elements in the two systems (drawn with a highlight line in Figures 2 and 3). However, the categories of exteroceptive and proprioceptive sensors (e.g., radar and CCD camera) and the external tracked element (e.g., tracked vehicle and road) represent the variable elements which differ from a system to another. For example, CCD camera, is present in LDW system, but is not used in ACC system. Thus, it is a variable element; it differs from a system to another. It is represented in the pattern using the specialisation relationship of the \textit{ExteroceptiveSensor} class (Figure 3).

3. ADAS DATA ACQUISITION PATTERN

In this section, we have to define here a new RT domain specific design pattern, entitled ADAS Data Acquisition (ADAS-DA), to be able to model ADAS data acquisition from the environment using sensors that are implanted in a vehicle. The definition of the appropriate solution, in terms of static and dynamic views, is based on the identification of similarities and differences of ADAS systems [Amditis et al. 2008] [Tsung-Ying et al. 2005] [Biral et al. 2010]. The common elements (e.g., objects, attributes and operations) must be defined in a standard and unified way. These elements are fundamental and must be present in this solution. Since the variable elements differ from an application to another, they can be omitted in a particular pattern instance: these elements are optional in the pattern. In order to
Fig. 2. Data acquisition pattern for ACC system.

make the pattern more flexible and understandable, we add some semantics to the used concepts by
the definition of a new UML profile, named UML-RTDB2, that specifies the RT requirements of ADAS
and the variability of patterns (see appendix).

We describe the proposed pattern using the following elements: name, context, problem, forces, so-
lution, examples code and consequences.

3.1 Name
ADAS Data Acquisition

3.2 Context
Data acquisition of advanced driver assistance systems.

EuroPlop, Article C6, Publication date: July 2012.
3.3 Problem
How ADAS systems can manage the data acquisition and the real-time constraints related to a lot of data and transactions efficiently.

3.4 Forces
- Several sensors are implemented in the vehicle in order to collect periodically data about the vehicle and its surroundings. These data must be updated regularly to reflect the current environment state.
- ADAS systems usually require the use of multiple sensors which provide redundant or complementary information. These data are merged to reduce the measurements noise and to produce more reliable and accurate information.
- After the data have been collected and updated, it must be processed fast so that the system can react in time.
3.5 Known uses

- The system manages a lot of data and several real-time constraints.
- The system must continue to operate despite a sensor is deficient.
- The system needs to minimize and accelerate the data acquisition.

3.6 Solution

Use a real-time database (RTDB) to manage real-time data and transactions efficiently. In fact, RTDB allows managing effectively a lot of data by taking into account the real-time constraints related to both data and transactions (e.g., validity duration of data and deadline of transactions). ADAS systems use several sensors. For this reason, the number of data acquisition increases significantly so the acquisition becomes very time consuming. Hence, the storage in a RTDB allows to accelerate the acquisition and to minimize the data acquisition (one data access operation). When a sensor is faulty, the data will not be updated regularly. So, it affects the reliability of the systems although this property presents the most important characteristic of automotive systems. In this case, RTDB ensures the reliability of the system [Flodström and Strömberg 2011]. In fact, the erroneous value will be replaced by another estimated relative to the stored data so the system can continue to function correctly.

3.6.1 Static specification. Figure 4 presents the data acquisition pattern static view.

Participants:

a. Sensor: The sensors are divided into exteroceptive and proprioceptive sensors. These categories of sensors constitute the variants of Sensor generic class. These variants are expressed through the specialisation relationship. These sensors constitute common elements in all ADAS. Thus, they are fundamental elements in the class diagram. Exteroceptive sensors can be a laser, a GPS, a radar and so on. In the same way, Proprioceptive Sensors can be an accelerometer, a wheel speed sensor and so on. The exteroceptive sensor has the Measure_Data_TrackedElem() method to indicate that it observes the external vehicle's environment. A proprioceptive sensor observes the vehicle's state. It has the Measure_Data_ContVehicle() method which acquires the measures related to the vehicle.

b. ObservedElement: The observed elements are classified into: (i) controlled vehicle, which is observed by the proprioceptive sensors, and (ii) the tracked element (e.g., road, fixed obstacle, driver and so on) which is observed by the exteroceptive sensors.

c. Measurement: it is divided into sensor data or derived data. The sensor data and derived data represent fundamental elements in the class diagram. In addition, CurrentSensorData subclass is stereotyped <<sensor>> to show that the data is acquired from sensors and CurrentDerivedData subclass is stereotyped <<derived>> to express that the data is calculated by the controller using sensor data.

We use notes to define two OCL (Object Constraint Language) [OMG 2003] related to the Measurement class. The first constraint (Context Measurement inv: self.Validity.Duration ≥ self.Sensor.Periodicity) indicates that the measurement validity duration must exceed the periodicity of sensor acquisition which is responsible of acquiring this data in order to maintain freshness and validity of the database [Ramamritham et al. 2004]. In fact, a data is considered fresh if the difference between the current value and the stored value is less than or equal to Maximum Data Error (MDE) [Amirijoo et al. 2006] defined by the DBA. Otherwise, the considered attribute must be updated. The second constraint (Context Measurement inv: self.MDE ≥ self.Sensor.Precision) indicates that MDE must be greater than the precision of the sensor in order to have an accurate data which reflects the change of the external environment, and to minimize the number of update transactions.
d. SensorData, DerivedData: these measurements are RT data which are characterized by the following attributes: (i) value, that represents the last value captured by the corresponding update method, (ii) timestamp, which represents the time at which the attribute’s value is updated, (iii) va-
credibility duration, which is the time interval in which the data is considered valid and (iv) MDE, that represents the maximum error associated with the attribute's value. We propose to take into account the QoD (Quality of Data) in ADAS systems by associating the MDE property to the data. In order to keep the database consistent with the current state of the environment and in accordance with the MDE, sensor data must be periodically updated using UpdateValSD() method. Whereas, the derived data are computed from sensor data and updated by UpdateValDD() method.

e. FusionUnit: this unit allows to merge the acquired data (e.g., reduces the measurements noise, synchronizes the sensors and merges redundant or complementary information issued from multiple sensors) in order to produce a consistent estimation of the controlled driving situation.

3.6.2 Dynamic specification. The dynamic specification of ADAS-DA pattern is described using sequence diagram as shown in Figure 5. We are interested to model the interactions of the components of data acquisition of ADAS and the updating of the RT database. First, the SensorDataFusionUnit receives data from passive sensors through the GetValue() method or from active sensors by the SetValue() signal. Secondly, it merges the data in order to produce a consistent estimation of the controlled vehicle and an accurate description of the external vehicle environment. Then, the SensorDataFusionUnit updates periodically the value of the sensor data. The DerivedDataCalculator reads sensor data and calculates derived data which are updated sporadically. The UpdateValue() method is stereotyped <<periodic>> if it modifies a sensor data value, or <<sporadic>> if it sets a derived data value.

3.7 Examples of code

We propose this code java to define Sensor class.

```java
public class Sensor {
    private double lambda;
    /* The generator which be used */
    private Random generator;
    public Capteur(double lambda, Random generator){
        this.lambda = lambda;
        this.generator = generator;
    }
    public Random getRandomGenerator (){ return generator;}
    public double getLambda (){ return lambda;}
    public void setLambda (double lambda){
        this.lambda = lambda;
    }
    public void setRandomGenerator (Random generator){
        this.generator = generator;
    }
}
```

Similary, we define the Measurement class as shown in following code. This code define the characteristics of a measurement in the database (e.g., the type of the measure, unit, timestamp, validity and MDE).
public class Measurement {
    private String Measure_Type;
    private String Unit;
    private Date TimeStamp;
    private int Validity;
    private int MDE;
    public String getMeasure_Type() {
        return Measure_Type;
    }
    public void setMeasure_Type(String measure_Type) {
        Measure_Type = measure_Type;
    }
    public String getUnit() {
        return Unit;
    }
    public void setUnit(String unit) {
        Unit = unit;
    }
    public Date getTimeStamp() {
        return TimeStamp;
    }
}
3.8 Consequences
ADAS-DA pattern represents clearly the real-time constraints related to the data and transactions. Modelling these constraints increases the reliability and the performance of the developed system (e.g., reduce the system failure).

4. CONCLUSION
The main objective of our work, described in this paper, is to define a RT domain specific design pattern for the data acquisition of advanced driver assistance systems. Design patterns provide general solutions which capture expert knowledge and enable large scale reuse independently of the development platform. We have proposed ADAS-DA, a domain specific pattern which models the structural, behavioral and real-time aspects of ADAS. ADAS-DA allows designing all specificities tied to any ADAS. It shows the common elements (e.g., exteroceptive sensors, proprioceptive sensors, controlled vehicle, sensor data and derived data) and the variable elements (e.g., the different categories of exteroceptive and proprioceptive sensors and the different tracked external elements) in all ADAS in order to facilitate and to maximize ADAS reuse. These elements are identified by studying and modeling two ADAS: ACC and LDW systems. ADAS-DA facilitates the modeling of any ADAS; it will be easy for the designer to reuse this pattern by adapting it to the needs of a particular advanced driver assistance system. In order to express such a pattern, with a maximum of expressivity and flexibility, we defined UML-RTDB2, a UML-Profile that contains a set of stereotypes which express the timing constraints, the non functional properties and the variability of the pattern.
In future work, we will propose other domain specific patterns to model the controller and the actuators subsystems of ADAS as well as the reuse of the proposed static and dynamic views in order to validate these models.

Appendix
We propose a new UML profile, named UML-RTDB2, that provides a precise semantic of UML concepts. This profile supports real-time database requirements. It takes into consideration both temporal aspects of data and transactions.
UML-RTDB2 profile includes stereotypes to express the variability of design patterns and temporal aspects of ADAS. In fact, the variability of patterns is an important criterion to obtain a flexible pattern and to maximize pattern reuse. To specify the variability of patterns, we extend the class diagram with the following stereotypes [Claub 2001]:

```java
public void setTimeStamp(Date timeStamp) {
    TimeStamp = timeStamp;
}
public int getValidity() {
    return Validity;
}
public void setValidity(int validity) {
    Validity = validity;
}
public String getMDE() {
    return MDE;
}
```
• <<mandatory>>: this stereotype is applied to a class interface. It specifies the fundamental classes that must be instantiated when the model is applied to a specific application.

• <<optional>>: this stereotype is used to express optional features (e.g., classes, attributes and methods). The optional element can be omitted in a pattern instance.

• <<extensible>>: this stereotype indicates that the class in the model may be extended by adding new attributes and/or methods during pattern reuse. This stereotype has the following tagged values: extensibleAttribute and extensibleMethod which are boolean. They indicate when they have true value that the model can be extended by adding new attributes and new methods in a pattern instance.

In order to model the RT features of ADAS, we also use the following stereotypes [Idoudi et al. 2008]: (a) <<sensor>>, which is applied to a class interface and indicates that the measurement is a sensor data, (b) <<derived>>, which is applied to classes and is used to express derived data which are calculated from sensor data, (c) <<periodic>> and <<sporadic>>, which are applied to express periodic and sporadic methods, respectively. The <<periodic>> stereotype is characterized by a deadline and a period. The <<sporadic>> stereotype is characterized by a deadline and a triggered time.

We use the following stereotypes [OMG 2007] to specify the non functional aspects: (a) <<nfp>>, that declares non functional requirements, and (b) <<nfptype>>, that extends the DataType metaclass. It is used to specify NFP values such as "NFP\_Duration" and "NFP\_Frequency". In addition, we express real-time constraints with OCL.

REFERENCES


OMG. 2003. Uml 2.0 ocl specification.


The path to patterns - introducing the path metaphor

CHRISTIAN KOHLS

This text aims at learning more about the nature of patterns by examining a very simple and intuitive example of a pattern: a path as a solution to reach a goal. A path is obviously bound to its environment and this happens to be a good starting point to emphasize why the context of a pattern is so important. By visual illustration one will find that it is the environment with its embedded forces that shapes the possible solutions. On a map this relation can literally be seen. Paths and their representations on maps are very well known concepts. This allows the use of common sense to understand properties of patterns that are otherwise hard to grasp. We will discuss and illustrate the context, problem and forces, solution and consequences sections for the path metaphor. Moreover, we will use the metaphor to reflect about pattern qualities, such as their level of abstraction, the granularity and composition of smaller patterns, wholeness and their empirical status.

Categories and Subject Descriptors: D.2.10 [Software Engineering]: Design – Methodologies
General Terms: Design, Human Factors
Additional Key Words and Phrases: Design Patterns, Quality, Abstraction, Composition, Software theory

Reference Format:
Kohls, C. 2012. The Path to Patterns - Introducing the path metaphor.

1. INTRODUCTION

I have found it helpful to use proven paths in a territory to explain how patterns capture, describe and explain good practices. A path is an example of a solution that leads to a specified goal and it is a simple concept that is very well known to everyone. This simplicity enables us to show how the territory and the overall situation define which paths are potential solutions and which properties each path has.

Pattern skeptics very often do not understand what is innovative about the pattern format. Best practices have been documented for a long time and scientific discovery is all about finding invariants – patterns – in nature. This text should show where patterns are different and which values the pattern approach offers.

Pattern beginners who are willing to share their knowledge by writing patterns will see that it takes more than just knowing a proven path to describe a reusable solution. The path metaphor helps to teach the core concepts to interested novices.

Pattern experts can use the propositions and visualizations to discuss their conceptions about patterns. Both agreement and disagreement will give new insights to how we see patterns.

Let us begin with Christopher Alexander's definition of patterns in A Pattern Language: „Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.“ (Alexander, p. X, 1977)

There are two parts of this definition that are essential to patterns:

1. a pattern is the solution to a problem in an environment
2. the solution structure can be used a million times over and over again without ever doing it the same way twice

Both statements, simple as they may appear, are rich in their meaning. Let us therefore consider a familiar environment, a hillside landscape, to explore the concept of patterns. In such an environment, if we intend to hike from one place to another, we may encounter the problem that we cannot directly move in a straight line to the destination because we cannot cross canyons or pass over steep walls. Each trail that guides us to the destination is therefore a solution to our problem: getting us from where we are to our desired destination.
The next sections will discuss the basic components of patterns, i.e. the context, problem, forces, solution, and consequences, pattern languages and the relations between those concepts. We will use a physical path in a landscape to illustrate each of the concepts. The path is a metaphor for a structural configuration that fits into an environment. In addition we will introduce qualities of patterns, such as the level of abstraction, composition of encapsulated parts, wholeness and the empirical status of patterns.

2. UNDERSTANDING CONTEXT

The key to describe the context is to ask when-questions. When can this pattern be applied? When should it not be applied? Who can use the pattern? In which situations can the pattern be used? Which other patterns have led to this context?

A context is the environment or the situation in which we find ourselves in. In the simple example of a hike, the context is the physical environment of the landscape as well as your current situation (where you are, how well trained you are, what your hiking skills are, which tools you have at hand etc.) and your goal or intent. Besides reaching your destination, your intention may be to find ways that are not too tiring or to find ways that are physically challenging. You may prefer a path along scenic views or cultural attractions and make this a requirement for the solution. This setting is the base for the forces: “the unitary description of the context is […] also a description of the required form” (Alexander, 1964, p.21). The problem statement and the forces of a context are so important that they are usually discussed in separate sections. But as the forces arise in a context they are really part of it.

The context depends to some extent on your personal intent and preferences while at the same time the environment – the hills, canyons, slopes, lakes, cabins - is out of your hands. Very often the context refers to other patterns. That is because a pattern has to fit to these other patterns as much as a path has to connect to other paths at intersections. Moreover, the current context or situation is the result of previous steps and decisions. The context “describes the space in which the pattern can be embedded. It often includes references to patterns at a higher level of granularity that have been considered before” (Schüümmer & Lukosch, 2007).

At first glance, the context of that landscape seems to be fixed and static. However, the context can vary. One can hike at different times of the year; there will be different animals that cross the path, and there may be different groups of people who take the same trail. Hence, there is a certain degree of variability in
The power of a pattern is that it is open enough to be adapted to specific needs of an actual situation. While the core matter of a general context – in our example the shape of the landscape - remains invariant, we must not forget that there is an infinite number of ways in which the “same” context will change dynamically.

If the differences of a context are significant then another pattern might provide a better solution. The form of a solution is only adequate and of value if it still fits the context. For example, the very same form is no longer a solution if an essential part of the context (such as our starting position) changes.

If the context changes (for example we start at a different position or we are in a different environment) the solution changes.

The core problem that we cannot pass all areas remains but the specific forces change.

3. UNDERSTANDING PROBLEM AND FORCES

The key to describe the problem and forces is to ask why-questions. Why do we have to follow this specific path? Why do we have to go there? Why do we need this specific form? Why can’t we do another thing instead? Each single force gives another answer to such why-questions. A force explains the cause for a specific design decision by giving the “because” to the “why”.

Explaining the reasons for the solution form helps us to understand the pattern rather than blindly following an advice. This understanding of a pattern is essential for judging whether it fits to the problem at hand and to adapt it to the specifics of the situation.

A change in the context can make new paths possible (e.g. a bridge over a canyon) or eliminate paths that have worked in the past (e.g. a rock slide blocks the way). The same solution form may not fit any longer. Another context sets new constraints, boundaries, introduces new requirements and offers different opportunities. For example, the equipment and training of a hiker is part of the context and implies different fields of forces. An untrained hiker might be forced to choose a path that is not too steep. The right equipment might offer new opportunities (e.g. climb a steep wall) but also introduce new limits. Constraints, boundaries, requirements and opportunities are different types of forces.

Forces can support each other or conflict if they imply different approaches: “Other prose styles emphasize the conflict between different forces, presenting first one assumption or desired outcome, and then contrasting it with another that is in conflict, often introduced with a ‘but’ or ‘however’” (Buschmann,
Henney, & Schmidt, 2007, p. 99). Such conflicts are problems that need to be resolved. The problem and forces of a given context are usually discussed in separate sections because they are important to understand why a pattern is shaped the way it is.

If forces are the reason to build a solution in a specific way that works, we can consider forces as the cause that leads to the pattern. For example, what is the cause for the problem that we cannot go straight to our destination? Originally, it was the rock face that blocks our way. It forces us to take another way. That particular force is the original problem, the one that is problematic in the situation we are in. That force is in conflict with our desire (or “force”) to reach the target directly. If we are looking for a way that brings us to our destination we have to account for all the forces that influence our chosen path in the given context. There are some forces that cause the original problem and there are more forces that shape the path of the solution: “we can express the problem as a conflict among forces which really do occur within the stated context, and cannot normally be resolved within that context” (Alexander, 1979). Thus, the problem section names the main forces – the ones that motivate us to act – and the forces section discusses the interplay of all forces in the given context. This should not be limited to negative forces. A water place or a historic site could be positive forces that shape the way.

4. UNDERSTANDING SOLUTIONS

The key to describe the solution is to ask what-, how- and who-questions. What is the general structure of the solutions? How can I generate the solution structure? How do the parts relate and interact? Who is participating in the solution? What are variations of the solution?

A solution is one known way that takes into account all the forces that matter in a specific context and balances them to a satisfying extent: “We see, in summary, that every pattern we define must be formulated in the form of a rule which establishes a relationship between a context, a system of forces which arises in that context, and a configuration which allows these forces to resolve themselves in that context. It has the following generic form: Context -> System of forces -> Configuration” (Alexander, 1979, p. 253). The form of the solution is a path that has proven in the past to actually lead to the intended goal and takes the forces appropriately into account. It does not mean that it is the only path that does exist nor does it claim to be the best path. There might be paths that even fit better to the context but that have not been found yet.
The term “pattern” suggests recurrent solution forms. And that is what happens if people use a path: no two hikes along a path will be identical. We will use detours, walk crisscross and unfold the hike every time in a new way while still following the same way. That refers to Alexander’s second element of the definition of patterns. It is a solution which occurs over and over … without ever doing it the same way twice.

Fig. 4. Variation of the same path.

While there is variance, the general structure is preserved. A pattern defines a “coherent yet infinite design space” rather than single realizations of a solution (Buschmann, Henney, & Schmidt, 2007, p. 76) Every concrete journey along a path has the same structural quality even if each instance unfolds differently. This structural symmetry lets us find abstracted representation of an actual path. However, the range of possible variation is different for the various parts of a path. At some points the space of a path may be broad, allowing many different steps to move along the same path. At other points, the path may be narrowed or obstacles only allow specific moves. Those points are critical and need more attention to make the path a success. It is therefore that we must find a medium level of abstraction. Like a map, a pattern description has to provide enough information to actually follow the solution path. It will leave out irrelevant details and concentrate on hallmarks that help designers to find their way. But it must not be too abstract. Otherwise we are losing the form and the specific structure that guides the designer through the territory: “A good solution has enough detail so the designer knows what to do, but it is general enough to address a broad context” (Coplien, 1996).

5. UNDERSTANDING CONSEQUENCES

The key questions of the consequences section is: Where are we now and what should we do next? What are the benefits, costs, drawbacks, tradeoffs and liabilities of the solution? Which forces have been resolved or weakened? Which forces remain or have been newly introduced? What needs to be done next and which other patterns could support this solution?

The context shown on the map allows more than one solution path. Each path has different advantages, disadvantages, liabilities, and suggests different next steps: “The consequences are the results and trade-offs of applying the pattern” (Gamma, Helm, Johnson & Vlissides, 1995, p. 3). For example, the path that leads to the historic site certainly has the advantage of visiting that place. However, it requires more time. The two alternative paths compete with each other.
In the process of searching a decision between different solutions, the pros and cons are weighed against each other; the consequences are compared. "In practice it is rare for solutions to be ‘right’ or ‘wrong’. Instead, a proposed solution will typically do some things well and other things less well." (Koenig, 1998) The overall situation will determine the actual path taken. If the hiker has a special interest in historic sites this interest "forces" him to take the longer way because he “has” to see it. In this context the path that runs along the historic site is best for that hiker. However, in a slightly different context there might be a time constraint to reach the goal. In that case the hiker might be forced to take the shorter path because reaching the goal in time (e.g. before sunset) might be more important than seeing the historic site. The better we understand the consequences the better we can find an appropriate decision. If we know the advantages (values) and disadvantages (costs) of a path we can better decide which one fits the problem at hand.

Which consequences matter depends on the specific context or situation. One consequence might be that it takes a long time to follow a particular path. But if you have enough time and energy this consequence does not matter. It is still a consequence but one that you can accept.

Small changes in the context (time constraints, personal interests) have impact on the path we consider best. There are also situations in which two paths seem to be equally valuable. For example, if one path leads to a historic site and the other to a scenic view, and both are equally interesting for the hiker then the decisions might depend on mood or previous experience (if the hiker has seen already five historic sites he might choose a scenic view). Both are contextual properties as well. Unless one flips a coin each decision is based on the fact that one path is favored over the other.
Once we have made our decision for one path we have to live with the resulting context and its liabilities. For example, we may encounter further obstacles on our way. The path may lead to our goal but we have to take care of some challenges. These challenges are local problems that occur in the context of the chosen solution. It is common to describe recurrent local problems and their solutions as separate patterns that can be used in the context of the current pattern. That is one reason why solutions frequently refer to other patterns of finer granularity.

Fig. 7. Obstacles to care about.

The choice of a path also has consequences for the next steps. Once it is decided where to go there is a resulting context that suggests which sub-paths can be followed and which preceding and succeeding paths are available. At any time, the steps that can be followed next depend on previous decisions. The start and end point of a path have to connect to surrounding paths. Intersections with other paths are as important as the path description itself. A path that does not properly connect to other paths is a dead end. That is the reason why many pattern descriptions refer not only to patterns on a more detailed level but also to patterns of the same level of abstraction. These other patterns support or surround the described pattern. An abstract pattern (such as crossing the mountains) can be made of small patterns (such as crossing a narrow passage, a canyon, a bridge etc). The abstract pattern can refer to the more specific patterns, i.e. as a consequence of crossing the mountain one has to cross a narrow passage, a canyon and a bridge. On the more specific level, choosing one alternative (such as crossing a bridge) could directly imply another pattern (if you cross the canyon you should also use the bridge). If we consider solutions as interventions to given situations, then the consequences describe the shift of forces after applying the solution form. In the same way as a magnet will shift the forces in a magnetic field, a solution will shift the field of forces into a favourable way. The most conflicting forces should be resolved. But there might be the need for additional patterns to address unresolved forces or new conflicts (side effects).

