PATTERNS 2012

The Fourth International Conferences on Pervasive Patterns and Applications

ISBN: 978-1-61208-221-9

July 22-27, 2012

Nice, France

PATTERNS 2012 Editors

Alfred Zimmermann, Reutlingen University, Germany

Pascal Lorenz, University of Haute Alsace, France

PATTERNS 2012

Foreword

The Fourth International Conferences on Pervasive Patterns and Applications (PATTERNS 2012), held between July 22 and 27, 2012 in Nice, France, targeted the application of advanced patterns, at-large. In addition, to support for patterns and pattern processing, special categories of patterns covering ubiquity, software, security, communications, discovery and decision were considered. As a special target, the domain-oriented patterns covered a variety of areas, from investing, dietary, forecast, to forensic and emotions. It is believed that patterns play an important role on cognition, automation, and service computation and orchestration areas. Antipatterns come as a normal output as needed lessons learned.

PATTERNS 2012 was aimed at technical papers presenting research and practical results, industrial small- and large-scale systems, challenging applications, position papers addressing the pros and cons of specific topics, such as those being discussed in the standard fora or in industry consortia, survey papers addressing the key problems and solutions on any of the topics, short papers on work in progress, and panel proposals.

We take here the opportunity to warmly thank all the members of the PATTERNS 2012 Technical Program Committee, as well as the numerous reviewers. The creation of such a broad and high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and efforts to contribute to PATTERNS 2012. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations, and sponsors. We are grateful to the members of the PATTERNS 2012 organizing committee for their help in handling the logistics and for their work to make this professional meeting a success.

We hope that PATTERNS 2012 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the area of pervasive patterns and applications.

We are convinced that the participants found the event useful and communications very open. We hope Côte d’Azur provided a pleasant environment during the conference and everyone saved some time for exploring the Mediterranean Coast.

PATTERNS 2012

Committee

PATTERNS Advisory Chairs

Herwig Manaert, University of Antwerp, Belgium
Fritz Laux, Reutlingen University, Germany
Michal Zemlicka, Charles University - Prague, Czech Republic
Alfred Zimmermann, Reutlingen University, Germany

PATTERNS 2012 Research/Industry Chairs

Teemu Kanstren, VTT, Finland
Guenter Neumann, DFKI (Deutsches Forschungszentrum fuer Kuenstliche Intelligenz GmbH), Germany
Markus Goldstein, DFKI (German Research Center for Artificial Intelligence GmbH), Germany
Zhenzhen Ye, iBasis, Inc., Burlington, USA
Cornelia Graf, CURE - Center for Usability Research & Engineering, Austria
René Reiners, Fraunhofer FIT - Sankt Augustin, Germany

PATTERNS 2012 Technical Program Committee

Junia Anacleto, Federal University of Sao Carlos, Brazil
Andreas S. Andreou, Cyprus University of Technology - Limassol, Cyprus
Senén Barro, University of Santiago de Compostela, Spain
Rémi Bastide, University Champollion / IHCS - IRIT, France
Bernhard Bauer, University of Augsburg, Germany
Noureddine Belkhatir , University of Grenoble, France
Hatem Ben Sta, Université de Tunis - El Manar, Tunisia
Félix Biscarri, University of Seville, Spain
Jonathan Blackledge, Loughborough University, UK
Manfred Broy, Technical University Munich, Germany
Michaela Bunke, University of Bremen, Germany João Pascoal Faria, University of Porto, Portugal
William Cheng-Chung Chu(朱正忠), Tunghai University, Taiwan
Bernard Coulette, Université de Toulouse 2, France
Karl Cox, University of Brighton, UK
Jean-Charles Créput, Université de Technologie de Belfort-Montbéliard, France
Mohamed Dahchour, National Institute of Posts and Telecommunications - Rabat, Morocco
Vincenzo Deufemia, Università di Salerno - Fisciano, Italy
Kamil Dimililer, Near East University, Cyprus
Eduardo B. Fernandez, Florida Atlantic University - Boca Raton, USA
Simon Fong, University of Macau, Macau SAR
Francesco Fontanella, Università degli Studi di Cassino e del Lazio Meridionale, Italy
Yukihisa Fujita, Central Research Laboratory / Hitachi, Ltd., Japan
Joseph Giampapa, Carnegie Mellon University, USA

Table of Contents

Growing Complex Software Systems

A Formal Argument for Piecemeal Growth in Software Engineering

Jerry Overton
Computer Sciences Corporation (CSC)
St. Louis, Missouri, USA
joverton@csc.com

***Abstract* – With piecemeal growth, complex systems are grown in a series of small steps rather than pieced together in one large lump. Although there are many specific examples (from agile methods) of piecemeal growth in software engineering; we argue that prior art has yet to produce general theoretical argument for building complex software systems this way. In this research, we propose a formal, theoretical argument for the general applicability of piecemeal growth to software engineering. As part of our argument, we infer both the requisites for piecemeal growth and some surprising connections between piecemeal growth and existing disciplines within software engineering.**

***Keywords* – Piecemeal Growth; Complex Systems; Software Engineering; Agile Methods; Software Design Pattern; Mathematics; Formal Method; POAD Theory**

I. Introduction

A. *The Nature of this Research*

This paper presents the results of software engineering research. We begin with a brief discussion of the nature of this research to establish the proper paradigm for evaluating this work. While scientific problems are concerned with the study of existing artifacts and phenomena (the behavior of subatomic particles, the motions of planets, etc), engineering problems are concerned with how to construct new artifacts (bridges, buildings, and, in our case, software systems) [1]. While scientific research problems have an empirical nature, engineering (specifically software engineering) research problems do not. It is not possible to apply the same empirical validation methods used for scientific research to software engineering research [1].

Software engineering research is the study of processes by which people turn ideas into software [1]. Empirical data collected about these processes necessarily contain social and cultural aspects. Although empirical data may serve as an example to clarify the concepts presented here, it cannot objectively validate our results. Producing any such data and determining its correspondence with our results requires subjective interpretation.

In this paper, we advocate for the effectiveness of a particular software engineering approach using a structured argument. Ultimately our work is validated by whether or not the argument we present is convincing among practicing software engineers. To be considered convincing, the argument will have to generate interest and credibility. It will have to be circulated among a wider audience, polished and refined. Parts or all of the argument must be used by engineers to justify design processes of their own. We consider this work to be the first step in the process – we have recorded an argument so that it can be read, circulated, and scrutinized. For this paper, our goal is to produce an argument lacks identifiable errors or contradictions.

B. *Piecemeal Growth and Software Engineering*

Piecemeal growth is the process of building a complex system in small steps [2]; where nothing is ever completely torn down or erased. Additions are made, existing structures are embellished and improved [3]. This is different from modular design [4] where the system is composed from individual pieces snapped together. Consider the process by which the St. Mark's Square in Venice (Fig. 1) was built – the example of piecemeal growth given in [5]. The process started in 560 A.D. with a small square basilica, where the castle of Doge (middle right in the picture) was built. In 976 A.D., two new buildings were added to the center of the basilica, including the tower shown in the middle of **Fig. 1**. By 1532 A.D., the tower became embedded in a rectangular building and the original basilica was extended.

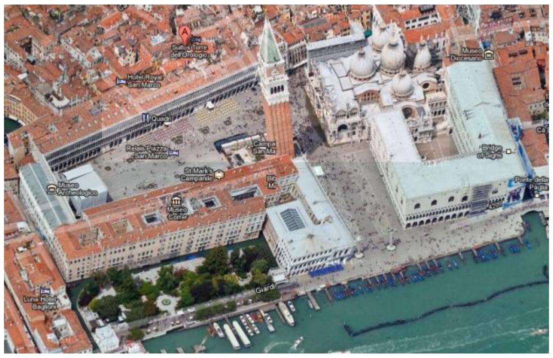


Fig. 1: St. Mark's Square in Venice [6].

St. Marks Square grew from a gradual sequence of changes rather than by assembling pre-fabricated parts. Each change mostly preserved the changes that came before. And each change contributed to the organic order seen in Fig. **1**. Because all acts of piecemeal growth have these characteristics in common [3]. We recognize a process as piecemeal growth, if it:

1. Specifies a sequence of operations
2. Each operation preserves the effects of all previous operations
3. Each operation solves part of a bigger problem
4. The sum effect of all operations solves the problem in its entirety

We will argue in the next section that although there are many specific examples of piecemeal growth in software engineering, there is nothing in the prior art that proposes an argument for the general applicability of piecemeal growth to building software systems. There is no theoretical argument that arbitrary, complex software systems can be built in a manner similar to the way St. Mark's Square was built. The general strategy of our argument is inspired by a technique used in mathematics – [6] argues that the Koch curve fractal exists by showing it to be the unique consequence of a particular equation. We argue that piecemeal growth is generally applicable in software engineering by showing it to be the unique consequence of a particular software engineering approach.

The rest of this paper is as follows. In Section II, we argue that software engineering is missing a general argument for the applicability of piecemeal growth to software engineering. In Section III, we introduce a system of mathematics needed to create that argument. In Section IV, we use our math to argue for the general applicability of piecemeal growth. In Section V and VI, we analyze our results and its significance.

II. State of the Art

The idea of piecemeal growth has made its way into software engineering practice through the adoption of agile methods [8] such as Extreme Programming (XP) [9], Scrum [10], and Crystal [11]. Agile methods assert that complex, well-designed software systems can be grown gradually through a process of continuous refactoring [12]. In this approach, software engineers do not put much emphasis on comprehensive analysis or design. Instead, they focus on building the highest-priority feature using the first reasonable approach that comes to mind. They refactor the results into a suitable design, and then repeat the process for the next highest-priority feature. The belief is that engineers can progress toward a solution piecemeal because refactoring makes it possible (and inexpensive) to make changes at any point.

Although the practice of piecemeal growth is known in software engineering as a part of agile methods, the actual idea of piecemeal growth is developed by prior works, such as [12], [13] and [14] that focus specifically on the practice of continuous refactoring. Instead of general arguments, these works all give detailed examples of how specialized refactoring techniques work to improve parts of a specific system. None of them propose an argument (although the premise is asserted) that, in general, continuous refactoring can be used to grow arbitrarily complex software systems.

For example, in [12], continuous refactoring is used as the basis for enabling piecemeal growth. The overall concept is developed using an introductory example. Although the example describes the basic idea of refactoring, it does not describe why this idea is useful beyond the specific example given. For the technique's broader application, the reader is asked to *"imagine [the example] in the context of a much larger system [12]."* In [13], continuous refactoring is asserted to be a proper basis for piecemeal growth in software engineering. The concept is illustrated by an example of evolving a new application framework for a legacy system, however, there is no argument for how to extend the techniques used in the example to the creation of systems not described in the example. In [14], the use of continuous refactoring for piecemeal growth is illustrated by an example of evolving a database for a simple financial institution. The work describes an example starting point for such a process, but the process itself – and its applicability to systems other than what is exemplified – is asserted without further argument.

The goal of this research is to add to the current state of the art an argument for the general applicability of piecemeal growth to software engineering. Our approach is to come up with a mathematical model that characterizes the general existence of piecemeal growth in software engineering and a mathematical argument that piecemeal growth follows naturally as the result of a specific engineering approach. In the next section, we establish the system of mathematics needed to make our argument.

III. POSE AND POAD THEORY

The arguments in the next section will be based on a system of mathematics known as Problem-Oriented Software Engineering (POSE) [15] and Pattern-Oriented Analysis and Design (POAD) Theory [16]. In this section, we provide a summary of both.

In POSE, a software engineering problem has context, W; a requirement, R; and a solution, S. We write $W, S \vdash R$ to indicate that we intend to find a solution S that, given a context of W, satisfies R. Details about an element of the problem can be captured in a description for that element and a description can be written in any language considered appropriate. The problem, P_0 of designing a complex system can be expressed in POSE as:

$$CSystem: W, S \vdash R \qquad (1)$$

where W is the real-world environment for the system, S is the system itself and R are the requirements the system must meet. Equation (1) says that we can expect to satisfy R when the system S is applied in context W.

In POSE, engineering design is represented using a series of problem transformations. A problem transformation is a rule where a conclusion problem $P: W, S \vdash R$ is transformed into premise problems $P_i: W_i, S_i \vdash R_i, i = 1, \dots, n\ (n > 0)$ using justification J and a rule named N, resulting in the transformation step $\frac{P_1 \dots P_n}{P} \begin{matrix}[N] \\ \ll J \gg\end{matrix}$. This means that S is a solution of $W, S \vdash R$ whenever $S_i, \dots, S_n$ are solutions of $W_i, S_i \vdash R_i, \dots, W_n, S_n \vdash R_n$. The justification J collects the evidence of adequacy of the transformation step. Through the application of rule N_i, problems are transformed into other problems that may be easier to solve. These transformations occur until we are left only with problems that we know have a solution fit for the intended purpose. POSE allows us to use one big-step transformation to represent several smaller ones. The progression of a software engineering solution described by a series of transformations can be shown using a development tree.

$$\cfrac{\cfrac{\overline{P_3: W_3, S_3 \vdash R_3} \quad P_4: W_4, S_4 \vdash R_4 \quad [N_2]}{P_2: W_2, S_2 \vdash R_2 \qquad \ll J_2 \gg} \quad [N_1]}{P_1: W_1, S_1 \vdash R_1 \qquad \ll J_1 \gg} \qquad (2)$$

In the tree, the initial problem forms the root and problem transformations extend the tree upward toward the leaves. There are four problem nodes in the tree: P_1, P_2, P_3, and P_4. The problem transformation from P_1 to P_2 is justified by J_1, the transformation from P_2 to P_3 and P_4 is justified by J_2. The bar over P_3 indicates that P_3 is solved. Because P_4 remains unsolved, the adequacy argument for the tree (the conjunction of all justifications) is not complete, and the problem P_1 remains unsolved. A complete and fully-justified problem tree means that all leaf problems (in this case P_3 and P_4) have been solved.

For the sake of clarity, we will show the context, solution, and requirement of a problem only when necessary to understanding a given transformation. In many of the equations in section IV, these details are omitted and only the problem's name is shown. In general we adopt the practice of omitting any detail not required to support our argument. For example, we recognize that systems requirements often compete and designers must consider details such as how to trace from business requirements to system requirement to architectural choices. Although these considerations are important in the day-to-day practice of software engineering, they were not necessary to complete our argument for the general applicability of piecemeal growth in software engineering and were, thus, omitted from representation in subsequent formal models.

An $AStruct$ (short for Architectural Structure) [15] is used to represent an architecture in the solution. An $AStruct$, $AS\ [g\ ](c_1, \dots, c_n)$ has a name, and combines a known structure, g (of arbitrary complexity), together with the c_i which are elements of the solution that are yet to be designed. Using the solution interpretation rule $SolInt$, we can modify the solution as follows:

$$\frac{W, AS[g](c_1 \dots c_n) \vdash R}{W, S \vdash R} \begin{array}{c}[SolInt]\\ \ll justify\ AS[\dots](\dots) \gg\end{array} \qquad (3)$$

Once an $AStruct$ has been applied, we can use the Solution Expansion transformation $SolExp$ to expand the problem context and refocus the problems to find the c_j that remain to be designed. For example, in the case where $n = 3$, we would have:

$$\frac{\begin{array}{l}W, g, c_2{:}\,null\,, c_3{:}\,null, c_1 \vdash R,\\ W, g, c_1{:}\,null, c_3{:}\,null, c_2 \vdash R,\\ W, g, c_1{:}\,null, c_2{:}\,null, c_3 \vdash R\end{array}}{W, AS[g](c_1, c_2\,, c_3) \vdash R}\,[SolExp] \qquad (4)$$

where $null$ is used to indicate that nothing is known about that particular component. Using $null$ allows us to isolate a particular unknown element without making assumptions about any of the other unknown elements. The $SolExp$ transformation creates a number of premise problems. Each new premise problem requires solving and each premise problem contributes its solution to the other premise problems. Note that because the architecture being expanded has already been justified, the expansion of the architecture requires no further justification.

Software design patterns record the engineering expertise needed to justify the substitution of a complex, unfamiliar problems with simpler, more familiar one [17]. The basis of POAD Theory is that software engineering design can be represented as a series of transformations from complex engineering problems to simpler ones, with software design patterns used to justify those transformations:

$$\frac{SimplerProblem}{ConplexProblem} \begin{array}{c}[SolInt]\\ \ll Pattern_1, \dots, Pattern_n \gg\end{array} \qquad (5)$$

In (5) the engineering expertise in patterns $Pattern_1, \dots, Pattern_n$ are used to justify the replacement of the $ConplexProblem$ with the $SimplerProblem$.

In the next section, we used the mathematics of POSE and POAD Theory to argue that piecemeal growth can be used to create arbitrary complex software systems.

IV. THE EXISTENCE OF PIECEMEAL GROWTH

A pattern $Pattern_i$ tells us how to solve a problem by introducing an architecture and components modeled as follows:

$$\frac{W_i, AS_i[g_i](c_i) \vdash R_i}{Problem_i} \begin{array}{c}[SolInt]\\ \ll\ Pattern_i \gg\end{array} \qquad (6)$$

We could apply the solution expansion rule to the architecture introduced by $Pattern_i$, similar to what we did in (4). But suppose, instead, we were to study $Pattern_i$and realize that there is a way to go about implementing the pattern's solution by breaking it into two problems: the problem of finding g_i (the problem of implementing the invariants of the pattern), and the problem of finding c_i (the problem of implementing the context-specific parts of the pattern). Suppose our research into $Pattern_i$ leads us to the engineering judgment (J_i) that there is a method for implementing the solution to the problem as follows:

$$\frac{Problemg_i\,, Problemc_i}{Problem_i{:}\, W_i, AS_i[g_i](c_i) \vdash R_i} \begin{array}{c}[SolInt]\\ \ll J_i \gg\end{array} \qquad (7)$$

Our research into $Pattern_i$ allows us to realize that we can solve $Problem_i$ by implementing $Pattern_i$ using a combination of $Problemg_i$ and $Problemc_i$. For the sake of clarity, we combine (6) and (7) into a single pattern of transformation.

$$\frac{\dfrac{Problemg_i, Problemc_i}{W_i, AS_i[g_i](c_i) \vdash R_i} \begin{array}{c}[SolInt]\\ \ll J_i \gg\end{array}}{Problem_i} \begin{array}{c}[SolInt]\\ \ll\ Pattern_i \gg\end{array} \qquad (8)$$

is shortened to

$$\frac{Problemg_i, Problemc_i}{Problem_i} \begin{array}{c}[SolInt]\\ \ll J_i,\ Pattern_i \gg\end{array} \qquad (9)$$

In the original application of $Pattern_i$ to $Problem_i$, the component g_i acts as context for the component c_i . The solution to $Problemc_i$will operate within the context of the solution for $Problemg_i$. This subtle relationship between the solution to $Problemg_i$ and the solution to $Problemc_i$ will be important later in our argument.

Suppose we had a $Complex\ Problem$ that we wanted to solve. Suppose that we found a set of transformations patterned after the one in (9) that we could apply in sequence to the $Complex\ Problem$ as follows:

$$\frac{Probg_1, \dfrac{\dots, \dfrac{Probg_n, Probc_n}{\dots} \begin{array}{c}[SolInt]\\ \ll J_n, Ptn_n \gg\end{array}}{Probc_1} \dots}{Complex\ Problem{:}\, W, S \vdash R} \begin{array}{c}[SolInt]\\ \ll J_1, Ptn_1 \gg\end{array} \qquad (10)$$

We know that our completed solution will be composed of $n+1$ interrelated problems and n patterns. The solution to $Probg_{i+1}$ will operate within the context of $Probg_i$. The $Complex\ Problem$ is solved by finding solutions to all leaf-level problems $Prob\ g_1, \dots, Prob\ g_n, Prob\ c_i$ For short, we can write (10) as the pattern sequence [17] :

$$[\ll J_1, Ptn_1 \gg, \dots, \ll J_n, Ptn_n \gg] \qquad (11)$$

The sequence of (11) is a model of the analysis process required to find a solution to the $Complex\ Problem$ – the patterns Ptn_i needed to solve the problem, the implementation strategies J_i that must be used for each pattern, and the order in which each pattern and implementation strategy must be applied.

Suppose we completed our analysis by finding solutions to all leaf-level problems as follows.

$$\frac{\overline{Solng_1}\ [SolInt]}{Probg_1 \ll Jg_1 \gg}, \dots, \frac{\overline{Solng_n}\ [SolInt]}{Probg_n \ll Jg_n \gg}, \qquad (12)$$

$$\frac{\overline{Solnc_n{}'}\ [SolInt]}{Probc_n \ll Jc_n \gg}$$

where$Solng_i$ is part of a specific implementation of the pattern Ptn_i, and Jg_i is convincing justification that $Solng_i$ is adequate to solve $Probg_i$. Just as we did with (10), we can (12) using the following sequence:

$$[\ll Jg_1, Solng_1 \gg, \dots, \ll Jg_n, Solng_n \gg, \qquad (13)$$

$$\ll Jc_n, Solnc_n \gg]$$

Whereas (11) is a model of the analysis process needed to find a solution to the $Complex\ Problem$, (13) is a model of the design process required to realize the solution. It describes the specific implementation needed to solve each outstanding problem and justification for why each implementation works. The sequence of (13) can be interpreted as the ordered steps of piecemeal growth required to solve the $Complex\ Problem$.

Recall from Section I, the criteria 1-4 for recognizing a process as piecemeal growth. Equation (13) specifies a sequence of operations. Each step $\ll Jg_i, Solng_i \gg$ in the sequence results in $Solng_i$ – a partial solution to the $Complex\ Problem$. We know that step $i+1$ of (13) preserves step i because, from earlier analysis, we know that $Solng_{i+1}$ acts entirely in the context of $Solng_i$. We also know from Section III that the solution to the $Complex\ Problem$ is the collection of all solutions $Solng_1 \dots Solng_n$. Thus, we have completed our argument that a piecemeal-growth solution to the $Complex\ Problem$ exists, and that the piecemeal-growth solution can be characterized as the sequence of steps given by (13).

In the last section, we analyze the significance of our efforts.

V. Analysis

We showed piecemeal growth to be a consequence of the engineering strategy of (9), and that piecemeal growth requires the analysis process modeled in (11). We started by looking for a solution to the $Complex\ Problem$. This problem is an arbitrary software engineering problem in that the only assumption that we made was that the $Complex\ Problem$ has an arbitrarily large number of requirements. We made a single assumption (9) about the strategy for solving the problem and found a solution by working through the consequences of that one assumption. As a consequence, we satisfied the n requirements of the $Complex\ Problem$ with the $n+1$ problem-solving transformations represented by the sequence in (13) – which happened to properly characterize piecemeal growth. The progression we went through is an argument that piecemeal growth is applicable to arbitrary complex problems in software engineering. Piecemeal growth has some very specific characteristics (the criteria 1-4 from Section I). Yet, by starting only with an engineering assumption made independently of the decision to use piecemeal growth, we were able to derive an abstract mathematical representation that matched our characteristics of piecemeal growth.

We recognize that the formal models presented in Sections III and IV would be easier to understand if accompanied by a comprehensive example. However, creating such an example is outside the scope of the current research – our goal, here, was to record the argument in enough detail to allow it to be circulated and scrutinized. In future research, as the argument is polished and refined, it will become essential to supplement the formal model with a running example.

VI. Conclusions

In (13) $Solng_{i+1}$ acts entirely in the context of $Solng_i$ – a relationship that can be satisfied using abstraction and refinement [18] – note that a refinement works entirely within the context of an abstraction. With this realization, we can begin to imagine specific tactics for general piecemeal growth: each step is a refinement of the previous step, and an abstraction for the next.

One of the more interesting questions in piecemeal growth is whether or not systems can be grown without the help of up-front planning [19]. Can we solve (or begin solving) the $Complex\ Problem$ without first completing some kind of detailed analysis? We know that the progression (13) from the $Complex\ Problem$ to its solution required the analysis shown in (11). If we were to proceed without up-front planning, we would have to derive the sequence of (11) as the result of progressing along the sequence of (13). As we implemented each of our interim solutions, we would have to be able to derive our next problem based on our current solution. More formally, we would need to be able to derive $Probg_{i+1}$ from $\ll Jg_i, Solng_i \gg$. Equation (10) implies that the minimum linking them is $\ll J_{i+1}, Ptn_{i+1} \gg$. That is, the minimum requisite for successful progression through the piecemeal growth of (13) is that one must be able to derive the $n+1^{st}$ step of the analysis (11) while one performs the n^{th} step of the design (13). This may be possible if one can anticipate how to structure $Solng_i$ so that it can act as the context for $Solng_{i+1}$. In other words, our argument implies that piecemeal growth without detailed planning is possible only if, at each step, one can successfully anticipate and accommodate the invariants of the next.

The idea that one must be able to anticipate future invariants suggests a potentially novel approach to piecemeal growth and a link between piecemeal growth and predictive analytics. Our argument suggests that all that is really needed to proceed with each step of piecemeal growth are the invariants of the next step. It may be possible predict all required invariants by performing a cluster analysis [20] on a complete set of system description documents. The resulting clusters and their dependencies may be interpreted as a map of the system's invariants. It may be interesting to explore whether or not it is practical to establish a community of software engineers that grow (piecemeal) complex software systems guided by architectures mined from collections of plain-text descriptions of what users would like the system to accomplish. For example, it may be possible to use crowdsourcing to efficiently produce a comprehensive set of description documents for a complex system, predictive and visual modeling to create a reliable map of that system's invariants, and piecemeal growth to build the system gradually over time using a long series of small, inexpensive acts of systems development guided by the derived map of invariants.

VII. References

[1] M. Lázaro and E. Marcos. *Research in Software Engineering: Paradigms and Methods*. Advanced Information Systems Engineering, 17th International Conference. Proceedings of the CAiSE 05 Workshops, 2005, pp. 517-522

[2] C. Alexander. *A Timeless Way of Building.* Oxford University Press, 1979.

[3] C. Alexander. *The Oregon Experiment.* Oxford University Press, 1975.

[4] K. Sullivan, Y. Cai, B. Hallen, and W. Griswold, *The Structure and Value of Modularity in Software Design*. Proceedings, ESEC/FSE, 2001, ACM Press, pp. 99-108

[5] C. Alexander. *A Vision of A Living World.* The Center for Environmental Structure, 2005.

[6] Maps.google.com. Last Accessed: 03/15/2012

[7] H.O. Peitgen, H. Jurgens, D. Saupe. *Chaos and Factals: New Frontiers of Science*. Springer, 2004.

[8] J.O. Coplien, N.B. Harrison. *Organizational Patterns of Agile Software Development.* Prentice Hall, 2004.

[9] K. Beck. *Extreme Programming Explained: Embrace Change,* Second Edition. Addison-Wesley, 2004.

[10] K. Schwaber, M. Beedle. *Agile Software Development with SCRUM.* Prentice Hall, 2001.

[11] A. Cockburn. *Agile Software Development.* Addison-Wesley, 2001.

[12] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 2000.

[13] J. Kerievsky. *Refactoring to Patterns.* Addison-Wesley, 2005.

[14] S. W. Ambler, P. J. Sadalage. *Refactoring Databases.* Addison-Wesley, 2006.

[15] J. G. Hall, L. Rapanotti, and M. Jackson. *Problem-Oriented Software Engineering: Solving the Package Router Control Problem.* IEEE Trans. Software Eng., 2008. doi:10.1109/TSE.2007.70769

[16] J. Overton, J. G Hall, and L. Rapanotti. *A Problem-Oriented Theory of Pattern-Oriented Analysis and Design.* 2009, Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, pages 208-213, 2009.

[17] F. Buschmann, K. Henney, and D. Schmidt. *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*, Volume 5. John Wiley & Sons, West Sussex, England, 2007.

[18] D. F. D'Souza, A. C. Willis. *Objects, Components, and Frameworks with UML.* Addison-Wesley, 1998.

[19] A. Dagnino, K. Smiley, H. Srikanth, A. I. Anton, and L. Williams, *Experiences in Applying Agile Software Development Practices in New Product Development*. In Proceedings of Proceedings of the Eighth IASTED International Conference on Software Engineering and Applications, Nov 9-11 2004 (Cambridge, MA, United States, 2004). Acta Press, Anaheim, CA, United States.

[20] I. H. Witten, E. Frank. *Data Mining Practical Machine Learning Tools and Techniques.* Elsevier, Inc. 2005.

Patterns in Safety Analysis

Tor Stålhane
Olawande Daramola
Vikash Katta
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
{stalhane, wande, vikash.katta}@idi.ntnu.no

Abstract **– This work proposes the use of a pattern-based hazard descriptions and generic failure modes, in combination with domain ontology and Jackson's problem frames for automating the identification of hazards. This is an extension of our previous work in the CESAR project where we built a tool that enables a requirements engineer to write requirements in a semiformal notation based on domain knowledge described as ontologies plus a set of predefined requirement templates. Our approach will enable automatic generation of the complete FMEA table based on system's requirements, pattern based hazard descriptions and domain knowledge formalized as domain ontologies.**

Keywords – safety analysis; HazId; generic hazards; generic failure modes.

I. INTRODUCTION

The reported work is based on the work on requirement patterns defined by E. Hull et al. [1] and extended and improved in the CESAR project. In this project we combined requirements patterns with domain ontologies. This allows us to check the requirements for e.g., consistency and completeness. The ontologies also enabled the identification of system components such as sensors, actuators and control units. By including a set of generic failure modes for each component, we are able to build a partly filled in Hazard Identification (HazId) table based on Failure Mode and Effect Analysis (FMEA) for the system specified by the requirements. In order to make further progress we identified three needs: we needed (1) to describe existing hazards in the environment where the specified system should operate, (2) to formalize the failure mechanisms that operate in this environment and (3) an algorithm that could bridge that gap between the local, generic failure modes from the FMEA and the global, domain specific hazards.

The rest of this paper is organized as follows: in Section II we give a short description of the two concepts generic failure modes and generic hazards. In Section III we discuss how to describe hazards while Section IV discusses the use of textual templates – boilerplates – for hazard description. In Section V we discuss possible ways to bridge the gap between system FMEA and environment hazards. In the last section (Section VI) we discuss how we can combine theoretical work with industrial experiments to validate and improve the work.

II. GENERIC DESRIPTIONS IN SAFETY ANALYSIS

The starting point of this work was the use of textual patterns – boilerplates – for requirements, ontologies for describing equipment and generic failure modes for each part of the equipment to semi-automatically construct an FMEA table. Our work on the application of boilerplates for requirements engineering is based on the work of E. Hull et al. [1] and further developed by the partners in the CESAR project – see [2]. The reader should consult this article for further information on the definition and use of boilerplates.

To apply textual patterns in the HazId process, we have used two concepts:

- Equipment ontology. This is used to identify the components that are part of the equipment. This is used for two purposes (1) to control that there are requirements for all components of the equipment and (2) to keep an updated ontology with the generic failure modes for each component.
- Generic failure modes, which are already used in a wide area of application domains – e.g. offshore industry [3], nuclear industry [4], aerospace [5] and automotive industry [6].

A generic failure mode is a failure mode containing a group of more detailed, specific failure modes that all have the same high level manifestation – e.g., all failure modes that will lead to a motor stopping can be included in the generic failure mode "motor stops". After having studied a large set of published generic failure modes, we have settled for the following:

- Actuators – no action, wrong actions
- Sensors – no info, wrong info
- Control systems – omission, commission, incorrect, too late.

Below is a small part of the requirements written using boilerplates and the generated part of a HazId table for a steam boiler. For further discussion of the boilerplate requirements and the semi-automatic generation of a system HazId, see [7].

- <controller> shall <read> <water level> from <water level sensor>

- If <water level exceeds TBD>, <controller> shall <turn off> <feeding pump>
- If <water level below TBD>, <controller> shall <turn on> <feeding pump>

TABLE I. EXAMPLE OF A GENERATED PART OF A HAZID TABLE

Element	**Failure mode**	**Risk from environment risk assessment**
controller	Omission	
	Commission	
	Incorrect	
	Too late	
Water level sensor	No output	
	Wrong output	
Feeding pump	No action	
	Wrong action	

Generic hazards are widely used in industry e.g., the offshore industry's generic hazards for blow-out [8] and subsea drilling [9]. Important industrial areas like aviation [10], chemical plants [11] and the building industry [12] also have lists of relevant generic hazards.

III. HAZARD DESCRIPTIONS

There are several ways to describe hazards. We have chosen an approach based on Ericson [13], which is illustrated by the diagram in Figure 1.

Based on this diagram, a hazard description must at least contain the three topmost components – hazardous element, initiating mechanism and target and threat.

- Control unit => ***initiating mechanism***, related to the control unit’s failure modes. The control unit receives info from the equipment under control.
- Controlled equipment => ***hazardous element***, related to the equipment’s failure modes. The equipment can move to a hazardous failure mode either due to a wrong control command or due to an internal failure
- Target and threat => equipment’s ***environment***, e.g., building and personnel. These are represented as having generic hazards.

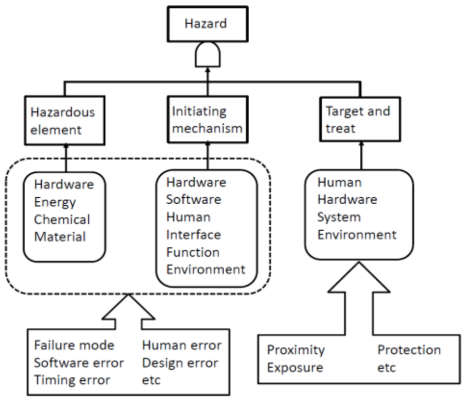


Figure 1: Hazard description pattern

We see that we need two descriptions in order to analyse an accident:

- How the equipment can harm the environment – cause an accident. Our starting point here is a list of generic hazards. The event sequence is as follows: (1) the equipment is brought into a hazardous state and (2) an event can then cause an accident – identified by one or more of the potential accidents contained in the list of generic hazards.
- How the equipment can reach a hazardous state. We need to consider how the equipment can do this alone, e.g., based on equipment characteristics or due to a faulty command from the controller.

Related to this, we need to consider (1) the controller's action and what causes it, e.g., an internal error or faulty information from equipment or from the equipment's environment via sensors and (2) the equipment entity that is affected, which will bring the equipment into a hazardous state – actuators such as pumps and valves.

The challenge is how much of the accident sequence we can describe in a generic fashion using one or more boilerplate patterns. The sequence of events that finally leads to an accident can also be described as a cause – effect chain.

In order to organize our hazard description, we have based our accident descriptions on Jackson’s problem frames [14]. We need to map:

- Hazardous element, something that is in or can be brought into a hazardous state.
- Initiating mechanism, something that happens to the hazardous element or to something that might affect the hazardous element.
- Target and threat, which is a description of a potential accident.

The use of boilerplates to describe hazards will enable the representation of hazards in a semi-formal way and will thus be an improvement over the use of tables. We believe that the use of hazard boilerplates provides a useful mid-level in order to go from manual hazard analysis to machine understandable hazards for automated safety analysis. It is possible to translate hazards expressed as free text or tables into hazard boilerplates, and also to generate table-based hazards from a set of hazard boilerplates.

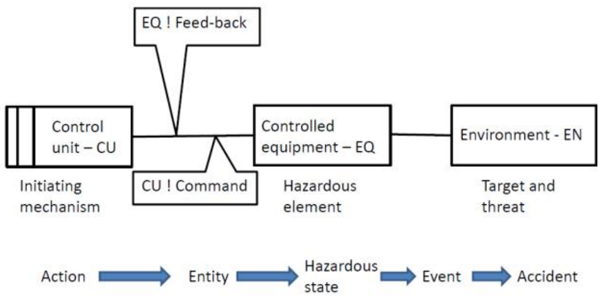


Figure 2: Pattern for equipment under control

The use of hazard boilerplates has the following advantages:

- It provides a unified structure and style of describing hazard and thereby reduces ambiguity. It also brings some consistency into the way similar hazards are represented.
- It will engender reuse of hazard descriptions since the semi-formal representation using boilerplates is more amenable to automated text processing. It creates a basis for pattern-based, structure-based, and semantic-based reuse in hazard analysis, which is useful in the context of product lines and variant systems in specific domains. Several automated safety analysis procedures for failure detection and prediction, hazard mitigation, and cause-effect analysis will benefit from reusable boilerplate hazard definitions.
- Increased completeness of hazard descriptions – ensuring that every hazard conforms to the requirements of the Hazard Classification Matrix used by Ericson [13] – see Figure 1.

IV. HAZARD BOILERPLATE DESCRIPTIONS

An element that is in a state where it has the potential to cause an accident is said to be in a hazardous state. Such an element is called a hazardous element. The element is brought to the hazardous state by an action – e.g., an equipment failure. This gives the following boilerplate formulation:

<action> **to** <entity> **in** <state> **can cause** <hazardous state>

A hazardous state does not necessarily lead to an accident. Instead it might just be the first step out of several that eventually leads to an accident. Thus, we might need several boilerplate statements in sequence to describe the full accident sequence.

If we stick to Ericson's model as shown in Figure 1, we see that an initiating mechanism applied to a hazardous element will create a threat to a target – an accident. We will use the notation {...} to indicate an alternative and since <action> **to** <entity> = <event>, we can write:

- {<action> **to**} <entity> **in** <state> **can cause** <hazardous state>
- <event> {**in** <hazardous state>} **can cause** <accident>

It is possible to have different events leading to the same hazardous state and to have several hazardous states leading to or enabling the same accident. When we have a chain of events finally leading to an accident, it is also possible to identify the event in the chain where it is most easy to stop the process and thus prevent the accident from happening.

Using the following steps, we can build the complete cause - consequence chain for any controlled equipment in any environment.

- ***Environment***: identify all generic hazards that are relevant for the environment under consideration, e.g., explosion, flooding and fire.
- ***Controlled equipment:*** use the two boilerplates "<action> **to** <entity> in <state> **can cause** <hazardous state>" and "<event> **in** <hazardous state> **can cause** <accident>" to describe how the equipment can cause an accident in the relevant environment. We start with the generic hazard and can write "<too high pressure> **to** <vessel> **can cause** <explosion>". When we consider the equipment ontology we see that <too much heat> **can cause** <too high pressure>.
- ***Control unit:*** The control unit can cause an accident by sending the wrong command to the equipment. The reason for the wrong command is either wrong input, e.g., from a sensor, or a wrong understanding of the current state of the equipment, which again is based on wrong info from the equipment.

The last step is to identify how the events leading up to the accident can be initiated. This involves understanding of how the controller works, i.e. the mapping from input sensor signals to the output actuator signals. E.g., <wrong value> **to** <temperature> **can cause** <wrong command> **to** <heater>

The last step can be used to map instruments – e.g., sensors – to an initiating event. The necessary knowledge can be taken from an equipment ontology where we find that temperature is measured using a temperature sensor. E.g., <sensor error> **can cause** <wrong value> **to** <temperature>.

With the examples above in mind we have the following event chain: EQ ! wrong temperature value => CU ! wrong command to heater => too high pressure => explosion. The event EQ ! wrong pressure value will have the same consequences.

V. BRIDGING THE GAP

We now have a semi-formal description of how the system can fail (Section II) and a semi-formal description of the environmental hazards (Section IV). We can thus bridge the gap between system failures and environment accidents – consequences.

The control unit's components are identified by using information from the control unit's requirements and from the equipment's ontology, see fig. 3. The equipment ontology will also contain failure modes – generic or specialized – for each component.

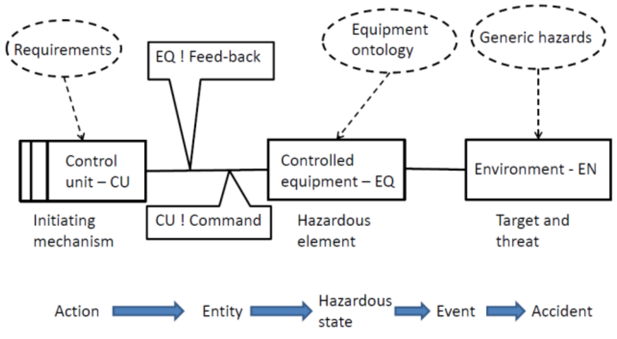


Figure 3: System-under-control pattern with necessary information

If we consider the examples at the end of the previous section, we see that the initiating event is an equipment sensor failure. A sensor registered in the tool's ontology base will have two generic failure modes: wrong output and no output and the first of these failure modes can cause an explosion.

Given that the control unit is correctly implemented, the control unit can move the equipment into a hazardous state in three ways:

- By acting correctly on a wrong signal from the equipment – e.g., a faulty sensor.
- By getting to a wrong state due to a wrong signal and then acting wrongly on a correct signal.
- By acting too late on a signal from the equipment.

Thus, the main activity for bridging the gap between hazard descriptions and the equipment and control unit's failures is to map generic failure modes onto the first action in the cause – consequence chain used in the hazard descriptions.

The typical event described in the previous section is EQ ! wrong temperature value => CU ! wrong command to heater => too high pressure => explosion. The local effect is related to the CU – send wrong command, while the global effect is the generic hazard – explosion. If we use the standard elements in an FMEA or HazId table we get the table shown in Table II below.

It is straight forward to include a detection strategy in the equipment ontology description for each component, thus extending the FMEA to a Failure Mode, Effect and Diagnostics Analysis (FMEDA). The detection strategy can be linked to each component or to each failure mode depending on the granularity of the information available. For our sensor example this could be a built-in self-test or sensor duplication.

Building on an already existing CESAR tool, we can add a new tool, which will enable the definition of hazards described as boilerplates and production of a HazId table using FMEA or FMEDA.

TABLE II. EXAMPLE OF COMPLTE HAZID TABLE

Component	Failure mode	Local effect	Global effect	p	C
Temperature sensor	No signal				
	Wrong signal	Wrong command to heater	Explosion		

VI. CONCLUSIONS AND FURTHER WORK

Our present work is based on our work in the CESAR project where we built a tool that enables a requirements engineer to write requirements in a semi-formal notation based on domain knowledge and stored as boilerplates. The equipment ontology can contain a set of generic failure modes, which allows a tool to automatically generate the first part of an FMEA table.

In the present work we have shown that it is also possible to describe hazards that stems from defined equipment failures using boilerplates and an equipment ontology. This enables us to complete the FMEA table. The new tool will also allow the engineers to add new knowledge and experience, thus making the tool an important part of the company's memory for safety analysis.

Our next steps will be to build a tool prototype and, in cooperation with an industrial partner, to enter a set of hazard definitions written as boilerplates. The tool prototype will be used in an experiment to identify strong and weak points plus identifying new functionality that need to be added in order to satisfy industrial users' needs.

REFERENCES

[1] E. Hull, K. Jackson, and K. Dick, (2004): "Requirements Engineering", Springer.

[2] O. Daramola, T. Stålhane, T. Moser, and S. Biffl (2011): "A Conceptual Framework for Semantic Case-based Safety Analysis", 16th IEEE Intl. Conf. on Emerging Technologies and Factory Automation, Toulouse France, IEEE Press

[3] SINTEF: "OREDA Offshore Reliability Data", 5th Edition

[4] J.D. Lawrence: "Software Safety Hazard Analysis", NUREG/CR-6430, February, 1996.

[5] C. Seguin, "Formal Notation Suitable to Express Safety Properties", ESACS technical report, September 17, 2001

[6] P. Johannesen, F. Tørner, and J. Torin: "Actuator Based Hazard Analysis for Safety Critical Systems", Proceedings of the 23th International Conference on Computer safety, Reliability and security, Potsdam, Germany September 2004.

[7] T. Stålhane, S. Farfeleder, and O. Daramola: "Safety analysis based on requirements", Extended Halden Reactor Project Meeting, Sandefjord, Norway, 2011

[8] H. Brant et al.: "Environmental Risk Assessment of Exploration Drilling in Nordland IV", DnV no. 2010-04-20

[9] J.L. Melendez: "Risk Assessmnet of Surface vs. Subsea Blowout prevneters (BOPs) on Mobil Offshore Drilling Units focusing on Riser Failure and the use of Subsea Shear Rams" Texas A&M University, May 2006

[10] N. Alvares and H. Lambert: "Realistic Probability Estimates For Destructive Overpressure Events In Heated Center Wing Tanks Of Commercial Jet Aircraft". 5th International Seminar on Fire and Explosion Hazards". Edinburgh, United Kingdom April 23, 2007 through April 27, 2007

[11] S. Rathnayakaa, F. Khana,, and P. Amyotte: "SHIPP methodology: Predictive accident modeling approach. Part I" Methodology and model description". Process Safety and Environmental Protection 8 9 (2011) 151–164

[12] B.E. Biringer, R.V. Matalucci, and S.L. O'Connor: "Security Risk Assessment and Management: A Professional Practice Guide for Protecting Buildings and Infrastructures". John Wiley & Sons, 2007

[13] C.A. Ericson II: "Hazard Analysis Techniques for System Safety". John Wiley & Sons, Inc., New Jersey, 2005

[14] M. A. Jackson: "Problem frames: analysing and structuring software development problems". Addison-Wesley 2001

Analyzing User Patterns to Derive Design Guidelines for Job Seeking and Recruiting Website

Yao Lu
École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
e-mail: yao.lu@epfl.ch

Sandy El Helou
École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
e-mail: sandy.elhelou@epfl.ch

Denis Gillet
École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
e-mail: denis.gillet@epfl.ch

***Abstract*—More and more competitive websites are targeting online job seeking and recruiting. In this paper, we discuss user pattern in an online job seeking website in Switzerland, by analyzing user profiles and actions. Then based on our findings, we derive design guidelines to improve both the interface usability and the efficiency of the embedded jobs/candidates recommender system.**

Keywords-patterns; job seeking; recruiting; Web design; recommender system;

I. INTRODUCTION

Nowadays, job seeking and recruiting websites are becoming more and more popular [1]. These websites not only provide a prospective platform for information gathering but also support rich interaction features leveraging user interests [2][3]. From a business point of view, it is crucial to understand user patterns and make use of them, in order to improve the provided services and satisfy the customers [4].

Analyzing user patterns is widely used not only for identifying group related activities but also for user behavior prediction based on previous users [5][6]. Patterns preserve common problem-solving knowledge to give recurring solutions for user-friendly design [7]. Unfortunately, there are only a few studies on online user patterns and design of the job seeking and recruiting websites [8]. In this paper, we analyze user profiles and interaction patterns on a job seeking and recruiting website.

Usually, users of job seeking and recruiting websites split into two main categories: employer and candidates. They participate in the job seeking and recruiting process by completing their profiles and performing different actions. By relying on usage statistics, graphical visualization and analysis, we derive typical user interaction and profile patterns for both user categories. Based on the discovered patterns, we generate design guidelines for jobs/candidates recommender system as well as for the website's interface. The guidelines proposed in this paper have contributed to increasing the number of overall visits of company, candidates, and job profiles by 300% within the period of one month.

The rest of the paper is organized as follows. Section 2 gives a general description of the website's dataset. Section 3 discusses user profile and interaction patterns. Section 4 provides guidelines for designing the interface and the embedded recommender system, based on the analyzed user patterns and the experience with our case study.

II. DATASET INTRODUCTION

The dataset used in our case study includes more than 7,000 candidate users (40% of which have provided CVs), 320 companies, and 7,000 jobs. The time span is from May 2008 to the beginning of February 2012.

Candidates, company, and job-related statistics are given below.

A. *Candidate Statistics*

- 80% of the candidates live in Switzerland.
- There are twice more male that female candidates.
- The average age is 27, and 88% of the candidates are between 22 and 35.
- 98.5% of the candidates are university students or graduates.

B. *Company Statistics*

- One in six companies is a startup.
- Most of the companies are located in Switzerland and Germany takes up another big part.
- 320 companies cover 50 industry branches. The biggest industry group is IT, Internet services, and computer hardware/software. The next biggest group is management consulting.

C. *Job Statistics*

- 89% of the posted jobs are hosted externally and the rest is directly posted on the website (using predefined forms).
- 61.7% of the jobs are located in Switzerland and 37.3% in Germany.

III. User Profile and Interaction Patterns

In this section we discuss user profile and interaction patterns in our case study.

A. *Input Patterns in User Profiles*

Of most job seeking and recruiting websites, user profiles are important for both employers and candidates. Taking candidates for example, they can either input their information online or upload CV files. Meanwhile, many websites support users to import their profile from other websites like LinkedIn [9].

Online profile forms contain specified fields. For candidates, the main profile fields typically consist of personal information (name, gender, birthday, etc.), educational background, and work experience. For companies and jobs, the main fields are location, industry, description, and requirements. Data processing is usually easier when dealing with structured and standardized profile information, as in our case study. Profile input patterns can be summarized as follows:

- Almost all companies and jobs have their basic information filled, and 70% provide more details regarding offers and requirements.
- Considering the candidates who filled their online forms, the profile completion rate is rather satisfying. 86% of these candidates input their personal information (name, gender, age). Moreover, 99.7% of them input their education while only 51.4% input their work experience (probably because most of them are graduating students). The completion rate for the different profile fields is illustrated in Figure 1.
- There is a preference for uploading CVs rather than filling online forms. In our case study, the website was supporting CV upload from November 2009 to October 2011. During this period, 57% of the candidate profiles were uploaded PDF files.

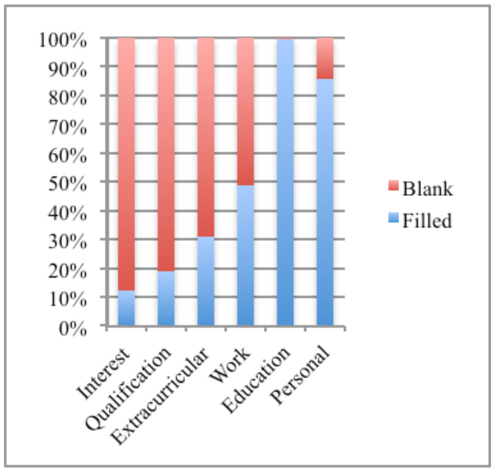
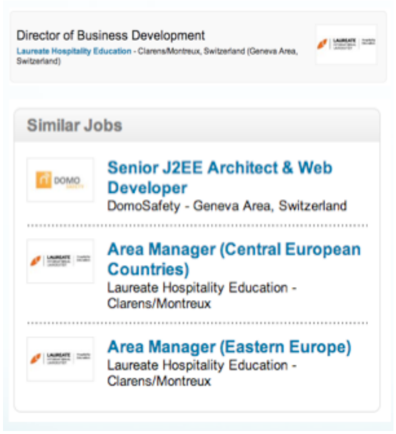


Figure 1. Profile completion rate per field

B. *Similarity Patterns in User Profiles*

On many websites, there is a recommendation list of items similar to the one a user is visiting [10]. For example, when a candidate is visiting a job page, similar jobs (in terms of location, industry, description) are recommended to that candidate. Similarly, when an employer is on a candidate's page, other candidates with similar profiles (in terms of educational background, skills, work experience) are also recommended to that employer. Figure 2 shows an example of similar job recommendation on LinkedIn [9]. A prerequisite to provide such features is to compute similarity between entities of the same type.



Director of Business Development
Laureate Hospitality Education - Clarens/Montreux, Switzerland (Geneva Area, Switzerland)

Similar Jobs

Senior J2EE Architect & Web Developer
DomoSafety - Geneva Area, Switzerland

Area Manager (Central European Countries)
Laureate Hospitality Education - Clarens/Montreux

Area Manager (Eastern Europe)
Laureate Hospitality Education - Clarens/Montreux

Figure 2. Similar job recommendation in LinkedIn[9]

In this section, we discuss profile similarity patterns illustrated using similarity graphs. Profile similarity consists of measuring the extent to which two user profiles are similar in terms of content. We construct similarity graphs for all types of users. In these graphs, an edge connects two user profiles if they have a similarity higher than a minimum threshold. We discovered that the graph of candidates and employers follow different patterns. We use an information visualization tool named aiSee [11] to build and visualize graphs of job, candidate, and company similarity. The profile similarity measurement is shown in Figure 3.

As it is illustrated in Figure 3， there are two types of profiles: structured and unstructured. Structured profiles consist of several predefined fields that can be filled online like name, age, and education. Unstructured profiles, on the other hand, are uploaded with no standardized format (e.g. uploaded CV files). For structured profiles, the overall similarity consists of a linear combination of weighted field similarity in the case of fields having predefined values and normalized content-based similarity in the case of free text fields. Table 1 shows the fields in candidate's profile we select and the weight assigned to each field. The choice of the fields and their corresponding weights are set after discussion with company representatives. For them, the educational background, the university, and the degrees are the most important fields since their target customers are graduating students. When it comes to companies and jobs, the industry field, geographic location, job title/position and its requirements are important fields to consider for similarity measurement [12]. When it comes to unstructured profiles, it is difficult to extract corresponding fields. So, when comparing two PDF files or one PDF file and one structured profile, both are parsed into unstructured data and LSA (Latent Semantic Analysis) is used to compute their similarity. Both structured and unstructured profiles have textual content (i.e. description fields in online filled forms). In this case, we also use LSA to compute similarity. More specifically, we call APIs provided by Salsadev [13]. Through the process described in Figure 3, we obtain similarity of structured online profiles and unstructured uploaded profiles.

TABLE I. SELECTED FIELDS AND WEIGHTS OF CANDIDATES' ONLINE PROFILES

Field	Weight (%)	Field	Weight (%)
Gender	5	Age	10
University	10	Study Course	20
Diploma	10	Language	5
Work Experience	20	Qualification	10
Extracurricular	10	Total	100

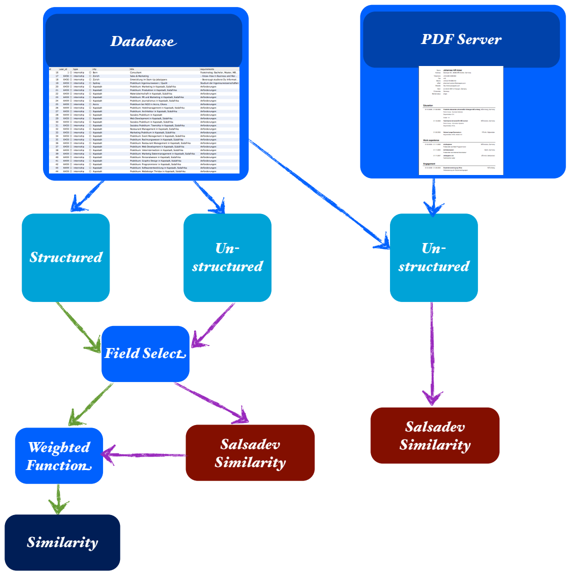
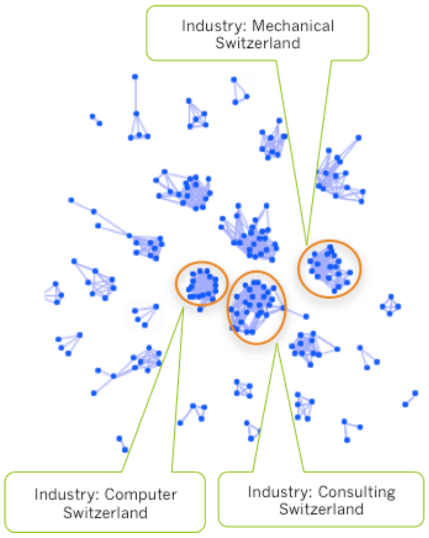
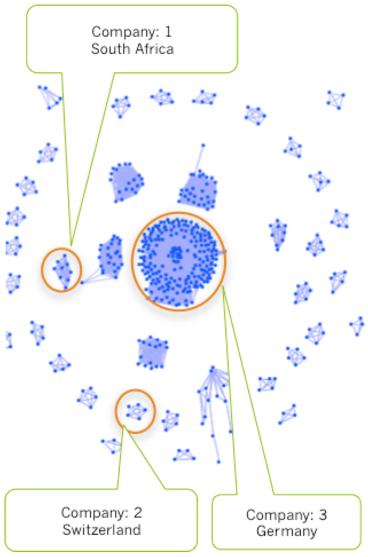


Figure 3. Similarity computation process for candidate profiles

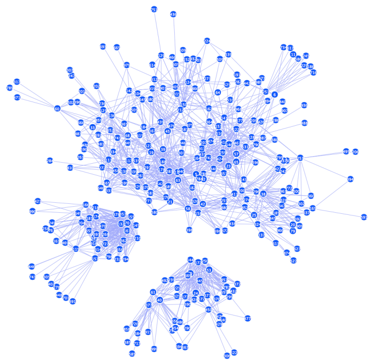
Figure 4 displays the similarity graphs for all user types. In these graphs, nodes represent users, and edges connect similar users. Radial layout is used and users with higher number of incoming edges (i.e. those having more similar users) are located in the center. Ignoring users that have no edges (not similar to any other user), we observe that the graph of candidates follow patterns that are different from other user types (jobs or companies). That is, company graph and job graph consist of clusters of small dense graphs, while the candidate graph consists of big radial clusters. However, according to a preliminary evaluation, this difference in patterns does not affect accuracy in any significant way. The use of structured candidates' CVs yields more accurate results than unstructured CVs.


Industry: Mechanical Switzerland
Industry: Computer Switzerland
Industry: Consulting Switzerland

(a) Company similarity graph



(b) Job similarity graph



(c) Candidate similarity graph

Figure 4. User similarity graphs

Reasons for the difference in similarity patterns are discussed hereafter. In the graphs of jobs (Fig. 4a) and companies (Fig. 4b), there are many small, dense graphs with a small number of nodes but most of them are connected to each other. But in candidates' graph, there are big, radial clusters. Splitting the candidates similarity graph into a graph for structured candidate CVs and another graph for unstructured CVs, still displays the same pattern. This indicates that the patterns observed in candidate graphs are not due to dealing with unstructured CVs. On the other hand, a closer look at company and job graphs, shows that company and job nodes tend to cluster by their dominant factor. In our case, the dominant factor of company is industry. That is because we use location, industry, and scale to compute the similarity and, as it is described in section 2, most of the companies are located in Switzerland. As a result, it is the industry field that makes the difference, and contributes the most in forming clusters of similar companies. When it comes to jobs, the dominant factor is the company that posts them. It is not only because the same company usually offers jobs with similar requirements related to the company's industry but also because job descriptions follow the same style and use the same set of terms. As a result, most jobs posted by the same company form one cluster. But when it comes to candidates, it is difficult to label clusters in the profile similarity graph. The main reason is that there is no dominant feature for candidates. There are much more fields considered for candidates than for companies (country, industry, scale) and jobs (company, location, title, description). In addition to the multiple fields contributing to the similarity pattern, many candidates have more than one input for each field. For example, they may have multiple interests, and may have studied in more than one university. Taking the field as an example, in our measurement, if two students have at least one field of study in common, they are considered as similar along this attribute. For example, if student A studies finance and business, student B studies finance and computer science, they are considered as similar in terms of their field of study. So a group of nodes with multiple, overlapping values are clustered. In contrast, companies and jobs usually have one value in most fields.

In addition, candidate similarity is not necessarily transitive. Let us consider a student C that has studied computer science and graphic design. Ignoring all other fields, C is similar to B because they both studied computer science but C is not similar to A. It is also observed that the nodes in the center of a radial graph tend to have multiple backgrounds. This is due to the fact that there are more nodes similar to them along different directions.

Based on the similarity patterns observed for profiles, potential discussions about the user classification, pattern improvement, and contribution to the recommendation will produce more interesting discoveries. We summarize some limitations exclusively relying on content-based analysis to produce recommendations:

- Candidates tend to mention multiple interests without the possibility to specify their order of preference.
- Two candidates with a high similarity in their profile may not attract employers in the same way.
- While computing profile similarity, it is impossible to set the field weights in such a way to satisfy every user, as all users have different priorities. For example, some candidates focus on employers in the same industry with their majors while other candidates are more concerned about the location of their jobs.

To better understand the users, and deliver personalized jobs recommendation, it is deemed important to leverage users interest by examining how they interact with the website. Actions such as rating a job, liking/disliking a company, adding a company event or a candidate profile to one's favorite list are useful indications of interest. In the next section, we discuss the user interaction patterns derived from logged user actions.

C. User Interaction Patterns

Job seeking and recruiting usually provide some social media features like connect, like, share, and recommend to friends. These features do not only help user discover interest and opportunities, but can also be exploited in recommender systems [14]. Figure 5 shows two screenshots from LinkedIn and Xing [15]. When it comes to our case study, the possible actions are summarized below:

- Visit
- Share
- Like/Dislike
- Rating
- Recommend to friends
- Add to favorite list (or bookmark)
- Apply (for a job)

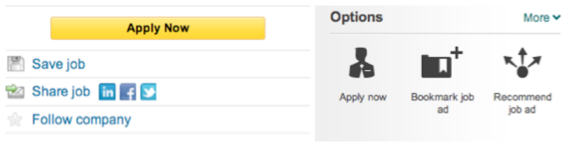


Figure 5. Social media features in LinkedIn and Xing[9][15]

On the interface, most of the buttons to perform interactions and express interest are located in easy-to-use places. According to user action logs, the interactions are sparse and the majority of the users are anonymous. Interactions follow a long tail distribution [16]. In our case, taking visits as an example, 20% of the users contribute to more than 70% of performed visits. Figure 6 shows the long tail by ranking users by the number of their page visits. Meanwhile, majority of users are anonymous. According to data logged between May 2011 and February 2012, on a total of 1.4 million visits, only 2278 were contributed by logged in users, i.e. 98.3% of the visits are anonymous. Interestingly, from April 2011 to September 2011, the website allowed anonymous users to bookmark candidate, job, or company profile pages. The result was that anonymous users were responsible for 99.7% of the actions of bookmarking.