The consequences describe the resulting context: “The Resulting Context is the wrap-up of the pattern. It tells us:
- which forces were resolved
- which new problems may arise because of this pattern
- what related patterns may come next

Each pattern is designed to transform a system in one context to a new context. The Resulting Context of one pattern is input to the patterns that follow. Contexts tie related patterns together into a pattern language” (Coplien, 1996, p.11).
6. UNDERSTANDING ABSTRACTION

The key to abstraction is to ask whether a reader can easily grasp the pattern as a whole. Can you still see the core structure? Can you see its gestalt? Does the pattern provide alternative choice? Is the pattern description instructive enough to implement it?

Patterns generalize over multiple cases and capture the essence of similar structures at a mid-level of abstraction. Too detailed solution descriptions are hard to transfer to new situations. Unless stated explicitly is not clear which structural qualities are required by the specific context and which parts of the solution structure are more general and qualify for generalization. Too abstract pattern descriptions are hard to grasp, difficult to understand and less instructive. As patterns are only general for the identified context, they are usually not universal (Lea, 1994). A pattern becomes more universal when it can be used in different contexts and its solution offers different options for its implementation. Yet if we let too many parts of the pattern’s structure unspecified we are losing its gestalt. That means we can no longer “see” its form as a self-contained whole. Think about abstract drawings one can find in comics. They leave out many details of the actual structure of a face. However, if essential parts are missing – such as eyes or a mouth – it becomes difficult to see a face. Likewise, if we describe a pattern without its essential parts we will get no picture of potential solution implementations.

Abstractions omit details of a particular solution and thereby leave them unspecified. As a result we have a choice between alternatives. These alternatives are the reason why patterns can be adapted for different situations. This choice has an important consequence. The designer needs to know which alternatives there are, how to use them and how to select the correct one. Too specific solution descriptions do no longer offer such alternatives because the design decisions are made. They are more instructive by providing the appropriate choice for the specific context. However, by making a particular choice they no longer show us at which points in the solution structure there are alternatives. They become less flexible for adoptions.

There are many ways of abstraction but let us consider three basic kinds:
- abstraction by isolation, i.e. omitting features from a structure that are not relevant;
- abstraction by generalization, i.e. omitting irrelevant variations of structural features;
- abstraction to emergent qualities, i.e. omitting the structure of a phenomena in favor for its emergent effect.

**Abstraction by isolation** is the most intuitive one. Features can be omitted if they are not relevant or it is clear how the appropriate option is selected, i.e. how the best alternative is chosen. Whether a bridge is painted black or green does not affect its purpose of crossing a river or canyon. Therefore, we can abstract from its color. That a bridge for pedestrians must be wide enough for at least one person to pass seems to be an implicit property or at least common sense. We can abstract the width of the bridge because we assume that any reasonable person comes up with a good measure. Another choice to be made is the building material. Should the designer use wood, metal or stone? This choice seems relevant and we can only omit it in a solution description if we assume that a designer knows how to select the right material. For a stone bridge we can reason about the particular requirements to build such a construction. A specific pattern description for stone bridges would be more instructive and more informative than a description of bridges in general. However, we limit the choice and applicable situations. If the material would not have been specified, the designer would still have the choice. In this case the designer would need to know which materials are available and which tradeoffs come with each. Hence, prior knowledge is required in order to unfold the abstract solution structure back to a particular instance of the pattern.

**Abstraction by generalization** means that we abstract over a range of possible values that are all similar while keeping the structural relations between elements intact. Let us consider the effects of this abstraction for our path. As the hiker traverses along the route it does not matter whether s/he walks on the very left, in the middle, or on the very right side of the path. The hiker can choose where to walk exactly. In order to offer this choice we need to abstract from the similar positions between the left and the right bounds of the path and consider each position as equal. The relative steps, however, remain fairly invariant. If a path leads south-west, the relations between two steps remain intact regardless of whether the hiker walks at the left or right bound of the path. While the abstract representation does not specify
whether a hiker keeps at the left or right boundary, the actual hike might be influenced by local conditions. An obstacle might force the hiker to keep at the left boundary. This does not invalidate the abstract representation which allows both options. Rather, by allowing both options, we ensure that local conditions can be taken into account.

If we want to abstract from actual positions of various hikes we can differentiate between different approaches. First, instead of using the actual positions, we could abstract to a range of similar positions. In this case we keep the exact boundaries and define a space of potential positions (abstraction 1 in figure 8). However, very often we do not have enough empirical data to know the exact boundaries are – people have a tendency to keep within a safe area instead of testing the exact limit. Therefore, instead of pretending to know the exact boundaries, we can use “prototypical” positions as abstract place holders representing a range of values. What is a good “prototypical” value? It could be the most frequent taken position (e.g., hikers may prefer to stay at the left side of the path) or the average position between the boundaries. The best choices, however, are values that resemble the invariant relations between each position. As long as we stay within the range of potential values we can select model positions which make it easy to grasp the essential directions of the path (abstraction 2 in figure 8).

If we use such a modeled abstraction of the path, there is another benefit. We can concentrate on the positions where we find a change of direction and omit (abstract from) positions where the direction remains invariant. We reduce our focus on the positions where the symmetry of the path is broken and the structural pattern emerges. These are the milestones, the essential elements that define the structure of the path (abstraction 3 in figure 8). Most maps are drawn from a bird’s eye perspective. However, informal sketches on a napkin (“demotic maps”) often focus on such turning points: milestones, landmarks, intersections. Likewise, if we describe a pattern it is important to focus on those turning points and key elements. They define the essential structure and must be clearly visible. If we would abstract from these key elements we would lose the gestalt of the pattern.

We can see that happens if we take away one of the turning point positions of our path. The new abstraction does not resemble the essential form any more. It is a different pattern on a more abstract level. While abstraction 3 (figure 8) still contains all the information we need to get to the next milestone, abstraction 4 (figure 8) does not reveal the actual structure of the last path section. Such an abstraction could still be meaningful. For example, there might be alternative paths that could be used to reach the final position. However, it is important to note that this abstraction step changed the pattern! It is a more abstract pattern and parts of its implementation will look very different depending on the actual sub path chosen.

What happened is an abstraction to emergent qualities. The same qualities can emerge from different configurations (Holland, 2008). You can take two different sub paths that look very different but have the same quality of reaching the goal. Let as assume the last part of the path is about crossing a river. One could cross the river “using a bridge” (sub path A) or by “jumping over stones” on a narrow section of the river (sub path B). While both ways lead to the final position by “crossing the river” they have very different forms – two different gestalts. While the more abstract notion “crossing the river” offers more choice it is important to realize that it is less instructive. It requires the hiker to figure out or know in advance that there is a bridge or alternatively a narrow section with stones.
The difference between abstraction by generalization and abstraction to emergent properties is the configuration of the micro-structure. On the macro-structure (the more abstract level) each of the sub-paths has the same effect of reaching the next milestone. However, the micro-structure (the less abstract level) may or may not have many variations. If we use abstraction by generalization, very similar micro-structures are abstracted to one representative macro-structure. In this case the macro-structure resembles the micro-structure and we can easily derive the actual structure by knowing the macro-structure. The important information for its implementation (“cross the river using a bridge”) is available. An abstraction to emergent qualities does not reveal such information because it refers to very different micro-structures (“cross the river” can unfold to very dissimilar ways).

If we would zoom to a very detailed level of abstraction of our path we would find that there are actually differences in the micro-structure as well due to the criss-crossing of hiking steps.

![Fig. 9. Abstraction to emergent qualities: Different wholes are considered to have the same quality on a higher level of abstraction](image)

When we are writing patterns it is important to understand that abstractions by generalization are structure-preserving. In spite of the abstraction, the required information is still there. Abstraction to emergent properties means that parts of a solution can be implemented in very different ways and no information is provided about the actual structure. This requires that a user of such an abstract pattern knows how to unfold it into a more specific one. A designer needs to know how to get from a more abstract pattern to a more concrete pattern. In order to do so s/he must know the patterns of lower abstraction levels and how to compose those to larger patterns.

7. UNDERSTANDING COMPOSITION

The key question is: how can I divide and conquer a solution into proper parts? Which elements do belong to one part? Where can I encapsulate a part? How do parts fit together? How can I provide choice for alternative parts?

A pattern can be composed of smaller patterns. By dividing a large structure into smaller parts we reduce the complexity, provide choice between alternatives, and can represent each part on different levels of abstraction.

![Fig. 10. A division into sub-paths.](image)

The paths in figure 10 divide the way to a goal into three sub-sections. One can concentrate on each subsection nearly independent which makes it much easier to handle the overall complexity. As a result of the division, one can separately study the details of paths A, B, and C. Moreover, subsections B and C are alternatives to reach the same goal.
As we have identified each of the nearly-independent sub-sections we can refer to them in a pattern on a more abstract level. Instead of describing A, B and C in detail, the abstract pattern would simply state “First take path A and then choose either path B or C”. References to other patterns are highlighted in pattern description by using CAPITALIZED LETTERS.

The abstract “Drive from Hamburg to Munich” can be expressed by “Drive to Hannover, Göttingen, Kassel, Fulda, Würzburg, Nürnberg, Ingolstadt, and then to Munich”. Each single milestone could then be further specified by a more particular route, e.g. “To get to Hannover, drive to Seevetal, Fallingbostel, and then to Hannover.” A benefit of the abstract representation is that we can replace each of the abstractions by different specializations to compose the whole. If the specializations are properly encapsulated, then we can independently choose an option. For example, the abstract “Drive to Hannover” can also be replaced by the alternative route “Drive to Lüneburg, Uelzen, Celle, and then to Hannover”.

However, the difficult question is what parts of a larger whole are meaningful and self-contained sub-wholes? To get answer, let us consider alternative partitions of path A.

Fig. 11. Parts that are not self-contained.

Part D as an arbitrary sub-section of B. While part D is a part of B it is not whole because it does not connect to other paths and it depends on B alone. D is not meaningful without the other parts of B because to get to D and to succeed from D one has to use the remaining parts of B. To reach another intersection, a point of choice, one has to follow B completely.

If we consider part E we find a partition that is even worse. Not only does E not connect properly to other paths it also misses the fact that there is a choice between alternatives. E starts and ends nowhere and is not open to alternative choices that take different situations into account.

Path Y, which is the composition of A-B, is a reasonable path. As it leads from one intersection point to another it is a whole solution. However, it does not take into account that there are alternative sub-sections. It might be more appropriate to describe A and B separately. Even if C is not described (or discovered) yet, a division of Y into A and B shows that there are alternatives for B. If A and B are described as separate solutions, then we can provide a very brief and abstract description of Y because it can refer to A and B.

\[ Y: \quad \begin{array}{c} \text{A} \\ \Rightarrow \\ \text{B} \end{array} \]

Fig. 12. Abstract representation by reference to other patterns.

While the description of Y uses abstractions that do not reveal the structure of either A or B, one can unfold the structure by referring to the more specific descriptions of A and B.

A more abstract pattern, let us call it pattern X, could integrate the choice between the two alternatives of B and C. Although the gestalt of X changes (it could either be A-B or A-C), it provides enough information...
how it can be composed by referring to required and alternative paths. The actual implementation leads to
two more specialized patterns, let us call them Y and Z.

\[
X: \quad A \rightarrow B/C \quad \quad Y: \quad A \rightarrow B \quad \quad Z: \quad A \rightarrow C
\]

Fig. 13. Refinement of patterns

This example shows different relations between patterns. The patterns X, Y and Z can be composed of
other patterns A, B, and C. The actual structure of X is not fully specified but it can be refined into patterns
Y or Z. The two different patterns B and C can be alternatives for the same goal. Pattern languages
highlight such relations and explicate that a “pattern uses another pattern, a pattern refines another
pattern, or a pattern conflicts with another pattern” (Noble, 1998).

We can see that it is always possible to split up a larger pattern into smaller patterns at points where
alternative implementations are possible. Each alternative implementation is encapsulated and can be
tackled nearly independent. Whenever there is a choice between independent alternatives then each
alternative is a self-contained pattern. However, it is not always pragmatic to describe all alternatives as
separate patterns for this could lead to an explosion of pattern documents.

Patterns of larger granularity, such as path X and its two specializations Y and Z, can either describe each
of its sub-parts (e.g. describe A in detail) or refer to a separate description of it (e.g. refer to a description of
A). Repeating the description of recurrent solution parts in several patterns leads to redundancy and
obscures alternatives. On the other hand, if we describe very small solution parts as separate patterns, the
descriptions become less accessible (a reader has to jump between descriptions), more abstract (as the
description only refer to actual solutions) and we end up with many patterns hat need to be described.

As a rule of thumb, whether we integrate variations into one pattern description or cover each variation in
another pattern description depends on the ratio between invariance and variation:

- If the variant parts of a solution are roughly the size as the invariant parts, then it is probably a
good idea to describe the invariant part and the variants as separate patterns.
- If two solutions differ in most aspects and share only some invariant parts, it is best to describe
both solutions separately and redundantly use the invariant parts in both descriptions.
- If large parts of a solution are similar and only small parts may vary, then we should write only one
pattern and include the variants in that pattern document.

The choice between alternatives is a sign of openness. Openness is important in order to take the specifics
of an actual situation into account. Openness refers to the agility and adaptability of patterns as well as
their compatibility with other patterns (Lea, 1994). In order to adapt to the specifics of a situation, a pattern
is more open if it allows a variety of sub-patterns and connects to various other patterns, i.e. it can be used in the context of alternative patterns and its resulting context offers choice between patterns.

A path that does not offer alternatives or connects to multiple different sub-paths means we have to “stick” to a particular track. If an obstacle blocks that a road we cannot continue. If there are, however, alternative sub-paths we can select an alternative. If a sub-path connects to a high number of succeeding paths we are more liberate to choose how to continue. Such alternatives also mean that the path can be used in more situations. For example a track might be suitable for both experienced and un-experienced hikers if at critical points (such as steep passages) there is an alternative route. The openness of a path may vary at different points. A narrow section close to a sheer offers less space for variation. A meadow allows running criss-cross.

Another degree of openness is achieved if a pattern does not only allow the combination of patterns in an additive way (i.e., combination of parts) but in a multiplicative way (i.e., merging and overlapping of parts). A path that can be used by DRIVING A CAR, BIKING or WALKING is more open. The physical space of a path could also overlap with patterns such as STANDPIPE, FIREPLACE, CAMPING ZONE and SCENIC VIEW.

8. UNDERSTANDING WHOLENESS

The key questions are: Do we achieve the quality without a name? Is the pattern whole? Is the solution alive? Do all parts fit? Is every force resolved? Are the parts in harmony? Can we use the pattern to generate infinite solutions, each one adapted to the specific needs of a situation?

We have discussed that in the process of abstraction and division into components the golden rule is to keep self-contained wholes. When we abstract from real forms, the abstract parts still need to resemble the gestalt. When we divide a larger whole into parts, each part needs to be self-contained, i.e. provide a whole solution part. Moreover, when we compose a larger whole out of parts, each part has to fit to all other parts to make it whole.

Alexander’s quest for wholeness can be traced back to his early work Notes on the synthesis of form: “[the design process] concentrates on structure, the process is able to make a coherent and therefore new whole out of incoherent pieces” (Alexander, 1964, p.110). It is discussed widely in The Timeless Way of Building, where the interplays of patterns are means to form whole structures: “Once we have understood how to discover individual patterns which are alive, we may then make a language for ourselves for any building task we face. The structure of the language is created by the network of connections among individual patterns: and the language lives, or not, as a totality to the degree these patterns form a whole” (Alexander, 1979, p. XII).

Wholeness unfolds to the most particular level, from abstract to the real. A map only abstracts the territory. The territory itself is the whole with all its components: the weather conditions, the skills of the hikers, wild animals, and all other factors. Every detail of the context, every pattern, every solution part plays together
as a whole. At no point of the path should there be an obstacle that cannot be managed. The path has to fit to its environment at all times. A single misfit – such as a blocked passage or a deep river without a bridge – will invalidate the whole path. The path also has to connect properly to other paths in order to be open for the composition of larger paths. Otherwise a path would be a dead end.

When we implement a solution, each of the abstract parts needs to be replaced by concrete counter-parts. While we can choose between alternative specializations for each part, each has to be in harmony with all of the other particularizations in order to keep the harmony between parts. Every solution unfolds step-by-step in a way of piecemeal growth and local adoptions: “A generative pattern is a means of letting the problem resolve itself over time, just as a flower unfolds from its seed” (Coplien, 1996). At any time in the process of design the current situation or context needs to be evaluated to progress. Each step into one direction changes the overall situation. The results cannot be fully foreseen and must be re-evaluated in order to select the next step.

When a path is followed its structure is created by performing a sequence of steps; it is a process in which the thing – the path itself – is generated. A particular hike along a path unfolds in the process of walking. It cannot be planned fully in advance. If a stone is on the road, a hiker needs to react and the particular course is adapted accordingly. If the hiker spots a beautiful flower or butterfly he stops at unpredictable times. A map of paths does not prescribe the exact sequence of steps but rather offers directions and constraints. The process is volatile; each step is a transformation of the current situation. At any time the current context needs to be re-evaluated to account the local forces. A path description that is generative tells you how to proceed in the sequence of circumstances: When you see the big oak tree, you should keep right until you find a place in the river that is not very deep, so that you can cross it barefoot.

A design pattern does not only capture good solutions in order to understand them; it also captures the ways of generating the solutions (Coplien, 1998). A pattern is at the same time a thing and process, “a rule which describes what you have to do to generate the entity which it defines” (Alexander, 1979, p. 182).

9. UNDERSTANDING THE REALITY OF PATTERNS

The key to more confidence for your patterns is to provide information about your pattern mining process. How many known uses are there for your pattern? Did you implement the solutions yourself or read about them?

As there is a huge variation in the contexts, situations and problems we are facing as designers, we can never be sure that a pattern always works in the way we expect. However, there are patterns we can put more confidence into because they have been tested many times whereas other patterns are just ideas or concept proposals. Many people in the pattern community disapprove of raw concepts or ideas as patterns because such pattern descriptions are often not declared explicitly as untested hypotheses.

To understand this, think about asking someone about the directions through a dark forest. Which path description would you prefer:

- A path that is assumed to work, a path that worked once, several times or a million times?
- A path that has been used by one person or by many different groups?
- A path that has been used only in summer or at all times of the year under various conditions?
- A path that is described by a person who has heard of it, read about it, or actually taken it by her/himself?

There are differences in the number of times a solution has been tested, the variations of contexts, and the research methods applied. To evaluate a pattern, it is crucial to learn in which environment a pattern was mined, which attitudes and beliefs the author held, and how many different views and artifacts have been examined to come up with the pattern. To assume that one designed solution will work in other cases and for other people depends on how stable the past cases have been and how similar the new cases and cultural settings are. Therefore, it is important to provide information about the pattern mining process used to find the patterns.

D-1: The path to patterns - 14
Hence, pattern papers should not only include the pattern descriptions but also document:
- The mining ground (variation of cases)
- The mining methods (validity of cases, confidence, objectivity)
- The known uses that induced a pattern (sample size and variation of cases)
- The degree of corroboration (successful application of patterns to similar cases as evidence)

Very often the mining process involves a mixture of methods and follows an iterative process of identifying, refining, criticising and cross-linking patterns (Retalis, Georgiakakis & Dimitriadis, 2006). Inductive and deductive methods often go hand in hand (Baggetun, Rusman & Poggi, 2004). First categories are inductively derived from the observed data. Thereafter deductive analysis can help the articulation as patterns (Schadewitz & Jachna, 2007). Deduction is needed to reason about the patterns – the recurrent patterns found by induction do not explain the relationships. It is important to point out that the generalization from single cases, the reasoning about causalities of working forms and the judgement between relevant and irrelevant features are hypothetical. The pattern itself is of theoretical nature. Or, as Brad Appleton (2000) puts it: “A pattern is where theory and practice meet to reinforce and complement one another, by showing that the structure it describes is useful, useable, and used”. As such, patterns are theories of good practices. Typical methods for the inductive inference of patterns can be found in qualitative research (Hughes, Rodden, Rouncefield, & Viller, 2000) and comprise techniques such as observation and analysis, retrospectives, expert interviews, and focus groups.

ACKNOWLEDGEMENTS
There is not enough praise for those who have helped forming this paper. Early versions of the first part have been presented at PLoP 2010 in Reno, Nevada, and the second part was discussed at PLoP 2011 in Portland, Oregon. The merge of both parts (the description fields and the qualities of patterns) has been resulted in this paper which was discussed at EuroPLoP 2012 in Irsee, Bavaria. Thanks to all participants of those workshops. My first two shepherds Rebecca Wirfs-Brock and Jenny Quillien both provided wonderful feedback of greatest value. The present version has been shepherded by Andi Fießer who has done an amazing job. It was a great pleasure to work with Andi and he has pointed out very important flaws in my discussion. He has inspired most of the “intersections” examples which are so important to illustrate that patterns build on each other and have to connect and fit.

REFERENCES
A Pattern Language for Teaching in a Foreign Language  
- Part 1

Christian Köppe and Mariëlle Nijsten

HU University of Applied Sciences Utrecht,  
Postbus 182, 3500 AD Utrecht, Netherlands  
{christian.koppe,marielle.nijsten}@hu.nl  
http://www.hu.nl

Abstract. Mastering foreign languages increasingly becomes a required skill in many working fields. This often is incorporated in curricula by integrating content and language learning. This paper describes some difficulties of this integration in order to raise awareness about them and offers some patterns which support teachers who begin with integrating content and language. The patterns are: Input Selection, Lucky Language Clover, Metatalk, and Language Role Model.

1 Introduction

The set of patterns introduced in this paper is aimed at lecturers who occasionally want to teach a class in a foreign language instead of their mother tongue. The patterns are meant to raise awareness of the difficulties and problems with integrating content and language learning and to support these instructors in preparing for these classes in such a way that a foreign language will not be a barrier to students’ understanding of the course content. This paper is part of a larger project for developing a pattern language for teaching in a foreign language. To date this pattern language consists of an introduction to the topic [10], this paper, and a second part which covers enhancing foreign language proficiency through integrated learning [9].

2 Foreign language teaching patterns

Subject teachers in higher technical education who occasionally need to switch to a foreign language as a teaching medium are often not aware of any language barriers. Most teachers are proficient in a single domain, either the foreign language or the content of a course. Without specific language pedagogy training, it is hard to be pedagogically aware of both domains — language and content. The students attending their classes often have different educational backgrounds and their proficiency in foreign languages will vary greatly. When the course is finished, a crucial question for lecturers is: should students fail their tests, is this due to lack of understanding of the subject content or lack of understanding of the second language used? The main focus of this work is therefore to help instructors teach their course in a foreign language without the risk of students falling behind due to low language proficiency. Yet, as attending courses in a foreign language also is an opportunity to further improve students’ mastery of this language, the second focus of this work is to identify ways of integrating the use of this language in such a way that language fluency increases while keeping the focus on the course content. A useful format for providing practical specific instructions for teachers is the so-called 'pattern' format, which originates from the architectural domain. Patterns are tools for documenting solutions and problems in such a way that users can fairly easily apply them to similar problems in their own context. The patterns in this paper aim to create awareness and help teachers design the right course for
each audience on two levels: content as well as language without needing any formal language pedagogy training. It is the foreign language teaching pattern format that is of practical use to technical subject teachers rather than their content (which is based on existing literature and personal experience).

2.1 Methods for integrating language and content

Teaching a course in a foreign language can serve a number of goals. Besides structural bilingual education, teachers in higher education may be asked to occasionally teach in a foreign language. Due to internationalization programs, teachers will need to teach in a second language understood by students from various countries — often, but not always, in English. In competence based educational systems, some subjects are taught in foreign languages to promote the use and acquisition of a foreign language in a relevant context. Such courses serve a double goal: conveying content and improving foreign language proficiency. This is a form of integrated learning, requiring course integration of language and content.