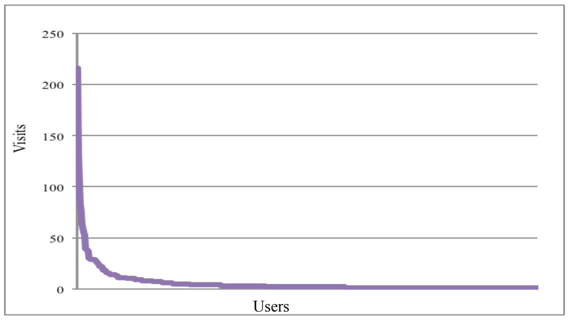


Figure 6. Long tail of performed visits

IV. WEB DESIGN GUIDELINES

In this section, we provide a list of design guidelines based on the case study, and more specifically: the analysis of user profile, interaction patterns and the interface usability analysis. These guidelines can help in designing usable interfaces and job recommendation services for online job seeking and recruiting websites. In our case for example, applying the guidelines provided below, contributed to an increase of 300% in the number of visits within the period of a month:

- Users tend to be anonymous which renders pattern analysis less personalized. So website registration should not be difficult or time consuming. Instead, low entry barriers should be adopted, and profiles could be filled progressively. User interest should be leveraged in a semi-automatic way by combining different techniques such as content-based and interaction-based analysis.
- Mining user interactions on the website complements content-based profile analysis in deriving user patterns and delivering personalized recommendations. Social media features such as bookmarking, liking/disliking and rating should be used to encourage interactions and infer user interest. Nevertheless, when it comes to job recruiting websites, these features should be "built-in" and should not be imported from other sites such as the Facebook "like" button [17] or the Twitter "share" button [18]. The main reason for that is that candidates do not necessarily want that everyone in their social network know that they "liked" a job. Thus, using a Facebook button would discourage them from expressing interest. Last but not least, using external features prevents the website from being able to log the corresponding user actions and exploit them for data mining and recommendation purposes.
- Obviously, the location of the interaction features is essential. Bookmarking or adding to favorites is deemed useful and widely used in many online sites (e.g. social sites, e-commerce, job seeking sites). Surprisingly, we had only very few records of such features in our study. Usability analysis showed that it was the feature's inadequate position that dramatically affected its usage. Instead of being placed on the homepage, the user had to go to settings to find his/her list of favorites. So it was time consuming for candidates to perform extra actions to access it in order to bookmark a company or job page.
- Based on observed patterns, users tend to upload CV files rather than fill online forms. Therefore, websites should provide various ways for users to provide their information, like online form, uploading, and file import from other existing sites. On the other hand, since it is easier to process structured data, incentives for filling online forms should be created for users.
- Content-based profile analysis is not by itself sufficient to leverage user interests, while interaction-based analysis has the problem of sparse data, as reported earlier. To overcome each method's limitations, a hybrid job recommender system, which combines the two techniques, should be adopted. All data types, including structured content, unstructured content, and recorded interactions should be explored. Using hybrid recommendation techniques help cope with the data sparseness and cold start problems.

V. CONCLUSION AND FUTURE WORK

In this paper, we analyze user profile and interaction patterns for an online job seeking and recruiting website. Based on our user pattern analysis, we provided guidelines for the design of the website's interface and the embedded recommender system. In the future, we will continue to analyze the data from the website for further user pattern analysis. A user study will also be conducted to assess the impact of our design guidelines on the website's usability and evaluate the accuracy of the embedded job/candidate recommender system.

References

[1] Jobvite 2011 Social Recruiting Survey. Dec. 2011. http://recruiting.jobvite.com/

[2] P. Lops, M. D. Gemmis, G. Semeraro, F. Narducci and C. Musto, "Leveraging the linkedin social network data for extracting content-based users profiles," Proceeding of the fifth ACM conference on Recommender Systems (RecSys '11), ACM, Oct. 2011, pp. 293-296, doi:10.1145/2043932.2043986

[3] N. B. Ellison, C. Steinfield, and C. Lampe, "The benefits of Facebook "Friends": Social Capital and College Students Use of Online Social Network Sites," Computer-Mediated Communication, vol. 12, iss. 4, July. 2007, pp. 1143-1168, doi: 10.1111/j.1083-6101.2007.00367.x.

[4] Edmond H. Wu, Michael K. Ng, Andy M. Yip, and Tony F. Chan, "A Clustering Model for Mining Evolving Web User Patterns in Data Stream Environment," Proceeding of Intelligent Data Engineering and Automated Learning (IEDAL 2004), pp. 565-571, doi: 10.1.1.4.8132.

[5] A. Ross, C. B. Owen, and A. Vailaya, "Models for User Access Patterns on the Web: Semantic Content versus Access History," Proceedings of 5th Annual World Conference on the WWW and Internet (Webnet 2000), pp. 464-469, doi: 10.1.1.117.7275.

[6] S. Han, A. Goker, and D. He, "Web User Search Pattern Analysis for Modeling Query Topic Changes," Information Processing & Management, vol. 38, iss. 5, 2002, pp. 727-742, doi: 10.1016/S0306-4573(01)00060-7.

[7] I. Wentzlaff, and M. Specker, "Pattern-based development of user-friendly web applications," Proceedings of ICWE '06 Workshop proceedings of the sixth international conference on Web engineering, ACM, July. 2006, doi: 10.1145/1149993.1149996.

[8] B. J. Jansen, K. J. Jansen, and A. Spink, "Using the web to look for work Implications for online job seeking and recruiting", Internet Research, vol. 15, iss.1, 2005, pp. 49-66, doi: 10.1108/10662240510577068.

[9] http://www.linkedin.com/

[10] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, "Item CF Item-based Collaborative Filtering Recommendation", Proceedings of the 10th international conference on World Wide Web (WWW '01), ACM, May 2001, pp. 285-295, doi: 10.1145/371920.372071.

[11] http://www.absint.com/aisee/

[12] S. L. Rynes, R. D. Jr. Bretz, and B. A. Gerhart, "The Importance of Recruitment in Job Choice: A Different Way of Looking," Personnel Psychology, vo. 44, iss. 3, Sept. 1991, pp. 487-521, doi: 10.1111/j.1744-6570.1991.tb02402.x.

[13] http://www.salsadev.com/

[14] R. Rafter, Keith. Bradley, and B. Smyth, "Automated collaborative filtering applications for online recruitment services," Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-based Systems, Springer-Verlag Press, Aug. 2000, pp. 363-368, doi: 10.1007/3-540-44595-1_48.

[15] https://www.xing.com/

[16] A. Enders, H. Hungenberg, H Denker and S. Mauch "The long tail of social networking. Revenue models of social networking sites", European Management Journal, vol. 26, iss. 3, Feb. 2008, pp. 199-211, doi: 10.1016/j.emj.2008.02.002.

[17] http://www.facebook.com/

[18] https://twitter.com/

Pattern Innovation for Architecture Diagnostics in Services Computing

Alfred Zimmermann
Reutlingen University, Faculty of Informatics
Architecture Reference Lab of the
SOA Innovation Lab, Germany
alfred.zimmermann@reutlingen-university.de

René Reiners
Fraunhofer FIT
User-Centered Ubiquitous Computing
Sankt Augustin, Germany
rene.reiners@fit.fraunhofer.de

Abstract **– Assessing the maturity of service-oriented enterprise software architectures is a problem since the current practice has been developed rather intuitively, providing only a sparse and rarely validated metamodel. In preliminary research, we have developed an original pattern language for supporting repetitive enterprise system architecture assessments. The aim is the evaluation and optimization of these kinds of architectures. For this purpose, we extended base frameworks like the Capability Maturity Model Integration and The Open Group Architecture Framework. Since we apply a pattern catalogue for the assessment of enterprise system architectures, we see ourselves confronted with the problem that patterns are traditionally derived after long experience by an expert group of pattern authors. In our view, this may lead to a decelerated reuse of available knowledge. Our approach intends to integrate available knowledge from services computing and software architects directly from the beginning of the pattern development process. Over time, these ideas are iteratively developed towards validated patterns by feeding back the insights of pattern applications. This allows the early integration of new findings and concepts into the pattern catalogue at an early stage whereas already existing patterns are continuously refined. In this work, we propose both, a clear maturity framework background for the developed architecture assessment patterns, and an early integration of new ideas as pattern candidates within a pattern innovation and evolution process.**

Keywords – service-oriented systems; architecture maturity framework; assessment patterns; pattern evolution.

I. INTRODUCTION

Innovation oriented companies have introduced services computing systems to assist in closing the gap between business and information technology and thus enabling business opportunities for service and emerging cloud computing paradigms in the context of novel enterprise architecture management approaches. One of the main problems is that until today the transparency of this innovation change to system architectures based on services and cloud computing in information technology is blurred. Our approach investigates the ability of heterogeneous enterprise services systems [1] and integrates system architecture elements from convergent architecture methods, technologies and related software patterns, as in [2], [3], and [4] with evaluation methods for service-oriented enterprise systems [5].

The SOA Innovation Lab - an innovation network of industry leaders in Germany and Europe - investigates the practical use of vendor platforms in a service-oriented and cloud-computing context. For this purpose we have researched a suitable set of architecture assessment instruments for services computing, leveraging and extending the Capability Maturity Model Integration (CMMI) [6] and the Open Group Architecture Framework (TOGAF) [7]. This set extends our previous work and consists of ESARC - our Enterprise-Services-Architecture-Reference-Model [8] and [9], an associated ESA-Architecture-Maturity-Framework [1] and [8], and an ESA-Pattern-Language [10] for supporting architecture evaluation and optimization.

Our research explores the novel hypothesis to relevantly support a major effort of software architects during architecture assessments of service-oriented systems:

1. CMMI [6] is well known as a suitable basic maturity framework to assess software processes. Nevertheless the metamodel of CMMI can be transformed to enable quality assessments for software architectures.
2. The idea of software patterns can consistently be applied and extended in service-oriented architecture assessments for capability diagnostics of service-oriented architectures. The collected architecture assessment patterns could be iteratively improved within our original pattern evolution process.

We are reporting in this research paper about our current research step to combine our previous evaluated architecture assessment metamodel with a newly introduced community-oriented pattern evolution process. In Section II we present related and preliminary work concerning service-oriented architectures and frameworks. Additionally, we present current findings on architecture maturity assessment. Section III gives a brief introduction to software patterns as best practices for application and software design also providing background information about the approach, a pattern's intention, structure and the combination of patterns. The application of architecture patterns as test cases for the maturity assessment of software architectures is shown in Section IV together with the derived SOA maturity model integration SOAMMI and results from a first practical validation process. We explain the pattern innovation process that is combined with the currently existing pattern catalogue providing collaborative means for the early integration of new concepts and their evaluation during a project's lifetime. Finally, we conclude in Section V on the current state and provide an outlook on future work and directions.

II. Architecture Maturity And Its Assessment

The Open Group Architecture Framework (TOGAF) [7] as the current standard for enterprise architecture provides the basic blueprint and structure for our enterprise software architecture domains of service-oriented enterprise systems, as in the ESARC reference model [9]: Architecture Governance, Architecture Management, Business & Information Reference Architecture, Information Systems Architecture, Technology Architecture, Operation Architecture, Security Architecture, and Cloud Services Architecture.

SOA is the computing paradigm that utilizes services as fundamental flexible and interoperable building blocks for both structuring the business and for developing applications. SOA promotes a business-oriented architecture style as promoted in [11] and [3]), based on best of breed technology of context agnostic business services that are delivered by applications in a business-focused granularity. To provide dynamic composition of services within a worldwide environment SOA uses a set of XML-based standards. A main innovation introduced by SOA is that business processes are not only modeled, but also combined services are executed from different orchestrated services.

In recent work, we have transformed the Capability Maturity Model Integration into a specific framework for architecture assessments of service-oriented enterprise systems. For this reason, we have combined CMMI with current SOA frameworks and maturity models. We used TOGAF and ideas related to the business and information architecture from [12] as a basic structure for enterprise architecture spanning all relevant levels of service-oriented enterprise systems. In contrast to the Enterprise Architecture Project in [12] we are focussing on standardized structures form TOGAF [7] and extend these in our ESARC Architecture Reference Model, as in [8] and [9], with currently researched new additional architectural views: Operation Architecture, Security Architecture, and Cloud Services Architecture.

The Architecture Capability Maturity Model (ACMM) framework, which is included in TOGAF [7], was originally developed by the US Department of Commerce. The goal of ACMM assessments is to enhance enterprise architectures by identifying quantitative weak areas and to show an improvement path for the identified gaps of the assessed architecture. The ACMM framework consists of six maturity levels and nine specific architecture elements, which are ranked for each maturity level, and are deviant from the understanding of maturity levels in CMMI.

Inaganti and Aravamudan [13] describe the following multidimensional aspects in their SOA Maturity Model: scope of SOA adoption, SOA maturity level to express architecture capabilities, SOA expansion stages, SOA return on investment, and SOA cost effectiveness and feasibility. The scope of SOA adoption in an enterprise is differentiated by the following levels: intra department or ad hoc adoption, inter departmental adoption on business unit level, cross business unit adoption, and the enterprise level, including the SOA adoption within the entire supply chain. The SOA maturity levels are related to CMMI, but used differently, applying five ascending levels to express enhanced architectural capabilities: level 1 for initial services, level 2 for architected services, level 3 for business services, level 4 for measured business services, and level 5 for optimized business services.

Sonic [14] distinguishes five maturity levels of a SOA, and associates them in analogy to a simplified metamodel of CMMI with key goals and key practices. Key goals and key practices are the reference points in the SOA maturity assessment.

ORACLE [15] considers in their SOA Maturity Model a loose correlation with CMMI five different maturity levels: opportunistic, systematic, enterprise, measured, industrialized and associates them with strategic goals and tactical plans for implementing SOA. Additionally, the following capabilities of a SOA are referenced with each maturity level: Infrastructure, Architecture, Information & Analytics, Operations, Project Execution, Finance & Portfolios, People & Organization, and Governance.

A. The SOAMMI Framework

The aim of the SOAMMI – SOA Maturity Model Integration - framework [1] is to provide an integral framework to assess architectures of service-oriented enterprise systems and to accord with a sound metamodel approach. The previously mentioned related work elements where developed following in contrast to SOAMMI only a pragmatic and intuitive approach, having no explicit metamodel and outside of common architecture standards, like TOGAF. The metamodel for architecture evaluation enlarges the standardized CMMI, which is originally used to assess the quality of software processes and not the quality of software architectures.

The SOAMMI architecture maturity framework introduces original architecture areas and organizes them within extended architecture domains, which are mainly based on TOGAF. Our intention was to leave most structural parts e.g. *Maturity Levels, Capability Levels, Specific Goals and Practices, Generic Goals and Practices* - of the original CMMI metamodel as untouched concepts. We extend these concepts of the metamodel by reclusively connected architecture patterns, as navigable architecture quality patterns of a pattern language, and enlarge these by other architecture specific structures and contents. The metamodel of SOAMMI is illustrated in Figure 1 also revealing that it has similarities with the original CMMI metamodel; we left the semantics of maturity levels and capability levels the same like in CMMI. Additionally, we added the following concepts: Architecture Domain, Architecture Area, Architecture Pattern, and replaced all the contents of related Specific Goals, Specific Practices, and the Generic Practices, to fit for our architecture evaluation purpose. We used multiplicity indicators for class relations to add a basic metamodel semantic. Not indicated multiplicities corresponds to the default 1 cardinality or a 1..1 multiplicity.

The semantics of these maturity levels as in [1] were adapted from [6] to conform to the architecture assessment scope for service-oriented enterprise systems. In terms of requirements from customer oriented domain-models and reference use scenarios, our model has introduced in [8] five maturity levels, which define architecture assessment criteria for service-oriented enterprise systems and help to measure the architecture maturity, like Initial Architecture, Managed Architecture, Defined Architecture, Quantitatively Managed Architecture, and Optimizing Architecture.

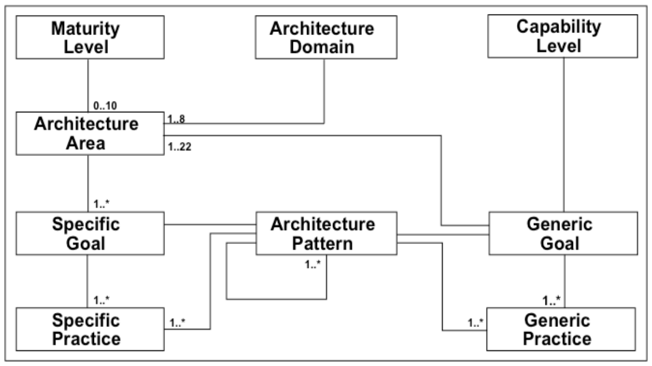


Figure 1: SOAMMI Metamodel – Main Concepts.

We have derived the architecture domains mainly from TOGAF where they are used as specific architecture subtypes and corresponding phases of the TOGAF-ADM (Architecture Development Method). Architecture areas cover assessable architecture artifacts and are correspondent, but very different, parts of process areas from CMMI. To fit our architecture assessment scope, we have defined 22 original architecture areas of the SOAMMI framework [1] and [8], linked them to our architecture maturity levels and ordered them in line with our specific enterprise and software architecture domains. Each of the delimited architecture area is accurately described in a catalog including *name* of architecture area, *short identification* of architecture area and a *detailed description*.

SOAMMI supports both the staged and continuous representations. The same staging rules as in CMMI apply to SOAMMI and should therefore enable the flexible adoption of both model representations: *Continuous* for assessing single architecture areas and *staged* for assessing the whole architecture maturity. The assessment of capability levels could be applied to iterate specific architecture areas or to assess or improve a focused innovation aspect, involving one or more architecture areas. To verify and support persistent institutionalizations of architecture areas we introduce architecture related generic goals and practices. All architecture areas are affected by the same generic goals and associated generic practices. In the following, two example architecture areas together with their goals and practices are presented.

B. Example of Architecture Area

Business Processes & Rules

Purpose: Structure, design, model, and represent business value chains and business processes to support business capabilities.

Maturity Level: 2

Specific Goals (SG) and Specific Practices (SP):

SG 1: Model Business Value Chains as Root of Business Capabilities and Business Processes

SP 1.1 Identify business value for business operations

SP 1.2 Structure value chains

SP 1.3 Optimize business considering customer channels and supplier networks

SG 2: Model and Optimize Business Processes

SP 2.1 Identify business activities for business processes: system activities, user interaction activities, and manual activities

SP 2.2 Structure business processes for business roles and organizational units

SP 2.3 Define business workflows and business process rules

SP 2.4 Model and represent business processes

SG 3: Model and Represent Business Control Information

SP 3.1 Identify and represent control information for product monitoring

SP 3.2 Identify and represent control information for process monitoring.

III. PATTERN COLLECTIONS AND LANGUAGES

Design patterns originated as an architectural concept introduced in the seminal book ”A Pattern Language” written by Christopher Alexander [16]. He captured his experience gathered over time and structured this knowledge in smaller units as patterns describing good qualities of a real world examples. The level of detail varied from landscape characteristics over areas, quarters up to single parts of houses and even rooms. Alexander structured the different patterns by means of size. This way, patterns explaining concepts of larger areas were explained first, connecting the currently read pattern to descendent patterns with a higher level of detail.

Alexander's intention was to describe best practices and effective design solutions in order to share design knowledge with other people facing similar problems in a related context. The solution proposed by a software pattern should be generic rather than specific, such that it can be implemented in numerous different ways. The benefit of using patterns is that they communicate insights into common design problems and reflect solutions that a community of experts has developed over time.

An important quality of a pattern was the readability by non-experts. Since every pattern was written in prose with a standard vocabulary, the concepts could be understood by a large readership that was not necessarily experts in the domain.

This thought of structuring knowledge in patterns was picked up in many different computer-science domains like software design, human-computer interaction design, website design and many others.

In particular, Gamma et al. extended the notion of patterns into the domain of software engineering, and constructed twenty-three classic software design patterns [2]. Since then, the concept of design patterns also became essential in the domain of Human-Computer-Interaction (HCI), where patterns are commonly used to describe and preserve solutions to recurring user interface design problems. Borchers transferred the pattern concept to human-computer interaction design for interactive exhibits [17]. From his point of view, especially patterns in HCI need to bridge the gap between users with conceptual knowledge and understanding of the problem domain and software engineers who are deeply involved in the technical development.

A given pattern is not always the optimal solution in every case, but tends to work in practice and supports user acceptance for the system. Tidwell describes the influence of patterns in user interface design stating that each implementation of the same pattern differs somehow in its characteristics although it comes from the same origin [18]. Thus, patterns should be seen as description of a problem solution as starting-point and not as fixed design rules.

A similar approach is introduced by Schümmer and Lukosch in the domain of computer-supported collaboration [19]. They structure their pattern language along the level of technical complexity: The more detailed a pattern describes a certain solution, the more technical this description becomes. Up to a certain degree of detail, they consider all patterns as relevant for all stakeholders. Beyond that point, the target group changes to engineers that need to technically implement the design suggestion.

In addition to working solutions, the description of *anti-patterns* is also a valid information source for application and interface designers. They document surprisingly failing approaches that turn out to be ineffective or counter-productive in practice [20]. Other collections, e.g., in UI design, focus on pointing out repetitions of design flaws [21]. Here, concepts that have intruded many designs but actually lead to rejection are discussed and the reasons for design failures are explained.

A collection of patterns, which are organized in a directed acyclic graph structure, is referred to as a *pattern language*. Elements of a pattern language are navigable sequences of patterns. In contrast to pattern language, pattern *collections* provide semi-structured clusters of patterns that are not interconnected in a hierarchy. This is for example the case in [2] who distinguish between structural, creational and behavioral patterns.

IV. Pattern Innovation for Architecture Diagnostics

Although design patterns are mainly used to inform the design of a system, they are also applied as test cases for assessing software. Software architecture assessment patterns are based on the seminal work of software patterns originated from the work of [16].

Our pattern language for architecture assessments of service-oriented enterprise systems provides a procedural method framework for architecture assessment processes and questionnaire design. This method framework of our new introduced pattern language was inspired from [20], and derived from the structures of the metamodel of SOAMMI as well as from our initial pattern catalog from previous research [10].

We have linked each specific and each generic goal within our assessment framework to a distinct pattern of our pattern language. We organize and represent our architecture assessment patterns according to the following structures: *Architecture Domains, Architecture Areas, Problem Descriptions* - associated with *Specific Goals, Solution Elements* that are connected to *Specific Practices* and *Related Patterns*, which are subsequent connections of applicable patterns within the pattern language.

Linking elements to specific practices of the SOAMMI framework indicate solutions for architecture assessments and improvements of service-oriented enterprise systems. This assessment and improvement knowledge is both verification and design knowledge, which is a procedural knowledge based on standards, best practices, and assessment experience for architecture assessments of service-oriented enterprise systems. It is therefore both concrete and specific for setting the status of service-oriented enterprise architectures, and helps to establish an improvement path for change. Patterns of our language show what to assess. Our patterns aim to represent verification and improvement knowledge to support cooperative assessments synchronizing people in cyclic architecture assessments.

Associated with our architecture assessment pattern language we have set up an assessment process to show how to assess architecture capabilities. This process is based on a questionnaire for architecture assessment workshops providing concrete questions as in [8], answer types, and helping to direct and standardize the related assessment process. Additionally, we have included process methods for workshops, result evaluations, improvement path information for technology vendors and for application organizations, as well as change support and innovation monitoring instruments.

We have identified and distinguished a set of 43 patterns as parts of a newly researched and introduced pattern language in the context of 7 Architecture Domains and 22 Architecture Areas. Even though our architecture quality patterns accords to the Specific Goals, the Specific Practices and the Generic Goals from the SOAMMI framework, they extend these structures by navigable patterns as part of an architecture assessment language. Only this pattern structure enables architecture quality assessors to navigate easily in two directions to support the diagnostics and optimization process, and to provide a clear link to questionnaire and the related answer and result concepts. The full collection of patterns of the architecture assessment pattern language was derived from the SOAMMI framework (cf. Section II).

A. *Evaluation and Findings*

The practical benefits of our SOAMMI assessment pattern language were demonstrated by the successful use as guideline for the questionnaire design in four major capability assessments of service-oriented vendor technology architectures, as in [1] and [8]. Architecture assessments need to address key challenges for companies during the built-up and management of service-oriented architectures.

SOAMMI seems to be complex in practice. Therefore a pragmatic simplification of the SOAMMI framework was particularly required in counting assessment results. Additionally, we have considered for our assessments specific user requirements from companies using and providing service-oriented enterprise systems.

Following these ideas, the basic structure of our questionnaire in [8] was taken from the SOAMMI architecture areas with one or more questions per Specific Goal. User requirements have been consolidated and mapped against specific goals. Wherever no user requirements could be mapped, Specific Practices have been used to generate questions on the level of specific goals. Through this procedure each Specific Goal could be related to at least one concrete question.

The assessment process takes about 3 months in total to complete for each software technology provider. The first step is a pre-workshop (2-3 hours) to make sure that the architecture provider can identify the appropriate experts for the assessment workshop itself. Then the actual assessment workshop (4- 6 hours) is held a few weeks later, so that the provider has enough time to identify the experts that should participate and prepare answers. Finally, a series of follow up workshops for specific questions (3-4 hours each) are arranged with the system technology provider.

B. *Shortcomings for Updating and Refining*

The pattern catalogue that serves as a basis for our assessments is continuously a subject of consideration with regard to pattern refinement, pattern improvement and catalogue extension. In parallel to the assessments, feedback on the state of the patterns that were used during the evaluation is gathered.

This way, we have a chance to update existing patterns or derive variants of them. However, we cannot be sure that a new pattern or derivation is really valid. On the other side, the variant or new formulation can be a promising pattern candidate and later be validated and therefore be integrated into the pattern catalogue in order to use and benefit from it as early as possible.

The current process, however, does not foresee the inclusion of non- or semi-validated patterns. The validation process of a pattern also is a time-consuming process with much iteration. It can partially be combined with additional SOA assessments but then still a subset of new patterns needs to be investigated in more depth.

So, our aim is to gather the feedback, adjust our current findings and preserve knowledge, feedback and new findings within our catalogue.

For this reason, we aim at establishing an evolution process that makes it possible to integrate early results into the existing pattern catalogue. Continuous refinement and therefore the lifelines of the pattern catalogue need to be ensured. The requirements for such a process were already defined in preliminary work [18] and [23]. The process itself is described in the following section.

Traditionally, pattern collections are published after a long period of development and validation where the essences from design experience can be extracted. This is mostly done by a small, closed group of design experts as described in [24]. In the approaches presented in the previous sections, much effort was put into the derivation and evaluation of mature and evaluated patterns.

However, we see the problem that many findings must be regarded earlier, at the state of an idea in order to be able to consider many findings in a flexible pattern set. This holds the chance to start working with patterns very early – even if it not yet fully proven. Our process [25] wants to include new ideas and concepts into the project's lifecycle as early as possible. Over time, the idea, which is directly formulated as a *pattern candidate*, gets refined and evaluated.

In this scenario, it may turn out that the candidate is not a pattern and needs to be rejected. Alternatively, after continuous refinement and evaluation the pattern candidate may become more mature, reaching a new state, e.g., being "under consideration". The counter-result is also possible: A promising pattern idea may also turn out to lead to a bad decision or concept. In this case, we speak of a surprisingly failing solution. In order to avoid similar failures in the future, we formulate this concept as an anti-pattern. This way, the pattern gets a warning character, allowing follow-up to directly cross out this idea and alter considerations.

V. Conclusion and future Work

In this work, we have motivated the necessity to extend existing SOA maturity models to accord to a clear metamodel approach due to the verified CMMI model. Based on the related work to CMMI, which is an assessment and improvement model for software processes but not for architectures, we have developed suitable models for assessments of service-oriented enterprise systems. Our specific architecture assessment approach of the SOAMMI framework was founded on current architecture standards like TOGAF and architecture assessment criteria from related work approaches.

The presented SOAMMI framework was validated in consecutive assessment workshops with four global vendors of service-oriented platforms and has provided transparent results for subsequent changes of service oriented product architectures and related processes. Our current research extends SOAMMI to support architecture diagnostics for complex integrated enterprise systems in the emerging context of services and cloud computing architectures.

Our empirical validation and optimization of the presented maturity framework is an ongoing process, which has to be synchronized with future cyclic evaluations of SOA platforms and their growing number of services. Extended validations of customers of service-oriented technologies are planned for the next phase of our framework research and development.

The need for iteratively updating our assessment pattern collection motivated us to merge the efforts done for SOA assessment with a flexible and iterative pattern refinement and creation process. After talking about SOA maturity and assessment, we looked at the concept of involving *many* stakeholders into the pattern creation and evolution process and to adapt already available knowledge and findings from the project's domain as early as possible.

Our presented *pattern-lifecycle process* allows for continuously evaluating gathered knowledge during the project's lifetime and makes patterns as well as pattern ideas available during the whole development process. This way, pattern collections can be formulated collaboratively without needing to wait for a closed author group that shares its well-evaluated design knowledge after a longer period of time.

Additional improvement ideas include an architecture pattern and knowledge repository, as well as patterns for visualization of architecture artifacts and architecture control information, to be operable on an architecture management cockpit. We are working at extending our pattern language to a full canonical form in order to support fully standardized cyclic architecture assessments for service-oriented products and solutions. The pattern evolution process represents a new aspect to the assembly and structuring our patterns and will further explored in the SOA assessment domain.

ACKNOWLEDGMENT

This paper extends ideas from the SOA Innovation Lab, which is a major research and innovation network on Enterprise Architecture Management for Services and Cloud Computing in Germany and Europe.

Parts of the research leading to these results has received funding from the European Union Seventh Framework Program (FP7/2007-2013) under grant agreement n°261817, the BRIDGE project (www.bridgeproject.eu) under the Security Program SEC-2010.4.2-1.

REFERENCES

[1] H. Buckow, H.-J. Groß, G. Piller, K. Prott, J. Willkomm, and A. Zimmermann, "Analyzing the SOA Ability of Standard Software Packages with a dedicated Architecture Maturity Framework," in *EMISA*, 2010, pp. 131-143.

[2] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software*, 1st ed. Amsterdam: Addison-Wesley Longman, 1994, p. 416.

[3] T. Erl, "*SOA Design Patterns*", Prentice Hall. 2009.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Chichester, UK: Wiley, 1996.

[5] P. Bianco, R. Kotermanski, and P. Merson, "Evaluating a Service-Oriented Architecture," *Engineering*, no. September, pp. 1-91, 2007.

[6] CMMI-DEV-1.3 2010 "*CMMI for Development, Version 1.3*", Carnegie Mellon University, Software Engineering Institute, SEI-2010-TR-033, 2010.

[7] TOGAF "*The Open Group Architecture Framework*" Version-9.1, The Open Group, 2011.

[8] A. Zimmermann, H. Buckow, H.-J. Gross, O. F. Nandico, G. Piller, and K. Prott, "Capability Diagnostics of Enterprise Service Architectures Using a Dedicated Software Architecture Reference Model," *Services Computing, IEEE International Conference on*, vol. 0, pp. 592-599, 2011.

[9] A. Zimmermann and G. Zimmermann, "*ESARC - Enterprise Services Architecture Reference Cube for Capability Assessments of Service-oriented Systems*", SERVICE COMPUTATION 2011 - The Third International Conferences on Advanced Service Computing, September 25-30, 2011 Rome, Italy, ISBN 978-1-61208-152-6, IARIA Proceedings of SERVICE COMPUTATION 2011, pp. 63-68.

[10] A. Zimmermann, F. Laux, and R. Reiners, "A Pattern Language for Architecture Assessments of Service-oriented Enterprise Systems," in *PATTERNS 2011, Third International Conferences on Pervasive Patterns and Applications*, 2011, no. c, pp. 7-12.

[11] D. Krafzig, K. Banke, and D. Slama, „*Enterprise SOA*", Prentice Hall, 2005.

[12] "Essential Architecture Project." [Online]. Available: http://www.enterprise-architecture.org. [Accessed: 11-Mar-2012].

[13] S. Inaganti and S. Aravamudan, "SOA Maturity Model," *BP Trends*, no. April, pp. 1-23, 2007.

[14] Sonic: "*A new Service-oriented Architecture (SOA) Maturity Model*", http://soa.omg.org/Uploaded%20Docs/SOA/SOA_Maturity.pdf, [Accessed: 11-Mar-2012].

[15] ORACLE, "ORACLE: 'SOA Maturity Model'."[Online]. Available: http://www.scribd.com/doc/2890015/oraclesoamaturitymodelcheatsheet. [Accessed: 11-Mar-2012].

[16] C. Alexander, *A Pattern Language: Towns, Buildings, Construction*. New York, New York, USA: Oxford University Press, 1977.

[17] J. Borchers, *A Pattern Approach to Interaction Design*, 1st ed. John Wiley & Sons, 2001, p. 268.

[18] J. Tidwell, *Designing Interfaces*, 1st ed. O'Reilly Media, 2005, p. 352.

[19] T. Schümmer and S. Lukosch, *Patterns for Computer-Mediated Interaction*. Chistester, West Sussex, England: John Wiley & Sons, 2007, p. 600.

[20] R. Reiners, I. Astrova, and A. Zimmermann, "Introducing new Pattern Language Concepts and an Extended Pattern Structure for Ubiquitous Computing Application Design Support," in *PATTERNS 2011, Third International Conferences on Pervasive Patterns and Applications*, 2011, pp. 61-66.

[21] J. Johnson, *GUI bloopers 2.0: Common User Interface Design Don'ts and DOS*, vol. 2, no. October. New York, NY, USA: Morgan Kaufmann, 2007.

[22] T. Grill and M. Blauhut, "Design Patterns Applied in a User Interface Design (UID) Process for Safety Critical Environments (SCEs)," in *HCI and Usability for Education and Work*, vol. 5298, A. Holzinger, Ed. Springer Berlin / Heidelberg, 2008, pp. 459-474.

[23] C. R. Prause, "Reputation-based self-management of software process artifact quality in consortium research projects," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 380-383.

[24] "The BRIDGE Design Pattern Library." [Online]. Available: http://pattern-library.sec-bridge.eu/. [Accessed: 11-Mar-2012].

[25] R. Reiners, "*A Pattern Evolution Process – From Ideas to Patterns*", Proceeedings Informatiktage 2012 Bonn - Germany, in Lecture Notes in Informatics, Vol. S-11, 2012, pp. 115-118.

Iris Recognition: Existing Methods and Open Issues

Sajida Kalsoom
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan
Email: sajida.kalsoom@comsats.edu.pk

Sheikh Ziauddin
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan
Email: sheikh.ziauddin@comsats.edu.pk

***Abstract*—Biometric authentication uses unique physical or behavioural patterns in humans to identify individuals. Though biometric is generally considered most reliable, stable and unique among all entity authentication means, it is not as stable and unique as is usually conceived. In this paper, we highlight the issues with current state-of-the-art iris-based biometric authentication systems. This survey covers the review of existing iris recognition methods with a focus on enumerating the open issues that must be addressed in order to be more confident in the performance, security and privacy aspects of iris-based biometric systems.**

***Keywords*-Pattern recognition; biometric authentication; iris recognition; template security**

I. Introduction

With the increase in use of biometrics for human identification, control shifts to identifying the factors that affect the performance of biometric authentication systems. Biometric authentication systems use behavioural or physical characteristics to authenticate a user. These systems have become more reliable sources of authentication as compared to the traditional means like passwords or hardware tokens such as smart cards. Reliability of biometric authentication systems lies in the fact that, unlike passwords and smart cards, biometrics cannot easily be forged, shared, compromised or forgotten. Biometric is considered to be highly unique among all human population. Genetically, same identities including twins and irises of left and right eye of the same person represent different iris patterns [8]. Another important property of biometric is its stability [23][24][25][34]. In this paper, we will critically analyse these claims by showing counter-examples from other researchers' work. These will be discussed in the problems and open issues section in detail.

The rest of the paper is organized as follows. Section II provides an overview of existing iris recognition methods. Section III presents issues, problems and challenges associated with template security and recognition performance. The paper is concluded in Section IV.

II. Iris Recognition

Iris recognition is considered as one of the most reliable biometric authentication technique [9][19][35][37]. An iris recognition system captures human eye image using a near infrared iris sensor which passes through three steps to be transformed into an iris template. These three steps are iris segmentation, iris normalization and iris feature encoding. The iris segmentation procedure segments the annular iris region from the entire eye image. First, it finds the inner and outer boundaries (the iris-pupil and iris-sclera boundaries) of the iris, then it marks the region of the annular iris ring that is not visible due to eyelids and eyelashes. The iris normalization procedure transforms the segmented iris region into a fixed size to cater for variations in iris sizes among different eye images. The feature encoding procedure extracts the most distinguishing features from normalized iris images and typically encodes the result as a binary string.

Recognition involves either verification or identification. Verification is one to one comparison where claim of an identity is verified, e.g., an employee of an office. On the contrary, identification is one to many comparison where an identity is watched against an entire database, e.g., a criminal surveillance system. In the verification step, the recognition time captured image is compared with the image taken at the enrolment time. The comparison is mostly done by calculating the Hamming distance where a value of 0 represents a perfect match and a value of 1 represents perfect non-match.

This paper is not primarily a survey on iris recognition techniques, but is to identify performance and security related issues with existing techniques. So, we will briefly describe just a couple of representative systems, followed by a table, reproduced from [5], providing a comparative analysis of a number of state-of-the-art iris recognition systems.

The most famous iris biometric system is due to Daugman [7][8]. In Daugman's system, iris segmentation is performed by using the following optimization:

$$\max_{(r,x_0,y_0)} \left| G_\sigma(r) * \frac{\partial}{\partial r} \oint_{C(s;r,x_0,y_0)} \frac{I(x,y)}{2\pi r}\, ds \right|,$$

where r and (x_0, y_0) are candidates for the radius and center of the iris, $G_\sigma(r)$ is the one-dimensional Gaussian with standard deviation σ, $*$ is the convolution operator, $C(s; r, x_0, y_0)$ is the circular closed curve with center (x_0, y_0) and radius r, parameterized by s, and $I(\cdot,\cdot)$ is the input eye image. Noise due to eyelids occlusion is avoided by restricting ds to the nearly vertical regions. The above optimization is performed twice to find both iris and pupil circles. For template generation, Daugman uses phase information of the image. After detecting the iris boundaries and removing the irrelevant region, 2D

Gabor wavelets is applied on normalized iris image to the iris template. For comparison of iris templates, Hamming distance metric is used. Most of the subsequent work on iris recognition, follows Daugman's approach of using Hamming distance for template matching

After Daugman's iris recognition system, one of the most important and popular systems is due to Wildes [38][39]. For iris segmentation, Wildes first detects edges in the eye image and then applies a circular Hough transform to find circular pupil and iris boundaries. Much of the subsequent work on iris segmentation follows Wildes approach where a common variation is the usage of a coarse-to-fine strategy. For template generation, Wildes uses Laplacian of Gaussian filter applied at multiple scales to extract unique information from iris texture. His system uses normalized correlation between the templates for template matching at verification time.

As mentioned above, most of the subsequent work in iris recognition follows the above-mentioned seminal approaches. Most work on iris segmentation is a variation and enhancement of Wildes' approach, while most feature extraction schemes are variations on Daugman's wavelet-based approach. A very nice detailed survey of iris recognition techniques is due to [5]. Table I (reproduced from [5]) provides a quick comparison of recognition results for some of the important iris recognition techniques. The interested reader is referred to [5] for a detailed study of existing iris recognition techniques.

III. Problems and Open Issues

We categorize issues of iris recognition systems in two broad classes namely those related to iris template security and those associated with iris recognition performance. Details are as follows.

A. *Iris Template Security*

As biometric is an integral part of human body, loss of one's biometric corresponds to loss of his/her identity. Therefore, security of biometric templates is one of the most important concerns in any biometric authentication system. In literature, we found four types of biometric systems which are described below along with related issues and challenges.

1) Traditional Biometric Systems: These are the conventional systems [7][25][38] which store users' templates in clear form to verify the identity. A template is generated at enrolment time, stored in the database without encryption/hashing and compared with the corresponding verification template at the verification time. As the template is used and stored in plaintext, a compromise of database has severe security and privacy implications. There are scenarios where users use the same biometrics for multiple applications or different organizations share data among themselves for their users. In such scenarios, cross-matching becomes feasible for tracking individual users [27][30][31].

2) Biometric Key Release: These are the systems where biometrics along with cryptographic keys are used for authentication and communication [32]. The effort lies in using biometric templates effectively to release cryptographic keys in a secure manner. Modern cryptographic keys are uniformly random and large in size, therefore it is not feasible for users to memorize them. In biometric key release systems, cryptographic keys are stored at some location and are released using biometric information of the user. When user inputs his/her biometric, cryptographic key is released for use in any security protocol. This way, the key would be released only to the authorized users.

Though these systems use biometric information effectively for cryptographic key storage and release, there are certain issues which are not addressed by these systems. First, though these systems secure cryptographic key using biometric template, the template itself still remains unprotected. This leads to all security and privacy issues discussed earlier. Second, these systems fail to provide revocability of biometric templates meaning that if it is known that biometric template of a

TABLE I
Comparison of the most cited iris recognition techniques [5]

First Author, Year	Database Size	Results
Alim, 2004	Not given	96.17%
Jang, 2004	1694 images including 160 w/glasses and 11 w/contact lenses	99.1%
Krichen, 2004	700 visible-light images	FAR/FRR: 0%/0.57%
Liu, 2005	4249 images	97.08%
Ma, 2002	1088 images	99.85%, FAR/FRR: 0.1%/0.83%
Ma, 2003	2255 images	99.43%, FAR/FRR: 0.1%/0.97%
Ma, 2004	2255 images	99.60%, EER: 0.29%
Ma, 2004	2255 images	100%, EER: 0.07%
Monro, 2007	2156 CASIA images and 2955 U. of Bath images	100%
Proenca, 2007	800 ICE images	EER: 1.03%
Rossant, 2005	149 images	100%
Rydgren, 2004	82 images	100%
Sanchez-Reillo, 2001	200+ images	98.3%, EER: 3.6%
Son, 2004	1200 images, (600 used for training)	99.4%
Sun, 2004	2255 images	100%
Takano, 2004	Images from 10 people	FAR/FRR: 0%/26%
Thornton, 2006	CMU database, 2000+ images	EER: 0.23%
Thornton, 2007	CMU database, 2000+ images	EER: 0.39%
Tisse, 2002	300+ images	FAR/FRR: 0%/11%
Yu, 2006	1016 images	99.74%

particular user has been compromised, it is not feasible for him/her to change his/her secret in contrast to password or hardware token-based systems.

3) Cancelable Biometrics: Cancelable biometric systems apply some transformation on the biometric template to secure the template [6][21][33][42]. The idea is that, instead of directly storing the template, a function is applied on the template and the output of that function (transformed template) is stored in the database. The transformation function must be non-invertible so that a compromised transformed template cannot be translated to the original template. The major advantage of cancelable biometric systems is that even if the transformed template is compromised, the original template still remains secure. In addition, the secret can easily be revoked by applying a different transformation to the original template resulting in a new transformed template. Moreover, a user can have different transformed templates for different applications he/she is using hence making cross-matching infeasible for any potential attacker.

Finding a suitable transformation function can be quite tricky in cancelable biometric systems. Standard non-invertible transformation functions (such as one-way cryptographic hash functions) do not work with biometric data due to intra-class variability of biometric data. Therefore, in most cases, transformation is user-dependent, i.e., user either has to remember a password/pin or to carry a token which stores the transformation parameters. This puts an extra burden on the user and effectively converts the system into two-factor authentication scheme. It is also desirable to observe user specific key to check the strength of user-provided secret.

4) Biometric Key Generation: In such systems, biometric template and cryptographic key are bind together [11][14][20][22][26][41]. Cryptographic key can either be generated directly from biometric template [14][20][22][41] or by using standard cryptographic techniques [11][26]. In former case, generated key is not uniform and hence may not be strong enough for use in many cryptographic protocols. In biometric key generation systems, neither biometric template nor cryptographic key is stored in cleartext. Instead, a value obtained by binding these two secrets is stored such that it is not feasible to get any of the two secrets from this bound value.

Though last three non-traditional systems described above are quite effective in resolving template security related issues in biometric recognition systems, in most cases, recognition performance is affected. In addition, speed of these systems is always slower as compared to conventional iris recognition systems. Moreover, most of these systems do not perform well with noisy iris image datasets. Due to all these issues, we can conclude that a reliable and efficient solution to solve template security related issues is yet to be achieved.

B. Iris Recognition Performance

An iris recognition system is considered ideal when match and non-match distributions do not overlap each other. There are a few factors which may lead to a significant drop in accuracy of iris recognition systems. These are detailed as follows.

1) Dilation: One of the important but often ignored factor is pupil dilation. Due to dilation effects, we have varying size of pupil, which results in decreased recognition performance. Dilation may occur due to many factors such as drugs, sunglasses, light illumination, etc.

Experimental studies are presented by Hollingsworth et al. identifying the effects of pupil dilation on iris recognition performance [15][16]. To produce dilation, they used ambient light for controlled intervals of time. Degree of dilation was measured by taking the fraction of pupil and iris radius. They conducted two experiments, one to find out the effects of dilation of same degree (between two templates to be matched) and second with varying dilation. Their findings indicate that 1) If both images have same but high pupil dilation, this results in lower recognition performance as compared to images with no dilation. 2) If images have different amount of pupil dilation, this results in further increasing of False Reject Rate (FRR).

Effects of pupil dilation on iris recognition performance have been studied by other researchers also [4][10][29]. Rakshit and Monro [29] have used eye drops to achieve the effects of dilation. For their experiments, they collected images before and after 5, 10 and 15 minutes of instilling of drops. In most cases, due to the instillation of drops, pupil lost its shape and they used their shape-description method to generate accurate normalized images. Their experiments also showed a decrease in recognition performance due to iris dilation. They also observed that, with the increase in time, dilation is increased leading to an increase in FRR. Dhir et al. [10] later extended their study with 15 subjects as compared to 11 in [29]. They found the same results namely dilation results in poor performance and false reject rate increases with increase in dilation which in turn increases with time after eye drops have been administered.

Bowyer et al. [4] categorized iris images in three classes based on amount of pupil dilation namely *small, medium* and *large*. For experiments with varying amount of dilation, their results show that the larger the difference in dilation ratio, the more the chances of false non-match. For experiments with same amount of dilation, their findings indicate that increasing the degree of dilation, increases the false match and false non-match.

From the above studies, it can safely be concluded that it is not that difficult to deceive iris recognition systems which is contrary to the popular belief in research community. Pupil dilation not only affects the recognition performance but an intruder can easily deceive the system by just wearing sunglasses or by using eye drops. Pupil dilation factor should be incorporated in iris recognition systems to increase confidence in recognition results.

2) Lenses: Around the world, approximately 125 million people use contact lenses. Therefore, iris recognition systems should be flexible enough to accommodate these large number of people. Designers of iris recognition algorithms claim that recognition performance of their systems is not affected by the use of contact lenses[1][8][28][40]. But, recently, Baker et al. [2] come up with a study showing that every type of lens negatively affects iris recognition performance. They used a dataset containing 51 subjects with contact lens and 64 without lenses. After visual inspection of iris images, they categorized lenses into four categories. First category includes lenses that are visible but have no effects on the iris. Second category includes images that result in light or dark outline around iris and sclera. Third category includes lenses with large artefacts on the iris that are mainly due to written logo/number or misfit lens. Fourth category is one having subjects with hard lenses.

They conducted two experiments. First experiment compares results of contact lenses and non-contact lenses subjects while the second experiment compares results of different categories of contact lenses. In first experiment, false reject rate for subjects with lenses is 9.42% and 0.719% for subjects without contact lenses. This shows that contact lenses have a severely adverse effect on iris recognition accuracy. The second experiment showed that second category is the one with the lowest FRR of 3.9% whereas fourth category has highest FRR of 45.44%. Category one and three have also shown high false reject rates of 10.64% and 14.37%, respectively. As is clear from results, lenses of all types affect the verification results little or more depending on the type.

Baker et al. [3] later conducted a larger study on the effects of lenses. They used three different systems for iris recognition namely IrisBEE, VeriEye and CMU. They also categorised lenses in four types. The results show that false reject rate for subjects with lenses is much higher than that for subject without lenses. In addition, category four of hard lenses showed worst recognition results among all lens types for all three iris recognition systems.

Bowyer et al. [4] conducted a similar study to evaluate iris recognition performance among subjects wearing contact lenses. Their findings are that false non-match score was almost same for contact lens and non-contact lens groups while false match score was 0.27% for non-contact lens group and 5.64% for contact lens group showing a significant drop in recognition accuracy. From the above reported studies and results, the effects of the contact lenses are apparent on recognition performance. All types of lenses result in performance degradation so there is need to introduce techniques that can handle such scenarios to strengthen iris biometric systems.

3) Twins: In [17][18], Hollingworth et. al presented studies identifying the texture similarities between irises of twins. Their work is in contrast to the previous work which focuses on identifying the differences between genetically same identities. To conduct their experiments, they collected the data on twins day festival Twinburg in Ohio in August 2009.

They also collected the data from unrelated people to do comparative analysis. At first step, they performed biometric system testing and their findings are same as those of the old researchers, i.e., for iris biometric system, irises of twins are more or like similar as those belonging to unrelated people. At the second step, they performed user testing to identify similarities between irises of twins.

They conducted two user studies. First, where only irises of subjects are presented to respond to the queries and second where periocular images are displayed to the user to respond to the queries. On the iris image experiments, the average success score is 81.3% and for periocular queries success score is 76.5%. Their findings indicate that there are similarities between the irises of genetically same users which can be visually identified, but current biometric systems do not identify them. It is required to explore further and establish techniques so that biometric systems may utilize this visual similarity between genetically similar irises for the benefit of performance enhancement.

4) Time Variability: Human iris is considered stable over time [23][24][25][34]; but, a recent study by Gonzalez et al. [36] shows results which contradict what has been demonstrated so far. For their experimental evaluation, Gonzalez et al. used BioSecurId [12] and BioSec [13] baseline datasets. The former dataset consist of 254 individuals (8128 images) captured in four different sessions and later has 200 subjects (3200 images) captured in two different sessions, both splitted by a time span of one to four weeks. Results show that errors rate is increased in inter-session experiments compared with the intra-session ones. Their finding indicates that, as the lapse time between enrolment and comparison is increased, false accept rate remains unaffected but false reject rate is increased up to more than twice. Bowyer et al. [4] conducted a similar research to find the effect of time variability on recognition performance. Their recognition results also showed that as the time between enrolment and verification increases, false reject rate of the system also increases though that increase is less significant than that reported by Gonzalez et al. Although, research results show that time variability affects verification performance but to be more confident in extent of this effect, more research with large datasets is desirable.

5) Cataract Surgery: In [10] Dhir et al. and [29] Rakshit et al. identified the effects of cataract surgery on recognition performance. In [29], they collected the images of 3 patients before and after two weeks of cataract surgery. The results of pre and post surgery images comparison shows that cataract surgery does not affect recognition performance. Later on, Dhir et al. [10] did similar experiments with 15 subjects and found same results. Although the study is significant, but as the dataset was not large, there is need to do more experiments on larger datasets to explore the effects.

6) System Portability: To check system portability related issues, Bowyer et al. [4] performed experiments on a set of iris images acquired using different sensors namely LG 2200 and LG 4400. Experiments show false reject rate is higher when both images (enrolment and verification) are from different sensors compared with the results where both images are from the same sensor. The study is done on limited dataset and only using IrisBEE software. There are chances that results may be affected differently by different software and hardware. A larger research is needed to explore effect of different sensors on iris recognition performance.

IV. Conclusion and Future Work

This review paper summarizes the issues and challenges with current iris biometric systems. In particular, we discussed security and performance related issues. We have shown that many popular beliefs about security, reliability, stability and performance of iris recognition systems are not correct and need to be revisited. The issues raised in this survey should be addressed in order to be more confident in working of iris recognition systems. In the future, we plan to explore the security and privacy concerns facing other biometric systems. This could lead to a design of multi-biometric system that overcomes the weaknesses of one by the strength of other biometric.

References

[1] J. Ali and A. Hassanien. An iris recognition system to enhance e-security environment based on wavelet theory. *AMO-Advanced Modeling and Optimization*, 5(2):93–104, 2003.

[2] S. Baker, A. Hentz, K. Bowyer, and P. Flynn. Contact lenses: Handle with care for iris recognition. In *International Conference on Biometrics: Theory, Applications, and Systems (BTAS)*, pages 1–8, 2009.

[3] S. Baker, A. Hentz, K. Bowyer, and P. Flynn. Degradation of iris recognition performance due to non-cosmetic prescription contact lenses. *Computer Vision and Image Understanding*, 114:1030–044, 2010.

[4] K. Bowyer, S. Baker, A. Hentz, K. Hollingsworth, T. Peters, and P. Flynn. Factors that degrade the match distribution in iris biometrics. *Identity in the Information Society*, 2(3):327–343, 2009.

[5] K. Bowyer, K. Hollingsworth, and P. Flynn. Image understanding for iris biometrics: A survey. *Computer Vision and Image Understanding*, 110(2):281–307, 2008.

[6] J. Bringer, H. Chabanne, and B. Kindarji. The best of both worlds: Applying secure sketches to cancelable biometrics. *Science of Computer Programming*, 74(1-2):43–51, 2008.

[7] J. Daugman. High confidence visual recognition of persons by a test of statistical independence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1148–1161, 1993.

[8] J. Daugman. How iris recognition works. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(1):21–30, 2004.

[9] S. Dhavale. Robust iris recognition based on statistical properties of walsh hadamard transform domain. *International Journal of Computer Science*, 9, 2012.

[10] L. Dhir, N. Habib, D. Monro, and S. Rakshit. Effect of cataract surgery and pupil dilation on iris pattern recognition for personal authentication. *Eye*, 24(6):1006–1010, 2009.

[11] H. Feng and C. Wah. Private key generation from on-line handwritten signatures. *Information Management & Computer Security*, 10(4):159–164, 2002.

[12] J. Fierrez, J. Galbally, J. Ortega-Garcia, M. Freire, F. Alonso-Fernandez, D. Ramos, D. Toledano, J. Gonzalez-Rodriguez, J. Siguenza, J. Garrido-Salas, et al. BiosecurID: a multimodal biometric database. *Pattern Analysis & Applications*, 13(2):235–246, 2010.

[13] J. Fierrez, J. Ortega-Garcia, D. Toledano, and J. Gonzalez-Rodriguez. Biosec baseline corpus: A multimodal biometric database. *Pattern Recognition*, 40(4):1389–1392, 2007.

[14] F. Hao, R. Anderson, and J. Daugman. Combining cryptography with biometrics effectively. *University of Cambridge Computer Laboratory, Tech. Rep*, 2005.

[15] K. Hollingsworth, K. Bowyer, and P. Flynn. The importance of small pupils: a study of how pupil dilation affects iris biometrics. In *2nd IEEE International Conference on Biometrics: Theory, Applications and Systems, 2008. BTAS 2008.*, pages 1–6. IEEE, 2008.

[16] K. Hollingsworth, K. Bowyer, and P. Flynn. Pupil dilation degrades iris biometric performance. *Computer Vision and Image Understanding*, 113(1):150–157, 2009.

[17] K. Hollingsworth, K. Bowyer, and P. Flynn. Similarity of iris texture between identical twins. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 22–29. IEEE, 2010.

[18] K. Hollingsworth, K. Bowyer, S. Lagree, S. Fenker, and P. Flynn. Genetically identical irises have texture similarity that is not detected by iris biometrics. *Computer Vision and Image Understanding*, pages 1493–1502, 2011.

[19] M. Hosseini, B. Araabi, and H. Soltanian-Zadeh. Pigment melanin: pattern for iris recognition. *IEEE Transactions on Instrumentation and Measurement*, 59(4):792–804, 2010.

[20] S. Kanade, D. Camara, E. Krichen, D. Petrovska-Delacrétaz, and B. Dorizzi. Three factor scheme for biometric-based cryptographic key regeneration using iris. In *Biometrics Symposium, 2008. BSYM'08*, pages 59–64. IEEE, 2008.

[21] S. Kanade, D. Petrovska-Delacrétaz, and B. Dorizzi. Cancelable iris biometrics and using error correcting codes to reduce variability in biometric data. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 120–127. IEEE, 2009.

[22] Y. Lee, K. Bae, S. Lee, K. Park, and J. Kim. Biometric key binding: Fuzzy vault based on iris images. In *2nd International Conference on Biometrics*, pages 800–808. Springer, 2007.

[23] I. Maghiros, Y. Punie, S. Delaitre, E. Lignos, C. Rodriguez, M. Ulbrich, and M. Cabrera. Biometrics at the frontiers: Assessing the impact on society. *Institute for Prospective Technological Studies, Technical Report EUR*, 21585, 2005.

[24] K. Miyazawa, K. Ito, T. Aoki, K. Kobayashi, and H. Nakajima. An effective approach for iris recognition using phase-based image matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(10):1741–1756, 2008.

[25] D. Monro, S. Rakshit, and D. Zhang. DCT-based iris recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(4):586–595, 2007.

[26] F. Monrose, M. Reiter, Q. Li, and S. Wetzel. Cryptographic key generation from voice. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 202–213. IEEE, 2001.

[27] K. Nandakumar, A. Jain, and S. Pankanti. Fingerprint-based fuzzy vault: Implementation and performance. *IEEE Transactions on Information Forensics and Security*, 2(4):744–757, 2007.

[28] M. Negin, T. Chmielewski Jr, M. Salganicoff, U. von Seelen, P. Venetainer, and G. Zhang. An iris biometric system for public and personal use. *Computer*, 33(2):70–75, 2000.

[29] S. Rakshit and D. Monro. Medical conditions: Effect on iris recognition. In *Multimedia Signal Processing, 2007. MMSP 2007. IEEE 9th Workshop on*, pages 357–360. IEEE, 2007.

[30] N. Ratha, S. Chikkerur, J. Connell, and R. Bolle. Generating cancelable fingerprint templates. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(4):561–572, April 2007.

[31] N. Ratha, J. Connell, R. Bolle, and S. Chikkerur. Cancelable biometrics: A case study in fingerprints. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 4, pages 370–373, 2006.

[32] O. Song, A. Teoh, and D. Ngo. Application-specific key release scheme from biometrics. *International Journal of Network Security*, 6(2):127–133, 2008.

[33] A. Teoh and C. Yuang. Cancelable biometrics realization with multispace random projections. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(5):1096–1106, 2007.

[34] J. Thornton, M. Savvides, and V. Kumar. A Bayesian approach to deformed pattern matching of iris images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(4):596–606, 2007.

[35] J. Thornton, M. Savvides, and B. Vijayakumar. Robust iris recognition using advanced correlation techniques. *Image Analysis and Recognition*, pages 1098–1105, 2005.

[36] P. Tome-Gonzalez, F. Alonso-Fernandez, and J. Ortega-Garcia. On the effects of time variability in iris recognition. In *IEEE International Conference on Biometrics: Theory, Applications and Systems, (BTAS)*, pages 1–6. IEEE, 2008.

[37] Z. Wang, Q. Han, and C. Busch. A novel iris location algorithm. *International Journal of Pattern Recognitionand Artificial Intelligence*, 23(1):59, 2009.

[38] R. Wildes. Iris recognition: An emerging biometric technology. *PIEEE*, 85(9):1348–1363, September 1997.

[39] R. Wildes, J. Asmuth, G. Green, S. Hsu, R. Kolczynski, J. Matey, and S. McBride. A machine-vision system for iris recognition. *Machine Vision and Applications*, 9(1):1–8, 1996.

[40] G. Williams. Iris recognition technology. *IEEE Aerospace and Electronic Systems Magazine*, 12(4):23–29, 1997.

[41] S. Ziauddin and M. Dailey. Robust iris verification for key management. *Pattern Recognition Letters*, 31(9):926–935, 2010.

[42] J. Zuo, N. Ratha, and J. Connell. Cancelable iris biometric. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

A Pattern-Based Architecture for Dynamically Adapting Business Processes

Mohamed Lamine Berkane[1] Lionel Seinturier[2] Mahmoud Boufaida[1]
[1] LIRE Laboratory
Mentouri University of Constantine, Algeria
{ml.berkane,mboufaida}@umc.edu.dz

[2]LIFL-INRIA ADAM
University of Lille, 59655 Villeneuve d'Ascq, France
Lionel.Seinturier@lifl.fr

Abstract— **The need to adapt a business process in applications has been a topic of interest in the recent years. Several approaches offer solutions to it. But, a limitation of most existing ones is the tight coupling of the adaptation logic with the execution one inside the engine implementation. In addition, they use the adaptation of business process only in the implementation phase (at runtime). To address these problems, we propose an architecture to develop a business process adaptation system. This architecture introduces modularity with an approach based on design patterns. We use some patterns to separate the adaptation logic and the functional one, and to address the adaptation at both the design phase and the implementation one. We show the feasibility of the proposed approach through the TRAP/BPEL framework.**

Keywords-Business Process; Design pattern; Abstraction Layers; Adaptation logic.

I. INTRODUCTION

Web services have evolved as a means to integrate processes and applications at an inter-enterprise level [17]. Several Web services can be combined to compose a new system. This last one can be seen as a composite Web service, which usually implements a business process.

A business process describes a sequence of tasks. Each task represents a coherent set of activities that fulfill a specific functionality. Tasks can be delegated to services and may require human interaction. Most business process languages assume that the tasks are executed in a static context. However, business process environments are often dynamic. For example, services can become unavailable, unexpected faults may occur or participating partners in the business process may not be known upfront, before some tasks are actually executed. In these situations, it is important to adapt a business process's behavior at run time in response to changing requirements and environmental conditions.