This integrative approach is supported by the communicative language teaching approach on foreign language pedagogy, in which the emphasis is put on learning foreign languages through meaningful communication and meaningful tasks. The role of the teacher is to provide class room activities for such communication processes and to monitor students’ progress, promoting peer assessment and self reflection rather than correcting mistakes. Teaching materials often include real-life documents and audiovisual means used by native speakers.

Various methods are available for integrating language and content in courses that provide useful support for teachers, referred to as Content Based Instruction (CBI) and Content and Language Integrated Learning (CLIL). However, the use of these methods requires a fairly large amount of teacher training in language pedagogy. Their target group is not a subject-specific teacher who occasionally teaches a class in a foreign language.

2.2 Preparing for foreign language instruction

When teachers in higher technical education are asked to teach a course in a foreign language, they often do not realize it involves more than just switching from one language to another. Their usual preparation for classes, however, must be adapted for teaching in a foreign language, to make sure students’ class performance will not be influenced by lack of language development. So the teacher’s preparation (his or her formal “talk”, as well as jokes and class room instructions), the instruction activities, the materials, and the assessment must all be adapted [2]. Furthermore, students’ foreign language level must be assessed before the course’ start, to adjust the course to the appropriate foreign language level. The language proficiency in the foreign language of both teachers and students may vary greatly. Teaching a course without taking these differences into account could have a negative impact on students’ understanding of the course content and be counter-productive. For occasional use, teachers need hands-on instructions, fit for their teaching context that can be applied without formal language pedagogy training. Such instructions can be provided by so-called patterns.

3 The Patterns

The patterns use a version of the Alexandrian pattern format, as described in [1]. The first part of each pattern is a short description of the context, followed by three diamonds. In the second part, the problem (in bold) and the forces are described, followed by another three diamonds. The third part offers the solution (again in bold), the (empirical) background, consequences of the pattern application — which are part of the resulting context — and
a discussion of possible implementations. In the final part of each pattern, shown in *italics*, we present some known applications.

These patterns were mainly mined in existing literature and experience reports. Therefore they often lack known uses and do not follow the *rule of three*. However, we intend to include additional known uses after these patterns have been applied and adjusted by a wider range of users, eventually publishing them as a comprehensive pattern language. They may be applicable outside the field of higher technical education; yet our current experience, selection and use of these patterns has been restricted to this field.

In the next sections we present the following patterns: **Input Selection**, **Lucky Language Clover**, **Metatalk**, and **Language Role Model**. After that the short versions of the other patterns — the patlets — are presented, which are published in [10,9].
INPUT SELECTION

Most courses make use of material — literature, websites, tutorials etc. — which covers the content of the course. You have identified both the Content-Obligatory Language and most parts of the Content-Compatible Language and you know the Language Status Quo of the students’ language levels. You now want to start to look for the material.

Available material often differs in both language levels and comprehensibility, and can be too difficult or too easy for students. Both cases will lead to problems during the course.

*Complexity.* Long sentences and words, academic words, complex concepts and a dense writing style requiring analysis, makes texts hard to understand.

*Text cues.* Cues such as headings, lists, signal words, and visuals help students understand course texts.

*Different Writing Styles.* Each author has a specific writing style, and as a result, some authors’ texts are much easier to read than others. Getting used to different writing styles takes getting used to.

*Different book audience.* Most of the textbooks available which cover the content of a course and are written in the foreign language are made for people using this language and not for people learning this language. These books therefore do not include language didactics, which might be necessary or helpful if this book is used as input.

Therefore: select comprehensible course input that explains the subject matter in a way that matches students’ language levels and interests.

Krashen looks upon comprehensible input as the primary motivator of language development, immersing students in meaningful input, without any explicit teaching of grammar [11]. This could be realized by e.g. incorporating newspaper or blog articles about the subject being taught.

According to Carrell, teachers should focus on the readers’ background instead of on the text [3]. Students need sufficient knowledge of text content as well as text structure and grammar. They may fail to understand texts due to lack of text cues or schemata, or culturally specific schemata. She suggests narrow reading, i.e. limiting the number of authors to one. She also advocates students previewing texts, which may include presenting difficult terms and expressions used in these texts [3].

According to Dale et al., as a rule of thumb, no more than 5 of the words on a page should be unfamiliar [5]. This is not easy to realize and should be seen more as a recommendation than as a fixed rule. However, readability instruments can be found online, helping instructors determine the level of difficulty of texts. Another way of using this rule is to have target students read a text and mark all unfamiliar words.

A consequence of applying this pattern is a longer preparation time, as the material has to be checked on appropriateness for both content and language.

---

1. [http://www.online-utility.org/english/readability_test_and_improve.jsp](http://www.online-utility.org/english/readability_test_and_improve.jsp)
Christian Köppe uses for a course on Patterns & Frameworks different kinds of literature. As the book by Gamma et al. [6] is a quite difficult reading for undergraduate students, the material was complemented with links to websites which describe the design patterns in a shorter and more comprehensible way. But in the later phase of the course the students had to use the Design Patterns book, but were better prepared for it as they understood the Content-Obligatory Language and the Content-Compatible Language.

Christian Decker from the HAW University of Applied Sciences in Hamburg provides via his twitter feed links to English articles containing interesting news related to a course’s topic. These articles are supporting the course material and complementary. Students are encouraged to read them, and because they often contain interesting news, students are for a large part motivated to read this extra material and therefore are exposed to a wider range of content and language input.
LUCKY LANGUAGE CLOVER

Also known as: The Four Skills

You are thinking about the tasks you want to include in the course design and want to ensure that they also cover the CONTENT-OBLIGATORY LANGUAGE and the CONTENT-COMPATIBLE LANGUAGE aspects appropriately.

Exposing the students to language comprehension only — reading and listening — is not sufficient for creating a lasting effect in learning the foreign language. They might be able to understand content input, but unable to produce content output in the foreign language.

Usage motivation. Even if a foreign language as medium of content instruction is used, some students will stick to their mother tongue and therefore will not improve their foreign language production skills. This is especially the case if they miss intrinsic motivation or if some students in the class are giving answers much more often than others.

Potential Hubris. Many students think that they master a foreign language quite well because they can read and understand the foreign language. These students often fail when it comes to speaking and writing.

Therefore: Promote reading and listening, and let students write and speak in the foreign language as well. Include all four types of linguistic competences in your course design.

Learning a language requires mastery of all types of linguistic competences: reading, listening, writing, and speaking — the four leaves of the LUCKY LANGUAGE CLOVER. This is referred to as exposure to input or comprehension (i.e., reading and writing) and so-called pushed output or production (i.e., writing and speaking). But language input does not always lead to language intake. It is by actively using the language input in stimulating assignments that help students grasp its actual meaning, the input is actually stored in students’ long term memory. So just giving lectures in the foreign language and requiring the students to read literature in the foreign language is not sufficient. Courses must allow for students to write and speak in the foreign language as well, as this promotes learning content and language at the same time [7]. According to Mehisto et al., a special focus should be put on speaking [14]. Producing output requires students to use their passive knowledge of the language to make themselves understood. Thus, their mastery of this language is enhanced [18].

Exercises, assignments, and didactics should therefore take all four leaves of the LUCKY LANGUAGE CLOVER into account. This could be achieved through the use of a variety of pedagogical patterns, e.g. PREFER WRITING, PEER FEEDBACK [16], and many others. Swain suggests a collaborative form of writing, as this implies the need for talking about the content [17] and therefore also promotes the use of METATALK.

Implementing LUCKY LANGUAGE CLOVER requires a careful selection of exercises, which are also aligned with the course material and should therefore be taken into account during the INPUT SELECTION. Coonan showed that GROUPS WORK [16] leads to a much higher oral output than a classical teacher-led lesson [4]. If LUCKY LANGUAGE CLOVER will be applied in teacher-led lessons, then an extended focus should be put on the questions asked
during the lesson, e.g. by applying Carefully Crafted Questions [12], which obliges
the students to elaborate their responses more richly [4].

Another important aspect is that not all students make use of the opportunity to speak,
most often we see a small group of students which answers most of the questions. The teacher
has to ensure that the oral participation is spread over all students equally if possible in
order to increase the language learning effect for the whole group. The patterns Groups
Work in combination with Shotgun Seminar [16] can help hereby. This also requires an
atmosphere where students dare to speak, another important aspect the teacher has to take
care of. It helps to explain in the beginning of a course that the most important aspect is to
use the language and to make oneself understood by the others, not a correct usage of the
language. This will lower the participation barrier.

But even in groups work do not all students automatically make use of the opportunity
to speak [4]. They are pedagogical patterns which help to increase the participation of all
students, like Think Pair Share [12] or Student Design Sprint [16].

Another question which needs to be addressed when applying this pattern is whether
the use of the native language is permitted at all and if so, under which circumstances.
Mehisto et al. suggest that especially when the students are exposed for the first time to
content being taught in a foreign language they also should be allowed to use their na-
tive language when necessary [14]. This should only be seen as bridging technique and the
students should always be encouraged to use the foreign language as much as possible. It
has to be clear that using the language at all is more important than using it 100% correctly.

The students in a course on Model Driven Development at the Hogeschool Utrecht had
to work on a longer lasting assignment which included a Model-to-Text transformation im-
plemented in a tool new to the students. After the first week all student groups (mostly 2
students) had to prepare and give a presentation about one of the problems they encountered
during the first week of the assignment. This included therefore writing (the content of the
presentation) and speaking (discussing the content and giving the presentation).

An earlier version of another course at the Hogeschool Utrecht on Patterns & Frameworks
which was given in English was based on classical lectures, exposing the students to reading
and listening only. In a newer version of this course the Lucky Language Clover was
implemented by having the students regularly give presentations on different topics, e.g. as
part of the implementation of Discover Your Own Pattern [8]. They were also given an
assignment which included answering a few questions about two articles in English. A survey
taken at the end of the course showed that their ability and self-consciousness regarding
speaking and writing improved remarkably.
METATALK

Students understand the content and are using the foreign language, making use of all four leaves of the Lucky Language Clover, but the language competences of the students still vary.

Students are not aware of their foreign language shortcomings and keep using incorrect language constructs and terms.

Hands-on activities. It seems common sense that hands-on activities are of benefit for students when learning new information and language expressions [13]. But this is rather based on general pedagogical principles. But the pure fact of hands-on activity does not necessarily stimulate language learning.

No self-reflection. Students often just apply the foreign language, but do usually not reflect on their language use.

Insufficient feedback. As teacher there is not enough time to correct all occurring incorrect language uses of students, either because the number of students is too large or the teacher is not present when students use the language. The students are therefore missing sufficient feedback needed for correction and improvement of their language use.

Therefore: Stimulate foreign language learning by including exercises or other appropriate course parts which require a collaborative reflection on language usage.

Westhoff [18] explains that language learners often do not apply grammatical rules they have learned in grammar classes, even though they understand them. It appears that in language learning, people in fact apply grammar rules they deducted from input they were exposed to. They have all kinds of assumptions and hidden knowledge in the back of their minds, which become clear the moment they try to express themselves in a foreign language. So even students who passed their grammar tests and did their vocabulary exercises, will not immediately use this knowledge in actual communication. However, when talking or composing a joint text, they will become aware of their lack of knowledge, by themselves or as pointed out by others. When given frequent corrective feedback, either by peers or by teachers, students’ accuracy, correct use of expressions and grammar rules will improve [18].

The usage of language to indicate an awareness about their own, or their interlocutor’s, use of language is called Metatalk [17]. Metatalk helps in acquiring Content-Compatible Language, as students discuss different ways of saying something and therefore broaden their vocabulary and means of expression. This helps students in making use of foreign language learning strategies.

It is important that metatalk is encouraged in contexts in which learners are engaged in making meaning. One way of implementing this pattern could be through a dictogloss task [17], where the teacher reads a short article — or some other text from the course material, for example the description of a pattern — to the students, let them write down familiar words and phrases and afterwards have them reconstruct the article based on their shared resources. These can then be compared with the original text. Another possibility is a jigsaw story construction task: give some pictures in unsorted order, let students sort them, and write the story down.
If one student in the group is really good, chances are that all the others are following this student, omitting discussions on language usage. So you have to be aware of this when forming groups of students which are to work together. Encourage all members to participate in applying Metatalk.

This pattern makes use of Lucky Language Clover, as discussions are mostly done orally and the results are manifested in written form. These written results could also be used for the Language Monitor.

Metatalk can also be stimulated in a Peer Feedback [16] situation, as also suggested by De Graaf et al. [7]. The artifacts feedback is given on should be made using the foreign language.

In a course on Model Driven Development, Christian Köppe let students document a problem that students experienced during the implementation of their assignments. These assignments were done in pairs. The students had to prepare a joint presentation of the problem. The requirements for this presentation were an accurate description in proper English of the whole problem including context and other relevant information, as well as possible solutions they had already tried. The preparation also required the students to not only discuss what their problem actually was, but also how to describe it in correct English. This process promoted Metatalk in combination with the Lucky Language Clover and led to good presentations.
LANGUAGE ROLE MODEL

You are asked to give a course in a foreign language.

Learning is also imitating, but imitating incorrect language usage of a teacher will affect the students’ learning of the language negatively.

*Mother Tongue.* Not all teachers who give a course in a foreign language are native speakers of this language. Their own language skills might be limited.

*Qualification.* Teaching content in a foreign language requires the ability of doing so. Evenly important is the ability to select and apply the appropriate instructional options.

*Linguistic Confidence.* The combination of teaching a specific content while using a foreign language correctly can be overwhelming, as it requires a greater repertoire of instructional options as fallback if observations indicate that the used option does not work for the students [15].

*Facilitation.* The teacher should have the facilities needed for teaching content in a foreign language. These facilities include material, time for preparation (including Input Selection and general course design), and, if necessary, additionally available language courses.

*Curriculum objectives.* If the language abilities of a teacher are sufficient for giving a specific course in a foreign language depends on the defined curriculum objectives regarding the foreign language.

*Content-Focused.* Swain observed that in cases when considerable teaching of content occurred, no or less attention was paid to the accuracy of the target language use [17].

Therefore: make careful language preparations to ensure that you can instruct students using a foreign language in a correct way for all related language parts. Use the language during the course always as correct as possible.

As teacher you should be a LANGUAGE ROLE MODEL for the students and use the CONTENT-OBLIGATORY LANGUAGE, the CONTENT-COMPATIBLE LANGUAGE and the general language as correct as possible.

The process of implementing this pattern consists of three steps: (1) assess your level in the foreign language (free online resources are available, such as the Dialang test\(^2\)) and compare it with the level of the curriculum objectives, (2) carefully prepare your classes to ensure that you as teacher are able to give the course using the foreign language in an appropriate way and (3) use your language consciously during the course while taking the different language aspects into account.

Proper language assessment will help you decide whether you are the right person to teach in a foreign language. Your level in the foreign language should clearly exceed the required level as defined in the language objectives of the curriculum.

People just acquiring specific skills are often unsure if they also can apply them sufficiently, even if the qualification to do so is sufficient. If this is the case it might help to

\(^2\) see [http://www.lancs.ac.uk/researchenterprise/dialang/about](http://www.lancs.ac.uk/researchenterprise/dialang/about), last visited on 13 June 2012
rehearse parts of your classes in front of colleagues and ask for feedback. You could also videotape your classes and review them, to detect points of improvement.

Christian Köppe used this pattern for a course on Patterns & Frameworks. As he’d given the course in English before it was ensured that his language skills were appropriate. During the course he made use of Content-Obligatory Language and Content-Compatible Language in general and also at specific Commented Actions.
Patlets of the other patterns

LANGUAGE STATUS QUO

Your assume that students are at a sufficient level of general foreign language abilities, i.e. they have knowledge of basic common vocabulary and grammar and can use the language. You now want to start teaching a course in this foreign language, with a foreign language as a medium of instruction so as to improve the use of this language in a professional setting.

Without knowing the actual level of foreign language abilities of the students it is likely that the language parts of the course design are either to difficult for the students which hinders them in grasping the content or are to simple for them which means that their language understanding probably does not improve.

Therefore: Get to know the language level of all students at the start of a course to obtain a realistic overview for your specific professional and educational goals. Use appropriate tests that include both general language capabilities and context specific abilities, such as class room language, formal academic language, and core professional activities in your field. This is the basis for an adequate language integration in the course design.

CONTENT-OBLIGATORY LANGUAGE

The content of a course is mostly focused on one domain, which often has specific terminology used in this domain. If students have a low general language proficiency, the chances of them failing to understand the real meaning of this terminology increases [15].

Some parts of the foreign language are so closely related to the content of a course that mastering them is crucial to students in order to achieve the course objectives.

Therefore: Define the content-obligatory language before and during course design. Expose the students to this language continuously in different ways with an emphasis during the beginning of the course. Let this language repeatedly come back during the whole course to improve acquisition and understanding of this language.

CONTENT-COMPATIBLE LANGUAGE

You identified the CONTENT-OBLIGATORY LANGUAGE and included in the course design opportunities so that the students can master it. However, most domains contain more elements of a specific language: synonyms, proverbs, expressions, phrases, metaphors, etc..
Only mastering the obligatory language of a course’s content might be sufficient to fulfill the course’s requirements, but it limits the students in their expressiveness and does not improve the overall quality of students’ language skills.

Therefore: Identify the language constructs and expressions of the course domain which are additional to the obligatory elements. Include opportunities for learning these in your course design and course execution.

**COMMENTED ACTION**

Also known as: Think Aloud Protocol, Show and Tell.

Often in courses or lectures you show or demonstrate some content-specific activities. You are aware that you are a Language Role Model and have identified the Content-Obligatory Language and the Content-Compatible Language of your course. Students do not yet know these language constructs or were just introduced to them.

Therefore: Do not only show or demonstrate complex abilities but give a spoken description of the steps you are taking. Use the earlier identified language terms when you show their meaning.

**LANGUAGE MONITOR**

Also known as: Formative Assessment.

You have designed a course with a focus on both content and language, identified the Content-Obligatory Language and the Content-Compatible Language and used this for Input Selection. You have chosen specific learning activities to stimulate Metatalk, support language learning with Commented Actions and are aware that you are a Language Role Model. You now want to assess whether your learning activities have had the expected result: an improvement of the students’ foreign language skills.

Therefore: Implement regular assessments on the language skills of the students to determine whether they grasped the content and whether their language skills have improved, and use these outcomes to intermittently adapt your course.
Acknowledgements

We want to thank Tim Wellhausen, our EuroPLoP 2012 shepherd. His comments helped in making the patterns more whole. Also the feedback of all members of the writers’ workshop on Education at the EuroPLoP 2012 conference is highly appreciated, therefore we want to thank all of them.

References

5. L Dale, W Van der Es, and R Tanner. CLIL skills. 2010.
Assessment Patterns

A pattern language for assessing students’ work in computer science courses

Uwe van Heesch*,
Richard van den Ham,
Pieter van den Hombergh
Fontys University of Applied Sciences
Venlo, The Netherlands
uwe@vanheesch.net, {r.vandenham, p.vandenhombergh}@fontys.nl

January 5, 2013

Abstract

This paper presents a set of interrelated educational patterns for assessing students’ performance in practice-oriented computer science classes. The patterns were mined in different software engineering courses at the Fontys University of Applied Science in Venlo, the Netherlands, where all authors are involved as lecturers.

The patterns in this pattern language can be used to solve problems related to assessing and grading practical students’ work, with reasonable effort for lecturers and maximum educational value for the students.

1 Introduction

Applied computer science (CS) courses should prepare students for a smooth transition into the computer science industry. Companies are looking for people with broad theoretical basic knowledge on the one hand. On the other hand, the ability to apply this knowledge in business relevant practical situations becomes more and more important. The industry appreciates students who can be used in software projects right away, without having to go through multiple months of trainings.

Computer science curricula have to be adjusted to serve these needs; practical work should be a substantial part of every CS curriculum. Nobody has ever learned programming or designing systems just by reading books or attending lectures. Students must get their hands dirty.

Properly assessing students’ practical skills, however, is not a trivial task. Computer science lecturers should strive for measuring and grading the competence level of each individual student, taking into account a mix of theoretical knowledge, skills and a professional attitude. Properly measuring practical skills and a professional attitude is particularly difficult and can hardly be achieved using traditional written exams. Instead, individual practical work assignments and group projects seem to be much more appropriate for training and measuring these aspects of a student’s competence level. Unfortunately,
prosperously assessing these activities is difficult and time consuming. Trade-offs have to be made between the lecturer’s grading effort, the quality of the assessment, and the educational value for the students.

A very important role of an educator is to inspire and to motivate students. Offering a practice-oriented curriculum and study environment, in our experience, boosts the students’ motivation. A fair grading system is another important factor of influence. Students are enticed to excel and to get the most out of themselves, if they are rewarded for doing their work properly and for performing better than other students. On the other hand, students who do not commit themselves to the improvement of their practical skills should not be able to pass the course.

Grading is an important task of lecturers, but their main focus should be on teaching, coaching and giving feedback. The majority of time should be spent on these aspects, rather than on the grading process. Students should receive frequent feedback to get an indication of how well they are performing and how they can improve.

Within our university of applied science, a number of practical approaches are applied to create an authentic and motivating learning environment with a strong practical component. We try to optimally prepare students for their careers as IT professionals. Group and project work is encouraged and widely applied in our course modules, but the students’ performance is assessed and graded individually, as far as possible.

In this paper, we present a pattern language for assessing students’ work in practical computer science courses. The patterns were mined in several courses in our university and have proven to work for several years. The language is intended for all lecturers and educators in the computer science domain, regardless of the educational level of the students. It can be particularly helpful for lecturers who would like to increase the proportion of practical work in their course, but don’t know how to measure students’ performance appropriately. Furthermore, it is useful for lecturers who already teach courses in a practice-oriented way, but who feel that their current grading approach is either not fair or not efficient enough.

The pattern language is comprised of the following patterns:

**Mini assessment:** A short individual assessment is done very often to make sure that students receive frequent feedback and commit themselves to the practical work assignments.
**Random assessment:** A subset of all students in a course is chosen for an assessment at random to reduce the assessment effort for the lecturer while keeping the students motivated.

**Collective feedback:** During a collective feedback session, lecturers give feedback to a larger group of students. Additionally, the session can be used to briefly explain practical assignment solutions. Students can get questions clarified and discuss their individual solutions.

**Performance assessment:** Students are assessed using a realistic working environment to ensure fair and authentic grading.

**Peer assessment:** Students in a project group grade each other regarding different criteria, serving the lecturer as input and/or verification for his/her own grading system.

**Student company:** Assessment of student performance in a real world environment by working on projects for external customers or working in companies autonomously run by students.

### 2 The Pattern Mine

The patterns presented in this paper were mined in the software engineering study program at the Fontys University of Applied Sciences in Venlo, the Netherlands. Universities of applied sciences, in the Netherlands, have a strong focus on practical applicability of the course contents. As a consequence, the vast majority of modules has a practical course component in addition to classical lectures. Additionally, a great part of the study’s workload is spent on projects, in which students work in groups of 3-6 persons to solve a complex case using and combining skills gained from different lecturing modules.

The patterns were mined in several lecturing modules, each of which being taught by a single lecturer who supervises between 25 and 30 students. In many cases, a second lecturer is assigned to the practical part of the modules.

### 3 The Pattern Language

In this section, we will present each of the patterns in the pattern language using a specific pattern template, which explicitly comprises the following elements:

**Context:** The context, in which the pattern can be applied.

**Problem:** The problem, which the pattern addresses including the forces that need to be considered when solving the problem.

**Solution:** The solution, as promoted by the pattern.

**Consequences:** A number of consequences that might ensue if the pattern is applied.

**Examples:** Real world situations, in which the pattern was applied.
Related patterns: Related patterns from the pattern language presented in this paper and from other pattern authors.

Each pattern is presented in a dedicated subsection. Cross links to other patterns are emphasized using capitalization (e.g. RANDOM ASSESSMENT).

3.1 Mini Assessment

This pattern promotes the use of very short assessments, which take place in a high frequency to check students’ practical work progress on a regular base.