Recently, various approaches have proposed to support the dynamic business process adaptation: AO4BPEL (Aspect-Oriented for Business Process Execution Language) [1][2][16], VxBPEL [3], TRAP/BPEL (Transparent Reflective Aspect Programming/Business Process Execution Language) [4], CEVICHE (Complex EVent processIng for Context-adaptive processes in pervasive and Heterogeneous Environments) [5], MASC (Manageable and Adaptable Service Compositions) [6], DYNAMO (Dynamic Monitoring) [7], MVC (Model-View-Controller) [15]. However, most of these approaches do not treat changes at the design phase, and focus on run-time adaptation in terms of process instances. In addition to this, the current lack of reusable adaptation expertise can be leveraged from one adaptation system to another further exacerbates the problem.

In this paper, we present a pattern-based architecture for designing the adaptation system of business process. In our architecture, the system is designed in a modular way based on design patterns [8][9][10][12]. These patterns offer flexible solutions to common system development problems [12]. They express solutions of a known and recurrent problem in a particular context. Some of these patterns are used to specify the components of the adaptation systems. These components are: monitoring, decision-making, and reconfiguration [9][10]. Monitoring enables an adaptation system to aware the bussines process and detect conditions warranting reconfiguration, decision-making determines what set of monitored conditions should trigger a specific reconfiguration response, and reconfiguration enables an adaptation system to change the bussines process in order to fulfill the business requirements. Based on design patterns, our architecture supports the design of the adaptation system in four levels: the requirement layer, the functional layer, the logical layer and technical one. These abstraction layers are ordered hierarchically starting with (very abstract) high layers and leading to (very concrete) low layers. Each abstraction layer provides concepts for representation of the adaptation information, which is specific for each development phase. During the transition from a higher layer to a more concrete layer, the model information is enriched.

The rest of this paper is organized as follows: Section 2 presents some of the related work, Section 3 presents the proposed architecture and shows how it is realized using patterns; in Section 4, we use a case study to demonstrate the feasibility of our architecture through the TRAP/BPEL framework, and Section 5 concludes and discusses some future work.

II. RELATED WORK

This section overviews selected efforts conducted by researchers to facilitate the development of dynamically adapting business process system.

Adaptation and patterns: Ramirez and Cheng [8][9][10] presented several patterns for developing adaptive systems. These patterns are classified into three key elements of adaptive systems: monitoring, decision-making, and reconfiguration. The authors do not offer an approach to use these patterns. Beside, these patterns are generic; they can be used in adaptive systems as the multi-agent systems, network applications and information systems. In our case, we use some of these patterns to define aspects which are relevant for running the business process.

Gomaa et al. [14][19] proposed some patterns to specify the dynamic behavior of software architectures (master/slave, centralized, server/client, and decentralized architectures). These patterns are helpful to the developers implementing dynamically adaptation systems. Moreover, these approaches support only some kinds of software architectures, and the proposed patterns are specific to these architectures.

The GoF (Gang of Four) patterns [12] are the most popular and widely used in the designed system, and also are used in the abstract level. However, these patterns do not provide a solution to the adaptation problems. But, we use some of these patterns, to identify objects of the business process adaptation system at a high level of abstraction.

Adaptation and business process: Charfi et al. [16] presented a plug-in based architecture for self-adaptive processes that uses AO4BPEL [1]. Each plug-in has two types of Aspects: the monitoring Aspects that will check the system to observe when an adaptation is needed and the adaptation Aspects that will handle the situations detected by the monitoring Aspects. Yet, this approach supports only two kinds of components (Aspects) to adapt the business process. However, this approach defines the adaption logic at run time, while in our approach, the adaption logic is defined both at design-time and at runtime. In addition, our approach defines three components to separate the functional logic from the adaptive one. This makes our approach more modular.

Koning et al. [3] presented a language, called VxBPEL. They extended the BPEL language to add new elements like VariationPoint and Variant to capture variability in a service-based system. The first element specifies the places where the process can be adapted, and the second define the alternative steps of the process that can be used. This approach defines the adaptive logic both at design-time and at runtime. Yet this approach defines a new language to support the adaptation, and extends an existing engine to support VxBPEL language. In our case we use the standard BPEL, and we keep the known process engine.

The work which is closer to our proposal is the one presented in [15]. The authors present a framework based on the Model-View-Controller (MVC) pattern to support the adaptation of BPEL processes in a dynamic and modular way. In this framework, a workflow process is designed as a template, where the tasks can be specified in an abstract level. Concrete implementations of the tasks, modeled as aspects, are then selected from a library according to policy-based adaptation logic. However, this approach uses the pattern notion (MVC) to support the adaptation of business processes, while in our approach, we use the pattern (MDR) to develop the adaptation system of business process. This makes our approach more generic.

Hermosillo et al. [5] present CEVICHE, a framework that combines Complex Event Processing (CEP) and Aspect Oriented Programming (AOP) to support dynamically adaptable business processes. The adaptation logic is defined as aspects (reconfiguration component), and adaptation situations are specified by CEP rules (monitoring component). However, the decision- making is not specified as component in this framework. It is integrate into the defined aspects.

Xiao et al. [18] propose a constraint-based framework for supporting dynamic business process adaptation. In this framework, process adaptations are performed in a modular way based on process fragments. Process fragments are standalone fragments of processes that can be reused across multiple processes. This approach separates between the functional logic and the adaptive one by using the process fragments. However, this framework presents the adaptation only at run-time; in addition it cannot apply changes to living process instances. When new process schemas are (re)generated, only new process instances will be created according to the new process schemas.

III. A LAYER-BASED ARCHITECTURAL MODEL FOR BUSINESS PROCESS ADAPTATION

In this section, we present a pattern-based architecture that permits the design of adaptation systems in a dynamic and modular way. This architecture is composed of four layers: the requirement layer, the functional layer, the logical layer, and the technical one. Each layer contains three components (except requirement level): monitoring, decision-making, and reconfiguration. These three components will be refined in three layers. The starting point is the requirement layer which is a set of requirements for a behavior of the adaptation system. These requirements can have different forms, for example the form of a textual documentation or a collection of Use Cases. Secondly, the functional layer provides the definition of the adaptation system's interfaces with business process. Thirdly, the logical layer provides an architectural view of the system by partitioning it into logical communicating components. It defines the total behavior of the system. Lastly, the technical layer represents the lowest level of abstraction. It focuses on aspects relevant for running the business process. This architecture can provide an appropriate level of abstraction to describe dynamic change in a business process, such as the use of components, rather than at the algorithmic level (Figure 2). In the proposed architecture, we focus on the functional, logical and technical layers.

A. Requirement Layer

The proposed approach imposes a clear separation of concerns between functional and adaptation requirements. The adaptation requirements are concerned with understanding how a system may either make a transition between satisfying different functional requirements depending on context, or continue to satisfy the same functional requirements in the face of changing context. Hence, the adaptation requirements are intimately related to, and derived from, the functional requirements. These requirements can have different forms, for example the form of a textual documentation or a collection of Use Cases.

B. *Functional Layer*

In this layer, we provide the main functions of our adaptation system. It comprises the definition of its interfaces with business processes. Our adaptation system contains three main functions: monitoring, decision-making, and reconfiguration; and also two interfaces: to monitor and to reconfigure the business process. These functions can be seen as components, which communicate between them to adapt a business process's behavior in response to changing requirements and environmental conditions.

C. *Logical Layer*

In this layer, we refine the adaptation system presented in the previous sub-section. It defines three components specified by the functional level: monitoring, decision-making, and reconfiguration. In addition, there are two relationships: one between monitoring and decision-making, and the second between decision-making and reconfiguration, as shown in Figure 3. The logical components can be obtained as the combination of sub-components (objects) with respect to the dependencies between them. In this level, we use the GoF design patterns [12] to model the components and the sub-components. These patterns were chosen because they define the abstract concepts of adaptation (such as different strategies of adaptation defined by "Strategy" pattern).

The first component is the monitoring. The main objective of monitoring component is to enable an adaptation system to observe business process and environmental conditions that may warrant reconfiguration. To monitor the business process in the logical layer, we use the Observer pattern [12], which uses Observer and Subject objects. The Observer object collects the information about the business process and its environment, and the subject object is used to represent any component that needs to perform monitoring in business process. This last object defines the detection conditions to specify the conditions that may warrant reconfiguration (Figure 1). When a detection condition is detected, the subject objects notifies the observer objects, which in turn notifies the corresponding decision-making component.

The first relationship between monitoring component and decision-making one is defined to permit the interactions between the objects defined in the first component and the objects defined in the second one. We can have multiple interactions between the objects defined in these components. An object of the monitoring component can communicate with multiple objects of the decisions making component. Thus, an object of decision-making component may receive several messages from objects of the monitoring component. To carry the number of interaction between these two types of objects, it becomes necessary to define an intermediate object that manages these interactions. The 'Mediator' pattern [12] responds in a good way to this situation. By applying this pattern, the monitoring and the decision-making components can be evolving independently.

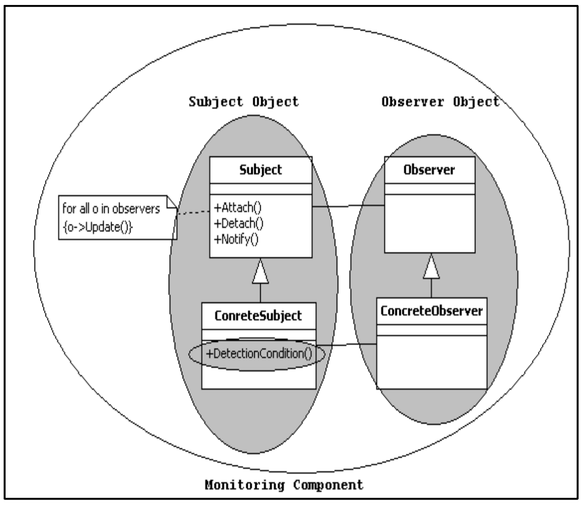


Figure 1. Monitoring Component in the logical layer

The second component (Decision-making) is the most important in the proposed architecture. The main objective of decision-making component is to determine when and how to reconfigure a business process in response to monitoring information. In this component, we define a family of algorithms. These one leverage a knowledge repository that associates specific monitoring scenarios with series of reconfiguration instructions. To define these algorithms, we use the 'Strategy' pattern [12]; this one creates a set of algorithms defined in the objects. Applying this pattern separates the functional logic from the decision-making one, thus clustering the set of reconfiguration responses for distinct events.

The second relationship between decision-making component and reconfiguration one is similar to the relationship between monitoring component and decision-making one, unless it manages the interactions between the objects of decision-making component and the objects of the reconfiguration one.

The last component (Reconfiguration) specifies in detail the actions defined in the algorithms of decision-making component. In this component, we use the 'Bridge' pattern [12] to separate between the algorithms (Abstraction) and the reconfiguration instructions (Implementor). In this pattern, we specify two kinds of objects: the Abstraction, and the Implementor. By applying this pattern, we can extend the Abstraction and the Implementor hierarchies independently.

D. *Technical layer*

This layer defines the aspects relevant for running the adaptation system. This layer also explains how the process adaptation is realized. Thereby, we use the design patterns defined for developing dynamically adaptation systems

[8][9][10]. These patterns were chosen because they use the platform-independent models to represent the adaptation solution.

A business process executes a series of activities in a sequence. It can be designed to show the sequence of service invocation. A Web Service is one of many types of services that a process can invoke. In general, the Web services are distributed application components that are externally available, and each Web service provides an interface that can be used to exchange the required information with business processes. In the monitoring component of this layer, we use the sensor factory pattern [8][9]; this pattern may be used when components (Web services) are distributed and each component (Web service) provides an interface that can be probed for the required information. In this pattern, a 'SimpleSensor' object can be used to sensor the component that needs to be monitored in business process. It replaces the Observer object of the 'Observer' pattern. In addition the methods 'Attach' and 'Detach' of 'ConcreteSubject' of Observer pattern were realized by 'SensorFactory', 'ResourceManager' and 'Registry' objects. The first object manages the addition and the removal of sensors in the business process, and the Clients interact with this object in order to gain access to a sensor. The second object determines if an existing sensor can be shared with one or more clients, and also, determines if the business process has enough resources to deploy a new sensor. The last object is responsible for tracking deployed sensors across the business process.

In the first relationship between monitoring component and decision-making one, we use the 'Content-based Routing' pattern [8][10]. This pattern should be applied when multiple clients require access to the same monitoring information. In our case, may be multiple monitoring components need access to the same decision-making component.

In the logical layer, we have used the strategy pattern to define a family of algorithms for decision-making components. In the technical layer, we use a 'Case-based Reasoning'pattern [8][10] to select the specific reconfigurations, and show how the reconfigurations can be executed at run time. The 'Case-based Reasoning' pattern can be applied when runtime scenarios that require reconfiguration can be reliably identified. The important objects of this pattern are: 'Trigger', 'Inference Engine', 'Decision', and 'Fixed Rules'. The 'Trigger' object contains relevant information about what caused the adaptation request. It should at least provide information about the error source, and the type of error observed. The 'Inference Engine' object is responsible for applying a set of 'Rules' to produce an action in the form of a 'Decision'. The 'Decision' object represents a reconfiguration plan that will yield the desired behavior in the system. The 'Fixed Rules' object contains a collection of 'Rules' that guide the 'Inference Engine' in producing a 'Decision'. These 'Fixed Rules' replace the strategy objects of the 'Strategy' pattern.

The second relationship between decision-making component and reconfiguration one is specified by the 'Divide and Conquer' pattern [8][10][20]. This pattern avoids potential business process inconsistencies, because the business process may require applying multiple reconfigurations in succession. The 'Divide and Conquer' pattern decomposes a complex reconfiguration into simpler reconfigurations, and it determines dependency relations between different reconfigurations. The 'Divide-and-Conquer' strategy is employed in many complex algorithms. With this strategy, a problem is solved by splitting it into a number of smaller sub-problems, solving them independently, and merging the sub-solutions into a solution for the whole problem. Conceptually, this pattern follows a straightforward approach. One task splits the problem, then forks new tasks to compute the sub-problems, waits until the sub-problems are computed, and then joins with the subtasks to merge the results.

The reconfiguration component uses two kinds of patterns 'Component Insertion' and 'Component Removal' [8][10]. In our case, we insert and remove the web service. For example, 'Component Insertion' pattern can be used to safely insert a new component (web service) at run time. The important objects of this pattern are: 'Adaptation Driver', 'Change Manager', 'Reconfiguration Plan', and 'Reconfiguration Rules'. The first object is responsible for ensuring that incoming Client requests are queued for further processing. The second object provides support for loading and unloading Components (web services) and their interconnections. The third object stores the specific sequence of instructions for reconfiguring the system at run time. This object replaces the 'Abstraction' object of the 'Bridge' pattern. The last object contains rules and instructions for specifying how basic reconfiguration operations are carried out in system. Some basic reconfiguration operations include Component insertion, removal, and swapping. This object replaces the 'Implementor' object of the 'Bridge' pattern.

IV. A Case Study:"TRAP/BPEL Framework"

In this section, a case study is used to demonstrate the feasibility of our approach. For this case, we have selected the TRAP/BPEL framework [4][13]. In our paper, we focus on an architectural approach not because the TRAP/BPEL framework is uninteresting or less promising, but we argue that the architectural level shows how the adaptation system components of this framework are separated and generality deals with the challenges posed. TRAP/BPEL is a framework that adds the autonomic behavior into existing BPEL processes. It aims to make an aggregate web service continue its function even after one or more of its constituent Web services have failed, and also adds the autonomic behavior to BPEL processes by using a generic proxy as an indirection layer to interact with the partner services. The generic proxy has a standard interface and works for all partner services of one or more adapt-ready BPEL processes. A recovery policy is used in the proxy to dictate the adaptation behavior for each monitored service [4]. This generic proxy can be reused for any BPEL processes. Therefore, it is possible to provide a common autonomic behavior to a set of services. Furthermore, an adapt-ready process monitors the behavior of Web service partners and tries to tolerate their failure by forwarding the failed request to its generic proxy, which in turn will find an equivalent service to substitute the failed one [4].

The main components of this framework are: adapt-ready process and generic proxy. The first one represents the monitoring component, and the second specifies in both the decision-making and the reconfiguration components (Figure 4 (a)).

In the monitoring component of the logical layer, we use the 'Observer' pattern to specify the adapt-ready process. This process (Observer object) monitors the behavior of Web service partners (Subject object). If one of the partner services fails (Detection condition) then the adapt-ready process forwards the failed request to its proxy. The proxy is generated specifically for this adapt-ready process and provides the same port types as those of the monitored Web services. This port is the mediator (Mediator object) between the adapt-ready process and the generic proxy. In addition, the generic proxy can provide behavior either common to all adapt-ready BPEL processes or specific to each monitored invocation using some high level policies. It may take one of the following actions according to the policy: invoke the service being recommended in the policy; find and invoke another service to substitute for the monitored service, or retry the invocation of the monitored service in the event of its failure. These three policies are considered as the different strategies (of 'Strategy' pattern) of decision-making component. The 'Bridge' pattern is responsible for the management of the policies (Abstraction object); it concretizes the actions defined in the policies (Implementor object). In this framework, there is not a mediator between decision-making component and reconfiguration one, because the proxy plays two roles at the same time (Figure 4 (b)).

In the technical layer, the TRAP/BPEL framework needs to incorporate some generic hooks (sensors of 'Sensor Factory' pattern) at sensitive joinpoints in the BPEL process (i.e., the invoke instructions). These joinpoints are points in the execution path of the program at which adaptation code can be introduced at run time. The operations and input/output variables of the proxy are the same as those of the monitored invocations. When more than one service is monitored within a BPEL process, the interface for the specific proxy is an aggregation of all the interfaces of the monitored Web services; this situation is specified by 'Content-based Routing' pattern. This last one defines an architecture (many-to-one) that gathers data from the different web services (one or more adapt-ready BPEL processes (multiple monitoring components)) and distributes it to the specific proxy (one decision-making component).

The proxy uses 'Case-based Reasoning' pattern to specify the behavior of decision-making component in the technical layer. This proxy (Inference Engine object) checks all the intercepted invocations (Trigger object) and tries to match these invocations with the specified policies (Fixed Rules object). If it finds a policy for that invocation, the proxy behaves accordingly to that, it selects one of three actions (Rule object); otherwise it follows its default behavior (Figure 4 (c)).

We use the 'Component Insertion' pattern to define the behavior of reconfiguration component. This pattern inserts a new web service (i.e. invoke a new web service). In the generic proxy, we cannot establish the relationship between decision-making component and reconfiguration one because it defines the behavior of two components (decision-making and reconfiguration) in one component.

Our Pattern-Based Architecture approach has several advantages over a framework-oriented approach (like TRAP/BPEL, AO4BPEL, CHEVICHE, etc) at developing dynamically adapting business processes. The design patterns provide general models that need to be instantiated and customized before they are implemented. Since models operate at a higher-level of abstraction than frameworks, they impose fewer initial constraints upon the system being developed. In addition, with our design pattern approach, developers select only those adaptation mechanisms their application will require. In contrast, adaptation-oriented frameworks provide infrastructure to perform the adaptation tasks for a wide range of applications; the overall infrastructure is needed for the adaptive application, even if not all the features are needed or used.

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented a pattern-based architecture for designing the adaptation system of business processes. The proposed architecture is composed of four layers: the requirement layer, the functional layer, the logical layer, and the technical one. Then, in each layer, we defined three components: monitoring, decision-making, and reconfiguration. Finally, in each component, and for each layer, we use patterns to facilitate the reuse of adaptation expertise. These patterns separate the adaptation logic from the functional one. This separation of concerns facilitates the reuse of adaptation designs across multiple adaptation systems.

In the future, we will try to propose a hybrid approach that allows the use of the various adaptation components (monitoring, decision-making, and reconfiguration) of the different adaptation approaches (like AO4BPEL, CEVICHE, etc).

REFERENCES

[1] A. Charfi and M. Mezini. "Aspect-oriented web service composition with AO4BPEL". In Proceedings of the 2nd European Conference on Web Services (ECOWS), volume 3250 of LNCS, pp. 168–182. Springer, September 2004.

[2] A. Charfi, B. Schmeling, A. Heizenreder, and M. Mezini. "Reliable, secure, and transacted web service compositions with ao4bpel". In Proceedings of the 4th IEEE European Conference on Web Services(ECOWS), December 2006.

[3] M. Koning, C.-a. Sun, M. Sinnema, and P. Avgeriou, "Vxbpel:Supporting variability for web services in bpel," Inf. Softw.Technol., vol. 51, no. 2, pp. 258–269, 2009.

[4] O. Ezenwoye, S. Sadjadi, "TRAP/BPEL: A Framework for Dynamic Adaptation of Composite Services", Tech. Rep. FIU-SCIS-2006-06-02, School of Computing and Information Sciences, Florida International University, 2006.

[5] G. Hermosillo, L. Seinturier, L. Duchien, "Using Complex Event Processing for Dynamic Business Process Adaptation" in Proceedings of the 7th IEEE 2010 International Conference on Services Computing (SCC 2010), Miami, Florida : United States, 2010.

[6] A. Erradi, V. Tosic, and P. Maheshwari. "Masc - .netbased middleware for adaptive composite web services". In ICWS International Conference on Web Services, pages 727–734. IEEE Computer Society, 2007.

[7] L. Baresi and S.Guinea. "Dynamo and self-healing bpel compositions". In ICSE COMPANION '07 : Companion to the proceedings of the 29th International Conference on Software Engineering, pages 69–70, Washington, DC, USA, 2007. IEEE Computer Society, 2007.

[8] A. J. Ramirez. Design patterns for developing dynamically adaptive systems. Master's thesis, Michigan State University, East Lansing, MI 48823, 2008.

[9] A. J. Ramirez and B. H. C. Cheng. "Developing and applying design patterns for dynamically adaptive systems". In 6th IEEE International Conference on Autonomic Computing, ICAC '09 Barcelona, Spain, 2009.

[10] A. J. Ramirez and B. H. C. Cheng. "Developing and applying design patterns for dynamically adaptive systems". In 5th International Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS'10, Cape Town, South Africa, May, 2010.

[11] A. Campetelli, M. Feilkas, M. Fritzsche, A. Harhurin, J. Hartmann, M. Hermannsdorfer, F. Holzl, S. Merenda, D. Ratiu, B. Schatz, and W. Schwitzer. "Model-based development – motivation and mission statement of workpackage zp-ap 1". Technical report, Technische Universität München, 2009.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley Professional, 1995.

[13] F. Baligand "Une Approche Déclarative pour la Gestion de la Qualité de Service dans les Compositions de Services ", Doctorate'thesis l'Ecole des Mines de Paris, 2008.

[14] H. Gomaa and M. Hussein. "Software reconfiguration patterns for dynamic evolution of software architectures". In WICSA'04: Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture, page 79, Washington, DC, USA, 2004.

[15] K. Geebelen, E. Kulikowski, E. Truyen and W. Joosen "A MVC Framework for Policy-Based Adaptation of Workflow Processes: A Case Study On Confidentiality" In 2010 IEEE International Conference on Web Services, 2010.

[16] A. Charfi, T. Dinkelaker, and M. Mezini, "A Plug-in Architecture for Self-Adaptive Web Service Compositions", in the Proceedings of IEEE International Conference on Web Services (ICWS'09), pp. 35- 42, 2009.

[17] M. Little, Transactions and web services, Communications of the ACM 46 (10) (2003) 49–54, 2003.

[18] Z. Xiao, D. Cao, C. You and H. Mei,"Towards a Constraint-based Framework for Dynamic Business Process Adaptation" In 2011 IEEE International Conference on Services Computing, 2011.

[19] H. Gomaa "Pattern-based Software Design and Adaptation". In PATTERNS 2011: Proceedings of the Third International Conferences on Pervasive Patterns and Applications, page 90-95, Roma, Italy, 2011.

[20] G. Mattson Timoth, A. Sanders Beverly, L. Massingill Berna. « Patterns for Parallel Programming ». Addison-Wesley Professional, 2004.

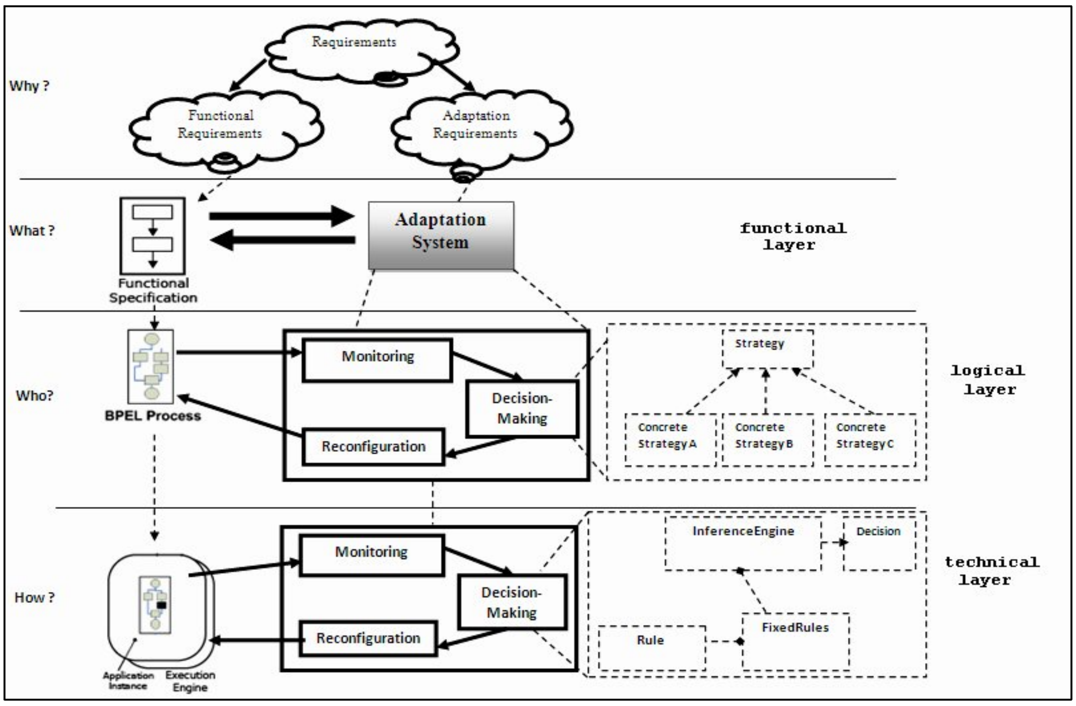


Figure 2. Overview of the proposed architecture

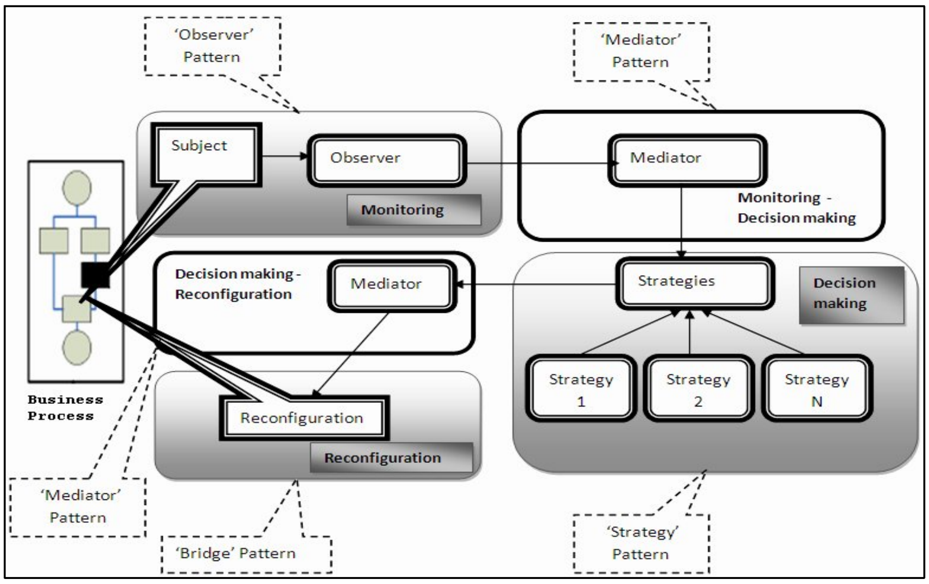


Figure 3. Representation of the logical layer

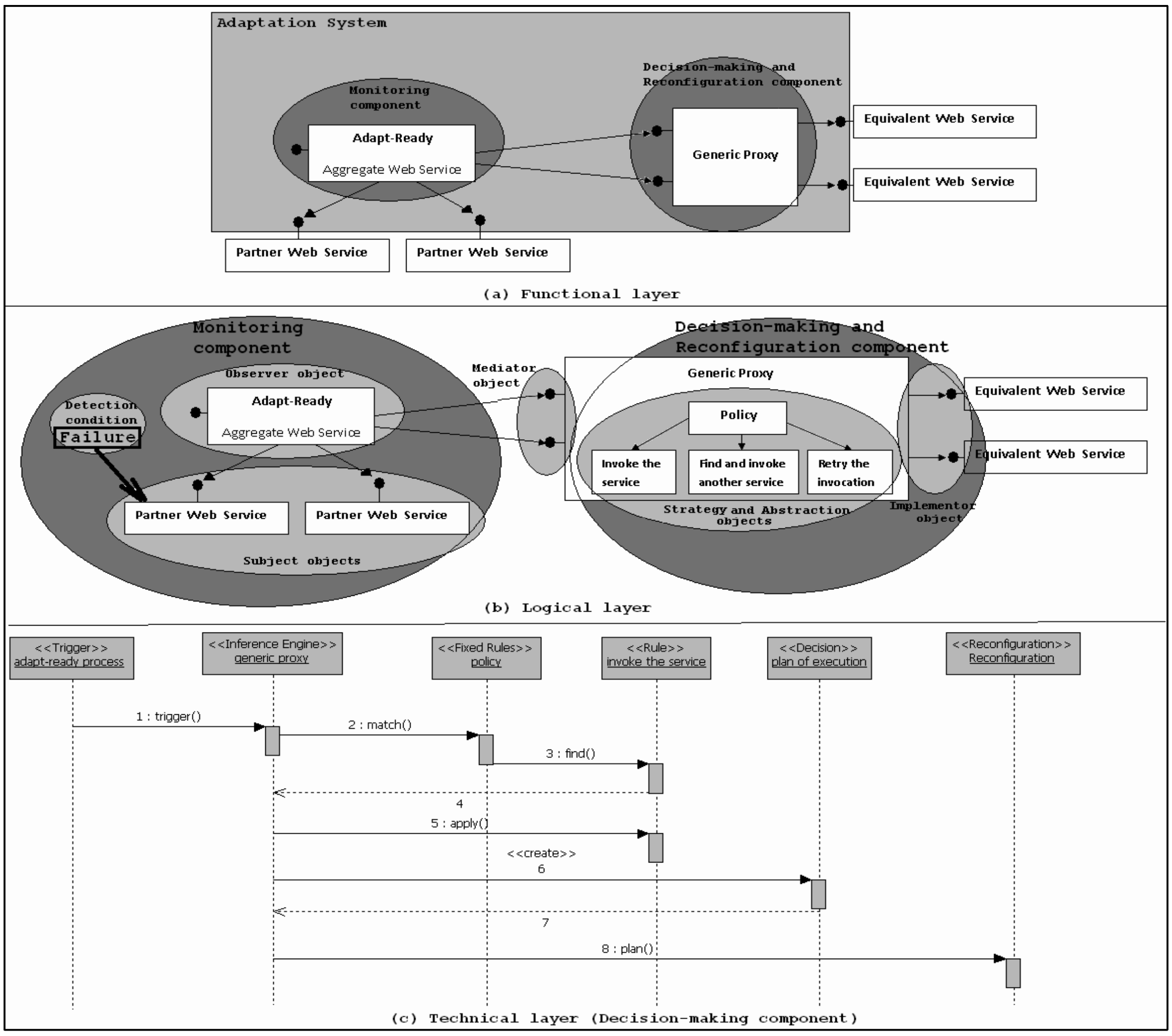


Figure 4. TRAP/BPEL framework in three layers

Exploiting HCI Pattern Collections for User Interface Generation

How well do Existing Pattern Definitions support Automated UI Construction?