3.1.1 Context

The pattern is applicable in situations where students have to deliver multiple practical assignments with fixed deadlines for a study course.

3.1.2 Problem

Typically, practical work assignments are identical for all students, since it’s hardly possible to provide all students with different assignments with the same level of difficulty. Even if this was possible, the lecturer’s work load would be unacceptably high. Using the same assignments, however, comes with the risk that students hand-in work which has not been solved personally or individually. Authenticity of the submitted exercises needs to be verified.

Using the same practical work assignments for all students, together with handing-in solutions in an asynchronous\(^1\) way, causes the lecturers to need a lot of time for grading and still not being able to judge the authenticity of the work.

3.1.3 Forces

The following forces need to be taken into account:

**Face-to-face assessment** The authenticity of work handed-in in an asynchronous way must be checked.

**Working in groups** The fact that students work together and, by doing this, motivate and teach each other, is actually desirable and should be encouraged. For good students, this way of working can have synergistic effects. For other students, the temptation to copy work without completely understanding it, is usually higher. In cases where students have high workload or very limited time, the temptation is particularly high. Group work needs to be encouraged, together with a mechanism ensuring individual understanding.

\(^1\)asynchronous: there is no personal contact at delivery time.
Limited time The lecturer’s time budget for a specific course module is limited. Especially with larger student groups, the time to have a look at each and everyone’s individual solution in detail is often not available. Furthermore, time that is spent on assessments, cannot be spent on teaching and coaching.

Working JIT Students tend, like most *homo sapiens*, to finish work Just-In-Time. Only having one or two semester deadlines causes students to start working just before these deadlines. This behavior is undesirable, since it bears the risk that they will run out of time, with all its consequences for the quality of the work and the quality of the learning process. It also causes peaks in workload at the end of the teaching period, both for students and lecturers. On the other hand, asking students to hand in solutions on a regular base comes with the lecturer’s duty to give feedback on a regular base as well, which is not always possible for time constraints.

Early feedback An involved lecturer continuously checks if the students are still on track, by following their study progress. Lecturers can only get a proper progress indication by assessing students in some way, starting early in the course module.

3.1.4 Solution

The described problem can be solved by applying the concept of mini assessments. Students are invited frequently, for example weekly, for a short oral assessment. The assessment typically does not take more than about 5-10 minutes. The student shows and explains his solution of the practical work assignment after its deadline has passed. The lecturer asks questions with respect to these assignments and the underlying theory. The purpose of the assessment is 3-fold. First of all, to check if the student completely solved the assignment before the deadline (which is typically the assessment date). Secondly, to check the authenticity of the presented work. Thirdly, to check if the student’s work can be considered as a serious attempt. Students are allowed to fail (see also GRADE IT AGAIN SAM [1]); therefore solutions don’t need to be 100% correct. The student has to convince the lecturer that he has made a viable effort to properly solve the assignments.

The mini assessment is not graded as such. Students who do their work properly and seriously, don’t have to fear the mini assessments at all. Students who didn’t do their work properly (work not finished, work is not own work, work cannot be considered as a serious attempt) or who do not show up at all, will be “punished” with a deduction from the final practical grade (which could be determined, for instance, using the PERFORMANCE ASSESSMENT pattern). To keep students motivated, this deduction should be substantial. A 10% deduction per failed mini assessment turned out to work well.

3.1.5 Consequences

Applying the mini assessment pattern might have the following consequences:

- In a face-to-face oral assessment, the authenticity of work can be checked. Students can work together to solve the assignments, but they are forced to fully understand the assignment solutions and therefore the underlying theories.

- Since assessments take place quite often, students start working with the module topics early, especially early enough to be properly prepared for the formal module assessment.
• The lecturer gets an impression of the level of knowledge of his students early, and can potentially adapt the course content or pace to the students’ understanding.

• Since the assessment itself is not graded, it can also be used as an individual feedback moment for the student. The lecturer’s time is used efficiently.

• It could be difficult and time consuming to assess each student every week, especially with larger groups. The usage of the RANDOM ASSESSMENT pattern could be recommended.

• In our experience, short oral assessments are very efficient, especially compared to long (30-60 minutes) assessments. Experienced lecturers will get a good impression of the student’s achievement in a short time.

• Since mini assessments are not graded, student assistants can also be involved. Only in cases where student assistants decide that the examinee has not done a sufficient job, a lecturer needs to cross check.

• The concept of a serious attempt has not been defined very strictly. In some cases, it’s difficult to determine if a student doesn’t show proper work because he has not put enough effort in finding a solution, or simply because he doesn’t have the understanding (yet) to solve the assignment. The latter case should still be appreciated as a serious attempt. In case of doubt, the lecturer can decide to give the student an additional chance to show and explain his work again.

3.1.6 Examples

An example of the application of this pattern can be found in the example section of the RANDOM ASSESSMENT pattern. This section describes how the MINI ASSESSMENT is used in a Relational Databases course module.

3.1.7 Related Patterns

This pattern is frequently applied together with RANDOM ASSESSMENT, to make the work load acceptable. If the mini assessment is not used for formal grading\(^2\), the pattern can be used together with the PERFORMANCE ASSESSMENT to determine the practical work grades.

3.2 Random Assessment

This pattern promotes to randomly choose students for assessments to reduce the lecturer’s assessment effort, while motivating students to work continuously on exercises.

3.2.1 Context

A course comprises a practical component, in which students work on exercises that have to be delivered by fixed deadlines.

\(^2\text{In our opinion, formal grading should be avoided in mini assessments, certainly in combination with RANDOM ASSESSMENT.}\)
3.2.2 Problem

It is important that the students deal with the exercises on time, because the assignments build up on each other and the experience gained during the practical work is beneficial for following the lectures. Students need to be made to work individually and independently on exercises and deliver them on time.

3.2.3 Forces

The following forces need to be taken into account:

Regular checks In our experience, students do not continuously work on practical assignments if the deliverables are not regularly checked by the lecturer.

Limited time The lecturer’s time budget does not allow checking the delivered assignments of all students after each deadline.

Individual feedback To create the best learning experience, students should receive individual feedback on their delivered solutions.

Independent solution The lecturer has to make sure that the student worked independently on the solution. Students copying solutions from fellow students must be avoided.

Exam situation Students are better prepared if they have to verbally explain their solutions rather than submitting them asynchronously.

3.2.4 Solution

Choosing students for individual assessments at random provides a good balance between the forces mentioned above. After each deadline for the exercises, randomly choose a significant subset of the students and assess their delivered solution. This assessment can either be done offline providing detailed written feedback, or preferably using the MINI ASSESSMENT pattern, in which a student has to orally defend the solution he delivered. In the latter case, the lecturer can ask in depth questions to make sure that the student solved the exercise himself and allow the student to ask questions for clarification himself.

To make sure that students, who were chosen previously, are still motivated to solve the future assignments on time, it makes sense to keep them included in the random selection procedure. That way, the chance of being chosen after each deadline stays equal for all students. In order to exert enough pressure on the students, the chance of being chosen should be greater than 30%, otherwise students seem to poker more often by not getting prepared very well.
To raise trust in the random choice, it is a good practice to allow students to attend the selection process. One way of achieving this is to use a hardware wheel of fortune. As CS lecturers we make do with a software that randomly chooses a subset of student numbers out of all course participants and show the execution of the software using a beamer at the end of a lecture (after the deadline has passed). The tool slowly and clearly picks student numbers one after the other like in a lottery, which is usually a funny happening for the students.

### 3.2.5 Consequences

When applying this patterns in a course, a number of consequences might ensue.

- Compared to the situation in which a lecturer has to grade handed-in work of each individual student, the use of random selection reduces workload, without losing quality.

- The risk of being chosen for an individual assessment causes students to work on the exercises much more carefully. We experienced that, on average, the students are much better prepared for exams than in courses where delivered assignments are just checked once at the end of the course.

- Compared to courses, in which the students have to present each exercise to the lecturer, the students seem to be less prepared in the case of random choice. This, however, is often an acceptable trade-off regarding the lecturer’s limited time-budget.

- A few students might be chosen more often, while others are chosen seldom or never. This is problematic mainly in two ways: Students who are chosen very often could get the feeling of being disadvantaged, especially if a bad performance during the assessment is punished by a bad grade. On the other hand, students who were not chosen very often or never, are unable to benefit from the personal feedback given by the lecturer. The latter problem can be mitigated by allowing students to voluntarily apply for the assessments.

- If students are chosen at random, the assessments cannot be used as the only basis for determining the grade. Therefore, this pattern is often combined with the PERFORMANCE ASSESSMENT pattern, which is then primarily used for grading a student’s achievements.

### 3.2.6 Related Patterns

- MINI ASSESSMENT

- PERFORMANCE ASSESSMENT

### 3.2.7 Example

In our course *Relational Databases*, the RANDOM ASSESSMENT pattern is used in combination with MINI ASSESSMENT and PERFORMANCE ASSESSMENT to determine the students’ practical grades. The students have to solve a number of practical
exercises (e.g. creating an entity relationship model or a relational design, writing SQL queries or developing stored procedures). Each of these exercises has a fixed deadline.

When the deadline has passed, we use RANDOM ASSESSMENT to determine a number of students, with whom we conduct a MINI ASSESSMENT. If the students cannot convincingly show that they at least made a serious attempt to solve the exercise, then the practical grade is reduced by one point. The main part of the practical grade is determined using two PERFORMANCE ASSESSMENTS at the end of each quarter.

3.3 Collective feedback

A collective feedback session provides students with the chance to clarify questions with regard to practical assignments.

3.3.1 Context

This pattern can be applied in situations, where multiple students work independently on the same practical task. It is particularly helpful where RANDOM ASSESSMENT is applied, thus not all students are actually assessed regarding specific exercises.

3.3.2 Problem

Providing students with individual feedback for practical tasks is often not feasible because of time constraints of the supervising lecturers. If, however, students do not receive feedback on their solutions soon after the submission of the exercises, the learning effect is drastically reduced.

3.3.3 Forces

The following forces need to be considered when finding a solution to this problem:

**Individual questions:** Students should be able to ask specific questions about their individual solutions.

**Common problems:** When solving exercises, many students stumble upon the same problems.
The disadvantages of correct solutions: When students successfully solve an exercise, they often avoid possible pitfalls without being aware of this. Therefore, they miss opportunities to learn from mistakes.

Time consumption: Lecturers usually have a limited amount of time for supervising practical work.

3.3.4 Solution
After each deadline for practical assignments has passed, invite students to take part in a collective feedback session. In this session, the lecturer explains common pitfalls and mistakes students made in the past. The students are also encouraged to show their individual solutions to the public. In such cases, the lecturer gives individual feedback on the shown solutions, while other students can compare this solution to their own.

3.3.5 Consequences
When applying this pattern, a number of consequences might ensue:

- Individual students get the chance to receive feedback on their individual solutions. Some students, however do not feel comfortable with discussing in front of the group. They might also be afraid of making a fool of themselves, in cases where they made apparently stupid mistakes.

- Mistakes that are usually made by many students only have to be explained once. This reduces the work load of the lecturer.

- Students can learn from each other’s mistakes. During the feedback sessions, other students could ask questions or discuss problems a student did not run into himself.

3.3.6 Related Patterns
This pattern is closely related to RANDOM ASSESSMENT. If RANDOM ASSESSMENT is applied, not all students receive feedback during the assessment. This can be partially compensated using COLLECTIVE FEEDBACK.

3.4 Performance Assessment

![Diagram](image)

This pattern promotes the assessment of practical skills, e.g. programming, in an authentic environment, where students can use tools like IDE’s which they would have available in practice as well.
3.4.1 Context

This pattern is applicable in study courses, where students have to solve practical work assignments using a computer.

3.4.2 Problem

Study courses often comprise a practical component that can hardly be assessed in written exams or oral examinations. This is particularly the case when practical skills need to be assessed that require the use of specific software tools, e.g. an integrated development environment (IDE), or a computer aided software engineering (CASE) tool.

When grading students, a lecturer should always endeavor to determine a fair grade that realistically reflects the students’ abilities. In some situations, classical examination strategies like written or oral exams are inappropriate, because they require a student to use an unfamiliar and unrealistic way of working. The following example illustrates the problem: In programming courses, students usually learn by solving practical programming exercises. Typically, programming is done using IDEs, which provide among others debugging facilities, show compilation errors, wizards, shortcuts, and code completion. When examining a student’s programming skills in a written exam, the student has to renounce these auxiliary tools. To make matters worse, paper does not allow for frequent corrections and inserting code lines. Thus, a student’s result in a written exam on programming does not necessarily reflect his programming skills in a realistic programming setting. To make matters worse, handwriting skills of CS students have not improved lately, causing lecturers who do not have a strong background in cryptography to despair.

Alternatively to exams, practical work is often graded using exercises that students need to submit. This, however, entails the serious problem that students copy from each other, which avoids fair grading upfront.

3.4.3 Forces

When dealing with these problems, the following forces need to be taken into account:

**Tool usage**  Students should be able to use the same tool stack in an exam as during exercises.

**No copying**  Copying solutions from fellow students should be prevented.

**Grading effort**  Grading should not take more effort than grading written exams or performing oral examinations.

**Allowed sources**  Students must not consult any other than the allowed and provided information sources for solving exercises.

**Archiveability**  Examination data must be archived and preserved unchanged to be accessible for future inspection.

**Authenticity**  Examination data must be clearly assignable to a specific student.
3.4.4 Solution

To solve the problem and balance the mentioned forces, the students’ practical skills should be assessed under examination conditions using exactly the same tools as usual. We call this type of examination *performance assessment*, in educational psychology referred to as *performance-based assessment*, which forces students to solve complex authentic tasks applying recent knowledge and relevant skills [2].

Compared to other disciplines like medicine, social sciences, or circus acts, realistic performance assessment settings are easily accomplished in computer science courses, because a computer with a specific software configuration is an authentic work environment. Thus, *simulation* of real world work settings is not necessary.

![Figure 2: Isolated workplaces prevent students from cheating and at the same time prepare them for future conditions of work in cubicles](image)

Preventing students from cheating or using unauthorized information sources is a special challenge in performance assessments, which can be faced using the following measures:

- Students receive one-time login credentials at the start of the exam, which are only usable for that particular exam. This reduces the risk of maliciously compromised accounts and prevents students from deliberately using a fellow student’s login data, e.g. to look at his solution or to support him during the exam.

- The machines used in the exam should be isolated from each other, e.g. by means of firewall rules or network switch configuration. In cases where students are not allowed to use web resources, internet access should be disabled completely. Particularly computer science students quickly find ways to communicate to each other over the network, even if no obvious communication software is installed on the exam machine.
• The exam machines should not provide any software to the examinee that is not necessary for the purpose of the exam. Each needlessly available software increases the risk of loopholes that students can use for cheating.

• During the exam, the students (partial) solutions to the exercises should be stored on an external server, which is constantly synced with a backup system. That way, a failure or crash of one exam machine does not result in data loss and the student can proceed with the exam on a different machine.

• Compared to paper exams, digital deliverables could easily be changed after their submission. To rule out this risk, a hash value of the submitted results can be generated at the end of the exam, which is either printed and signed, or digitally signed by the student.

• Computers provide vast logging facilities which should be used during a performance assessment. That way, the logs can be analyzed later, if fraud or malfunction are suspected.

• Like in other exams, students should be prevented from getting the chance to observe each others screens. This can be achieved using cardboards, which are temporarily placed between desktops (see Figure 2).

• Finally, at the moment where students leave the room, the accounts must be disabled to avoid students from accessing the exam server in the rest of the available exam time outside of the exam room.

3.4.5 Consequences

When applying this pattern for an exam, a number of consequences might ensue:

• Students usually feel much more comfortable in the exam when they are allowed to use the tools they are familiar with.

• As opposed to correcting paper exams, grading is simplified as the lecturer can use software tools like unit testing, checkstyle etc., to support the analysis of the students’ submitted results.

• In computer science, there are often many ways to solve a problem at hand. Sometimes students come up with results the lecturer had not thought of before. The availability of digital solutions allows the lecturer to verify solutions much more easily than results submitted on paper (e.g. by compiling and executing).

• Compared to paper based exams, preparing a performance assessment takes more effort and requires careful planning. The software environment has to be configured, exam accounts need to be created, and the exam setting has to be thoroughly tested to assure seamless operation. This is particularly challenging in the first couple of executions. The whole process should be automated as far as possible. Automatic testing and verification of the exam setting using software is desirable.
• The risk of technical failure during the exam can never be completely alleviated, especially in comparison to paper based exams. Therefore, fallback procedures should be defined, which can be used in case of a severe technical problem, e.g. an alternative examination date.

• We sometimes observe that students are more likely to get stuck when trying to solve certain assignments, since they get immediate feedback that certain solutions do not work. The probability of wasted time is higher, compared to traditional exams.

• The work handed in is readable.

3.4.6 Examples

In the software engineering study course at our university, we use performance assessments for grading the students’ abilities in all courses that have a practical component. In the following, we describe the setting used in two courses: Programming in Java and Relational Databases.

In our university, we have enough computer work places for each student in a study year. If this was not the case, the students from one year could not be assessed at the same time, requiring two different sets of questions for the student groups to avoid cheating. This is not desirable, as it requires higher correction effort and it has the potential of unfair grades, as question sets can hardly have exactly the same level of difficulty. As shown in Figure 2, we use flexible cardboards to separate the work spaces from each other. They can easily be removed after the assessment to create a friendlier atmosphere during normal practical work.

The lab computers have a dual boot configuration. Usually, Ubuntu linux [3] is used for performance assessments, because the exam environment (e.g. restricted shell, limited user rights, special routing tables, allowed tools) can easily be configured using script languages. The exam data, students create during the assessment is submitted to an individual subversion [4] repository, which was created for each student upfront and which is used solely for one assessment. For each student, an exam account is generated that is used to log into the machine and to access the subversion repository.

In the programming course, the students use the Netbeans IDE [5] for Java. Netbeans has an integrated subversion client, which is used to access the repository. Other than that, the students may use a web browser to access their repositories and check if the assessment data was submitted correctly.

In the databases course, the students have to construct SQL queries and implement stored procedures, cursors, and triggers. In this case, we configure an external Oracle DBMS [6], which students can access from the exam computers. For each student, a DBMS account is created and a database dump is imported into the respective user space. The students uses Oracle’s SQL Developer [7] software for the queries and implementations. The integrated subversion client is used to access the personal subversion repositories.

3.4.7 Related Patterns

For assigning the practical grade for a course, this pattern can be combined with MINI ASSESSMENT and RANDOM ASSESSMENT.
3.5 Peer Assessment

This pattern promotes the active use of the students’ insights in the performance and quality of their peer students during group work.

3.5.1 Context

A course comprises a practical component, in which students work in groups. The students get some guidance by lecturers (depending on their maturity), but most hours spent on the tasks is done without supervision.

3.5.2 Problem

The reasons to choose group tasks are amongst others:

- Groups allow bigger, more complex, and therefore more interesting tasks.
- Group work is an essential competence in many disciplines and should be used in all practice-oriented courses.
- Group work reduces the number of similar artifacts to be reviewed, which makes tutors work less boring.

Assigning individual grades to students who work in groups, purely based on a lecturer’s impressions and handed-in artifacts, is particularly challenging. The individual contribution to group artifacts (like a UML class diagram) can be difficult to measure. Furthermore, some important group work skills like team competence, reliability and a constructive attitude are sometimes hard to be observed by a lecturer.

3.5.3 Forces

The following forces affect possible solutions to this problem:

Work behind the scenes  In group work, students spend most of their time without supervision by the lecturer. Most of the interesting group interaction therefore takes place when the group is unobserved. Think for example of discussions and decision making processes and the role of each individual within these processes.

Attitude is influenced by the environment A professional attitude of each of the team members is a key success factor for group performance and atmosphere. The observable attitude during the lecturer’s presence however can be different from the attitude during his absence. Some students perform better in an unobserved environment, while others tend to perform worse. In both cases, the lecturer’s impression will be biased.
Differentiation is necessary Although it’s reasonable that a group is responsible for all deliverables and therefore each individual is graded based on those, individuals should be motivated to excel and they should be rewarded for above-average performance or penalized for below-average performance.

Open critique Many students have problems criticizing each other in the open. Especially introverted students can hardly be pushed to provide critical feedback to their group mates openly.

Anonymous feedback Anonymous criticism, as far as the receiver of the critique is concerned, is potentially more objective, but also has the potential danger of peer bashing.

3.5.4 Solution

The students evaluate each other, except themselves, in a so called Peer Assessment on a few preselected criteria. Most of these criteria are rather universal and can be used in non computer science courses as well, e.g. team competence, quality or quantity of delivered work, reliability, or writing skills. The student anonymously assigns a grade for each of these criteria ranging from 1 (bad) to 10 (best possible) to each of his or her fellow students.

Figure 3.5.4 shows the results of a (fake) peer assessment, the authors conducted with each other using a web based tool, which is also used in our university.

Optimally, peer assessments are conducted multiple times in a course, so that peers have a chance to improve themselves after receiving constructive feedback (see also MISTAKE [1]).

3.5.5 Consequences

When applying this pattern in a course, a number of consequences might ensue:
• You will need some kind of (software) tool, in which the students can enter their peer assessment grades.

• Peer assessment grades cannot be taken for granted. Lecturers should always have an educated look at the total group result before using them as (a component) in a course grade. In case of big deviations from the tutor’s own assessment, extra investigation, for instance using an interview, should be done.

• Quantifying someone’s performance can be difficult and doesn’t give any insight in the reason why certain grades are given by peer students. Asking students to write a short individual reflection document can be considered, complementary to the peer assessment.

• Anonymous peer assessments are useful for both student and lecturer. Complementary to this, a project group should always try and be motivated to ensure transparency and create an atmosphere in which students can be pointed to their responsibilities, strengths and weaknesses face-to-face.

3.5.6 Related Patterns

• ROLE PLAY[1] which is often applicable in group work,

• REFLECTION on self and others is stimulated by giving and receiving feedback.

• MISTAKE[1] applies if the PEER ASSESSMENT is applied more often during one course.

3.5.7 Examples

In our software engineering and business informatics courses we typically use the following four criteria:

**Team competence**  His/her participation in the group process adds a lot to the success of the project.

**Quality**  His or her contribution was of high quality.

**Quantity**  His or her contribution was of high quantity.

**Reliability**  He or she sticks to the appointments and is a reliable team member.

To reduce the workload on the lecturers in collecting the student data, we developed peerweb, a self developed web portal (see Figure ??). Peer assessments have been successfully used in our course for more than five years.

3.6 Student company

This pattern promotes the use of 'real-world' projects, in which external (non university, non course related) people are involved, for example in the role of principal, customer or expert.
3.6.1 Context

This pattern is applicable in situations where students have to show that they are able to apply gained knowledge and skills from several modules in prior study phases. It’s therefore typically used in later semesters.

3.6.2 Problem

Theoretical knowledge and certain skills can be properly developed and assessed in a ’closed world’ university environment, in which lecturers act as principal / customer and define project case studies. Some other skills, as well as a professional attitude, are important parts of students’ competence level for which the university environment does not always suffice. Especially in later phases of a study program, aspects like for example customer communication and dealing with uncertainty need to be practiced.

3.6.3 Forces

The following forces play when solving the problem:

- Students have to develop skills to act in an environment, in which they have to deal with processes and/or experts and/or colleague students from other (non IT) domains.

- Students have to practice their social skills. Working in longer termed and more elaborate projects, within bigger project groups, involves group dynamics, like communication (with its difficulties), consensus finding and conflict management.

- Students have to learn to deal with uncertainty, limited resources, changing environments and incomplete or changing requirements, setbacks or disappointments.

- Students have to develop a professional attitude towards project stakeholders.

- The students have to deepen their skills and competences particular to the domain. For software engineering for instance, you may think of analysis, design and realisation. For social skills you can think of group leadership, customer contact and the like.