Juergen Engel, Christian Herdin, Christian Maertin
Faculty of Computer Science
Augsburg University of Applied Sciences
An der Hochschule 1
86161 Augsburg, Germany
[Juergen.Engel | Christian.Herdin | Christian.Maertin]@hs-augsburg.de

***Abstract*—In the field of human-computer interaction various pattern languages and pattern collections have evolved during the last years. They include patterns for domain-independent and domain-specific graphical user interface design, interaction design, interactive application structure and navigation, design for usability, and design for user experience. However, the description and specification structure and attributes vary between existing pattern languages. In order to unify the various approaches and make them exploitable for automated graphical user interface construction, we have mapped the information structure of the different pattern collections to the pattern language markup language. By adding some additional attributes, the resulting pattern specifications can be used for facilitating and automating the development process for real world interactive applications, e.g., for knowledge sharing systems.**

Keywords – HCI patterns; pattern language; standardized pattern definition; pattern attributes; pattern relations; user interface generation.

I. INTRODUCTION

Human-computer interaction (HCI) patterns, pattern languages, and pattern collections recently have gained much influence in the field of model-based user interface development approaches (MBUID) [2]. HCI pattern languages and their various types of user interfaces related patterns provide abstract, semi-abstract and/or concrete modeling information for the (semi-)automatic construction of domain-specific, highly-usable, adaptable, and accessible user interfaces, e.g. for web-based and mobile applications. In order to simplify the selection of useful pattern types for a specific development project, ways for hierarchically structuring such languages and grouping their patterns by specific inter-pattern-relationships have been proposed [14]. Also, tools for visually defining patterns and their attributes, for browsing pattern hierarchies and exploiting the selected patterns for automated user interface generation purposes have been introduced [9].

Most available and popular pattern collections, however, as useful for the developer as they may be, still consist of rather unrelated pattern sets. Neither do such collections provide hierarchical or content-based relationships between similar or aggregated patterns, nor do pattern definitions provide attributes for using the structural, contextual and content information provided by a pattern for exploiting the pattern description for automated user interface generation.

Nevertheless, the pattern collections discussed in the following sections provide valuable information for structuring and designing the user interfaces and the navigation paths between the interaction objects of various types of interactive applications. Patterns from different collections should therefore be easily combinable within the target model of an interactive application. The patterns should be accessible from tool environments, browsers, and generators. In order to reach this rather practical goal, standardization efforts for reaching a common HCI pattern definition approach are necessary.

In the following section, several major standard HCI pattern collections are introduced and compared. Section III discusses the PLML approach for standardizing pattern definitions used in HCI and discusses several extensions to PLML. Section IV examines whether the PLML pattern description approach can be used for specifying real world interactive applications, e.g., for knowledge sharing systems. Finally, Section V presents concepts and specification extensions for HCI pattern definitions that can be used for automated user interface construction and generation.

II. ANALYSIS OF PREVALENT PATTERN COLLECTIONS

During the last decades, various HCI pattern collections have been developed, introduced and published providing valuable and reusable design know-how. Among others there are Jenifer Tidwell’s *Designing Interfaces, Patterns for Effective Interaction Design* [15], Martijn van Welie’s *Patterns in Interaction Design* [17], Douglas van Duyne’s *The Design of Sites, patterns for Creating Winning Web Sites* [6], Todd Coram’s and Jim Lee’s *A Pattern Language for User Interface Design* [3], *Yahoo! Design Pattern Library* of Yahoo! Inc. [18], or the community-driven UX and UI pattern library *Quince* operated by Infragistics [11].

Besides our own general and domain-specific pattern languages used for facilitating the design of knowledge sharing systems and portable mobile applications [8, 12], we predominantly have utilized patterns from Tidwell [15], van Welie [17], and van Duyne [6] in our previous work. Due to that fact we have analyzed and evaluated these three pattern collections in detail to assess their suitability to be used in automated and semi-automated user interface generation process steps respectively.

A. *Pattern Collection by Jenifer Tidwell*

The pattern collection by Jenifer Tidwell is one of the most extensive pattern libraries and consists of 125 patterns organized in 11 categories from “What users do” to “Make it look good”. Compared to the previous version the over-worked pattern collection described in the second edition of her book [15] is extended by three additional categories. Many patterns have been revised and several new patterns have been introduced. A digest is available on the Internet [16].

TABLE I. TIDWELL PATTERN COLLECTION ORGANIZATION

Tidwell pattern collection organization		
Category	***#***	***Patterns***
What users do	14	Safe exploration; Instant gratification; Satisficing; Changes in midstream; Deferred choices; Incremental construction; Habituation; Microbreaks; Spatial memory; Prospective memory; Streamlined repetition; Keyboard only; Other people’s advice; Personal recommendations
Organizing the content	10	Feature, search and browse; News stream; Picture manager; Dashboard; Canvas plus palette; Wizard; Settings editor; Alternative views; Many workspaces; Multi-level help
Getting around	13	Clear entry points; Menu page; Pyramid; Modal panel; Deep-linked state; Escape hatch; Fat menus; Sitemap footer; Sign-in tools; Sequence map; Breadcrumbs; Annotated scrollbar; Animated transition
Organizing the page	13	Visual framework; Center stage; Grid of equals; Titled sections; Module tabs; Accordion; Collapsible panels; Movable panels; Right/left alignment; Diagonal balance; Responsive disclosure; Responsive enabling; Liquid layout
Lists of things	12	Two-panel selector; One-window drilldown; List inlay; Thumbnail grid, Carousel; Row striping; Pagination; Jump to item; Alphabet scroller; Cascading lists; Tree table; New-item row
Doing things	11	Button groups; Hover tools; Action panel; Prominent “done” button; Smart menu items; Preview; Progress indicator; Cancelability; Multi-level undo; Command history; Macros
Showing complex data	11	Overview plus detail; Datatips; Data spotlight; Dynamic queries; Data brushing; Local zooming; Sortable table; Radial table; Multi-Y graph; Small multiples; Treemap
Getting input from users	11	Forgiving format; Structured format; Fill-in-the-blanks; Input hints; Input prompt; Password strength meter; Autocompletion; Dropdown chooser; List builder; Good defaults; Same-page error messages
Using social media	12	Editiorial mix; Personal voices; Repost and comment; Conversation starters; Inverted nano-pyramid; Timing strategy; Specialized streams; Social links; Sharing widget; News box; Content leaderboard; Recent chatter
Going mobile	11	Vertical stack; Filmstrip; Touch tools; Bottom navigation; Thumbnail-and-text list; Infinite list; Generous borders; Text clear button; Loading indicators; Richly connected apps; Streamlined branding
Make it look good	7	Deep background; Few hues, many values; Corner treatments; Borders that echo fonts; Hairlines; Contrasting font weights; Skins and themes

In the following, we consider the fully-fledged suite of patterns [15]. In Table I, an overview of the diverse pattern categories is given. In addition, it is shown how many and which patterns are assigned to the particular groups.

The actual pattern definitions consist of the eight attributes “Name”, “Figure”, “What”, “Use when”, “Why”, “How”, “Examples”, and “In other libraries”. Brief descriptions of these attributes are provided in Table II.

TABLE II. TIDWELL PATTERN DEFINITIONS

Tidwell pattern definitions	
Attribute	***Brief description***
Name	Name of the pattern
Figure	Meaningful example of pattern application, i.e. a screen shot
What	Brief description of the pattern
Use when	Information about the problem to be solved and the context of application
Why	Information about why one should use this particular pattern
How	How to use the particular pattern
Examples	Instances of application of the particular pattern
In other libraries	Links to other pattern collections where the selfsame pattern can be found

The Tidwell pattern collection draws a bow over a huge variety of possible fields of applications. The patterns are well-illustrated and described in detail. For all patterns, several meaningful examples of pattern instantiations are included. Descriptive category names support the user when browsing the catalog and help to locate individual patterns. Information about relations between the patterns are occasionally included in the “How” and “Why” attributes. References to the same or very similar patterns in other collections are provided via the “In other libraries” attribute.

B. *Pattern Collection by Martijn van Welie*

The pattern library of Martijn van Welie is available in the Internet [17]. It comes with 131 patterns grouped in the three main categories which are again divided into a total of 15 sub categories.

Table III shows the organization of the main category “User needs”, while Table IV illustrates “Application needs” and Table V “Context of design”.

TABLE III. MAIN CATEGORY “USER NEEDS”

Organization of pattern category “User needs”		
Category	***#***	***Patterns***
Navigating around	25	Accordion; Headerless menu; Breadcrumbs; Directory navigation; Doormat navigation; Double tab navigation; Faceted navigation; Fly-out menu; Home link; Icon menu; Main navigation; Map navigator; Meta navigation; Minesweeping; Panning navigator; Overlay menu; Repeated menu; Retractable menu; Scrolling menu; Shortcut box; Split navigation; Teaser menu; To-the-top link; Trail menu; Navigation tree
Basic interactions	7	Action button; Guided tour, Paging; Pulldown button; Slideshow; Stepping; Wizard
Searching	13	Advanced search; Autocomplete; Frequently asked questions (FAQ); Help wizard; Search box; Search area; Search results; Search tips; Site index; Site map; Footer sitemap; Tag cloud; Topic pages
Dealing with data	14	Carrousel; Table filter; Collapsible panels; Detail on demand; Collector; Inplace replacement; List builder; List entry view; Overview by detail; Parts selector; Tabs; Table sorter; Thumbnail; View
Personalizing	3	Customizable window; Login; Registration
Shopping	9	Booking; Product comparison; Product advisor; Product configurator; Purchase process; Shopping cart; Store locator; Testimonials; Virtual product display
Making choices	5	Country selector; Date selector; Language selector; Poll; Rating
Giving input	3	Comment box; Constraint input; Form
Miscelleaneous	5	Footer bar; Hotlist; News box; News ticker; Send-a-friend link

TABLE IV. MAIN CATEGORY “APPLICATION NEEDS”

Organization of pattern category “Application needs”		
Category	***#***	***Patterns***
Drawing attention	8	Captcha; Center stage; Color coded section; Premium content lock; Grid-based layout; Liquid layout; Outgoing links; Alternating row colors
Feedback	2	Input error message; Processing page
Simplifying interaction	2	Enlarged clickarea; Font enlarger

TABLE V. MAIN CATEGORY “CONTEXT OF DESIGN”

Organization of pattern category “Context of design”		
Category	***#***	***Patterns***
Site types	14	Web-based application; Artist site; Automotive site; Branded promotion site; Campaign site; E-commerce site; Community site; Corporate site; Multinational site; Museum site; Personalized ‘my’ site; News site; Portal site; Travel site
Experiences	8	Community building; Information management; Fun; Information seeking; Learning; Assistance; Shopping; Story telling
Page types	13	Article page; Blog page; Case study; Contact page; Event calendar; Forum; Guest book; Help page; Homepage; Newsletter; Printer-friendly page; Product page; Tutorial

The pattern definitions contain the nine attributes “Name”, “Problem”, “Solution”, “Use when”, “How”, “Why”, “More examples”, “Literature”, and “Comments”. Brief descriptions of these attributes are provided in Table VI.

TABLE VI. VAN WELIE PATTERN DEFINITIONS

van Welie pattern definitions	
Attribute	***Brief description***
Name	Name of the pattern
Problem	Decription of the problem to be solved
Solution	Description of the intended solution of the problem
Use when	Information of the context of application
How	Information about how to apply the pattern
Why	Information about why one should use this particular pattern
More examples	Instances of application of the particular pattern
Literature	Links to background literature
Comments	Comments from users of the pattern

In contrast to the renewed Tidwell collection the patterns, in the van Welie pattern library have not been revised during the last four years. According to an entry on the Internet page [17], the last change was performed in September 2007.

The two-tiered categorization supports the user in terms of searching and finding appropriate patterns. Collection-internal relationships between patterns are occasionally included in the attributes “Use when”, “How”, and “Why”. References to other pattern catalogues do not exist. A potentially valuable resource of information is the attribute “Comments”, which contains annotations that pattern users have left on van Welie’s website.

C. Pattern Collection of Douglas K. van Duyne

The pattern collection of Douglas van Duyne consists of 107 web-related patterns separated in 13 categories from "Site genres" to "The mobile web" [6]. All patterns are also published in the internet [7] but in a shortened and therefore less exhaustive manner.

Again, we consider the fully-fledged patterns library as published in [6]. Table VII provides an overview of how many and which patterns are available and how they are classified.

The pattern definitions consist of the seven attributes "Pattern ID", "Name", "Figure", "Background", "Problem", "Solution", and "Other patterns to consider". Brief explanations are provided in Table VIII.

The target audience of the pattern collection is the web designer community. This is already indicated by the category names. The actual pattern descriptions are extensive and detailed. Very positive is the treatment of collection-internal dependencies and relation between patterns which are explicitly listed and accentuated in terms of font color. References to external pattern collection are not included.

TABLE VII. VAN DUYNE PATTERN COLLECTION ORGANIZATION

van Duyne pattern collection organization		
Category	***#***	***Patterns***
Site genres	12	Personal e-commerce; News mosaics; Community conference; Self-service government; Nonprofits as networks of help; Grassroots information sites; Valuable company sites; Educational forums; Stimulating arts and entertainment; Web apps that work; Enabling intranets; Blogs
Creating a navigation framework	9	Multiple ways to navigate; Browsable content; Hierarchical organization; Task-based organization; Alphabetical organization; Chronological organization; Popularity-based organization; Category pages; Site accessibility
Creating a powerful homepage	2	Homepage portal; Up-front value proposition
Writing and managing content	11	Page templates; Content modules; Headlines and blurbs; Personalized content; Message boards; Writing for search engines; Inverted-pyramid writing style; Printable pages; Distinctive HTML titles; Internationalized and localized content; Style sheets
Building trust and credibility	9	Site branding; E-mail subscriptions; Fair information practices; Privacy policy; About us; Secure connections; E-mail notifications; Privacy preferences; Preventing phishing scams
Basic e-commerce	9	Quick-flow checkout; Clean product details; Shopping cart; Quick address selection; Quick shipping method selection; Payment method; Order summary; Order confirmation and thank-you; Easy returns
Advanced e-commerce	7	Featured products; Cross-selling and up-selling; Personalized recommendations; Recommendation community; Multiple destinations; Gift giving; Order tracking and history
Helping customers to complete tasks	13	Process funnel; Sign-in/new account; Guest account; Account management; Persistent customer sessions; Floating windows; Frequently asked questions; Context-sensitive help; Direct manipulation; Clear forms; Predictive input; Drill-down options; Progress bar
Designing effective page layouts	6	Grid layout; Above the fold; Clear first reads; Expanding screen width; Fixed screen width; Consistent sidebars of related content
Making site search fast and relevant	3	Search action module; Straightforward search forms; Organized search results
Making navigation easy	17	Unified browsing hierarchy; Navigation bar; Tab rows; Action buttons; High-visibility action buttons; Location bread crumbs; Embedded links; External links; Descriptive, longer link names; Obvious links; Familiar language; Preventing errors; Meaningful error messages; Page not found; Permalinks; Jump menus; Site map
Speeding up your site	6	Low number of files; Fast-loading images; Separate tables; HTML power; Reusable images; Fast-loading content
The mobile web	3	Mobile screen sizing; Mobile input controls; Location-based services

TABLE VIII. VAN DUYNE PATTERN DEFINITIONS

van Duyne pattern definitions	
Attribute	***Brief description***
Pattern ID	Unique identifier of the pattern
Name	Name of the pattern
Figure	Meaningful example of pattern application, i.e. a screen shot
Background	Information of the context of application
Problem	Decription of the problem to be solved
Solution	Description of the intended solution of the problem
Other patterns to consider	List of patterns which correlate with the particular pattern.

III. PLML: A STANDARDIZATION APPROACH FOR HCI PATTERN-DEFINITION

Like most of the available pattern collections, the three libraries described above lack a common organizational structure in order to facilitate appropriate and convenient pattern retrieval and selection. Manageability aspects of various existing UI pattern catalogues are discussed and compared in [3].

Another significant obstacle is that the authors describe their patterns in different and inconsistent styles. This makes it hard or even impossible to search, select and reference patterns across pattern collections. Hence, in a workshop [10] held within the scope of the CHI 2003 conference the attendees aimed for unification of pattern descriptions and guidance for the authors. As a result, the Pattern Language Markup Language (PLML) has been constituted and documented [10].

Version PLML v1.1 specifies that the documentation of a certain pattern should consist of the following elements: a pattern identifier, name, alias, illustration, descriptions of the respective problem, context and solution, forces, synopsis, diagram, evidence, confidence, literature, implementation, related patterns, pattern links, and management information [10]. Brief descriptions of these elements are provided in Table IX.

A. Extensions and changes to PLML

Over the years, PLML continued to develop and new extensions were added in order to adapt PLML to developing trends and needs of authors and users.

TABLE IX. PLML ATTRIBUTES

PLML pattern description elements	
Element	***Brief description***
Pattern ID	Unique pattern identifier
Name	Name of the pattern
Alias	Alternative names; also known as
Illustration	Good example of instantiation of the pattern
Problem	Description of the problem to be solved
Context	Situations in which the pattern can be applied
Forces	Description of forces in the environment that the use of the pattern will resolve
Solution	Description of how to resolve the problem
Synopsis	Summary of the pattern description
Diagram	Schematical visualization of the pattern
Evidence Example Rationale	Verification that it is in fact a pattern by at least three known uses of the pattern discussion and any principled reasons
Confidence	Rating of how likely the pattern provides an invariant solution for the given problem
Literature	References to related documents or papers
Implementation	Code fragments or details of technical realization
Related-patterns	Relationship to other patterns
Pattern-Link Type Pattern ID Collection ID Label	Catenation of patterns kind of relationship identifier of related pattern identifier of pattern collection name of the pattern link
Management Author Credits Creation-date Last-modified Revision-number	Authorship and change management name of the pattern author Merits date of pattern compilation date of last change version of the pattern definition

B. PLML version 1.2

An extension to PLML was suggested by Deng et al. [4]. New in this version 1.2 is an improved pattern- and attributes-description. This change allows for the attributes <chance-log> and <force> to be used by the pattern management tool MUIP.

C. PLMLx

The first published extension to PLML is PLMLx, the Extended Pattern Language Markup Language by Bienhaus [1]. The extension allows for better support of search functionality and classification of the patterns. The additional attributes that were added or modified are shown in Table X.

TABLE X. ADDITIONAL OR MODIFIED PATTERN DESCRIPTION ELEMENTS FROM THE PLMLX

Additional or modified PLMLx pattern description elements	
Element	***Brief description***
<organisation>	Meta information about the category, classification and the collection. A pattern can be part of different collections
<resulting context>	Describes the context in which we find ourselves when we applied the pattern. By applying a pattern it can create new problems that must be solved again.
<acknowledgments>	Acknowledgment for each who contributed significantly to the develeopment of the pattern or the techniques desription. If the pattern has been through a “shepherding process” or "writer's workshop", significant contibutors are candidates for the acknowledgment.
<management>	Added two new Subtaks copyright and license. The <last modified>-Tag from the origninal PLML<management>-Tag was deleted. The data from the last chance and the pertinent pattern name were integrated in the <revisions-number>-Tag
<example> and <rationale>	<example> and <rationale> are now separate tags. In the original PLML 1.1.2 they are Subtasks of the tag <evidence>.

The extensions and modifications in PLMLx make sense, because the <organization>-Tag helps to give a better overview of the different patterns from separate collections. However, an attribute that would be helpful for describing relationships and hierarchy between patterns or pattern catalogs is not included in PLMLx.

D. XPLML

Another extension to PLML is XPLML that is especially appropriate for human-computer interaction (HCI) design and for networking. XPLML describes the patterns and their relationship to each other with the help of Open Source standards such as XML, RDF and OWL [13].

The XPLML-Framework is a summary of specifications and tools to formalize HCI Patterns. The framework is intended to close the gap between the textual specification of patterns and their application in software. The conversion of an existing design pattern to XPLML does not work automatically.

The implementation of the XPLML framework is based on the following seven modules [13]:

- Unified HCI Pattern Form
- Semantic Metadata
- Semantic Relations among Patterns
- Atomic Particles of HCI Design Patterns
- Requirements engineering in HCI community
- Survey of HCI Design Pattern Management Tools
- Specification Documentation

XPLML is based on PLML, but adds components (e.g., Metadata) not in the HCI pattern form. The new components were added outside in a separate form. This is a good approach to define the relationships between patterns and collections.

The problem is that XPLML uses the older and not enhanced PLML backbone for pattern description. In addition, many pattern collections are not available in an appropriate specification and need to be translated separately for the use in XPLML. An automatic translation is not possible.

Furthermore, there neither exists a detailed description of the individual components of the framework, nor a release version until today.

IV. Comparison of Pattern Definitions

For all three discussed pattern collections the particular pattern definition attributes can be mapped almost entirely to corresponding PLML 1.1 elements. In some cases pattern attributes show 1:1 relationships to particular PLML elements, while not rarely exist equivocal correlations, i.e. 1:n, n:1, or n:m. We identified just one single pattern attribute within van Welie's pattern library that has no appropriate counterpart in PLML, i.e., the "Comments" attribute. Contrariwise four PLML elements are totally unconsidered, i.e., <alias>, <forces>, <confidence>, and <management>. Content that maps to the <implementation> PLML element is solely provided by van Duyne patterns.

Within the Tidwell pattern collection the "Use when" attribute provides information that partly maps to both, the <problem> and the <context> PLML elements. On the other hand the "What" and "How" attributes provide content that fits into <solution>. Data from the "In other libraries" attribute relates to the PLML element <literature>. However, not all Tidwell patterns feature this type of information. The <pattern-link> PLML element is completely disregarded.

Van Welie's patterns keep information within the "Solution" and "How" attributes that maps to the <solution> PLML element. Moreover the "How" attribute occasionally delivers figures or sketches that refer to <diagram> in PLML. Likewise the "Literature" attribute is not specified within every pattern definition. Information that maps to the <related-patterns> PLML element is provided by the "Use when", "How", and "Why" attributes albeit <pattern-link> data is missing.

Patterns within the van Duyne collection possess a <pattern ID> type property which consists of a capital letter indicating the particular pattern category and a consecutive number. The PLML <context> is split over the "Background" and "Problem" attributes. Notably, "Problem" is a very rich attribute whose content maps to various PLML elements, i.e., <problem>, <context>, <evidence>, <implementation>, and <related-patterns>. PLML <pattern-link> data is merely rudimentary available. Notably, the <type> specification of the pattern links are provided not at all.

The complete comparison of the pattern attributes of the discussed pattern collections and their mappings to PLML elements is consolidated in Table XI.

TABLE XI. PATTERN DEFINITION COMPARISON

Comparison of pattern attributes			
PLML 1.1	***Tidwell***	***van Welie***	***van Duyne***
<pattern ID>	-	-	pattern category + consecutive number
<name>	"Name"	"Name"	"Name"
<alias>	-	-	-
<illustration>	"Figure"	included in "Solution"	"Figure"
<problem>	included in "Use when"	"Problem"	"Problem"
<context>	included in "Use when"	"Use when"	partly in "Background" and partly in "Problem"
<forces>	-	-	-
<solution>	"What" and "How"	"Solution" and "How"	"Solution"
<synopsis>	-	-	-
<diagram>	-	occasionally in "How"	included in "Solution"
<evidence> <example> <rationale>	yes "Examples" "Why"	yes "More examples" "Why"	to some extent included in "Problem" -
<confidence>	-	-	-
<literature>	occasionally included "In other libraries"	occasionally available in "Literature"	-
<implementation>	-	-	occasionally included in "Problem"
<related-patterns>	occasionally included in "Why" and "How"	occasionally included in "Use when", "How", and "Why"	"Other patterns to consider", and contained in "Problem" and partly in "Background"
<pattern-link> <type> <pattern ID> <collection ID>	-	-	rudimentary "Pattern ID"

Comparison of pattern attributes			
PLML 1.1	***Tidwell***	***van Welie***	***van Duyne***
<label>			
<management>	-	-	-
-	-	"Comments"	-

V. Does PLML Meet Our Demands?

In this section, we would like to approach this question on the basis of an application in the area of knowledge sharing systems, which has been developed in the context of the project p.i.t.c.h. (pattern-based interactive tools for improved communication habits in knowledge transfers) [12]. It was conducted by the Automation in Usability Engineering group (AUE) at Augsburg University of Applied Sciences in cooperation with two medium-sized enterprises with engineering and production background and several partners from communication sciences and the knowledge management domain.

As an introductory example, Figure 1 shows a screenshot of the search function of the p.i.t.c.h. application. The red rectangles indicate the essential patterns which have been used to design the search capabilities.

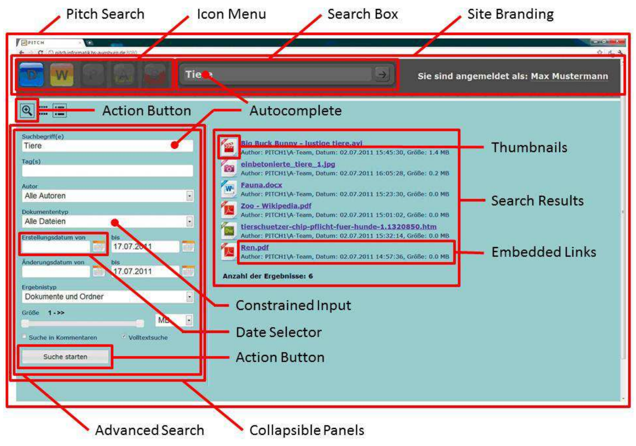


Figure 1. Screenshot of the p.i.t.c.h. Search Functionality.

Figure 2 gives an overview of the intrinsic pattern hierarchy of the p.i.t.c.h. search dialogue. Root element is the "p.i.t.c.h. Search" pattern which contains three further patterns, i.e., "Site Branding" [6], "Action Button" [17], and "Search Results" [17]. "Site Branding" in turn contains the two patterns "Icon Menu" [17] and "Search Box" [17], which finally relates to the "Autocomplete" pattern [17]. The other branches of the tree are constructed in analogous manner. The capital letters in the lower right corners of the boxes indicate from which pattern collection the particular pattern has been retrieved. The arrows represent the respective relationships between the patterns.

Such relationships are domain-independent and can be exploited for guiding structural and layout-related decisions during the automated pattern-based construction of the interactive application.

As yet, we have identified different relation types including "contains", "invokes", "employs", and "is-similar-to" relationships. Certainly there are respective inverse relations, i.e., "is-contained-by", "is-invoked-by", and "is-employed-by".

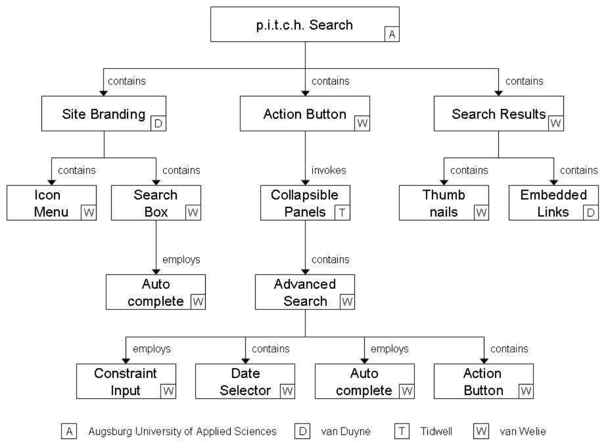


Figure 2. Excerpt of the Pattern Hierarchy of the p.i.t.c.h. Search Functionality.

A "contains" relationship implies that one pattern is integral part of another while "invoke", respectively "is-invoked-by" means that a pattern is activated owing to an event that happened within the correlated pattern. The "employs" relation means that the parent pattern avails itself of a functionality provided by the child pattern. Finally, "is-similar-to" predicates that the related patterns are either identical or can be utilized alternatively.

As outlined in Section I, we intend to use PLML-style pattern definitions that allow for (semi-)automated processing and UI construction purposes. Besides the fact that the "implementation" element is not yet adequately structured and specified, PLML shows various shortcomings concerning the given options to model the previously described pattern relationships.

Automated pattern processing requires for distinguishing two fundamental types of pattern links. On one hand, there are permanent links to other patterns, which can be regarded as "hard-coded" and generally, will not change for a long period of time, e.g., a "is-similar-to" relationship to the identical pattern within a different pattern collection. If a permanent link is to be changed it would lead to a new revision of the pattern. As soon as a respective parent pattern is applied, all child patterns referenced by permanent links are also applied automatically.

On the other hand, there is a need for defining temporary pattern relationships when the pattern is applied respectively the pattern hierarchy is constructed. For instance, the "Autocomplete" pattern occurs twice within Figure 2. In both cases, it is a matter of the selfsame pattern, but one time it has an "is-employed-by" relationship with the "Search

Box" pattern while the other time it indeed correlates in the same manner, but to a different pattern, i.e., "Advanced Search". In order to meet these requirements, we defined two child elements belonging to the PLML <related-patterns>, i.e., <permanent-links> and <temporary-links>.

In addition, PLML incorporates information about the pattern revision number within the <management> element. It is a flaw that the data is not used in the <pattern-link> element which allows for referencing other patterns, but not certain revisions of that particular pattern. This can be resolved by simply adding a child element to <pattern-link>, i.e., <revision>. Since an individual pattern may, of course, possess multiple relationships to other patterns, we propose to specify another child element <linkID>.