- Students have to learn to deal with imperfections. The ’perfect world’ doesn’t exists; business environments and systems arise and develop, influenced by many factors, constraints and limitations. Here imperfection is the norm instead of the exception. Students need to learn to cope with these kinds of imperfections.
In practice, a lot of software projects deal with changes to or extensions of existing solutions. Students need to understand and accept a solution as is. They should also learn to properly hand-over their work in a way that makes their contribution valuable to the future of the project.

3.6.4 Solution

To solve the problem, taking into account the mentioned forces, a student company concept is applied; students work together in a project group solving problems from industry. The group is self-organizing, meaning that the students define roles and procedures themselves, to optimally fulfill a business request.

Different parameters can be used for tuning to specific learning goals and for tuning the degree of lecturer involvement. Students can for example be asked to apply for a certain position within a certain project, or students can alternatively be assigned by lecturers. The group size can be increased in line with project complexity. Assignment descriptions can be developed in advance, or be crystallized during the project by students themselves. Groups can work on assignments from scratch, or participate in existing longer lasting projects for a certain period only.

At the end of the project, students present their results to all project stakeholders. Beside group products and artifacts, a student’s personal contribution to these as well as his personal competence development during the project are taken into account for grading. In the beginning of the project, each student defines his personal competence development goals. He chooses a role within the project enabling him to reach his goals (e.g. someone who wants to develop his management skills could be project manager). At the end of the project, the student prepares a personal assessment dossier, making his contribution and his competence development visible. During an oral assessment, authenticity of his work is verified.

Since the intra group processes are partially unobserved by the lecturer, usage of the PEER ASSESSMENT pattern is highly recommended in combination with this pattern.

3.6.5 Consequences

When applying this pattern, the following consequences must be taken into consideration:

- Students are forced to work in domains they are not necessarily familiar with. This will especially be interesting in cases where the target domain is not IT-related.

- Social skills are a very important project success factor. Students will differ in talent, quality and drive. These are potentially conflicts of interest. Dealing with these is what is required in a professional setting.

- Aspects which can normally only be simulated, like uncertainty, resource limitations, dependencies, changes and imprecise stakeholder statements, will now influence the project in a natural way.

- Students have to accept that they have to work with existing systems and imperfect resources or earlier versions. There is typically not time to redo a lot of work or to change the environment.
• They will experience that documentation is very important, especially in the earlier project phases; they will often be confronted with unclear and/or missing information or documentation.

• With a (larger) group, more complex and hence more interesting projects/products can be tackled.

• The coaching effort needed to keep students on track can be reduced. Given the fact that these projects typically take place in senior study phases, this is even desirable from a didactic perspective; students should be given the chance and responsibility to run their own project.

• Involving external people is only meaningful if they have a serious interest in the problem and are willing to invest (mainly time, possibly other resources). Partnerships with companies can be considered.

• As lecturer, you are less ‘in control’ than you would be in regular student projects. For instance problem complexity and effort largely depends on project as formulated by the client. Client involvement is also out of your control.

• Professional attitude and taking responsibility is developed.

• Working as a self steering team requires a professional attitude towards all stakeholders in the project, in particular towards the customer and the other team members.

• The customer gets a software solution for his problem relatively cheap. Especially proof-of-concept or prototyping projects may be interesting, since getting budget for these types of projects can be difficult within companies.

• Companies come into contact with students (potential future employees) and vice versa.

• The university gets an excellent opportunity to show its ‘product’, being well educated directly employable young professionals.

In particular the last two points are a win-win-win situation for the three parties involved and is an excellent enabler for this kind of project settings.

3.6.6 Related patterns

• ROLE PLAY [1] is typical in student companies. For instance, the customer will not be pleased if he is contacted by every member but will like a fixed contact.

• REFLECTION [8]

• EXPLORE YOURSELF [8] by applying for and executing roles.

• PEER ASSESSMENT is applied to assess various aspects.

• Teacher Selects Teams [8] is applied for several reasons, such as getting the required set of team competences.
• ACTIVE STUDENT or WORK FORMS [8] without activity of the students the student company will fail.

• REAL WORLD EXPERIENCE [8] which is brought in by the external customer.

3.6.7 Examples

In our university, the student company is applied in two different courses. In the so-called software factory module, the students work in software projects for external, mainly industrial customers. This is already interwoven into our previous sections on forces, solution and consequences.

Within the University we also know the Mini Company module, which is part of a business study program. Students set up their own company to produce and market their own products. Here the students do not have an external customer with an existing problem. Instead, they must define the product themselves by doing market research, prototype development and the complete business organisation including funds finding, production (organisation), accounting, marketing and sales etc. The products are typically gadgets, but the setting has aspects similar to the engineering projects. For instance, students also assume roles, produce artifacts (reports, accounting) and are assessed at the end. Peer Assessment is applied too.

Acknowledgements

We would like to thank all lecturers in the software engineering study course at the Fontys University of Applied Science in Venlo for giving us insights into their assessment approaches and for discussing the pattern descriptions with us. We would also like to thank our shepherd Yishay Mor for providing encouraging and valuable feedback at EuroPLoP 2012.

References


A Pattern for Learning Phraseology of a Foreign Language

Karen Velázquez
Universidad Complutense de Madrid - Universität Hamburg
alatispatent@gmail.com

Abstract. Due to their idiomatic nature and fixed structure, phraseological units are the most elusive lexical items to apprehend when learning a foreign language. Offering students an equivalency-based instruction, that is, presenting phraseological units attending to the similarities found between the learner's mother language (ML) and the foreign language (FL), has been proven to give rise to higher comprehension and production levels of these lexical units.

Key words: lexical items, phraseology, idioms, phraseological units, locutions, learning a foreign language, foreign language teaching.

1. Introduction

Every human being is born with the innate capacity for acquiring any natural language within a relatively short amount of time. Likewise, and by means of an entangled construction process influenced by cognitive, attitudinal, social and affective factors, achieving an excellent command of a foreign language is within reach of every individual, with the element of time being the only limiting variable.

As language teachers the focus of our investigation should be confined to the learning process itself, by posing questions to determine in which way it is possible to optimize the construction of the interlanguage, so that a more efficient and effective instruction can be offered to students who have to face the escalating demands of a growing globalized world.

Learning a language does not consist of memorizing grammatical rules and dreary lists of vocabulary; it means discovering a different culture and keeping an open mind towards a different way of thinking and living.

This pattern can be applied by language instructors whose students currently either learn a foreign language in a non-immersion environment or acquire a second language in a mixed immersion context. Their stadium of interlanguage shall fluctuate between B2 and advanced C1 level of the CEFR¹.

¹ For more clarification see Appendix.
The following pattern is aimed at language instructors who have a command of linguistics and language didactics. If you are not familiar with these fields take a look at the appendix to clarify the most important concepts and terms, which have been marked in bold throughout the text.

2. **Background**

The cultural and historical components which are found in phraseological units - such as “to know the ropes” or “to be worth one’s salt” - make them the most propitious lexical items for allowing a student to become impregnated with the culture of the target language, not to mention the expressiveness and lexical enrichment that the usage of these units means to any speaker. In addition, phraseological units have a high frequency of use in the discourse and therefore they must be gradually learnt as a part of the lexicon of the foreign language.

Furthermore, the urge of simplifying the aforementioned learning process and creating teaching proposals based on similarities between languages have also triggered the conception of this pattern. Throughout the last decades linguists and editors have only emphasized differences between the learner’s first language and the language to be mastered (false friends are the best example), although the key to success when it comes to language learning lies in remarking the similarities between both languages. The underlying principle supporting similarity-based language learning is the fact that the mother language - along with other second language(s) and/or foreign language(s) the user might have already acquired/learnt - is used as scaffolding during the learning process regarding the new foreign language, so that the new data is built upon the existing linguistic competences. Thus, the cognitive effort while learning the new language is reduced and the storage and consolidation of the new linguistic system occur in a faster and more efficient way.

**Related work**

Until recently, there had been few language learning proposals based on similarities between languages. One of the latest is that from Cecilia Ainciburu [1] which is aimed at the early stages of the interlanguage construction and deals with lexical units in general rather than focusing on phraseological units.

Nan Jiang’s model of lexical acquisition in adult students of English [2] also regards the fact that the lexical competence of a second language (SL) is built upon the foundations of the ML.

For Michael Lewis [3] the transfer of lexical meaning is an instinctive method the mind uses to acquire a SL. Thus, Lewis pleads for exploiting the natural tendency of falling back into the ML and translating chunks i.e. segments of information to facilitate the transfer from phraseological units (and further lexical items).
3. **The Pattern**

This pattern proposal is composed of the following sections: the context where the problem appears, the problem itself, the forces, the solution to the problem, the consequences of applying the solution and finally known applications in the section evidence.

**EQUIVALENCY-BASED PHRASEOLOGY LEARNING**

Also known as **Phraseological Twins**

**Context**

In every natural language phraseological units appear in the spoken and written speech, intertwined with many other lexical items. The learner of a FL is gradually exposed to samples of language which contain these lexical units, which the learner is expected to comprehend and eventually reproduce.

**Problem**

Without the right input the learner of a FL cannot fully comprehend and reproduce phraseological units effectively. This deficiency in the lexical competence within a given stadium of the interlanguage could jeopardize both the sociolinguistic and pragmatic competences and therefore compromise the integrity of communication.

**Forces**

- **Idiomaticity.** Idiomaticity is one of the features which stand in the way of comprehension of phraseological units for the learner of a foreign language. These sort of elusive lexical items possess phraseological or idiomatic meaning, i.e. the meaning of the whole unit is not conveyed by the sum of the meanings of its components but formed as a result of generalized figurative meaning. The idiomatic or phraseological meaning generally stems from the cultural background and/or the figures of speech contained in the phraseological unit, such as similes, metaphors, metonymies, hyperboles, etc.

- **Fixation.** Phraseological units are unique and fixed combinations of at least two constituents which do not function in the same way in any other combination or combinations of the kind, or occur in a highly restricted number of contexts or in a single setting only. Fixation (internal or external), although there is always a certain degree of variation to be taken into account, is an inherent feature of phraseological units which make their production in writing and speech more difficult than other lexical items.

- **Exclusion.** Despite being a crucial part of any language in terms of frequency of use, phraseological units are seldom included in syllabuses and hardly treated at length, mainly due their aforesaid properties. This exclusion hinders the learner from developing his lexical competence to the same extent as other linguistic competences in a given stadium of the interlanguage.
Solution

Therefore: Offer an equivalency-based instruction, i.e. present phraseological units attending to syntactical, morphological and lexical similarities between the learner’s mother language (ML) and the foreign language (FL).

Using an equivalency-based input, having previously conducted a contrastive analysis of such units between both languages and having classified the results according to their type of similarity, enhances the comprehension and production levels of phraseological units in a given stadium of the interlanguage.

Implementation

In order to implement the solution, the following steps shall be performed:

I. Selecting a corpus of phraseological units, according to the institutionalization and frequency criteria.

II. Transferring the phraseological units from the source language to the target language and classifying the obtained results into different types of equivalency.

III. Tailoring the equivalency-based instruction and putting it into practice.

I. Corpus selection

The first step is to select a corpus of phraseological units of the language which is meant to be taught. The corpus is constructed in accordance with the following criteria:

- Institutionalization criterion: the phraseological unit must be included in an up-to-date normative lexicographical catalogue. By doing this we ensure that the phraseological unit is still in force and has being used on a regular basis among the speakers of that particular language.

- Use frequency criterion: the phraseological unit must have a high frequency of use in the discourse of the speakers of that particular language.

II. Contrastive analysis and results classification

Once the corpus has been selected, a systematic study between the source language and the target language is necessary to determine their structural similarities and differences.

---

2 The language instructor shall delimit the kind of phraseological units to be taught and their semantic field according to the course contents and purposes.
3 During the implementation, the language to be taught will be addressed as source language.
4 If the language to be taught is subjected to a high degree of diatopic variation (which is the case of Spanish) it is also necessary to opt for one of its linguistic variants.
5 In the present analysis the corpus of phraseological units was restricted to verbal locutions and the selected semantic field was that of somatisms.
The transfer of phraseological units from the source language to the target language is made in two steps:

* Semasiological phase (from the signifier to the signified): the meaning of each phraseological unit of the source language is noted.
* Onomasiological phase (from the signified to the signifier i.e. reconstruction of signified in the target language): each noted meaning is given an expression in the target language. With this two-phase procedure the phraseological unit of the source language is linked to a semantic equivalent in the target language.

When transferring a phraseological unit to the target language, different degrees of equivalency can be found. The correlates can fall into the following main categories:

* **Total equivalency (TE);** there is a phraseological identical correlate in the target language for the phraseological unit of the source language. In both phraseological units:
  - The phraseological meaning is identical
  - There is syntactical coincidence of their components.
  - There is semantic coincidence of their components.
  
    For example:  
    
    SPA. *perder los nervios* (*to lose the nerves)*
    GER. *die Nerven verlieren* (*to lose the nerves)*
    ENG. *to lose one’s temper*

* **Partial equivalency (PE);** there is a phraseological similar correlate in the TL for the phraseological unit of the source language. Regarding the degree of equivalency, the correlates can fall into the following subcategories:
  
  o **Type 1:**
    - The phraseological meaning is identical.
    - Syntactical variation or grammatical category variation of its components.

    For example:  
    
    SPA. *asesinar a alguien a sangre fría* (*to assassinate someone with cold blood)*
    GER. *jemanden kaltblütig ermorden* (*to assassinate someone cold-blooded)*
    ENG. *to kill someone in cold blood*

  o **Type 2:**
    - The phraseological meaning is identical.
    - Semantic variation of the verb or its components.

    For example:  
    
    SPA. *pillarse los dedos.* (*to catch one’s fingers)*
    GER. *sich die Finger verbrennen.* (*to burn one’s fingers)*
    ENG. *to catch one’s finger*6

---

6 Note that there is total equivalency between the English and the Spanish phraseological unit. Hence the importance of conducting contrastive analyses between each pair or languages.
Type 3:
- The phraseological meaning is identical.
- Variation of the somatic lexeme.
  For example: SPA. *llevar el corazón en la mano* (*to carry the heart in the hand*)
  GER. *das Herz auf der Zunge tragen* (*to carry the heart on the tongue*)
  ENG. *to wear one’s heart on one’s sleeve.*

* Zero equivalency (ZE); the phraseological unit of the source language has either a phraseological correlate or none:

- Type A: There is a phraseological correlate in the target language with the same phraseological meaning but no formal equivalency.
  For example: SPA. *hablar por los codos* (*to talk through the elbows*)
  GER. *wie ein Wasserfall reden* (*to talk like a waterfall*)
  ENG. *to have verbal diarrhea*

- Type B: or no phraseological correlate at all, in which case the only way to reproduce the phraseological unit of the source language is by means of a periphrasis, circumlocution or explanation.
  For example: SPA. *No dejar títere con cabeza* → To attack and kill everyone (figuratively) by cutting their heads off [Origin: novel El Quijote]

<table>
<thead>
<tr>
<th></th>
<th>TE</th>
<th>PE1</th>
<th>PE2</th>
<th>PE3</th>
<th>ZEa</th>
<th>ZEb</th>
</tr>
</thead>
<tbody>
<tr>
<td>phraseological meaning</td>
<td>identical</td>
<td>identical</td>
<td>identical</td>
<td>identical</td>
<td>identical</td>
<td>completely different</td>
</tr>
<tr>
<td>syntactical / morphological level</td>
<td>identical</td>
<td>slightly different</td>
<td>identical</td>
<td>identical</td>
<td>completely different</td>
<td>completely different</td>
</tr>
<tr>
<td>semantic level</td>
<td>identical</td>
<td>identical</td>
<td>slightly different (verb)</td>
<td>slightly different (lexeme)</td>
<td>completely different</td>
<td>completely different</td>
</tr>
</tbody>
</table>

Fig. 1. Summarize of phraseological equivalencies and their features
III. Instruction

The instruction, which is based on the equivalency input, is built upon a constructivist approach\(^7\); hence it is divided into a theoretical and a practical phase. In the first phase the previously classified phraseological units are presented to the students and they are made aware of the fact that they also exist in their mother language (ML), if some phraseological units show certain variations.

- In the case of TE they are encouraged to figure out the phraseological equivalent in the FL when they are given a phraseological unit in their ML.
- In the case of PE and after having made them aware of the nature of semantic and syntactic changes, they are encouraged to reconstruct a phraseological unit of the FL by offering them the equivalent phraseological unit in their ML.
- In the case of ZE they are encouraged to complete sentences in the FL which include a phraseological unit in the FL, putting emphasis on the existence of a phraseological equivalent in their ML (ZE, Type A) or on the historical or cultural background from which the phraseological unit in the FL has evolved (ZE, Type B).

Consequences

Benefits

- After having received an equivalency input instruction, comprehension levels improve significantly, since the learner has linked the phraseological meaning of the lexical unit in this ML to the one in the FL.
- After having received an equivalency input instruction, the production levels of the students improve satisfactorily, since the learners are given the possibility to discover for themselves the close inner structural connection between the phraseological units in both languages and to observe thoroughly their syntactical, lexical and morphological composition. However, there are still minor errors in word selection (mainly prepositions and word order).
- After being instructed at length in phraseological units of a given semantic field, the learner has stored them and is aware of their existence, developing evenly the several linguistic competences within a given stadium of the interlanguage.

\(^7\) Traditionally, the student has been thought to be a passive recipient of information. Nevertheless, teachers nowadays should plead for a constructive approach when dealing with the learning process and transform the instruction into functional learning. Knowledge is only built through experience, and therefore, the instructor should act as guidance, providing his students not only with the key components of a language, but also with the necessary skills so that they can learn by themselves.
Liabilities

- The lack of contrastive studies as well as the shortage of lexicographical catalogues between the source language and the target language\(^8\) might present a drawback when implementing the pattern. The few available corpora of phraseological units Spanish-German are obsolete and thus, they do not reflect the current state of the word-stock of the language.
- On the other hand, carrying out a contrastive analysis between the two languages involved in the instruction and tailoring equivalency-based teaching materials requires a considerable deal of work on the instructor's part.

Evidence

This pattern was conducted in June 2011, as experimental part of a master's thesis, with a group of German students who were getting their master’s degree in Intercultural Studies and had a C1 level of Spanish according to the CERF. These students had been learning Spanish in a classroom-based environment and had had no or few experiences in Spanish speaking countries.

The corpus of phraseological units was restricted to verbal locutions\(^9\) and the selected semantic field was that of somatisms, i.e. verbal locutions which contained a lexeme which referred to a body part, for instance “to catch one’s finger”.

Pretest, instruction and posttest were conducted within four weeks.
- The aim of the pretest was to determine their comprehension and production levels of somatic verbal locutions.
- The control group was offered a traditional instruction, whereas the experimental group received an equivalency-based instruction.
- The aim of the posttest was to determine if there had been an improvement of their comprehension and production levels of somatic verbal locutions and to measure to which extend the results achieved in the experimental group varied from the ones obtained in the control group. As a matter of fact, the subjects who received an equivalency-based instruction came up with better results than the subjects who were offered a traditional instruction\(^10\).

This pattern has been proven to work in the context of German native speakers who learn Spanish as a foreign language. Nevertheless, it can be also applied in contexts in which the mother language of the learner and/or the foreign language meant to be mastered belong to the Indo-European language family, i.e. Greek, Romance, Germanic and Balto-Slavic languages.

---

\(^8\) In this case Spanish as source language and German as target language.
\(^9\) According to Corpas Pastor [5] phraseological units can be classified into three sub-categories: collocations (e.g. *To fulfill/achieve/shatter a dream*), locutions (e.g. *To catch one’s fingers*) and phraseological statements (e.g. *The early bird catches the worm*).
\(^10\) Due to the intriguing results, a dissertation examining this issue in depth is currently being written.
Acknowledgements

I would like to thank Christian Köppe for shepherding this paper. Not only did he guide and encourage me along the process of writing this (my first) pattern, but he also improved it immensely with his comments, suggestions and experience in pattern writing. I am also thankful for all the enriching feedback I received in our workshop at EuroPLoP 2012.

References

List of abbreviations (In order of appearance)

ML  mother language
FL  foreign language
SL  second language
CEFR Common European Framework of Reference for Languages
SPA Spanish
GER German
ENG English

Appendix

Phraseology Branch of linguistics dealing with phraseological units.

Phraseological Unit Stable or fixed combination of words with idiomatic meaning.
E.g. to lose one’s temper
to burn one’s fingers
to wear one’s heart in one’s sleeve

Mother language A person’s native language; the language acquired as a child.

Second language A nonnative language acquired after the mother language in a context where the first is spoken. In a second language situation the person is exposed to the target language in a variety of settings, e.g. a German graduate moves to Spain and acquires Spanish as second language through immersion as he works in the country.

Foreign language A nonnative language learnt in classroom environment and in a context where the language is not spoken outside the classroom, e.g. a German graduate who lives in Hamburg takes Spanish lessons twice a week.

CEFR The Common European Framework of Reference for Languages (abbreviated as CEFR) is a guideline used to describe achievements language learners across Europe and, increasingly, in other countries around the world. Its main aim is to provide a unified method for learning, teaching and assessing which applies to all languages in Europe. The CEFR divides learners into three broad groups which can be divided into six levels:

A Basic User A1 Breakthrough or beginner
A2 Waystage or elementary

B Independent User B1 Threshold or intermediate
B2 Vantage or upper intermediate

C Proficient User C1 Effective Operational Proficiency
C2 Mastery or proficiency

A detailed description of the four competences a learner is expected to fulfill at each level can be found at: ec.europa.eu/dgs/jrc/downloads/jobs_traineeships_cefr.pdf
False friends
Pair of words in two languages that are similar in spelling and/or pronunciation but have different meanings, e.g. in German Mantel means ‘coat’ but the Spanish mantel means ‘tablecloth.’

Interlanguage
Type of linguistic system produced by nonnative speakers in the process of acquiring a second language or learning a foreign language and characterized by interferences (Aspects which nonnative speakers incorporate from their native language).

Competences
General competences of a language user comprise four sub-categories:
- Declarative knowledge; resulting from experience or formal learning
- Skills and know-how; implying the ability to carry out tasks and apply procedures.
- Existential competence; comprising individual characteristics, personality traits and attitudes towards oneself and others engaged in social interaction.
- Ability to learn; is the ability to engage in new experiences and to integrate new knowledge into the existing knowledge.

Communicative languages competences of a user/learner involve knowledge, skills and know-how for each of the following components:
- Linguistic competence; deals with formal characteristics of a language, such as phonology, morphology, lexicon and syntax.¹²
- Sociolinguistic competence; concerns the socio-cultural conditions of the language use, such as politeness rules or social group repertoires.
- Pragmatic competence; covers the functional use of language in specific scenarios, such as how to act in a job interview.

Simile
Trope or figure of speech whereby two elements are compared based on their similarities. Such comparison is explicitly marked by the use of as or like, e.g. They fought like cats and dogs.

Metaphor
Trope or figure of speech whereby one element is substituted by another based on their similarities, e.g. She’s a diamond (highly valuable, esteemed).

¹² The linguistic competence comprises the following sub-categories: lexical competence, grammatical competence, semantic competence, phonological competence, orthographic competence and orthoepic competence.

Further information on this subject can be found in GREEN, A. (2012): Language Functions Revisited: Theoretical and Empirical Bases for Language construct definition across the ability range. Cambridge: Cambridge University Press.
| **Metonymy** | Trope or figure of speech whereby one element is not designated by its own signifier, but by the signifier of another closely associated element, based on the contiguity between the two elements. E.g. *The old Rembrandt has been hanging in the gallery for weeks* (a painting made by Rembrandt). Note that a metaphor creates new links between otherwise distinct conceptual domains, whereas a metonymy relies on the existing links within them. |
| **Hyperbole** | Trope or figure of speech based on exaggeration in which an extravagant statement is intentionally made for emphasis or effect, e.g. *He is older than those hills*. |
| **Somatic (adj)** | Relating to the body (Greek σωματικός; “of the body”). |
A Pattern Language for Designing Pattern Dialogue Workshops

Takashi Iba
iba@sfc.keio.ac.jp
Faculty of Policy Management, Keio University

Abstract
In this paper we present a pattern language for designing workshops where participants mine their own experience and talk about them with a pattern language. This language is composed of ten patterns describing the design knowledge for carrying out an effective workshop to use a pattern language: Pattern Dialogue Workshop, Experienced or Not, Talking in Patterns, Expanding with Patterns, Connecting the Unconnected, Open Space, Hot Music, Missing Patterns, Distribution of Experience, and Future Design. We anticipate that this language will help people introduce a pattern language into their community by carrying out the workshop.