The described changes to PLML are illustrated in Figure 3. Please, note that the given <permanent-link> is not included within the graph in Figure 2.

```
…
<name>Search Box</name>
<problem>
The users need to find an item or specific
information.
</problem>
<solution>
Offer search functionality consisting of a search
label, a keyword field, a filter if applicable and
a "go" button. Pressing the return key has the
same function as selecting the "go" button.
</solution>
<related patterns>
  <permanent-links>
    <pattern-link>
      <linkID>W00037PL001</linkID>
      <revision>1.0</revision>
      <type>is similar to</type>
      <patternID>D00079</patternID>
      <collectionID>van Duyne</collectionID>
      <label>search action module</label>
    </pattern-link>
  </permanent-links>
  <temporary-links>
    <pattern-link>
      <linkID>W037TL001</linkID>
      <revision>1.3</revision>
      <type>employs</type>
      <patternID>W00034</patternID>
      <collectionID>van Welie</collectionID>
      <label>Autocomplete</label>
    </pattern-link>
  </temporary-links>
</related patterns>
…
```

Figure 3. Fragment of PLML-like Definition of Search Box Pattern.

As already described above, a pattern might occur more that one time in a pattern hierarchy. This means that this particular pattern is applied several times or, in other words, multiple instances of the pattern do exist. In this sense, PLML merely specifies the <patternID> element which is not suitable for distinguishing two or more instances of the same pattern. Therefore, we suggest to introduce a new element named <UID> with two child elements, i.e., <patternID> and <instanceID>. In addition, the <pattern-link> element has to be extended, respectively.

Figure 4 provides a pattern definition fragment that illustrates all proposed PLML changes required for better support of automatic pattern processing. This example specifies the "Advanced Search" pattern in the context of the pattern hierarchy displayed in Figure 2.

```
…
<UID>
  <patternID>W00033</patternID>
  <instanceID>0001</instanceID>
</UID>
<name>Advanced Search</name>
<illustration>
Please refer to Figure 1
</illustration>
<problem>
Users need to find a specific item in a large
collection of items.
</problem>
<solution>
Offer a special advanced search function with
extended term matching, scoping and output
options.
</solution>
<related-patterns>
  <permanent-links>
  </permanent-links>
  <temporary-links>
    <pattern-link>
      <linkID>W00033TL001</linkID>
      <type>employs</type>
      <patternID>W00078</patternID>
      <revision>1.0</revision>
      <instanceID>0001</instanceID>
      <collectionID>van Welie</collectionID>
      <label>Constraint Input</label>
    </pattern-link>
    <pattern-link>
      <linkID>W00033TL002</linkID>
      <type>contains</type>
      <patternID>W00073</patternID>
      <revision>1.1</revision>
      >instanceID>0001</instanceID>
      <collectionID>van Welie</collectionID>
      <label>Date Selector</label>
    </pattern-link>
    <pattern-link>
      <linkID>W00033TL003</linkID>
      <type>employs</type>
      <patternID>W00034</patternID>
      <revision>1.0</revision>
      <instanceID>0002</instanceID>
      <collectionID>van Welie</collectionID>
      <label>Autocomplete</label>
    </pattern-link>
    <pattern-link>
      <linkID>W00026TL004</linkID>
      <type>contains</type>
      <patternID>W00026</patternID>
      <revision>1.0</revision>
      <instanceID>0002</instanceID>
      <collectionID>van Welie</collectionID>
      <label>Action Button</label>
    </pattern-link>
  </temporary-links>
</related patterns>
…
```

Figure 4. Fragment of PLML-like Definition of Advanced Search Pattern.

VI. Conclusion

In this paper, we have analyzed three prevalent pattern collections in detail. Subsequently, we have discussed whether and how pattern definitions of these collections could be mapped into a PLML-compliant description format. We have identified some shortcomings of PLML in terms of suitability for automated pattern processing and we have proposed some changes and extensions in order to overcome these obstacles.

In our current research, we put our emphasis on the <implementation> element of PLML and target our efforts to enriching pattern definitions with code and model fragments and other valuable design resources.

References

[1] Bienhaus, D.: PLMLx Doc., available at http://www.cs.kent.ac.uk/people/staff/saf/patterns/plml.html Retrieved on February 3, 2012.

[2] Breiner, K. et al. (eds.): Proceedings of the 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems (PEICS ´10), June 20, 2010, Berlin, Germany, ACM Int. Conf. Proc. Series, 2010

[3] Coram, T. and Lee J.: "A Pattern Language for User Interface Design", available at http://www.maplefish.com/todd/papers/Experiences.html Retrieved on March 13, 2012

[4] Deng, J., Kemp E., and Todd E.G. (Hg.): Focusing on a standard pattern form: the development and evaluation of MUIP: Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: design centered HCI, 2006

[5] Deng, J., Kemp E., and Todd E.G.: Managing UI Pattern Collections. Proc. CHINZ ´05, Auckland, New Zealand, ACM Press, pp. 31-38, 2005.

[6] van Duyne, D., Landay J., and Hong J.: "The Design of Sites, Patterns for Creating Winning Websites", 2nd Edition, Prentice Hall International, ISBN 0-13-134555-9, 2006

[7] van Duyne, D., Landay J., and Hong J.: "The Design of Sites, Patterns for Creating Winning Websites", available at http://www.designofsites.com/home/ Retrieved on March 13, 2012

[8] Engel, J., Märtin C., and Forbrig P.: HCI Patterns as a Means to Transform Interactive User Interfaces to Diverse Contexts of Use, in: J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2011, LNCS 6761, Springer-Verlag Berlin Heidelberg pp. 204-213, 2011

[9] Engel, J., Märtin C., and Forbrig P.: Tool-support for Pattern-based Generation of User Interfaces, in [1], pp. 24-27, 2010

[10] Fincher, S. et al.: Perspectives on HCI Patterns: Concepts and Tools (Introducing PLML), CHI '03 Workshop Report, 2003

[11] Infragistics, Inc., "Quince", available at http://quince.infragistics.com Retrieved on March 13, 2012

[12] Kaelber, C. and Märtin C.: From Structural Analysis to Scenarios and Patterns for Knowledge Sharing Applications, in: J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2011, LNCS 6761, Springer-Verlag Berlin Heidelberg, pp. 258-267, 2011

[13] Kruschitz, C.: XPLML - A HCI Pattern Formalizing and Unifying Approach, Proceedings of CHI EA, 2009

[14] Märtin, C. and Roski A.: Structurally Supported Design of HCI Pattern Languages, Jacko, J. (Ed.) Human-Computer Interaction, Part I, HCII '07, Springer LNCS 4550 (2007), pp. 1159-1167, 2007

[15] Tidwell, J.: "Designing Interfaces, Patterns for Effective Interaction Design", 2nd Edition, O'Reilly Media Inc., ISBN 978-1-449-37970-4, 2011

[16] Tidwell, J.: "Designing Interfaces, Patterns for Effective Interaction Design", available at http://www.designinginterfaces.com Retrieved on March 13, 2012

[17] van Welie, M.: "Patterns in Interaction Design", available at http://www.welie.com Retrieved on March 13, 2012

[18] Yahoo! Inc.: "Yahoo! Design Pattern Library", available at http://developer.yahoo.com/ypatterns/ Retrieved on March 13, 2012

On A Type-2 Fuzzy Clustering Algorithm

Leehter Yao and Kuei-Sung Weng
Dept. of Electrical Engineering
National Taipei University of Technology
Taipei, Taiwan
e-mail: ltyao@ntut.edu.tw; gsweng@mail.nihs.tp.edu.tw

***Abstract*—A Type-2 fuzzy clustering algoritm that integreates Type-2 fuzzy sets with Gustafson-Kessel algorithm is proposed in this paper. The proposed Type-2 Gustafson-Kessel algorithm (T2GKA) is essentially a combination of probabilistic and possibilistic clustering schemes. It will be shown that the T2GKA is less susceptive to noise than the Type-1 GKA. The T2GKA ignores the inlier and outlier interruptions. The clustering results show the robustness of the proposed T2GKA since a reasonable amount of noise data does not affect its clustering performance. A drawback of the conventional GKA is that it can only find clusters of approximately equal volume. To overcome this difficulty, this work uses an algorithm called The Directed Evaluation Ellipsoid Cluster Volume (DEECV) to effectively evaluate the proper ellipsoid volume. The proposed T2GKA is essentially a DEECV based learning algorithm integrated with T2GKA. The experimental results show that the T2GKA can learn suitable sized cluster volume along with a varying dataset structure volume.**

Keywords-ellipsoids; probabilistic; possibilistic; fuzzy c-means; Gustafson-Kessel algorithm; Type-2 fuzzy clustering

I. INTRODUCTION

Clustering shows powerful capabilities to determine a finite number of clusters for partitioning a dataset. Hruschka et al. [1] proposed a survey of evolutionary algorithms for clustering, we can see the clustering area profile by focusing more on those topics that have received more importance in the literature. Based on the partition-based concepts, the fuzzy clustering algorithm can be classified into probabilistic fuzzy clustering and possibilistic fuzzy clustering. The fuzzy c-means (FCM) algorithm proposed by Bezdek [2] is a widely used and efficient clustering method for clustering and classification. Because FCM employs the Euclidean norm to measure dissimilarity, it inherently imposes a spheroid onto the clusters regardless of the actual data distribution. In [3] and [4], Gustafson and Kessel proposed the G-K algorithm (GKA) using an adaptive distance norm based on the cluster center and data point covariance matrices to measure dissimilarity. Because the distance norm employed in the GKA is in the Mahalanobis norm form, GKA can be considered as utilizing ellipsoids to cluster prototype data points. However, GKA assumes fixed ellipsoid volumes before iteratively calculating the cluster centers.

FCM and GKA are probabilistic fuzzy clustering approaches. In a noise environment, the probabilistic fuzzy clustering will force noise to belong to one or more clusters, therefore seriously influencing the main dataset structure. To relieve the probabilistic clustering drawbacks, Krishnapuram and Keller proposed a possibilistic fuzzy clustering called the Possibilistic c-means (PCM) [5-6]. The possibilistic fuzzy clustering can evaluate a datum to a cluster depending only on the distance of the datum to that cluster, but not on its distance to other clusters. The possibilistic fuzzy clustering can alleviate the noise influence, but it is very sensitive to initialization, sometimes generating coincident clusters.

To avoid the various FCM and PCM problems, Pal et al. proposed a new model called the possibilistic fuzzy c-means (PFCM) model [7]. The PFCM is a hybridization of the PCM and FCM models. The PFCM solves the noise sensitivity defect of FCM and overcomes the coincident clusters problem of PCM. However, the PFCM model has four parameters that must be learned. For an uncertain environment how to search for the best four parameters is difficult. All aforementioned fuzzy clustering methods have membership values called Type-1 membership values. In a real application domain, the prototype data may have many uncertain factors. Owing to the Type-1 fuzzy sets, their membership functions are crisp and they cannot directly model the uncertainties. On the other hand, the Type-2 membership functions are fuzzy, and they can appropriately model the uncertainties.

The Type-2 fuzzy set concept was introduced by Zadeh [8]. The advances of the Type-2 fuzzy sets and systems [9] are largely attributed to their three-dimensional membership function to handle more uncertainties in real application problems. Recent researches [10-13] have shown that the uncertainty in fuzzy systems can be captured with Type-2 fuzzy sets. In [14], the interval Type-2 fuzzy set was incorporated into the FCM to observe the effect of managing uncertainty from the two fuzzifiers. Type-2 fuzzy sets have been used to manage the uncertainties in various domains where the performance of Type-1 fuzzy sets is not satisfactory. For instance, [15-17] used the Type-2 fuzzy set for handling uncertainty in pattern recognition. Zarandi et al.

[18] presented a systematic Type-2 fuzzy expert system for diagnosing human brain tumors.

When clustering methods are combined with Type-2 fuzzy sets the prototype data can be clustered more properly and accurately. We extend the Type-1 membership values to Type-2 by assigning a possibilistic-membership function to each Type-1 membership value. The possibility theory, introduced by Zadeh [19] appears as a mathematical counterpart of probability theory that deals with uncertainty using fuzzy sets. The Type-2 membership values are obtained by taking the difference between each Type-2 membership function area with the corresponding Type-1 membership value. In this paper we use the unbounded normal distributions Gaussian function as the secondary membership function [20-21].

Using the aforementioned cencepts we combined probabilistic and possibilistic methods to build Type-2 fuzzy sets. We present a Type-2 GKA (T2GKA) that is an extension of the conventional GKA. The membership values for each prototype datum are extended as Type-2 fuzzy memberships by assigning a membership grade to the Type-1 memberships. The higher the membership value for a prototype datum, the larger the prototype datum contribution possesses in determining the cluster center location. The experimental results show that the T2GKA was less susceptible to noise than the Type-1 GKA.

To overcome the T2GKA's inability to determine appropriate ellipsoid size, a Directed Evaluation Ellipsoid Cluster Volume (DEECV) scheme is proposed in this paper, so that the proper cluster volume can be directly evaluated instead of each cluster using equal cluster volume in the clustering learning. The Mahalanobis norm inducing matrix determinant is utilized in this paper to measure the ellipsoid size [22, 23]. The DEECV is developed to intelligently estimate the proper ellipsoid size value. With the proper ellipsoid size value determined by the proposed DEECV, the learning efficiency can be further improved. The proposed T2GKA is essentially a DEECV based learning algorithm integrated with T2GKA.

II. COMBINED PROBABILISTIC AND POSSIBILISTIC TO BUILD TYPE-2 FUZZY SET

We focus on providing a Type-2 fuzzy set model to avoid uncertain outliers affecting the clustering learning results. We explain how to build the Type-2 fuzzy sets based on the following concept. For every prototype data point, the ordered set of memberships to each of the clusters $\{\mu_1,\ldots,\mu_c\}$ spans a c-dimensional space. Sets of specific membership values in this space are represented as points. The possibility distribution transform of the Type-1 probability distribution on unbounded normal distributions Gaussian function around the Type-1 membership value. For each given point, the possibilistic type membership value indicates the strength of the attribution to any cluster independent from the rest. Figure 1 shows that two points x_1 and x_2 have the same Type-1 membership value but have different possibility values.

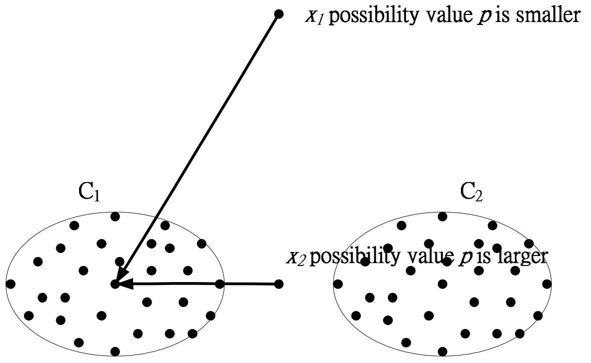

x_1 possibility value p is smaller
C1
C2
x_2 possibility value p is larger

Figure 1. The points have the same membership value but have different possibility values

The idea in building Type-2 fuzzy sets is based simply on the fact that, for the same Type-1 membership value, the secondary membership function should make the larger possibility value more than the smaller possibility value. The secondary membership function based on the competitive learning theory proposed here originates from the rival-penalized competitive learning (RPCL) in [24]. The basic idea of RPCL is that, for each input, the winner unit is modified to adapt to the input and its rival is delearned using a smaller learning rate, so, RPCL rewards the winner and punishes the rival. A Type-2 fuzzy set is defined as an object $\widetilde{A}$ which has the following form:

$$\widetilde{A} \equiv \left\{ \langle u, t, \xi_A(\bullet) \rangle \right\}, \tag{1}$$

where $\xi_A(\bullet)$ is an unbounded normal distributions Gaussian function representing the secondary membership function of the element $(u,t), u \in \boldsymbol{U}, \xi_A(\bullet) \in [0,1]$ in $\widetilde{A}$. We set the Type-1 membership value and Type-2 membership value relation as following equations:

$$u = u \times \max(\xi_A(\bullet)), \tag{2}$$

$$t = u \times \xi_A(\bullet), \tag{3}$$

where u represents the primary membership value and t represents the Type-2 membership value. The $\xi_A(\bullet)$ is an unbounded normal distribution Gaussian function representing the secondary membership function:

$$\xi_A(\bullet) = \exp -\frac{1}{2}\left(\frac{a-b}{\sigma}\right)^2. \tag{4}$$

Under the aforementioned concepts, reducing the Type-2 fuzzy sets involves complicated operations. We use the input/output data points $\mathbf{x_k}$, $k = 1 \ldots N$, set as the possibility value, p_{ik} as the unbounded normal distribution Gaussian function standard deviation, σ and the $(p_{ik} - 1)$ denotes the distance between p_{ik} to the central unbounded normal distribution Gaussian function, then design the secondary membership function $e^{-0.5\times(\frac{p_{ik}-1}{p_{ik}})^2}$.

The confidence intervals for varying possibilistic values p_{ik} built around the same prototype datum x_{ik} with membership value μ_{ik} are nested. A unimodal numerical possibility distribution may also be viewed as a nested set of confidence intervals. The unbounded normal distribution Gaussian function’s confidence intervals are 2σ and have a 95% confidence level. For example, the Type-1 membership value $\mu = 0.5$, has a secondary membership function with different possibility values as shown in Fig. 2.

The Type-2 membership values can be obtained using the following equation;

$$t_{ik} = \mu_{ik} \times \xi_A \Rightarrow t_{ik} = \mu_{ik} \times e^{-\frac{1}{2}\times\left(\frac{p_{ik}-1}{p_{ik}}\right)^2}, \qquad (5)$$

where t_{ik} (μ_{ik}) denotes the Type-2(1) memberships, p_{ik} denotes the membership degrees for one datum resembling the possibility of its being a member of the corresponding cluster. For example, for the Type-1 membership value $\mu = 0.5$, the following evaluations process interprets that Type-2 fuzzy sets evaluate their secondary membership values with different possibility values. The prototype data points $\boldsymbol{x_k}$, $k = 1,\ldots,N$, have Type-1 membership value $\mu_{ik} = 0.5$ and possibility value $p_{ik} = 1.0$ then the Type-2 membership values $t_{ik} = 0.5$ are obtained using (5). For the same Type-1 membership value $\mu_{ik} = 0.5$, and possibility value $p_{ik} = 0.1$ we obtain the Type-2 membership values as $t_{ik} = 1.2884e-018 \cong 0$.

We know that in our design the secondary membership function, for the same Type-1 membership value, a larger possibility value can make the Type-1 membership value larger than the smaller possibility value does. Using the aforementioned concepts, we combined the probability and possibility membership values and propose the Type-2 Gustafson-Kessel Algorithm (T2GKA).

III. THE TYPE-2 G-K ALGORITHM (T2GKA):

To overcome the drawback of the GK algorithm, it is used to find only clusters of approximately equal volumes. In this paper an algorithm called The Directed Evaluation Ellipsoid Cluster Volume (DEECV) is proposed to effectively evaluate the proper ellipsoid volume. The proposed T2GKA is essentially a DEECV based learning algorithm integrated with T2GKA.

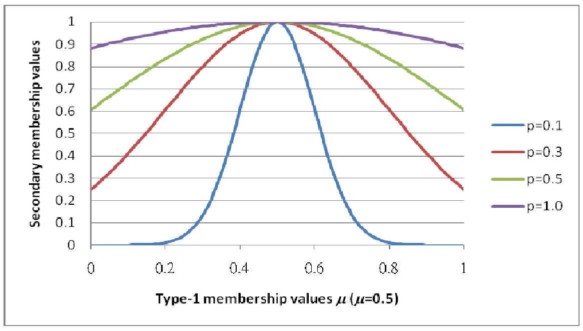


Figure 2. The secondary membership function with the different possibility values

A. *The Type-2 G-K Algorithm (T2GKA):*

Based on the prototype data points $\boldsymbol{x_k}$, $k = 1,\ldots,N$, given the random initial Type-1 fuzzy partition matrix $\boldsymbol{U}^{(0)} = \boldsymbol{T}^{(0)}$, T2GKA is to learn the Type-2 fuzzy partition matrix $\boldsymbol{T}$, the coordinates of all cluster centers $\boldsymbol{V}$ and the norm inducing matrix $\boldsymbol{A_i}$ by minimizing , $i = 1,\ldots, c$

$$J_{T2GKA}(\boldsymbol{T},\boldsymbol{V},\boldsymbol{A}) = \sum_{i=1}^{c}\sum_{k=1}^{N}(t_{ik})^m \boldsymbol{D}^2_{\boldsymbol{ikA_i}} + \sum_{i=1}^{c}\omega_i(|\boldsymbol{A}_i| - \rho_i) + \sum_{k=1}^{N}\gamma_k(\sum_{i=1}^{c} t_{ik} - 1), \qquad (6)$$

where t_{ik} has the same meaning of membership and constraints as FCM. The distance between the *k-th* prototype data point and the *i-th* cluster center is defined as the Mahalanobis norm:

$$\boldsymbol{D}_{\boldsymbol{ikA_i}} = ((\boldsymbol{x_k} - \boldsymbol{v}_i)^T \boldsymbol{A}_i (\boldsymbol{x_k} - \boldsymbol{v}_i))^{1/2}. \qquad (7)$$

For the *i-th* cluster, the ellipsoid $\phi_i(\cdot)$ is defined as

$$\phi_i(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{v}_i)^T \boldsymbol{A}_i (\boldsymbol{x} - \boldsymbol{v}_i) = 1,\ i = 1,\ldots, c. \qquad (8)$$

Since the volume of $\phi_i(\cdot)$ is inversely proportional to the determinant of $\boldsymbol{A_i}$, det($\boldsymbol{A_i}$) is thus utilized as a measure of the ellipsoid volume for T2GKA. If the determinant of $\boldsymbol{A_i}$ is given as ρ_i, $\boldsymbol{A_i}$ is constrained by

$$\det(\boldsymbol{A_i}) = \rho_i,\ \ \rho_i > 0,\ i = 1,\ldots, c. \qquad (9)$$

The optimization in $(\boldsymbol{T},\boldsymbol{V},\boldsymbol{A})$ can be solved using differentiations as follows:

$$\boldsymbol{A}_i = \left[\rho_i \det(\boldsymbol{F}_i)\right]^{1/n} \boldsymbol{F}_i^{-1} \quad i = 1,\ldots,c, \qquad (10)$$

$$\boldsymbol{F}_\mathrm{i} = \frac{\sum_{\mathrm{k}=1}^{\mathrm{N}} (t_{ik})^\mathrm{m} (x_k - v_i)(x_k - v_i)^\mathrm{T}}{\sum_{\mathrm{k}=1}^{\mathrm{N}} (t_{ik})^\mathrm{m}}. \qquad (11)$$

To avoid the covariance matrix being singular in the iterative process, a scaled identity matrix is added to the covariance matrix, i.e.,

$$\boldsymbol{F}_i = (1-\kappa)\boldsymbol{F}_i + \kappa \det(\boldsymbol{F_0})^{1/n} \boldsymbol{I}, \qquad (12)$$

where $\kappa \in [0,1]$ is a tuning factor with a small value and $\boldsymbol{F_0}$ is the whole data set covariance matrix with fixed value. The coordinate of each cluster center as well as the membership element in the partition matrix can be updated using the following equations:

$$\boldsymbol{v_i} = \frac{\sum_{k=1}^{N} (t_{ik})^m \boldsymbol{x_k}}{\sum_{k=1}^{N} (t_{ik})^m}, \qquad (13)$$

$$t_{ik} = \left(\sum_{j=1}^{c}\left(\frac{\boldsymbol{D}_{\boldsymbol{ikA_i}}}{\boldsymbol{D}_{\boldsymbol{jkA_i}}}\right)^{2/(m-1)}\right)^{-1}, 1 \le i \le c; 1 \le k \le N. \qquad (14)$$

For each given point, the possibilistic type membership value, indicating the strength of the attribution to any cluster, is independent from the rest. We calculate the possibilistic type membership value simultaneously using

$$p_{ik} = \frac{1}{1+\left(\frac{\boldsymbol{D}^2_{ikA_i}}{\varepsilon_i}\right)^{\frac{1}{(m-1)}}}. \tag{15}$$

We determine the reasonable number of ε_i by computing

$$\varepsilon_i = K\frac{\sum_{k=1}^{N} t_{ik}^m \boldsymbol{D}^2_{ikA_i}}{\sum_{k=1}^{N} t_{ik}^m}, \tag{16}$$

usually K=1 is chosen. For each given point, using the possibilistic type membership value, the Type-2 membership values can be updated using equation (5).

B. *The Directed Evaluation Ellipsoid Cluster Volume (DEECV)*

Without knowing the prototype data point distribution range a priori, a tentative value ρ_a is first assigned to every parameter, ρ_i, *i = 1,…, c*. With $\rho_i = \rho_a$, *i = 1,…, c,* T2GKA is applied to calculate the tentative ellipsoid $\hat{\phi}_i$ with center $\hat{\boldsymbol{v}}_i$, the covariance matrix $\hat{\boldsymbol{F}}_i$, and the norm inducing matrix $\hat{\boldsymbol{A}}_i$, *i = 1,…, c*. Denote B_i as the set of data points belonging to the cluster corresponding to $\hat{\phi}_i$ and $\boldsymbol{x}_j^i$ as the *j-th* data point belonging to B_i. Let $\hat{\boldsymbol{x}}^i$ be the data point with the largest Mahalanobis distance $\hat{L}_i$ among all data points in B_i, i.e.

$$\hat{\boldsymbol{x}}^i = \underset{\boldsymbol{x}_j^i \in B_i}{Argmax}(\|\boldsymbol{x}_j^i - \hat{\boldsymbol{v}}_i\|_{\hat{A}_i}) \tag{17}$$

and $$\hat{L}_i = \underset{\boldsymbol{x}_j^i \in B_i}{max}(\|\boldsymbol{x}_j^i - \hat{\boldsymbol{v}}_i\|_{\hat{A}_i}), \tag{18}$$

where $\|\bullet\|_{\hat{A}_i}$ denotes the Mahalanobis norm with the norm inducing matrix $\hat{\boldsymbol{A}}_i$ as in (7). According to (7) and (10),

$$(\hat{\boldsymbol{x}}^i - \hat{\boldsymbol{v}}_i)^T(\rho_a \det(\hat{\boldsymbol{F}}_i))^{1/n}\hat{\boldsymbol{F}}_i^{-1}(\hat{\boldsymbol{x}}^i - \hat{\boldsymbol{v}}_i) = \hat{L}_i. \tag{19}$$

It is thus obvious that if the initialization process appropriately adjusts the initial ellipsoid volumes so that the farthest data point $\hat{\boldsymbol{x}}^i$ with the largest Mahalanobis norm is right on the initialized ellipsoid, all of the ellipsoid volumes will be scaled to the range of solutions. As shown in (8), the data points on the ellipsoids have a Mahalanobis distance of 1. Divide $\hat{L}_i$ at both sides of (19),

$$(\hat{\boldsymbol{x}}^i - \hat{\boldsymbol{v}}_i)^T(\frac{\rho_a}{\hat{L}_i^n}\det(\hat{\boldsymbol{F}}_i))^{1/n}\hat{\boldsymbol{F}}_i^{-1}(\hat{\boldsymbol{x}}^i - \hat{\boldsymbol{v}}_i) = 1. \tag{20}$$

Therefore, the appropriate initial volume for the *i-th* ellipsoid leading to the result that all data points are included by the ellipsoid with tentative value ρ_a can thus be defined as:

$$\rho_{i\_initial} = \frac{\rho_a}{\hat{L}_i^n}, \text{ i = 1,…, c.} \tag{21}$$

It is worth noting that if $\hat{x}^i$ is an outlier for the cluster corresponding to $\hat{\phi}_i$, $\hat{L}_i$ will be unreasonably large. This results in an inaccurate initial ellipsoid volume ρ_i according to (21). For the data points with too much noise, an outlier detection scheme is required to determine the outliers and filter them out before applying the directed initialization. Let $\bar{d}_i$ be the average Mahalanobis distance among all data points belonging to B_i, then

$$\bar{d}_i = \frac{\sum_{j=1}^{|B_i|}\|\boldsymbol{x}_j^i - \hat{\boldsymbol{v}}_i\|_{\hat{A}_i}}{|B_i|}, \tag{22}$$

where $|B_i|$ denotes the number of data points in B_i. For all data points in B_i, the farthest data point and its maximum Mahalanobis distance can be respectively determined using (17) and (18). Removing the outliers affects the clustering learning results. With a predetermined threshold γ, any data point $\boldsymbol{x}^i$ belonging to the *i-th* cluster and its possibility membership value P_{ik} is larger than a predetermined threshold possibility membership value $P_{ik} \geq \alpha$ (in this paper, we set $\alpha = 0.1$), satisfies the following criterion:

$$\frac{\|\boldsymbol{x}^i - \hat{\boldsymbol{v}}_i\|_{\hat{A}_i}}{\bar{d}_i} \geq \gamma \tag{23}$$

is considered as an outlier and can be removed from B_i. The outlier detection scheme, as shown in (22) and (23), is recursively applied to every cluster of data points until no outlier has been detected. After filtering out the outliers in every cluster, the accuracy of calculating proper ellipsoid volume according to (21) for T2GKA's directed evaluation can be greatly improved.

IV. COMPUTER SIMULATIONS

We used the following computational conditions for all datasets: 1. The termination tolerance $\varepsilon = 0.000001$, the $\boldsymbol{D}_{ikA_i}$ for the FCM, FCMPCM, and PFCM is the Euclidean norm. 2. The $\boldsymbol{D}_{ikA_i}$ for the GK and T2GKA is the Mahalanobis norm. 3. The number of *c* clusters *c* is 7 for 7cluster. 4. *c* is 5 for 5 same-circle and sinusoidal sets. 5. *c* is *2* for all other datasets.

Example 1: The artificial 2-dimensional datasets X_{400} and X_{550} are designed. The X_{400} is a mixture of two 2-variate normal distributions with mean vectors $\begin{pmatrix}5.0\\6.0\end{pmatrix}$ and $\begin{pmatrix}5.0\\12.0\end{pmatrix}$. Each cluster has 200 points, while X_{550} is an augmented version of X400 with an additional 150 points uniformly distributed over $[0,15]\times[0,11]$. For data set X_{400} the clustering results in Table I show that the terminal centroids learned by all five algorithms produce good centroids.

When we cluster dataset X_{550}, we hope that the 150 noise points can be ignored and the cluster center will be found closer to the true centroids $\boldsymbol{V}_{true}$. From Table I, we can see that all five algorithms clustered the dataset X_{550} terminal centroids. Because PCM is very sensitive to initialization and it sometimes generates coincident clusters, we utilized the FCM clustering results to initialize PCM. The other four clustering methods ran the algorithm directly. To make a rough assessment of how each method accounted for inliers and outliers, we estimated $E_A = \|\boldsymbol{V}_{true} - \boldsymbol{V}_A\|^2$, where *A* denotes FCM, FCMPCM, PFCM, GK, and T2GKA. The E_{FCM}=0.4173, E_{FCMPCM}=0.0001, E_{PFCM}=0.3714 (*a*=1, *b*=0.1, *m*=2, η=2), E_{PFCM}=0.1699 (*a*=1, *b*=1, *m*=2, η=2), E_{GKA}=0.4825, and E_{T2GKA}=0.0066. The T2GKA clustering results with the proper cluster volumes for the datasets X_{550} are shown in Fig. 3. We compared the five clustering method's E_A values, the E_{FCMPCM} value is smaller than that in other methods, but its membership values are independent of the other clusters. We cannot depend on the membership values to classify the data points belonging to which cluster. Except for the E_{FCMPCM}, the E_{T2GKA} value is smaller than that in other methods. The clustering results show the robustness of the proposed T2GKA because a reasonable amount of noise data does not affect its clustering performance.