1. Introduction
When introducing a pattern language to your community, only the most active members will be willing to learn the patterns. People who do not feel the need are reluctant to learn the patterns, because it requires them to put effort into it. The dialogue workshops introduced in this paper is intended for members of a community to adapt to a pattern language with enjoyment. For example, you can hold a workshop where participants talk about their experiences with each other using the pattern language. The enjoyment in the workshop will help participants to think, learn, and talk about the patterns.

The Pattern Language for Designing Pattern Dialogue Workshops we present in this paper is based on our experience of 25 workshops for various targets, in various domains, and in various places¹. In what follows, we will explain the types of workshops we held, and then present the patterns for designing such a workshop.

¹ The breakdown is as follows: 19 workshops with the Learning Patterns [1-5], a workshop with the Presentation Patterns [6,7], 2 workshops with both of the Learning Patterns and Presentation Patterns, 1 workshop with the patterns of Fearless Change [8], and 2 workshops with pattern languages that students made in the class “Pattern Languages” at Keio University from 2010 to 2012.
2. Dialogue Workshop with Pattern Languages

The workshops described by the following patterns will be called the “Pattern Dialogue Workshops.”

The workshops consist of the following three phases: Experience Mining, Dialogues, and Reflection. In the first phase, Experience Mining, participants recall their experiences using the pattern language and make a list of patterns they have already experienced. In addition, they make a list of patterns they want to master in the near future. The second phase is Dialogues, the heart of this workshop. Each participant seeks others who experienced patterns he/she wants to master, and listens to their episodes. It is worthwhile to emphasize here that the pattern language functions to encourage participants to talk about their experiences, although people hardly talk about their experiences in their daily life. In the third phase, Reflection, participants talk about their discoveries from the workshop concerning their experiences and learn from the whole distribution of experiences of all participants.

As an example, we would like to introduce the case of the Learning Patterns, a pattern language for creative learning [1-4]. The Learning Patterns consist of 40 patterns describing practical knowledge for noticing problems, and to solve it in learning. We have held 19 workshops so far using the Learning Patterns. Holding a workshop using the Learning Patterns provides a good opportunity to encourage learners to talk about their ways of learning in their community. The workshop is designed so that the participants can talk about their experiences of learning in the light of the patterns [9,10] (Figure 1). First in the workshop, all participants would make a list of patterns that they have already experienced, and chooses 5 patterns that they want to gain in the near future. Then, they would walk around looking for a person who has experienced the learning pattern that they’ve put in the list of patterns to use in the future. When a match is found, they would listen to the episode of the learning experience from them. Through the workshop, participants acquire a deeper understanding of the patterns. Furthermore, the workshop provides shared experience using the pattern language in their community, and thus the participants may come to talk easily about their experiences and knowledge in their community.

Whenever we hold these workshops, we are surprised at the excitement of the participants. Based on our observations and surveys, the pattern language was used as a tool for reflecting on their experience, and the workshops provided a good opportunity to understand the meaning of each pattern and to talk about how to learn with others.
Figure 1: Pattern Dialogue Workshops with the Learning Patterns
3. Patterns

The pattern language proposed in this paper is composed of ten patterns describing the tips for carrying out an effective workshop to introduce a pattern language into a community. Figure 2 shows the overview of this language.

![Figure 2: The structure of the language](image)

The language is started off by Pattern Dialogue Workshop (No.0) at its base. It describes when and why this Pattern Language for Designing Pattern Dialogue Workshops is necessary. The first group of patterns is related to the mission of the workshop: Experienced or Not (No.1), Talking in Patterns (No.2), and Expanding with Patterns (No.3). Then, the second group is related to the actual workshop settings: Connecting the Unconnected (No.4), Open Space (No.5), and Hot Music (No.6). Finally, the third group describes how the reflection session for the workshop can be brought out: Missing Patterns (No.7), Distribution of Experience (No.8), and Future Design (No.9).

Each pattern is written in the same form: Pattern Name, Introductory Sentence, Picture, Context, Problem, and Solution. The Pattern Name is an attractive and memorable label for the pattern which it can be referred to by; the Introductory Sentence and the Picture serve as an introduction to the pattern which the reader can get a grasp of the pattern at a short glance; the Context is the environmental conditions for applying this pattern; the Problem describes a difficulty that often occurs in the situation of the Context; and finally the Solution describes how the Problem can be resolved. Note that the Problem statement is written in bold-typed sentence just after the heading ▼. In this context,” and Solution statement is also in bold just after the heading “Therefore.”
No.0

Pattern Dialogue Workshop

Make an opportunity for people to become familiar with the pattern language.

* * *

You want to introduce a pattern language into your community.

▼ In this context

There are passive members of the community who will not actively read the pattern language. People who do not feel the need are reluctant to learn the patterns, because it requires effort for them to read and think about new concepts. In result, the pattern language will not function as a common language in your community

▼ Therefore

Hold a workshop for members to use the pattern language with enjoyment. For example, you can hold a “Pattern Dialogue Workshop,” where participants talk about their experiences with each other using the pattern language. The enjoyment in the workshop will help participants to think, learn, and talk about the patterns.
Experienced or Not

It is easy to answer whether
one has experienced a pattern or not.

*      *      *

You've decided to offer a workshop to help participants adapt to a pattern language with enjoyment, and are designing the mission of the workshop. You are thinking of how the participants can learn and understand the patterns.

▼ In this context

The participants will become exhausted trying to read through the catalog of patterns. Many of the patterns would require special experience in order to understand them.

▼ Therefore

Make the participants pick up patterns they have already experienced and make a list of them, as a preparation for the workshop. This task is easily completed since they only need to focus on the patterns they can understand. In other words, they don't have to distinguish between the patterns they haven't experienced and the patterns they cannot understand.
You've decided to offer a workshop where participants adapt to a pattern language, and are designing the mission of the workshop. You are thinking of how the participants can come to communicate using the pattern language.

In this context

Participants will not use the patterns as vocabulary in their daily life, even if they understand the patterns. It is probably because they are novices in using pattern languages.

Therefore

Design a mission where participants can practice talking in the pattern language at the workshop. Through this experience, the participants are expected be able to talk in the language without hesitation.
Expanding with Patterns

Patterns can expand the experience in the near future.

* * *

You've decided to offer a workshop where participants adapt to a pattern language with enjoyment, and are designing the mission of the workshop. You are thinking how the participants come to use the pattern language as a guide for designing the target of the pattern language.

▼ In this context

Participants will not use the patterns as a guide for designing in the target domain, even if they understand the patterns. It is probably because they are novices in using pattern languages.

▼ Therefore

Make participants choose and create a list of patterns they want to master in the near future. Making a list of patterns they want to gain is the first step to designing their own way using a pattern language.
Connecting the Unconnected

Set game rules to encourage participants to perform out of the ordinary.

* * *

You've decided to offer a workshop where participants adapt to a pattern language with enjoyment, designed the mission, and are deciding the specific rules of the workshop.

▼ In this context

Some of participants are too shy to speak to others in the workshop. So they tend to talk only with people that they already know well, and then they will miss a good opportunity to know new things.

▼ Therefore

Forbid participants from talking to their friends or acquaintances. This rule encourages participants to speak to other participants that they do not know. Once they start to talk, the pattern language and the workshop settings should make them comfortable to keep talking.
Open Space

The environment changes the mood among the people.

* * *

You've decided to offer a workshop where participants adapt to a pattern language with enjoyment, designed the mission, and are thinking about the place for the workshop.

▼ In this context

The participants will feel too formal to enjoy talking if the workshop place is ordinary. It is because the nature of the space affects the mind of participants.

▼ Therefore

Choose an open-air space or glass-sided room where the participants can feel good and relax. Consequently, they will talk easily and the workshop experience will be memorable.
No.6

Hot Music

No music, no workshop.

* * *

You've decided to offer a workshop where participants adapt to a pattern language with enjoyment, designed the mission, and are thinking about the environment for the workshop.

▼ In this context

The participants won't walk and talk actively, especially at the beginning of the workshop.

▼ Therefore

Play groovy music that augments a scene where the participants will talk and walk more actively. It is good to play the music at a comparatively high volume at the beginning, because you can immediately make the mood of the scene livelier. And, when closing the dialogue session, you can emphasize the end of the workshop to the participants by stopping the music.
No.7

Missing Patterns

Where are the missing patterns?

* * *

You've decided to offer a workshop where participants adapt to a pattern language with enjoyment, designed the mission and environmental settings, and are designing the reflection session.

▼ In this context

The participants often have patterns that they could not find anyone who had experienced the pattern within the given time. As a result, they feel uncomfortable and may think that those patterns are insignificant.

▼ Therefore

Ask for people with patterns that they could not find anyone who with the experience, and match them up with those with experience of the pattern. By finding and meeting people with experience with the pattern, they will realize that the pattern is indeed important.
Distribution of Experience

Understand yourself in respect to
the whole distribution of experiences.

*      *      *

You've decided to offer a workshop where participants adapt to a pattern language with enjoyment, designed the mission and environmental settings, and are designing the reflection session.

▼ In this context

Participants have no chance to understand the patterns that they did not choose.

▼ Therefore

Show graphs that summarize the distribution of the experience of all participants for all patterns.
If you use a program to analysis the data from participants, you can summarize the data on the spot, and show the graphs at the end of the workshop.
Future Design

Design a plan to achieve your goal using the pattern language.

* * *

You've decided to offer a workshop where participants adapt to a pattern language with enjoyment, designed the mission and environmental settings, and are designing the reflection session.

▼ In this context

The participants will easily forget the episodes they listened to in the workshop.

▼ Therefore

Encourage participants write their future vision based on what they learned in the workshop with the pattern language. It can be homework after the workshop.
4. Conclusion

In this paper, we presented a pattern language for designing workshop of experience mining and dialogue with a pattern language. Through this kind of workshop, participants will experience a new way of applying pattern languages.

In our view, the evolution of the method of pattern languages can be divided into three generations: Pattern Language 1.0, 2.0, and 3.0 [11]; In the Pattern Language 1.0, pattern languages were used as media for bridging the gap between designers and users; In the Pattern Language 2.0, pattern languages were used as media for bridging the gap between expert and non-expert designers; In the Pattern Language 3.0, pattern languages are used as a media for connecting people who have different experiences (Figure 3). We anticipate that these workshops would be a good demonstration of typical applications of the Pattern Language 3.0.

<table>
<thead>
<tr>
<th>Pattern Language</th>
<th>Object of Design</th>
<th>Act of Design</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3.0</strong></td>
<td>Forms of Human Action (Innovation, Education, Learning, Presentation, Collaboration)</td>
<td>Design Act is Embedded in Actions over Time</td>
<td>Connecting People who have Different Experiences</td>
</tr>
<tr>
<td><strong>2.0</strong></td>
<td>Non-Physical Forms (Software, Interface, Organization)</td>
<td>Design Act is Iterated over Time</td>
<td>Bridging the Gap between Expert and Non-Expert Designers</td>
</tr>
<tr>
<td><strong>1.0</strong></td>
<td>Physical Forms (Architecture)</td>
<td>Design Act is Basically Carried out in a Period</td>
<td>Bridging the Gap between Designers and Users</td>
</tr>
</tbody>
</table>

Figure 3: Three Generations of Pattern Languages
Acknowledgement

I would like to do our best to thank Georgina Holden for shepherding our paper, and all participants of writers’ workshop in EuroPLoP2012, especially Christian Kohls and Christian Köppe. I also would like to thank to Mami Sakamoto and Eri Shimomukai for collaborating to hold several workshops, and to Taichi Isaku for helping to polish the expressions in the paper.

References

[1] Learning Patterns Project, Learning Patterns: A Pattern Language for Creative Learning, in Japanese, Faculty of Policy Management & Faculty of Environment and Information Science, Keio University, 2009


Besides, knowing their methods and tools, (software) engineers need to be competent in soft skills such as communication, empathy and self-awareness. As of today, most trainings focus on teaching methods and tools, where the soft skills are left to be trained by some pedagogical training that itself often focus on the introduction of methods as well, but does not apply soft skills to the actual setting in which they will occur in normal day life.

This pattern introduces a guideline for a training format as a solution to train soft skills in the participants’ specific setup. The introduced format is flexible in that way, that its execution can be adjusted to specific backgrounds by using the storytelling approach.


General Terms: Human Factors, Documentation

Additional Key Words and Phrases: Training, Format, Requirements Engineering, Soft skills

ACM Reference Format:
Hoffmann, A. 2012. Game Language. – L(January YY), 23 pages.

Imagine you are a trainer and you are standing in front of a bunch of people in a classroom-alike atmosphere. Your task is to train them in some factual knowledge as well as the corresponding soft skills. Corresponding soft skills are those competencies your participants need to have to successfully apply their newly gained factual knowledge outside the training room.

Your participants might not be especially motivated to pay attention to your training. It was their boss who send them, and their daily work might still be there to be done while they are attending your training. Consequently, some of your participants might even be sitting behind their laptops expecting to be working while listening to you. And now you want to gain and keep their attention and get their focus to whatever your training topic is.

In addition, you know very well, that learning theory in a classroom-style training will not enable sufficiently the learning you want (see e.g. [1]): Immediately understanding the factual knowledge’s application within their real world scenarios is what the participants will benefit most from.

Throughout the last years, I developed a workshop format to train soft skill awareness in project environments. This workshop format is called ImProject, and a variant of it is called REIM. ImProject stands for “Improvisation Theater Techniques in Projects” and REIM stands for “Requirements Engineering and Improvisation Theater Techniques”.

The workshop originates from my professional experience as a project manager and requirements engineer. It is based on my knowledge about and passion for improvisation theater.

The workshop format addresses an open point within the trainer’s community of industrial training: it teaches factual knowledge and soft skills related to factual knowledge together. Several workshops using the formats have been successfully executed. A detailed description about the workshop can be found in [7]. A guideline for trainers how to run such a workshop is described in [6].

Being a trainer in the above scenario, you might want to use one of the workshop formats or simply some interactive game to make your training more efficient and to attract the participants.

In addition, you know that interactive games create corporate feeling. As corporate feeling means each participant wants to be included, wants to be part of it, this is what you want to aim for. In addition, corporate feeling creates an
atmosphere where the focus of the participants automatically moves to the session itself. Consequently, you have
their attention!

Imagine, you decided to attract your participants’ attention by using some games from the workshop formats
REIM or ImProject, but you are rather unexperienced using such interactive elements in general. Thus you might
be asking yourself:

Where do I learn about interactive games? And how do I know efficiently which ones to choose?

Interactive elements such as interactive games can be very efficient to channel learning in training environments.
The trainer simply need to choose the right one, and execute it. But how do you know which games are the right
ones?

The answer, which game suits you best, highly depends on the training situation, such as participants’ back-
ground, topic to train, anticipated learning objectives and so forth.

Imagine, as a trainer you have your criteria for choosing the games prepared. Now, you simply need to choose
from a set of games. You would probably like to have the opportunity to screen existing games by your criteria and
select the most suitable ones.

So, what you are basically looking for is a catalogue of games which are structured by these criteria. In addition,
this catalogue should come with an instruction how to choose from these games the most suitable game for your
situation. This would allow you to select appropriate games easily.

1. THE GENERAL PROBLEM STATEMENT: OR WHAT THIS GAME LANGUAGE SHALL BE THERE FOR

This paper introduces a structure for describing games not only used in the workshop formats REIM and ImProject.
The structure is divided into so-called structural elements. These structural elements are defined in Appendix A.
The structural elements are named in accordance with the trainer’s language and were build upon buzzwords
industrial trainers are used to.

While developing the structure and its structural elements, the following criteria had been identified as criteria
for a “good” structure:

—Easily understand what problem can be addressed or solved with the game.
—Easily determine dependencies between games.
—Easily understand to vary a game to adopt if participants have a different background.
—Easy re-use of a game sequence (called Game Chains) that have been successfully executed.
—Easily compare games to select the optimal one.

As a result, the reader can use this format to easily retrieve all information needed to

—access a game’s learnings,
—choose the appropriate ones for the given situation and
—understand how to execute them.

In its workshop version for discussion during EuroPLoP 2012 in Kloster Irsee in Germany, this paper presented
two similar games using two different structures. One structure was in alignment with a typical pattern structure,
namely using Problem, Solution, Forces etc. as section headlines. In this structure the structural elements were
added only as sub-headlines for better orientation. The second structure focussed on the headlines of the structural
elements and the pattern structure remained (invisible) in the background.

The main aim of that workshop version was to receive feedback on the two structures, and to consequently
decide upon the format to be used to document further games.

As a result from the workshop’s feedback, the second - non-typical pattern structure - was selected. The
assumption, that the typical reader (who is a trainer!) will - in most cases - not be experienced in patterns, was
verified during the workshop.
Thus this paper presents the final structure where the typical pattern structure remains in the background, and the buzzwords the reader would typically be searching for are used as headlines.

The mere description of the structural elements is somewhat dry and boring, and does not give a real benefit for the reader in the first place. Therefore, the structural elements are introduced by using them to describe the games “Company Game” and “Requirements Game” in the Section Training Games (see [7] for details). The definition of the structural elements can be found in Appendix A for reference.

Both games are based on the same improvisation theater game, called Pattern Game (see [15], p. 48f), see Figure 4.

Audience
This language and its pattern is dedicated to Industrial Trainers who would like to execute REIM or ImProject, or just would like to re-use one of the games as an interactive element for their workshop or training.

The workshop formats REIM and ImProject have been developed in the industrial training environment and proved to be of value in these types of training. It is however assumed that this approach can be useful for educational lecturing e.g. of undergraduates as well. Therefore, Lecturers and Teachers are seen as an second audience as well.

Additionally, Facilitators, Project Managers and other roles in charge of executing workshops receive new ideas as well.

2. BACKGROUND

One cannot teach a man anything,
One can only enable him to learn from within himself.
- Galileo Galilei

In this section, some of the underlying aspects and thoughts are recapped in order to make the motivation of the author more transparent.

2.1 A Few Words About Learning

Why should factual knowledge and softs skills be taught together in the first place? What is the motivation for doing so, instead of remaining with the concept of dividing trainings into factual training (such as learning about project management methods) and soft skill training (such as learning about my leadership style)?

The need to combine these two into one shall be briefly exemplified on the International Competence Baseline (ICB) of the International Project Management Association (IPMA). The IPMA is one of the biggest (if not the biggest) community of project managers. The IPMA is owner of the four-step education and certification model for project managers which is like the Project Management Institute (PMI) internationally recognized and accepted.

In their recently approved IPMA Competence Baseline Version 3.0 [2], the IPMA introduces their “Eye of Competence”, which distinguishes different areas of competence in order to live up to expectations that arise from “fast changing context with many interested parties and external influencing factors” [2]. This “Eye of Competence” is divided into

—Behavioural Competence,
—Contextual Competence and
—Technical Competence.

Aligning these technical terms with the ones used within this paper Behavioral Competence equals to soft skills, Technical Competence to factual knowledge.

On page five of their IPMA Competence Baseline Version 3.0 technical competence and behavioral competence are clearly connected: “To develop and realise good project plans and results, the project manager’s behavioural competences, such as motivation and leadership, are an essential addition to his technical competence”.

Game Language — Page D7-3
2.2 Why Games?

The reader might wonder, why especially games serve the purpose of transporting the knowledge and drive the experience of soft skills. Games are generally good enablers for learning, but why?

—While the participants are playing a game, they are gaining immediate experience.
—To experience something is much more than being told something is true, it is best described by reformulating it to “knowing it by your own experience”.
—Experiencing something yourself also implicitly includes the aspect of reflecting over what you did, how you did and why you did do something.

It is widely recognized that learning is made in different steps and that in the end experience is the resulting consequence. For example, in his book Serious Play ([12]) Michael Schrage connects the need of playing with prototypes and learning as he states “… the value of a prototype arises from how people behave around prototypes…” ([12], p. 15). Interacting with and around prototypes is what Schrage calls “Serious Play” Thus, by playing around innovation is nothing else than the result of continuously learning while playing around. Schrage further quotes a Yahoo! engineering vice president: “Experience is essential. It is the only thing that lets you see how the whole system works” ([12], p. 87).

Especially on computer games, some investigation has been undertaken that goes even as far as a patterns on best practices to mathematician games for students have been made available [10].

Furthermore, we use games to integrate fun into the learning environment. Having fun is the channel to experience in flow. This goes back to Csikszentmihalyi and his well known research on flow [3].

2.2.0.1 Please Note:. You might see participants that object against games during workshops, because they simply consider this activity childish, indicating that this is “something for children” and thus does not have at all something to do with learning.

to avoid this situation never ask, if participants would like to play a game, but simply starting to play it. Throughout more than 10 workshops with participants from all over the world there has been one single person only who refused to play.

2.3 The Storytelling Aspect

Storytelling refers to the ancient skill [13], and has recently been put into new context for knowledge managers [14], product managers [8] and others, e.g. [11], [4].

In short, storytelling is the art of telling a story in the right way to transport a message. The probably most known purpose are fairy tales, in Germany the fairy tales from the Grimm brothers [5].

In context with our games, storytelling needs to be considered on three different structures:

(1) Definition and General Introduction: What is storytelling, and how does it work?
(2) The Technique for the Games: What is important to be told in the stories that are used in the games?
(3) Concrete Example: Examples of stories how they have been successfully applied in the games.

2.4 General Structure of the Games

With respect to the anticipated learning, each game has a similar structure: The game creates a small learning (“AHA Effect”) within the game’s setup. This “AHA Effect” is then transferred into the learning experience within the factual knowledge area under consideration.

This “Real World Experience” is created by the trainer by first addressing the learning experience within the game’s setup, and then aligning this learning experience with an example of the factual knowledge area (“Real World Example”). Understanding the “Real World Example” finally enables the learning within the participants’ daily work life (“Real World Reality”). Figure 1 illustrates this mapping from a game to the real world.

Game Language — Page D7-4
Through the game, the participants gained experience within the game’s setup and environment. Associating the game’s AHA Effect to their reality, the participants are transferring their awareness (and experience) into real life.

As the awareness and experience are associated with the game, the learning remains more present and is recalled better than in case of “just being told the facts”.

For instance, if Requirements Engineering (RE) should be the factual knowledge area under consideration, then the Requirements Game in Section 4.2 is one of the games to be used. Having played this game, initial transfer for awareness from the participants about situations in RE is created.

As described above, the storytelling element is used to enable the transfer from the just experienced game behavior to the related situation e.g. requirements engineering.

For an example, the reader is referred to “Motivation” in the game in Section 4.
2.5 Transfer Task
After having played one or several games, the transfer into real life, meaning daily work life environment is needed. Following each game a quick wrap up of the game should be done by the trainer. This ensures immediate feedback on the participants observations and experiences during the game. Most often participants report that they observed (or even experienced) a situation, which they would like to change in the future.

Thus, it has become a good practice to close the entire session, by recapitulating all games and then ask each participants for his or her major take away of the day, followed by a concrete situation in which the participant will notice his or her change of behavior.

2.6 Forces
This section introduces general drivers for using the games. The following forces are general in the sense, that they are part of each and every game to come.

1. Theoretical teaching by stating facts does not bring the teacher nor the participants forward, if the insight or enlightenment is missing.

2. Even if participants have already gained some factual knowledge, can we see weak points during the execution. This is especially true in situations, where the application of methods (learned as factual knowledge) creates negative emotions between two parties.

3. The weak spots in one’s application of factual knowledge and corresponding soft-skills are often difficult to detect, because they are context-dependent.

4. The participant’s unawareness of his ability or inability to correctly apply both a “correct” factual knowledge and an “appropriate soft skill” in a certain situation is often difficult to uncover.

5. Because it goes into the personnel self understanding, for some of the participants it might be difficult to give or receive feedback on how he or she behaves while applying factual knowledge.

6. Each participant has a different weak spot. Thus an approach to address several aspects at the same time is needed.

7. The trainer’s preparation time is limited. The solution needs to allow quick access to the issues to be trained without requiring in depth understanding of each participant’s background.

8. In industrial training participants either do not know each other or are working together on a daily basis. In the former case, some team building and warming up needs to be included, in the latter the solution additionally needs to be very “face saving” and must not at all include some finger pointing.

The remainder of this paper is structured as follows:
Section 3 gives general information about the games and introduces the different types of games as in Figure 3. In Section 4 the structural elements are introduced by documenting the “Company Game” and the “Requirements Game” using this elements. The structural elements themselves are defined in Appendix A.
Section 5 concludes the paper and outlines the activities for future work to further enrich this game language.

3. TYPES OF GAMES
The workshop format from which the games originate follows a typical improvisation theater training session. Each improvisation theater training session has several phases: Warm-Up, Training and Feedback. For each phase, there are specific types of games, see [7] for further details. The workshop format is thus divided into several phases as well, and each of its phase requires to use specific types of games.

In the workshop format, Warm-Up Games are used during “Warm up” and “Getting to know each other”, see Figure 2, and Training Games are used during “Main Section”.

Game Language — Page D7-6
Towards the "Main Section", there is another type of games to be used. These games are called Closure Games. Closure Games focus on the learning of improvisation competencies such as learning to tell "good" stories, create "good" characters. The name Closure Games stems from the fact, that these games are closing the "Main Section". Figure 3 illustrates how the type of games are connected.

3.1 Building Structure

Throughout each workshop format a General Context should be set for all games. This General Context is determined by the participants' background and their main roles within their work life or by the anticipated learning area. For the Company Game (Section 4.1) these are project members in general, where the Requirements Game (Section 4.2) is dedicated to Requirements Engineers.

Each game consists of three parts: an improvisation theater game, a game instruction, and a storytelling element that uniquely maps the game to the general context and the problems you want to address. The game instruction describes the needed interactive steps plus instructions how to execute the game. The instructions might be variants of already known games or interactive workshop elements within the trainer community. The storytelling element addresses the general context and describes story elements with which to address the anticipated learning objectives.
The remainder of this section introduces the identified types of games.

3.2 Warm Up Games
This section describes games that are used to warm up the group, help to loosen up and forget about the rest of the day.

The games set the full concentration of the participants towards the workshop and create the initial corporate feeling.

3.3 Training Games
This section contains the core games. These games are used to train soft skill competencies using the factual knowledge. The factual knowledge environment is created by telling the right story. This is described in the Storytelling Element of each game.

If two games originate from the same improvisation theater game, they will be very similar in setup and execution. It is then the storytelling element that makes these games distinguishable. In order to allow the reader to easily detect these relationships, these games have a reference to each other in “Variants”.

3.4 Closure Games
In this section games are described that train aspects of ‘good’ storytelling or skills an improvisor utilizes on the stage to perform a good improv show.

These games are mostly executed in smaller groups. To run these games, the trainer typically introduces the game’s rules and then steps back to let the participant experiment, train and learn within their groups. The trainer observes the interactions within each group. Occasionally, the trainer might step in and highlight specific moments within one group either to the corresponding group only or to all groups. Because the trainer steps back, these games do not include storytelling elements as a structural element. But they train the participants in telling better stories themselves.
4. TWO TRAINING GAMES

4.1 The Company Game

Context. This game is dedicated to project managers and project team members who shall be trained in both factual knowledge about project management and soft skills. It is suitable for beginners and experienced participants.

Motivation. This might be a typical but stereotyped dialogue between a trainer and a participant that would call for playing this game:

Trainer: “Implicit prioritization is normal. You as project manager need to especially pay attention to implicit prioritization as it occurs naturally, and can be found everywhere.”

Participant: “But that is stupid. People should follow procedures, they formerly agreed upon and not do what they want, and simply call it implicit prioritization just because the are not paying attention. This is just not professional. Besides, we do have a project management method within our organization that clearly states what to do when and why.”

In a normal training, this discussion might go on, the trainer and the participant might argue with each other. The trainer might argue with a participant who does not see an issue with potential unawareness, and who is convinced that implicit prioritization occurs only because people are not paying attention and are not willing to obey rules and follow procedures. The participant might argue that in his company there are clear rules how to prioritize, that he and his colleagues have been training in prioritization methods and that they are thus fully aware what they are prioritizing when and why.

By using this game, the trainer creates awareness that each and everyone is implicitly prioritizing no matter how many prioritization methods and rules one knows.

Consequently, not following the rules does not necessarily implies that someone does not want to follow them, but this could simply mean that someone is not aware that he is not following them.

Short Description. The number of participants shall be 6 to 20, where 10-15 is optimal. The participants shall be ready to try out something new. Thus, the game can be played as part of the workshop formats during the “Main Section”. No further preliminaries are needed.

Duration Time. This game needs some time. An experienced trainer might be able to train this game within 45 minutes. But the bigger the group of participants is, the longer it takes to play this game. A group of 20 people might need up to 90 minutes to complete this game.

Applicability. This game is dedicated to topics related to general situations that occur both within companies or projects. As project-related knowledge is mainly trained in the context of project management or general management courses, this game has a broad application area.

It can especially be used, whenever the roles of the participants are not clear to the trainer or very heterogenous.

Purpose and Learning Objective. This game trains and visualizes feedback to communication behavior immediately.

It particularly focusses on the aspect of awareness. It trains awareness among the participants, the awareness towards the given constraints of the game and awareness towards communication behavior of oneself and other, and ones communication type towards others. It thus also unveils unconscious attitudes related to passing information or collaborating with other persons or colleagues during daily work tasks.

Learning Objective

—The game demonstrates how little is needed, to be overloaded, and how long one needs to become aware that he is overloaded, and
—It unveils unconscious priorities of participants.
—It trains to become conscious about one’s own focus.
—It trains focusing on others, what they are doing, how they are feeling and what their needs are.
—The game trains awareness for setting the focus to what is needed in specific situations, and
—It trains the participant awareness to their own behaviors.
—It trains to be more aware about the needs of the other participants.

After having played this game

—Each participant is more aware how two-sided communication takes places.
—The participants experienced the consequences of not considering an other person’s needs while communicating with him or her.
—The participants have experienced how quickly they can become “busy” or even stressed by handling only a few (factual related) tasks at the same time.
—The participants became aware that they are not obeying rules in stressful situations even though, they think they are.
—The participants learnt that cultural behavior can interfere with factual related tasks without anyone noticing it.

**Challenge.** The technical principles and facts about priorities and priority setting within project environments are easily understood by the participants of a project management training.

It has however been frequently observed that project members, employees and the like do not follow these principles in their daily work. This is especially true where situations become emotionally intense and stressful.

**How can you as a trainer address the issues related to priority setting into your project management training?**

**How can you emphasize on soft skills and emotions that only arise while your participants are put under a certain stress?**

**Forces - What drives this challenge?**

1. In project management trainings we as a trainer often talk about the strong need for communication, but rarely ever train it - at least not in combination with the project management topic itself.
2. The own lack of knowledge of the participants of his or her own behavior in specific Project or Project Management-related situations.
3. Arrogance or Aburteilung of the communication partner like “I do know this topic better than you” creates imbalance between the communication partners and can be visualized and noticed.
4. Participants know they need to be competent in soft skills, but are not trained in it.
5. Participants believe, they are obeying the rules giving by the trainer, but are not aware that and when they are breaking them.
6. Participants believe they are respecting rules given in the daily work environment, but do not notice when they are breaking them.
7. If communication is part of a training, it is trained the technical way as well. E.g. the participants are trained in “active listening” by learning to keep eye contact, nodding to what the opponent says etc. This way, the trainer aim to train them how to actively create perception, to actively listening. But this does not train any awareness nor knowledge nor understanding about the participant’s own behavior in specific RE-related situations.
8. Prioritization sounds easy during the training session, but is difficult to realize in real life situations.
Consequences

1. Communication skills and factual Project Management topics are trained together.
2. The participants become aware of their own behavior in certain situations, such as forgetting to respect certain rules.
3. The participants are aware that respecting a lot of aspects at one time, causes them being overloaded, and consequently not being able to manage all of the aspects at one time.
4. The participants become aware of their own behavior, e.g. they realize they are breaking rules even though they formerly thought they were obeying them.
5. Participants receive an indication when and how often they might be forgetting or not respecting rules.
6. Participants are trained and consequently know, that only a few tasks to be solved in one time are needed, to be overloaded and loose track of the overall goal(s).
7. The participants experience two-sided communication.

Solution. Play a game, that requires to follow multiple rules and executes multiple tests at the same time. Use storytelling to set the game in the context to the project of company environment.

Implementation. Use the improvisation theater game “Patterns” (see [15], p. 48f) and tell the story that relates the game to the project managers’ environment.

The Pattern Game works as follows: Everyone is standing in a circle. Each participant puts his hand on his head. This gesture indicates that he is “free”. The trainer starts circulating a ball by throwing it to a participant. This participant takes his hand down, catches the ball and then throws the ball to another participant who is still “free”. This is continued until each participant has received the ball exactly once. The last participant throws the ball back to the trainer. Then the circulation starts again in the exact same order.

This is called the “introduction of a ball”. Whenever a new ball is introduced, this is repeated but the participant need to ensure they are throwing to another person than before. Thus, different circles are created.

Throughout the game you tell stories with project-specific topics. Observe the participants and correlate things that are happening while playing with the real world situation in a project or company (vgl. Section 2.3, Figure 1). Things to highlight can be manifold such as a special reaction of a participants, an unrecognized move, an disclosure about oneself (Johari-Window, see e.g. [9]) .

Some balls have associated names (such as the name of an animal or the name of the receiver).

1. Introduce the circle as a company or project where each participant is a department, and each ball is a tasks that needs to pass each department to be fulfilled. The given manner in which the ball is circulated is called “the established way through the company or project”. Each participant must receive the ball exactly once.
2. Start by throwing one ball and establish its way through the company. You might want to repeat the introduced circle until every single participant recalls it.
3. Stop throwing this ball, and remind the participants to remember the ball’s way.
4. Introduce a second ball in the same manner.
   You might further associate this ball with the name of an animal or the like: Each participant shouts an animal’s name before throwing the ball. Each name is to be used exactly once. Again, repeat this ball until each participant recalls it way. Then stop throwing it.
5. Throw the first ball in its “established way” again.
6. While the first ball is circulating correctly again, start the second ball as well.
7. Continue to introduce further balls in the described manner until your groups gets stuck.
8. After having played with several balls for a while, stop the circulation of the balls. Then take a glass, introduce it in the same way as you introduced the balls before.
Start introducing the balls again, as soon as the participants became used to throwing the glass.

This game is setup such that it provides an environment where the participants feel comfortable, are enjoying themselves and are trying out something new.

At the same time, the participants experience themselves, experience their behavior towards others as well as their behavior together with others during prioritization.

**Implementation - Tips and Tricks**

— Extend the trawling of the ball rules by the following statement right at the beginning in order to avoid that the participants try to pass on the glass instead of throwing it: Direct neighbors are not allowed to be chosen.

— Do not introduce all rules at the beginning. The participant will not obey all of them anyway. Additionally, violating a rule that had not been made explicit before, is a learning in itself (Just recall how often we do have this situation during daily work: there is no explicit rule, but everyone knows not to do certain things).

— Some of the participants might urge to understand what is going to happen during the game before actually starting the game. Instead of telling them beforehand the complete outcome of the game, you should explain that the rules of the game will be introduced as the game proceeds. This additionally prevents participants’ impression that “you had forgotten to introduce some of the rules”.

To ensure learning this game needs a trainer who is experienced in projects and can align corresponding consequences to observed behavior during this game.

**Limitations.** This game should not be used, if the participants haven’t be woken up and are not into the shape of playing an interactive game In case you want to use it, play a Warm up Game (see Section 3.2) first.

This game cannot be played without an experienced trainer who ensures to talk about the consequences in work live of observed behavior during this game (The game needs to be instructed and guided.).

**Storytelling Element.** The storytelling element first sets the scene, e.g. the following story is introduced with the game: “Imagine you are all working in the same company. Each of you is working in a different department. A new task is introduced. The task affects the whole company. This means, in order to fulfill this task, it needs to pass every department exactly once.”

Do not forget to correlate things that are happening while playing with the real world situation in a project or company (vgl. Section 2.3, Figure 1). Things to highlight can be manifold such as a special reaction of a participants, an unrecognized move, an disclosure about oneself (Johari-Window, see e.g. [9]) .

Some of the rules will be forgotten or broken as the game goes along and the participants are introduced to more and more rules and activities. Whenever a rule is broken, the trainer uses the storytelling element to align the demonstrated behavior during the game to normal day work. Considering the rule that all tasks are un-prioritized implies in fact, that all tasks shall take the same time through the organization and non shall be prioritized over the other. In fact, the game is setup such that some balls (that represent the tasks) are more easy to be handled than others. Thus, it provokes balls being faster than the others. Here, the storytelling element is used to set the correlation to daily work, for instance by saying: “You did not notice that this ball was faster than the others. Imagine this happens during your normal work as well. Before this game you might have neglected that this happens, but can you guarantee that it does not?”. You could and should even become a bit provocative.

Some of the rules are repeated with the introduction of the story: The tasks are un-prioritized, meaning they have the same priority. But, the tasks are also very fragile, so they must not fall down.” While the trainer talks about a task, he shows the corresponding ball.

**Storytelling Example**

1. Asynchronous Communication. Ensure the Receiver is receiving messages.
(a) Even though the balls were introduced as being breakable, the balls continuously fall on the ground. Often the trainer can observe that the sending participant pass the balls as if in a hurry, not taking the time to wait until the receiving participants pays attention and “is ready to catch the ball”.

(b) Make the analogy to the information model: “The sender sends a message, the receiver receives it. Or he does not receive it.” Dropping a ball equals to dropping (and loosing) a message!

(c) Make the analogy to real world’s general context and tell your participants: “When you are handing over your finished tasks to your project manager, you need to ensure, your project manager is truly listening to you. Delivering a tasks include, to ensure your project manager understood that you said “I am ready.”, and you will not be working on this task anymore unless he tells you so. This is also true vice versa: As a project manager you always need to ensure, that your team member understood a task you gave him exactly how you meant it. The same situation occurs also whenever you are talking to your stakeholders: Are you listening? How well are you listening? Are you understanding what they want to say? Or are you just hearing what you want to hear?”

LO: In stakeholder management it is very important to remember the exact quote of what a stakeholder desired from a project. The stakeholder will not remember what you as a project manager noted down, he will only remember what they said, often exactly how he said it. Therefore: Know the requester and his original statement, because this is and will be what the requester will be looking at at any time during the development process and especially at the end when finally testing and accepting your system.

(2) “Asynchronous Communication” - The information who was the sender of the information, a message or the requirement is lost throughout the time.

(a) While passing the ball around, the participants might become lazy, and you as a trainer might observe that they become less aware the longer the balls are being passed around. Some participants might even state something like “Great. I only need to remember whom I have to throw the ball to.”.

(b) The participants nevertheless implicitly know from whom they are receiving the ball. Introduce the need for asking questions back and thus traversing back to the origin of the task. Let the participants circulate a ball in reverse order (“backwards”). Whenever they are asked, to circulate in “reverse order” this does not work well. It takes some time to unveil the implicit known information. Until then some confusion among the participants can be observed.

(c) When you do not remember from whom you should receive a task, how will you manage to answer questions that might relate to that task efficiently? Imagine, the task is to gradually fill out a form and the form is ill-filled when it arrives in you inbox. Now, you need to know the original sender, so you can go back and ask for questions. If you do not know immediately who to talk to, this task might become very time consuming. Or, imagine, you will leave for vacation tomorrow, and you know there is a task you need to fulfill before leaving, but it did not yet “arrive” at your desk. Now, knowing who would have had to send it to you, helps.

LO: Why is it important to know the original sender? To know to whom you will need to go back for question, to know the most important stakeholder. Additionally, if something is wrong, you want to know to whom you can go back to. Or imagine, you know the task should be passed to you, but it simply does not arrive. If you know from whom you get it, you can go into that department and ask for it.

(3) “Equally Prioritized versus Implicit Prioritization” - Implicit Prioritization

(a) Different forms of prioritization can be observed. E.g. the ball that has an animal name attached is passed on faster. Some participants might recall one ball better than the other, so in case they receive two balls, the one that is remembered easier is given priority. The glass is often observed to be a prioritizer as well: While handling the glass, it is often observed, that all other activities (including talking and laughing) are stopped, because the full focus is on the glass.
(b) Give examples for the observed priorities. Get confirmation from the participants, if they did noticed the described behavior as well. Explain that it is natural to easier receive attention from the receiver if shouting out his name or the animal name. Point out that you are adding communication channels, thus enriching the information transferred.

(c) Everyone knows very well: if you want to get things done quickly, approach the person who need to do the next step and ensure he or she is doing his or her part immediately. Here, you are manipulating someones priority list in your own interest. So, are you aware, how often you priority list is being manipulated by someone else, or even by yourself? Starting with the tasks that appear to be “easiest”, “fastest” or what ever you prefer to put as an adjective here, already gives priority to the tasks. This is normal. And the key behind all this is, to know and be aware that these implicit priorities exist, happen and will always be there.

LO: The participants understand that they are adding an implicit prioritization to their tasks. Always. This exercise demonstrates this behavior to each participant.

Suited or Not Suited for Group Situations such as .... This game is an ideal “icebreaker” within team-building situations, e.g. if participants of a workshop never have met before.

This game requires some hand-eye-coordination. The group can usually cope with one or two participants not being able to catch balls. If the group has a significant number of participants that are or might be handicapped with respect to throwing and catching a ball, do not play this game.

Requisites needed. The room shall have at least 40 square meters. Any chairs and tables in the room need to be put to the wall, so that there is enough space for a big circle formed by the participants and the trainer.

The following material is needed for the game: Up to three distinguishable, hand-sized balls (e.g. juggle or tennis balls), one old drinking or mustard glass and two board markers are needed.

Additionally, it shall be organized that a vacuum cleaner and dustpan and brush are at hand, in case the glass falls down and breaks.

Variants. This game is derived from the improvisation theatre game called “Pattern Game”([15], p.48f). A variant of the Company Game is the game called Requirements Game, see section 4.2.

4.2 The Requirements Game

Context. This game is dedicated to requirements engineers that shall be trained both in requirements engineering (RE) factual knowledge and soft skills.

It is suitable for beginners and experienced participants.

Motivation. This might be a typical but stereotyped dialogue between a trainer and a participant that would call for playing this game:

Trainer: “Implicit prioritization is normal. You as requirements engineers need to especially pay attention to implicit prioritization as it occurs naturally, and can be found everywhere.”

Participant: “But that is stupid. People should follow procedures, they formerly agreed upon and not do what they want, and simply call it implicit prioritization just because they are not paying attention. This is just not professional.”

In a normal training, this discussion would now probably go on, and the trainer would argue with the participant. The trainer would argue with a participant who does not see an issue with potential unawareness, but is convened that implicit prioritization occurs only because people are not paying attention and are not willing to obey rules and follow procedures.

By using this game, the trainer creates awareness that not following the rules does not necessarily implies that someone does not want to follow them, but this could simply mean that someone is not aware that he is not following them.
Short Description. The number of participants shall be 6 to 20, where 10-15 is optimal. The participants shall be ready to try out something new. Thus, the game can be played as part of the workshop formats during the “Main Section”. No further preliminaries are needed.

Duration Time: This game needs some time. An experienced trainer might be able to train this game within 45 minutes. But the bigger the group of participants is, the longer it takes to play this game. A group of 20 people might need up to 90 minutes to complete this game.

Applicability. This game is specifically addressed towards requirements engineers. It can however used for a group that consists of the different roles of a software development process as well (e.g. architects, testers and even developers). This makes especially sense when you would like to highlight or create the awareness with respect to the cross-role responsibly of requirements, the need to communicate about requirements in a decent manner.

Purpose and Learning Objective. This game trains immediate feedback to communication behavior related to requirements engineering.

It particularly focusses on the aspect of awareness. It trains awareness among the participants, the awareness towards the given constraints of the game and awareness towards communication behavior of oneself and other, and ones communication type towards others. It thus unveils unconscious attitudes related to passing information on or collaborating with other persons or colleagues during daily work tasks.

Learning Objective Generally, this game focusses on training awareness in various aspects of the requirements engineering tasks.

—The game demonstrates how little is needed, to be overloaded, and how long one needs to become aware that he is overloaded, and
—It unveils unconscious priorities of participants.
—It trains to become conscious about one's own focus.
—It trains focusing on others, what they are doing, how they are feeling and what their needs are.
—The game trains awareness for setting the focus to what is needed in specific situations, and
—It trains the participants awareness of their own behavior.
—It trains to be more aware about the needs of the other participants.

After having played this game

—Each participant is more aware how two-sided communication takes places.
—The participants experienced the consequences of not considering another person's needs while communicating with him or her.
—The participants have experienced how quickly they can become “busy” or even stressed by handling only a few (factual related) tasks at the same time.
—The participants became aware that they are not obeying rules in stressful situations even though, they think they are.
—The participants learnt that cultural behavior can interfere with factual related tasks without anyone noticing it.

Challenge. The technical principles and facts about priorities and priority setting within requirements engineering are easily understood by the participants of a requirements engineering training.

It has however been frequently observed that requirements engineers do not follow these principles in their daily work. This is especially true where situations become emotionally intense and stressful.

How can you as a trainer address the issues related to priority setting into your requirements engineering training?
How can you emphasize on soft skills and emotions that only arise while your participants are put under a certain stress?

Forces - What drives this challenge?

(1) In RE trainings we as a trainer often talk about the strong need for communication, but rarely ever train it - at least not in combination with the RE topics.

(2) The own lack of knowledge of the participants of his or her own behavior in specific RE-related situations.

(3) Arrogance or Aburteilung of the communication partner like “I do know this topic better than you” creates imbalance between the communication partners and can be visualized and noticed.

(4) Participants know they need to be competent in soft skills, but are not trained in it.

(5) Participants believe, they are obeying the rules giving by the trainer, but are not aware that and when they are breaking them.

(6) Participants believe they are respecting rules given in the daily work environment, but do not notice when they are breaking them.

(7) If communication is part of a training, it is trained the technical way as well. E.g. the participants are trained in “active listening” by learning to keep eye contact, nodding to what the opponent says etc. This way, the trainer aim to train them how to actively create perception, to actively listening. But this does not train any awareness nor knowledge nor understanding about the participant’s own behavior in specific RE-related situations.

(8) Prioritization sounds easy during the training session, but is difficult to realize in real life situations.

Consequences

(1) Communication skills and factual RE topics are trained together.

(2) The participants become aware of their own behavior in certain situations, such as forgetting to respect certain rules.

(3) The participants are aware that respecting a lot of aspects at one time, causes them being overloaded, and consequently not being able to manage all of the aspects at one time.

(4) The participants become aware of their own behavior, e.g. they realize they are breaking rules even though they formerly thought they were obeying them.

(5) Participants receive an indication when and how often they might be forgetting or not respecting

(6) Participants are trained and consequently know, that only a few tasks to be solved in one time are needed, to be overloaded and loose track of the overall goal(s).

(7) The participants experience two-sided communication.

Solution. Play a game, that requires to follow multiple rules and executes multiple tests at the same time. Use storytelling to relate the game to requirements engineering.

Implementation. Use the improvisation theater game “Patterns” (see [15], p. 48f) and tell the story that relates the game to the project managers’ environment.

The Pattern Game works as follows: Everyone is standing in a circle. Each participant puts his hand on his head. This gesture indicates that he is “free”. The trainer starts circulating a ball by throwing it to a participant. This participant takes his hand down, catches the ball and then throws the ball to another participant who is still “free”. This is continued until each participant has received the ball exactly once. The last participant throws the ball back to the trainer. Then the circulation starts again in the exact same order.

This is called the “introduction of a ball”. Whenever a new ball is introduced, this is repeated but the participant need to ensure they are throwing to another person than before. Thus, different circles are created.
Throughout the game you tell requirements related stories. Observe the participants and correlate things that are happening while playing with the real world situation in a project or company (vgl. Section 2.3, Figure 1). Things to highlight can be manifold such as a special reaction of a participants, an unrecognized move, an disclosure about oneself (Johari-Window, see e.g. [9]) .

1. Introduce the circle as a company or project where each participant is a department, and each ball is a requirement. The given manner in which the ball is circulated is called “the established way through the company or project”. Each participant must receive the ball exactly ones.

2. Start by throwing one ball and establish its way through the company (see description above). You might want to repeat the introduced circle until every single participant recalls it.

3. Stop throwing this ball, and remind the participants to remember the ball’s way.

4. Introduce a second ball in the same manner.
   You might further associate this ball with the name of an animal or the like: “Each participant shouts an animal's name before throwing the ball. Each name is to be used exactly ones.” Again, repeat this ball until each participant recalls it way. Then stop throwing it.

5. Throw the first ball in its “established way” again.

6. While the first ball is circulating correctly again, start the second ball as well.

7. Continue to introduce further balls in the described manner until your groups gets stuck.

8. After having played with several balls for a while, stop the circulation of the balls. Then take a glass, introduce it in the same way as you introduced the balls before.

9. Start introducing the balls again, as soon as the participants became used to throwing the glass.

This game is setup such that it provides an environment where the participants feel comfortable, are enjoying themselves and are trying out something new.

At the same time, during this game they experience something about themselves, they experience their behavior towards others and together with other during the specific situation of requirements prioritization.

In summary, this game provides the comfort zone to have fun and allow the participants of a training to experience their own behavior (and emotions) while handling requirements, dealing with too many things at the same time and interacting with colleagues in non-easy situations.

**Implementation - Tips and Tricks**

—Extend the trawling of the ball rules by the following statement right at the beginning in order to avoid that the participants try to pass on the glass instead of throwing it: Direct neighbors are not allowed to be chosen.

—Don’t introduce all rules at the beginning. The participant will not obey all of them anyway. Additionally, violating a rule that had not been made explicit before, is a learning in itself (Just recall how often we do have this situation during daily work: there is no explicit rule, but everyone knows not to do certain things.).

—Some of the participants might urge to understand what is going to happen during the game before actually starting the game. Instead of telling them beforehand the complete outcome of the game, you should explain that the rules of the game will be introduced as the game proceeds. This additionally prevents participants’ impression that “you had forgotten to introduce same of the rules”.

**Limitations.** This game should not be used, if the participants haven’t be woken up and are not into the shape of playing an interactive game In case you want to use it, play a Warm up Game (see Section 3.2) first.

This game cannot be played without an experienced trainer who ensures to talk about the consequences in work live of observed behavior during this game (The game needs to be instructed and guided.).
**Storytelling Element.** The storytelling element first sets the scene, e.g. the following story is introduced with the game: "Imagine you are all working in the same company. Each of you works in a different department. A new project is initiated, where the requirement shall go through every single department exactly once."

Do not forget to correlate things that are happening while playing with the real world situation in a project or company (vgl. Section 2.3, Figure 1). Things to highlight can be manifold such as a special reaction of a participants, an unrecognized move, an disclosure about oneself (Johari-Window, see e.g. [9]).

Some of the rules will be forgotten or broken as the game goes along and the participants are introduced to more and more rules and activities. Whenever a rule is broken, the trainer uses the story telling element to align the demonstrated behavior during the game to normal day work. Considering the rule that all requirements are unprioritized implies intact, that all requirements shall take the same time through the organization and non shall be prioritized over the other. In fact, the game is setup such that some balls (that represent the requirements) are more easy to be handled than others. Thus, it provokes balls being faster than the others. Here, the story telling element is used to set the correlation to daily work, for instance by saying "You did not notice that this ball was faster than the others. Imagine this happens during your normal work as well. Before this game you might have neglected that this happens, but can you guarantee that it does not?". You could and should even become a bit provocative.

Some of the rules are repeated with the introduction of the story: The requirements are unprioritized, meaning they have the same priority. But, the requirements are also very fragile, so they must not fall down. " While the trainer talks about a requirement, he shows the corresponding ball.

**Storytelling Example**

(1) Asynchronous Communication. Ensure the Receiver is receiving messages.

(a) Even though the balls were introduced as being breakable, the balls continuously fall on the ground. Often the trainer can observe that the sending participant pass the balls as if in a hurry, not taking the time to wait until the receiving participants pays attention and “is ready to catch the ball”.

(b) Make the analogy to the information model and tell your participants: “The sender sends a message, the receiver receives it. Or he does not receives it.”

(c) Make the analogy to real world’s general context and tell your participants: “When you are handing over your “finished requirements” to your manager, you need to ensure, your manager is truly listening to you. Delivering a requirement includes, to ensure your manager understood that you said “I am ready,”, and you will not be working on this task anymore unless he tells you so. This is also true vice versa. And: The same situation occurs also whenever you are talking to your stakeholders: Are you listening? How well are you listening? How well are you listening? Are you understanding what they want to say? Or are you just hearing what you want to hear?”

LO: In stakeholder management it is very important to remember the exact quote of what a stakeholder desired from a project. The stakeholder will not remember what you as a requirements engineer noted down as requirements, he will only remember what they said, often exactly how he said it. Therefore: Know the requester and his original statement, because this is and will be what the requester will be looking at at any time during the development process and especially at the end when finally testing and accepting your system. (The requester will be looking for his requests.)

(2) “Asynchronous Communication” - The information who was the sender of the information, a message or the requirement is lost throughout the time.

(a) While passing the ball around, the participants might become lazy, and you as a trainer might observe that they become less aware the longer the balls are being passed around. Some participants might even state something like “Great. I only need to remember whom I have to throw the ball to."

(b) The participants nevertheless implicitly know from whom they are receiving the ball. Introduce the need for testing and traversing back to the origin of a requirement , and let the participants circulate a ball in
reverse order (“backwards”). Whenever they are asked, to circulate in “reverse order” this does not work well. It takes some time to unveil the implicit known information. Until then some confusion among the participants can be observed.

(c) When you are gathering requirements, ensure you know who says what. You might need this information later. Even though you think this information is obvious or know by everybody, this information is lost or becomes at least fuzzy over time, so you better document it together with the requirement.

LO: Why is it important to know the original sender? To know to whom you will need to go back to for question, to know the most important stakeholder for this request. And of course, to know at the time of testing, what to test and why. (And again: If there are questions on the way, you know whom to ask.)

(3) “Equally Prioritized versus Implicit Prioritization” - Implicit Prioritization

(a) Different forms of prioritization can be observed. E.g. the ball that has an animal name attached is passed on faster. Some participants might recall one ball better than the other, so in case they receive two balls, the one that is remembered easier is given priority. The glass is often observed to be a prioritizer as well. While handling the glass, it is often observed, that all other activities (including talking and laughing) are stopped, because the full focus is on the glass.

(b) Give examples for the observed priorities. Get confirmation from the participants, if the did noticed the described behavior as well. Explain, that it is natural to easier receive attention from the receiver if shouting out his name or the animal name, point out that you are adding communication channels, thus enriching the information transferred.

(c) As a Requirements Engineer you are often confronted with a lot a stakeholders or requirements you shall handle at the same time. When deciding where to start, it is natural to start with those stakeholders or requirements that seem more familiar to you than others.

LO: The participants understand that they are adding an implicit prioritization to their RE-related tasks. Always. This exercise demonstrates this behavior to each participant.

Suitied or not Suited for Group Situations such as .... This game has been used successfully during requirements engineering trainings both on beginner and on advanced level as an “icebreaker” after lunch.

Additionally, it proved useful in a context where the participants where either experienced requirements engineers or experienced testers. In this specific situation, the testability of requirements could be highlighted by using the traceability.

This game requires some hand-eye-coordination. The group can usually cope with one or two participants not being able to catch balls. If the group has a significant number of participants that are or might be hand-icaped with respect to throwing a ball, do not play this game.

Remark: The author assume that this game can be used to highlight existing conflicts within a software project team, if there are issues with handling the requirements. - This has however not yet been tested.

Requisites needed. The room shall have at least 40 square meters. Any chairs and tables in the room need to be put to the wall, so that there is enough space for a big circle formed by the participants and the trainer.

The following material is needed for the game: Up to three distinguishable, hand-sized balls (e.g. juggle or tennis balls), one drinking or mustard glass and two board markers are needed. Additionally, it shall be organized that a vacuum cleaner and dustpan and brush are at hand, in case the glass breaks.

Variants. This game is derived from the improvisation theatre game called “Pattern Game”([15], p.48f). And it is a variant of “The Company Game”.

Game Language — Page D7-19
5. CONCLUSION AND FUTURE WORK

In this paper, we a structure to catalogue games that have been used within the workshop formats REIM and ImProject was proposed. The structure eases trainers and others who would like to use these games to choose the most appropriate one for their situation.

This paper did however not cover, HOW the reader makes his selection. It leaves the decision criteria to the experience of the reader. In the future work, the games' description could be enhanced by such criteria. This would ease access of the games even more.

This is however only the start of further work where more games shall be documented in the proposed format. In addition, there are some aspect, that have not been part of this paper and need to subsequently be addressed in future work. These are aspects of dependencies or relationships between games. So far, we identified the following dependencies between Games:

—The derived from Relationship
—The variant of Relationship and
—The played together with Relationship.

For instance, the formerly mentioned relationship of the Company Game and the Requirement Game (see Figure ??) is a derived from Relationship, as the Company Game was first developed and only later the Requirements Game was derived from it, see Figure 4.

The played together with relationship also is the basis to provide the so-called Game Chains (see Sections) Documenting complete Game Chains using the played together with relationship provides important additional information to trainers.

As a first step, the sequences successfully used in the workshop structure of the workshop formats REIM [7] and IMPROject [6] will be documented.
Acknowledgments

I would like to thank my shepherds for feedback, special thanks to Dietmar “Didi” Schütz for numerous “Communicator discussions” on the games, their forces, and how to improve the documentation of the workshop elements in general. A big thank you goes to each and every participant during the EuroPloP2012’s workshop session at Kloster Irsee, Germany: With all the fruitful comments, they supported the decision on the final game pattern structure, and to the sharpness of the games’ description! This is usually also the place to thank all improvisors I have been and will be improvising with, it is always an inspiration.

Thanks to Sebastian Ihle for his never ending support and patience while writing this (and other) patterns, and for spending his life with me.

A. APPENDIX

The Structuring Elements of the Game Patterns

The idea is to document the games as a pattern. Looking at the games from the perspective of the trainer (the audience of this paper), the following headlines are assumed to be what the reader will be looking for. Some elements are optional. These are marked “optional”.

Title. For easy reference, and to be easily recognized, each game has a unique and descriptive title.

Context. The context describes the circumstances when this game can be used. It further explains the preliminaries you need to setup before you can start the game.

Motivation (optional). This section motivates the game by giving a not always artificial example of a communication between a trainer and a participant. This conversation leads to a lack of understanding that the trainer can address when playing this game.

Short Description. This section contains a summary of the game, briefly describes the content and gives important information such as duration time, number of participants, suited audience etc..

Applicability. This section describes the applicability of the game. Here, limiting conditions as well as hindering elements are described. These could be for instance, that a game is proven to be suitable for group where a specific role is dominating, where with other roles dominating, this games is proven to not be applicable.

Purpose and Learning Objective (LO). This section describe when the game is useful, and what you can achieve with it. It categorizes the game such as “icebreaker” or the like. Here, you also find further information, if a game can be used as a loosening up game outside a full workshop situation (such as certain team situations) as well.

The learning objectives (LO) of the game are addressed here. An (learning) objective is an intended goal you can reach with your participants by playing this game. This is either an experience the participants make or a knowledge about an experience they gain.

As the underlying original idea and principle to develop these games was to combine factual knowledge (formerly called hard facts) and soft skill training, this section especially focus on highlighting which combination of factual knowledge and soft-skills are aimed to be trained.

Challenge. This section describes which problems have been successfully addressed with this game. As opposed to the learning objective section, this section contains “approved” situations.

In the terminology of the pattern community, this would be a combination of the problem statement and aspect of the problem solved.
Forces - What drives this challenge?. The games inherit the focus as stated above in Section 2.6. A force formulates conditions, under which the addressed problem is solved by the given solution (combined with “Requisites needed”). The set of all sources make the solution unique to the problem while obeying the forces.

Consequences. This section describes the consequences of using the game. It states the (anticipated or formerly observed) reactions among participants?

A.0.1 Solution. What is the solution you are providing using especially THIS game. Note, the solution proposed to a problem might differ, depending on the game you chose - and vice versa: A game might be used to address different problems.

Implementation. This section describes how the solution is to be implemented. Often, a high-level step by step description is given here.

The details for the stories to be told by the trainer are however moved to a separate section, called the Storytelling Element.

Implementation - Tips and Tricks. The section contains tips and tricks from former usage for you. It reports on tricks that worked out successfully to solve somehow tricky situations. This section supports you to not trap into too obvious traps.

Limitations. This section outlines the limits for using the game. Here, situations where to use or not to use the game are listed. Especially, if setups of groups or participants are known which proved or did not prove to be suitable, this is stated here.

Storytelling Element. As already stated before, a game becomes unique through the storytelling used. So, this section describes the elements being used in order to tie the learning objective to the interactive elements.

Storytelling Example. The Storytelling Example gives a concrete real fragment of a story how it was told in a given workshop. This section additionally addresses, how specific Learning Objectives where addressed within the story being told by the trainer. The storytelling examples are structured as follows:

(1) The situation or behavior as observed by the trainer during the game is described.

(2) The story element used by the trainer to highlight this situation or behavior and its consequences is described.

In some cases, this might not be a real story but highlighting directly to a communication fact.

(3) The analogon to situations in the real worlds general context e.g. Requirements Engineering is given.

LO: The learning objective (LO) within the domain that is addressed by the game is described. In case there is more than one LO, they are numbered.

Resulting Context. This section describes how you as the trainer can recognize when the objectives of the game are reached. One might be tempted to call it “postcondition of the game”.

It gives you criteria or hints to criteria how you notice that it is time to move from one game to the other. Thus, this section becomes especially important for the games with “intends”, as the intend is not a clear criteria but a range of xxx to be met.

Suited or Not Suited for Group Situations such as ... This section highlights general situations for which the game has been used successfully.

This section also describes situations where the game has been used, but did not work out as intended. Similarly, this section outlines, when the game does not go well with any other game.

This section can be seen as the Anti-Pattern of The Known Uses.
Requisites needed. What do you need in order to execute the game? This section contains information about room size required, needed material such as board markers, tennis balls or the like. In case something special is needed (such as a vacuum cleaner) this is stated here as well.

Variants (optional). In this section variants of the game are pointed out. Often, this will be a variant in the storytelling element, which is an adjustment to the participants’ background. There might be a shift in the learning objective as well.

Cross References (optional). This section provides the references to other games. The use of this section will be detailed in the future work, see Section 5.

REFERENCES


A. Hoffmann. A trainer's guideline to teaching soft skills using improvisation theater - a workshop format exemplified on a requirements engineering game. accepted for Publication EuroPLoP 2011: 16th European Conference on Pattern Languages of Programs, 2011.


Crossing The Jungle

A Retrospective of Web 2.0

Andreas Rüping
Sodenkamp 21 A, 22337 Hamburg, Germany
andreas.rueping@rueping.info
www.rueping.info

What The Focus Group Was About

There is little doubt that the Web is the single most influential technology of the last decade. It became popular when the first websites emerged in the early 1990ies and gained true momentum after it started to involve its users and became Web 2.0. Today there is consensus that Web 2.0 has changed our society quite dramatically, though whether that is to the better or worse is open for debate.

Enthusiasts are convinced that Web 2.0 has had a positive effect on our society. More and more people world-wide enjoy the active participation in blogs, message boards, online chats and social networks. Web 2.0 is seen as a pathway to free information, new business models and a more open and more democratic society.

Sceptics are not so sure. Newspaper articles report on the dangers of putting personal information online. Record and film companies complain about copyright infringements. Conservative media criticize the quality of online material and speak of amateurisation and exhibitionism.

Some of the criticism sounds like a lament from those who always despise change and who have no desire for an open society anyway. It would be too easy though to dismiss the
concerns as being purely anti-web. In fact, there are real risks. Omnipresent surveillance and an increasing disregard of privacy do represent threats, especially to an open society. The quality of much of what is posted online is indeed questionable. And some people fail to see that there still is a world outside the Internet. The influence that the Web exerts on us seems to get bigger and bigger. Will we all become Web junkies whose lives are governed by online lifestyles and online ethics?

Web 2.0 is a jungle. In some places it’s wild, in some places it’s beautiful; in some places it’s the wilderness that makes it beautiful. Sometimes the jungle is dangerous, yet sometimes it exhibits the most astounding examples of symbiosis and collaboration.

What The Focus Group Did

The focus group set out to review an essay on the topic that I had prepared. The essay is a retrospective of Web 2.0 as we know it today. It describes cultural values that Web 2.0 has helped bring to fruition, while at the same time addressing the risks and dangers of Web 2.0.

The focus group consisted of a “structured discussion”, during which we first looked at the cultural values described in the essay, and then moved on to the more problematic side of the Web.

The focus group participants were able to confirm many of the phenomena that I had described in the essay. Thankfully, however, the in-depth review also spawned new ideas when the participants offered personal experiences and shared their views of the topic.

What The Next Steps Will Be


Acknowledgements

I would like to thank all focus group participants for their comments, ideas and suggestions. Your feedback was most welcome.
Focus Group Report: Evaluating the Consequences of Applying Architectural Patterns

JAAP KABBEDIJK, MATTHIAS GALSTER and SLINGER JANSEN, Utrecht University

1. INTRODUCTION
Patterns are proven solutions for reoccurring problems. Ideally, the consequences of applying a certain architectural pattern are known and documented but when a possible pattern (proto-pattern) is found within only a limited number of case companies yet, not all consequences may be known yet. We propose and tested two different consequence evaluation techniques that can help in identifying the consequences using domain experts. The techniques are aimed at making the implicit knowledge of experts explicit and quantitative in a structured and light-weight way.

2. FOCUS GROUP SET-UP
The focus group took place during EuroPLoP 2012, the 17th European Conference on Pattern Languages and Programs. To focus group took approximately 100 minutes and 6 people participated in it. All people were seated around 1 table and a flip-over was used at one end of the table to keep all discussions central.

The focus group had two main goals. One, the evaluation of a set of quality attributes of one predefined architectural pattern. Two, the testing of and discussion on the two different evaluation methods used during the focus group. The focus group facilitator opened to focus group by explaining the goals and introducing the difficulties and possibilities related to estimating the consequences of applying a certain architectural pattern on quality attributes. The introduction was similar to the introduction of this report. Within this introduction the facilitator also gave all participants a short moment to explain their background. In total, the introduction took around 10 minutes.

After this the META-DATA BASED MULTI-INSTANCE PATTERN was explained. The explanation took around 5 minutes and all participants received a one A4 handout stating all important characteristics of the pattern. The handout can be found in the appendix. All participants were then asked list the quality attributes they felt would be influenced by implementing this pattern. All ideas were written down on the flip-over and every was free to comment on the attributes and extend the list. In the end 7 attributes were identified, which were all written down on a separate A4 paper (see appendix) and laid out on the table (see figure 1). The 7 attributes were the basis for both evaluation techniques performed later in the focus group. The attributes were:

---Scalability
---Maintainability
---Extendibility
---Reuse

This research is part of the ‘Product as a Service’ project (www.productasaservice.org).
Corresponding author: J. Kabbedijk (j.kabbedijk@uu.nl)

©Copyright granted to Hillside Europe for use at/with the EuroPLoP 2012 conference.
Following this, the first evaluation technique was used, based on the 100 dollar test used for requirements prioritization [1]. More on this technique and the results can be found in section 3. This evaluation technique was followed by a short discussion on the pros and cons of the technique and the usefulness of this technique for evaluating the consequences of applying a certain pattern. The second evaluation was inspired by a likert scale based prioritization technique for project consequences [2]. The details of this technique can be found in section 4. This technique again was discussed on usefulness, after which a general discussion was held on what technique is most useful and if the techniques (or combinations of both) are useful at all.
3. TECHNIQUE 1: MONOPOLY GAME

Using the first technique, participants were handed out 100 credits in Monopoly money (see figure 2), which they had to divide among the 7 quality attributes that were written on A4 papers on the table. The 100 credits were handed to the participants as follows:

—5 notes of 1 credit
—1 note of 5 credits
—2 notes of 10 credits
—1 note of 20 credits
—1 note of 50 credits

The participants were asked to place the most money on the attributes they thought would be influenced most positively. They did not have to place credits on all attributes and were even allowed to place all their money on one attribute. Because of the specific notes given to the participants, they were restricted a little in how they could divide their credits.

3.1 Results

After all money was divided by all participants, the total sum of money was calculated per quality attribute. The final scores, sorted from highest to lowest, can be found in table I.
Table I: **META-DATA BASED MULTI-INSTANCE PATTERN** Monopoly test scores

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variability</td>
<td>275</td>
</tr>
<tr>
<td>Reuse</td>
<td>127</td>
</tr>
<tr>
<td>Extendibility</td>
<td>118</td>
</tr>
<tr>
<td>Implementation Effort</td>
<td>36</td>
</tr>
<tr>
<td>Performance</td>
<td>30</td>
</tr>
<tr>
<td>Scalability</td>
<td>7</td>
</tr>
<tr>
<td>Maintainability</td>
<td>2</td>
</tr>
</tbody>
</table>

3.2 Discussion

During the discussion on the usefulness of the Monopoly technique, the following things were mentioned by the participants:

—It really gave good insight in the most important implementation consequences
—You were influenced by others in placing your bets. Participants were not sure if this was a good or bad thing
—This technique may be difficult to extend
—You could only assess positive influences with this technique
—Besides the total amount placed on an attribute, the different notes and betting order may also be of value to observe.

4. TECHNIQUE 2: POKER GAME

Using the second technique, all participants received 2 ‘poker chips’ of each of the following values:

—2
—1
—0
—1
—2

They were asked to place the highest value chips (ie. 2) on attributes they think would be influenced most positively by applying this pattern and the lowest value chips (ie. -2) on the attributes they think would be influenced most negatively. Participants had to bet all their chips, including the 0 value chip.

4.1 Results

After all bets were placed, all values per attribute were summed up. This could results in both a positive or negative score. The sorted scores per attributed can be found in table II.

4.2 Discussion

When the Poker technique was discussed, the following things were mentioned:

—You can also express negative influences with this technique
—You have to think more using this technique
—You feel a pressure of placing a bet on every attribute
—Some participants had the feeling the results of this technique were of a higher quality (compared to the Monopoly technique)
—You could express a neutral influence as well
—Performing this technique felt less sure than the Monopoly technique
—It felt more serious
—The 0 value can be interpreted in multiple ways. It could mean ‘no effect’ or ‘I can not say anything about it’

5. CONCLUSION

Based on the feedback of the participants of the focus group and the results of the discussion, we could conclude both techniques are useful to evaluate the consequences of applying a certain architectural pattern. Applying both techniques in combination with each other, like in the focus group set-up, may
be even more useful than applying the techniques separate from each other, because the insight of
the experts grows during performing the techniques. All participants thought that performing these
techniques in an industrial setting would work well and would be an effective way of evaluation the
consequences of applying architectural patterns.

REFERENCES
C. Wohlin and A.A. Andrews. Prioritizing and assessing software project success factors and project characteristics using sub-
Appendix

Meta-data Based Multi-instance Pattern

**Context:**
An ERP company of 300 employees, offering a bookkeeping product to approximately 10,000 SME users.

**Problem:**
All customers want to have a software product that suits their specific needs, but do not want to pay a large amount of money for the service.

**Forces:**
— The software should respond fast
— Implementation should be easy
— Customers may have different functional preferences
— Customers may have need different data models

**Solution:**