Example 2: To verify that the proposed method can accord the prototype dataset structure to learn the proper cluster centers, 5 same-circles were designed with each cluster containing 300 prototype data points. The dataset 5 same-circle is a mixture of two 2-variate normal distributions with mean vectors $\begin{pmatrix}0.0\\3.0\end{pmatrix}, \begin{pmatrix}5.0\\3.0\end{pmatrix}, \begin{pmatrix}0.0\\-3.0\end{pmatrix}, \begin{pmatrix}5.0\\-3.0\end{pmatrix}$, and $\begin{pmatrix}2.5\\0.0\end{pmatrix}$.

The T2GKA clustered results with the proper clusters centers for the 5 same-circle datasets are shown in Fig. 4. For the 5 same-circle datasets, the E_{FCM}=0.0042, E_{FCMPCM}=0.0003, E_{PFCM}=0.0039 (*a*=1, *b*=0.1, *m*=2, η=2), E_{PFCM}=12.2009 (*a*=1, *b*=1, *m*=2, η=2), E_{GKA}=0.0036, and E_{T2GKA}=0.0026. We compared the five clustering method's E_A values. Except for the E_{FCMPCM}, the E_{T2GKA} value is smaller than that in other methods. The clustering results show the robustness of the proposed T2GKA because a reasonable amount of noise data does not affect its clustering performance.

Example 3: To verify that the proposed method can accord the prototype dataset structure to learn the proper cluster volumes, 2 artificial datasets named 7cluster and sinusoidal were designed. There are 700 and 200 prototype data points in the 7cluster and sinusoidal datasets, respectively. There are 700 prototype data points in the 7cluster datasets clustered into 7 clusters with different sizes and orientations. Each cluster contains 100 prototype data points. The 7cluster dataset is a mixture of two 2-variate distributions with varying deviation, its mean vectors are $\begin{pmatrix}5.0\\1.0\end{pmatrix}, \begin{pmatrix}1.0\\5.0\end{pmatrix}, \begin{pmatrix}1.0\\1.0\end{pmatrix}, \begin{pmatrix}5.0\\5.0\end{pmatrix}, \begin{pmatrix}2.0\\-2.0\end{pmatrix}, \begin{pmatrix}-2.0\\2.0\end{pmatrix}$, and $\begin{pmatrix}4.5\\3.0\end{pmatrix}$.

The prototype data points in the dataset sinusoidal are generated by $x_2 = 10^{-4}\, sin(0.001x_1^2)x_1^3 + \varepsilon$, where $x_1 \in [0,100]$ and $\varepsilon \sim Normal(0,25)$ is a normally distributed random noise. The T2GKA clustered results with the proper clusters volumes for the 7cluster and sinusoidal datasets are shown in Figs. 5 and 6, respectively. The proposed T2GKA is essentially a DEECV based learning algorithm integrated with the T2GKA. The experimental results show that the T2GKA can learn suitable sized cluster volume along with dataset varying structure volume.

TABLE I. THE TERMINAL CENTROIDS LEARNED BY FCM, FCMPCM, PFCM, GK, AND T2GKA IN THE DATASETS X_{400} AND X_{550}, EXAMPLE 1

Clustering Algorithm	**Data sets**			
	X400 (centroid)		**X550 (centroid)**	
	x1	**x2**	**x1**	**x2**
FCM: m=2	**4.9794**	**5.9531**	**5.5711**	**5.4143**
	4.9407	**12.0593**	**5.1885**	**11.6395**
FCMPCM: η=2	**5.0017**	**6.0094**	**5.0076**	**6.0091**
	4.9973	**12.0102**	**4.9968**	**12.0103**
PFCM: a=1, b=1, m=2, η=2	**4.9843**	**5.9746**	**5.3716**	**5.7308**
	4.9566	**12.0506**	**5.1281**	**11.6642**
PFCM: a=1, b=0.1, m=2, η=2	**4.9800**	**5.9558**	**5.5410**	**5.4604**
	4.9427	**12.0582**	**5.1804**	**11.6445**
GKA: m=2	**4.9782**	**5.9538**	**5.1064**	**5.4443**
	4.9397	**12.0568**	**5.5502**	**11.4151**
T2GKA: m=2	**5.0048**	**6.0239**	**5.0137**	**5.9593**
	5.0097	**12.0837**	**4.9743**	**12.1031**

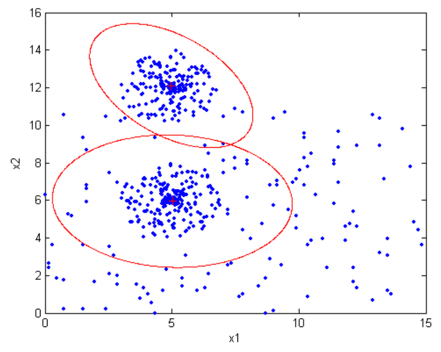


Figure 3. The T2GKA clustering results with the proper clusters volumes for the dataset X_{550}, Example 1

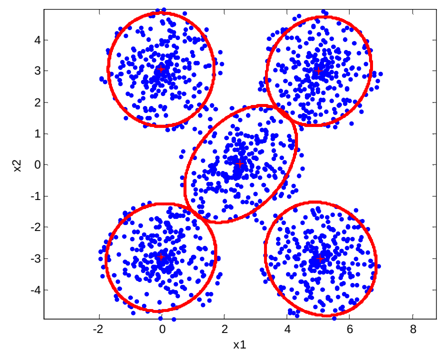


Figure 4. Clustering results using 5 ellipsoids for the prototype data points in the dataset 5samecircle, Example 2

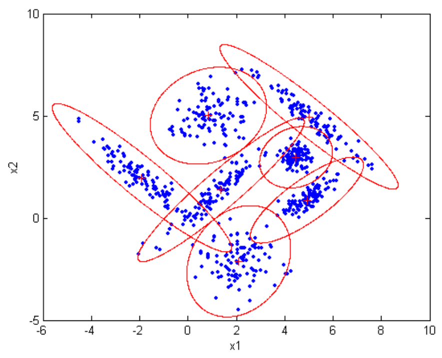


Figure 5. Clustering results using 7 ellipsoids for the prototype data points in the dataset 7cluster, Example 3

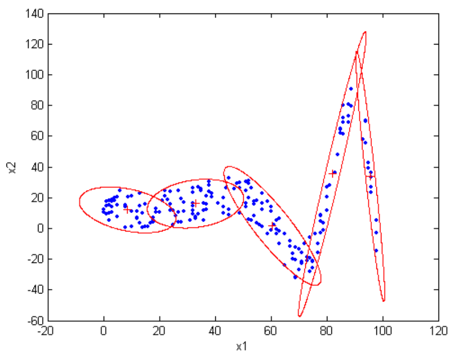


Figure 6. Clustering results using 5 ellipsoids for the prototype data points in the dataset sinusoidal, Example 3

V. CONCLUSIONS

This paper presented an efficient combined probabilistic and possibilistic method for building Type-2 fuzzy sets. Utilizing this concept we presented a Type-2 GKA (T2GKA) that is an extension of the conventional GKA. The experimental results showed that the T2GKA was less susceptible to noise than the Type-1 GKA. The clustering results showed the robustness of the proposed T2GKA because a reasonable amount of noise data does not affect its clustering performance.

The DEECV is proposed to effectively evaluate proper ellipsoid volume. The proposed T2GKA is essentially a DEECV-based learning algorithm integrated with T2GKA. The experimental results showed that the T2GKA can learn suitable sized clusters volume along with varying dataset structure volume.

REFERENCES

[1] E. R. Hruschka, R. J. G. B. Campello, A. A. Freitas and A. C. P. L. F. de Carvalho, “A Survey of Evolutionary Algorithm for Clustering,” IEEE Trans. Syst., Man, Cybern., pt. C, vol. 39, no. 2, pp.133-155, March 2009.

[2] J. Bezdek, Pattern Recognition with Fuzzy Objective Function, Plenum Press, New York, 1981.

[3] D. E. Gustafson and W. C. Kessel, “Fuzzy clustering with a fuzzy covariance matrix,” in Proc. IEEE Conf. Decision Contr., San Diego, CA, pp. 761-766, 1979.

[4] R. Babuška, Fuzzy modeling for control, Kluwer Academic Publishers: Massachusetts, 1998.

[5] R. Krishnapuram and J. Keller, “A possibilistic approach to clustering,” IEEE Trans. Fuzzy Sys., vol. 1, no. 2, pp. 98-110, May 1993.

[6] R. Krishnapuram and J. Keller, “The possibilistic c-Means algorithm: Insights and recommendations,” IEEE Trans. Fuzzy Sys., vol. 4, no. 3, pp. 385-393, August 1996.

[7] N. R. Pal, K. Pal, J. M. Keller and J. C. Bezdek, “A Possibilistic Fuzzy c-Means Clustering Algorithm,” IEEE Trans. Fuzzy Sys., vol. 13, no. 4, pp. 517-530, August 2005.

[8] L. A. Zadeh, “The concept of a linguistic variable and its application to approximate reasoning-I,” Inform. Sci., vol. 8, no. 3, pp. 199-249, 1975.

[9] J. Mendel, “Advances in Type-2 fuzzy sets and systems,” Inform. Sci., vol. 177, pp. 84-110, 2007.

[10] N. N. Karnik, J. M. Mendel and Q. Liang, “Type-2 fuzzy logic systems,” IEEE Trans. Fuzzy Sys., vol. 7, no. 6, pp. 643-658, December 1999.

[11] Q. Liang and J. M. Mendel, “Interval Type-2 fuzzy logic systems: theory and design,” IEEE Trans. Fuzzy Syst., vol. 8, no. 5, pp. 535-550, October 2000.

[12] J. M. Mendel, Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions, Upper Saddle River, NJ: Prentice-Hall, 2001.

[13] S. Coupland and R. John, “Geometric Type-1 and Type-2 fuzzy logic systems,” IEEE Trans. Fuzzy Sys., vol. 15, no. 1, pp. 3-15, February 2007.

[14] C. Hwang and F. C. H. Rhee, “Uncertain Fuzzy Clustering: Interval Type-2 Fuzzy Approach to C-Means,” IEEE Trans. Fuzzy Sys., vol. 15, no. 1, pp. 107-120, February 2007.

[15] H. B. Mitchell, “Pattern recognition using type-II fuzzy sets,” Inform. Sci., vol. 170, pp. 409-418, 2005.

[16] J. Zeng and Z. Q. Liu, “Type-2 Fuzzy Sets for Pattern Recognition: The State-of-the-Art,” Journal of Uncertain Systems, vol. 1, no. 3, pp. 163-177, 2007.

[17] J. Zeng, L. Xie and Z. Q. Liu, “Type-2 fuzzy Gaussian mixture models,” Pattern Recognition, vol. 41, pp. 3636-3643, 2008.

[18] M. H. Fazel Zarandi, M. Zarinbal and M. Izadi, “Systematic image processing for diagnosing brain tumors: A Type-II fuzzy expert system approach,” Applied Soft Computing, vol. 11, pp. 285-294, January 2011.

[19] L. A. Zadeh, “Fuzzy Sets as a Basis for a Theory of Possibility,” Fuzzy Sets and Systems, vol. 1, no. 1, pp. 3–28, 1978.

[20] D. Dubois, L. Foulloy, G. Mauris and H. Prade, “Probability-possibility transformations, triangular fuzzy sets and probabilistic inequalities,” Reliab. Comput., vol. 10, no. 4, pp. 273-297, 2004.

[21] G. Mauris, “Expression of Measurement Uncertainty in a Very Limited Knowledge Context: A Probability Theory-Based Approach,” IEEE Trans. Instr. Measu., vol. 56, no. 3, pp. 731-735, June 2007.

[22] L. Yao, “Nonparametric learning of decision regions via the genetic algorithm,” IEEE Trans. System, Man, and Cybernetics, vol. 26, no. 2, pp. 313-321, April 1996.

[23] L. Vandenberghe, S. Boyd and S. P. Wu, “determinant maximization with linear matrix inequality constraints,” J. SIAM, vol. 19, no. 2, pp. 499-533, 1998.

[24] L. Xu, A. Krzyak and E. Oja, “Rival penalized competitive learning for clustering analysis, RBF net, and curve detection,” IEEE Trans. Neural Netw., vol. 4, no. 4, pp. 636-649, July 1993.

Pattern-oriented Enterprise Architecture Management

Tobias Brunner
Reutlingen University, Faculty of Informatics
Architecture Reference Lab of the
SOA Innovation Lab, Germany
tobias1.brunner@student.reutlingen-university.de

Alfred Zimmermann
Reutlingen University, Faculty of Informatics
Architecture Reference Lab of the
SOA Innovation Lab, Germany
alfred.zimmermann@reutlingen-university.de

Abstract – **Current Enterprise Architecture Frameworks are limited in their ability of providing reference architectures and architecture development methods to assist for developing comprehensive, guided and fully-fledged enterprise architectures. The outcome is that in the majority of cases present architecture modeling approaches are rarely validated, have sparse reference and meta-models as well as general and limited statements for building and designing a comprehensive enterprise architecture. Furthermore these frameworks do not include recent approaches for developing enterprise architectures, like cloud computing, service-orientation and broad security reference architectures. This is a real problem of Enterprise Architecture Frameworks and their process models for designing and developing wide-ranging enterprise architectures. The Enterprise Services Architecture Reference Cube (ESARC) defines a integral framework for enterprise architecture artifacts with their main relationships. Our new idea and contribution is to extend existing architecture frameworks and their development methods with a new pattern-oriented Architecture Development Method approach. Based on our research on the ESARC architecture framework we have leveraged a new pattern-oriented Architecture Development Method and have extracted a coherent set of enterprise architecture patterns.**

Keywords – Enterprise Architecture Management; Enterprise-Services-Architecture-Reference-Cube; ESARC; Pattern-oriented Architecture Development Method; Core-Pattern-Catalog; Architecture Framework.

I. INTRODUCTION

Individual software solutions, legacy applications, and different infrastructure components lead to high costs and limited ability to respond quickly to new business requirements. Many companies start initiatives of Enterprise Architecture Management (EAM) and use Enterprise Architecture (EA) Frameworks like The Open Group Architecture Framework (TOGAF) [2, 3] to address these problems. But present Enterprise Architecture Frameworks are limited in their modeling approaches. These approaches are historically grown and do not cover current standards. New topics like cloud computing, service-orientation and especially the security aspect for enterprise architectures are not considered. In addition to these problems most of the EA-frameworks do not provide or provide an insufficient guided development method for building useful and suitable enterprise architectures.

The Enterprise Services Architecture Reference Cube (ESARC) [1] is a new Enterprise Architecture Framework derived from TOGAF and present standards like essential [4], the service model of ITIL [5] and from resources for service-oriented computing [6, 7, 8]. The current release of the ESARC is an original abstract architecture reference model, provides a integral approach for designing, developing, monitoring, evaluation and optimization of enterprise architectures over eight abstract Reference Architectures. Due to the new enterprise framework ESARC the hypothesis of this research integrates the idea to develop a new pattern-oriented [9] Architecture Development Method (ADM) [10] based on the TOGAF-Architecture Development Method [11] and the ITIL service-oriented-lifecycle and provides a relevant basis for high enterprise architecture innovation impacts for practice.

Based on the ESARC, the new Enterprise Architecture Development Method provides a process to create and manage enterprise architectures and integrates different patterns for a full iteration over the eight ESARC Reference Architectures and predefines basic architectural work products and artifacts. The whole development method and all integrated patterns are derived from current Enterprise Architecture Frameworks, process models and best practices from the Enterprise Architecture Management environment.

The aim of this research is to provide a core-pattern-catalog for a structured and guided development of enterprise architectures. Enterprise architects can expand the pattern-catalog for their own needs and requirements. The existing core-pattern-catalog comprises 26 patterns for developing essential artifacts of enterprise architectures.

Section II sets the base of our pattern approach by describing the ESARC – Enterprise Services Architecture Reference Cube. In Section III, we describe our new original pattern-oriented Architecture Development Method as a procedural framework. This architecture method mostly consists of two different types of patterns: "Architecture Development Method Patterns" represented in Section III and "Reference Architecture Development Patterns" embodied in Section IV. Section V describes the "Pattern Evolution Process" for the developed core-pattern-catalog. The conclusion in Section VI summarizes major achievements of this paper.

II. Enterprise Services Architecture Reference Model

ESARC – the Enterprise Services Architecture Reference Cube – is an abstract architecture reference model, which provides an integral view for main interweaved architecture types, and is derived primarily from state of art architecture frameworks. ESARC defines main architecture classification categories and abstracts from a concrete business scenario or technologies. The Open Group Architecture Framework (TOGAF) is the current standard for enterprise architecture and provides the basic blueprint and structure for our extended service-oriented enterprise software architecture domains like: Architecture Governance, Architecture Management, Business & Information Architecture, Information Systems Architecture, Technology Architecture, Operation Architecture, Security Architecture, and the Cloud Services Architecture.

The ESARC – Enterprise Services Architecture Reference Cube as seeded by [1], in Figure 1, unifies a set of close related reference models for essential architecture domains. The ESARC description in this section is fundamentally based on our previous research and extends this with our current presented pattern approach in a manageable and more procedural way. ESARC provides a coherent aid for examination, comparison, classification and quality ratings of specific architecture categories.

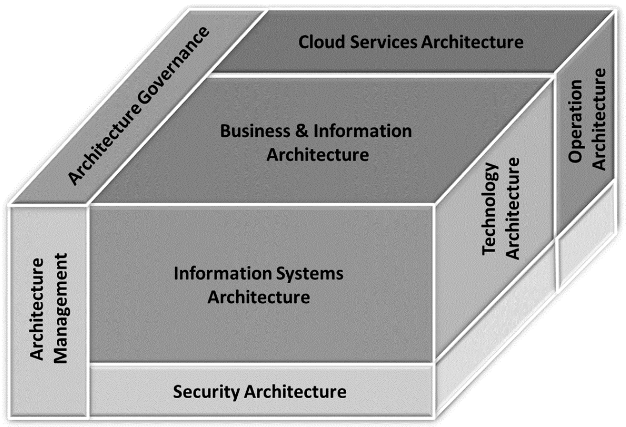


Figure 1. ESARC - Enterprise Software Architecture Reference Cube.

The Architecture Governance and Management framework provides the organizational context for the following main types of enterprise software architectures like Business & Information Architecture, the Information Systems Architecture, and the Technology Architecture.

Architecture Governance defines and maintains the Architecture Governance cycle. The Architecture Governance cycle sets the abstract governance frame for concrete architecture management activities within the enterprise or a product line development and specializes the more abstract Deming Cycle, as in [12], to the following management activities: plan, define, enable, measure, and control (see Section V. Reference Architecture Development Patterns with the Governance Cycle Pattern).

The second aim of Architecture Governance is to set rules for architecture compliance with internal and external standards. In addition policies for governance and decision definition are set to allow a standardized and efficient process for architecture decisions within the enterprise architecture organization. Enterprise and software architects are acting on a sophisticated connection path coming from business and IT strategy to the architecture landscape realization for interrelated business domains, applications and technologies. Architecture Governance has to set rules for the empowerment of people, defining the structures and procedures of an Architecture Governance Board, and setting rules for communication (see Section V. Reference Architecture Development Patterns with the Governance Board Pattern). With specifications from Architecture Governance we define main ESARC Architecture Management procedures for service-oriented enterprise software architectures: service strategy and life cycle management, service security, risk management, quality insurance for services, service testing, and service monitoring and control. Main management aspects include as well the metamodel-based management of service contracts and registries, and the reuse management of services in the enterprise.

The ESARC – Business & Information Reference Architecture defines the link between the enterprise business strategy and the results of supporting strategic initiatives through information systems. The Business & Information Reference Architecture provides a single source and comprehensive repository of knowledge, from which concrete corporate initiatives will evolve and link. This knowledge is model based and defines an integrated enterprise model of the business, which includes the organization and business processes. The Business & Information Reference Architecture sets the base for the business IT alignment. Important concepts of Business & Information Reference Architecture are: business and information strategy, the organization, and main business requirements for information systems like key business processes, business rules, business products, services, and related business control information.

The ESARC – Information Systems Reference Architecture provides an abstract blueprint of the individual solution architecture for application systems to be deployed and individually customized. The ESARC – Information Systems Reference Architecture contains the main application specific service types and defines their relationship by a layer model of building services. The core functionality of domain services is linked with application interaction capabilities and with the business processes of the customer's organization.

The ESARC – Technology Reference Architecture describes the logical software and hardware capabilities that are required to support the deployment of business, data, and application services. This includes IT infrastructure, middleware, networks, communications, processing, and standards. The layers of the ESARC – Technology Reference Architecture and the layers of the ESARC – Information Systems Reference Architecture correspond to each other.

III. Architecture Development Method Patterns

At this point it should be noted that the various Reference Architectures of the ESARC are strongly connected with each other. Therefore, the individual areas of Reference Architectures have to be developed in coordination with each other. All individual Reference Architectures will be developed in terms of their interdependence.

The pattern-oriented Architecture Development Method offers an entry point and a navigable process for iteration through the eight ESARC Reference Architectures. The whole Architecture Development Method is based on canonical structured patterns (Name, Classification, Problem, Solution, and Description) for the iteration through and for developing and designing architecture artifacts within the eight ESARC Reference Architectures.

The "Architecture Development Method (ADM) Pattern" is based on [11] and describes the pattern-oriented Enterprise Architecture Development Method. This pattern represents a procedural method (see Figure 2) for iterating through the eight Reference Architectures of the ESARC [1].

In addition, we derived from the "Architecture Development Method (ADM) Pattern" an iteration-loop (See Figure 3) by describing five subsequent patterns for iteration within each of the eight Reference Architectures of ESARC.

The aim is a core-pattern-catalog with architecture patterns for developing, structure- and designing integral enterprise architectures. Every enterprise architect can update, expand and enlarge the core-catalog with new patterns for its own needs and requirements.

The novelty of our new-presented pattern approach for Enterprise Architecture Management is its close fit with the integral classification framework from our previous research of ESARC and our specific coarse-granular EAM-process. Pattern approaches for Enterprise Architecture Management from the state of art are more visualization oriented and therefore more detailed, but lacking the overall integration in a more complex EAM-process.

Architecture Development Method Pattern

How can it be realized to set up an Architecture Development Method for developing a holistic enterprise architecture based on the ESARC?

Classification: Architecture Development Method

Problem: The absence of structured and concrete behavior guidelines for developing enterprise architectures ends up in complexity.

Besides these problems, the ESARC has eight highly aligned Reference Architectures they have to be developed in a structured, organized and controlled way.

Solution: An enterprise architect gets a structured way to iterate through the ESARC to build up integral enterprise information architectures. Enterprise architects should use practical guidelines for every Reference Architecture and descriptions for specified outcomes. The Architecture Development Method should be used as a support guideline and be adaptable for every enterprise.

Description: The Architecture Development Method maps the service-lifecycle of ITIL to a structured architecture development process based on the eight ESARC Reference Architectures. The "Architecture Development Method Pattern" is represented in Figure 2.

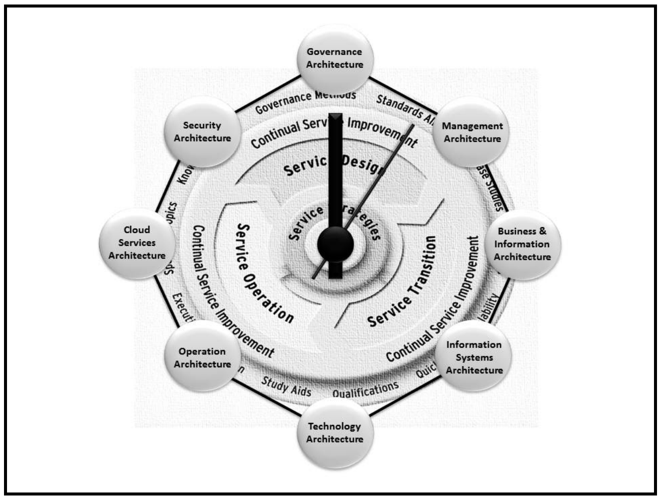


Figure 2. ESARC – Architecture Development Method Pattern.

The "Architecture Development Method Pattern" describes a structured and guided process to iterate through the ESARC and build up an enterprise architecture. The next step shows a further development of the already mentioned "Architecture Development Method Pattern".

The enhancement is a uniform iteration-loop for each step within the eight Reference Architectures of the ESARC.

The iteration-loop is derived from the Deming Cycle [12] respectively the PDCA-Cycle and it also contains artifacts from TOGAF/ADM and the service-oriented lifecycle of ITIL. The iteration-loop defines a structured development cycle for every step within the "ADM-Pattern". The combination of the "Architecture Development Method Pattern" with the mentioned iteration-loop is represented in Figure 3.

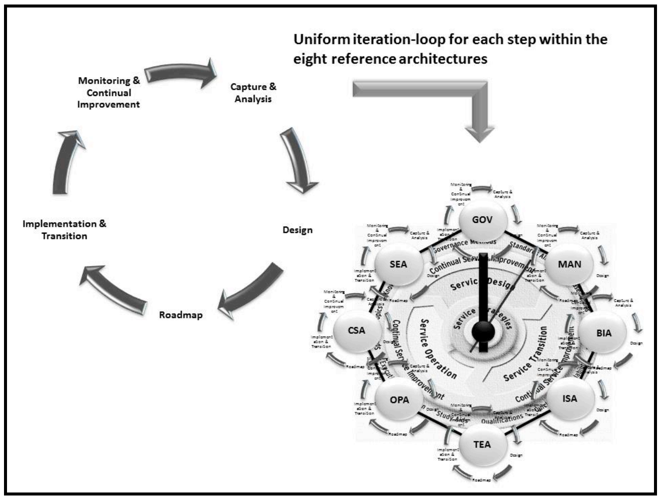

Uniform iteration-loop for each step within the eight reference architectures
Monitoring & Continual Improvement
Capture & Analysis
Design
Roadmap
Implementation & Transition
GOV
MAN
BIA
ISA
TEA
OPA
CSA
SEA

Figure 3. ESARC – Architecture Optimization Pattern.

The newly added iteration-loop relates to the eight Reference Architectures and contains five different steps for developing each of the Reference Architectures of ESARC consistently in the same way, but with a different particular focus.

For each step within the iteration-loop a pattern exists:

1) Architecture Capture & Analysis Pattern
2) Architecture Design Pattern
3) Architecture Roadmap Pattern
4) Architecture Implementation & Transition Pattern
5) Architecture Monitoring & Continual Improvement Pattern

In conclusion, the Architecture Development Method is defined by the “Architecture Development Method Pattern”, as a recommended development sequence for the various Reference Architectures of the ESARC. Furthermore there is an iteration-loop defined by five patterns to build each of the eight ESARC Reference Architectures.

Within the Architecture Development Method the enterprise architect has to determine first the scope of Reference Architectures. Each of the Reference Architecture can have a different scope and maturity level, because the developed and recommended sequence of development in the architecture method is an iterative one. The maturity of the enterprise architectures and their outcomes will increase cyclically.

In this section, we have seen a procedural, pattern-based Architecture Development Method for iterating through the eight Reference Architectures of the ESARC by the “Architecture Development Method Pattern”. In addition there was defined an extended iteration-loop for developing and designing architecture artifacts for each of the eight reference architectures of the ESARC.

IV. Reference Architecture Development Patterns

This section provides patterns for developing entities/artifacts for each Reference Architecture of the ESARC. The following mentioned patterns are designing and developing concrete architecture artifacts, so they are categorized as “Reference Architecture Patterns” (RA-Patterns). Currently, patterns are available for four Reference Architectures of the ESARC. Patterns available for building and designing the:

- Governance Reference Architecture
- Management Reference Architecture
- Business & Information Reference Architecture
- Cloud Services Reference Architecture

Following is shown a scenario of the core-pattern-catalog. There are RA-Patterns to develop, manage, evaluate and redesign the mentioned reference architectures. Two RA-Patterns (**Governance Board Pattern, Governance Cycle Pattern**) refer to the “Governance Reference Architecture” one RA-Pattern (**Service Lifecycle Pattern**) refers to the “Management Reference Architecture” and the last RA-Pattern that is shown in this paper (**Cloud Service Model Pattern**) refers to the “Cloud Services Reference Architecture”. The following mentioned patterns show a small but typical part of the core-pattern-catalog.

Governance Board Pattern

How can it be ensured that important command and control tasks are addressed within an enterprise?

Classification: Reference Architecture - Governance

Problem: If command, control and other governance tasks are not transferred from one central location, chaos will result. It is important to form a central governance steering-position.

Solution: A Governance Board must be set up. The Governance Board owns all command and control functions within the enterprise and has all necessary rights and abilities to fully perform these activities. The board may transfer rights and delegate tasks to subsequent instances.

Description: A Governance Board should be created in each company. A Governance Board should fulfill the following tasks:

- Defining a clear mission/vision to lead and strategically align the company.
- Acquisition of corporate assets.
- Definition of corporate programs, tasks and services.
- All business opportunities and chances have to be analyzed and a strategic has to be created on the existing results and resources.
- Monitoring and compliance of legal and financial requirements. This includes the monitoring of the budget and providing the investments.
- Review of financial statements.
- Development of appropriate risk management practices and activities. Risk factors must be recognized and protected in the company.
- Determine individuals for leadership positions (e.g. the position of the CEO).

Governance in a company is not about implementation of tasks or requirements. Rather it is about monitoring, that the requirements be done and completed.

Governance Cycle Pattern

How can fundamental control- and management processes be implemented uniformly in an enterprise?

Classification: Reference Architecture - Governance

Problem: Companies are formed by complex structures and exchange relationships. A procedure to accomplish all enterprise command and control tasks is necessary.

Solution: The aim of the Governance Cycle is a fixed-cyclical approach for planning, defining, evaluating, measuring and controlling of all control, governance and management measures that established by the Governance Board.

Description: The Governance Cycle comprises the following architecture processes:

- Plan: The Governance Board schedules all necessary control and management tasks.

- Define: All tasks must be planned in detail. Actors and roles must be defined to be responsible for implementing and achieving planned goals.
- Enable: The Governance Board communicates all the objectives and tasks to the responsible persons, they ensure and guarantee that all objectives and tasks going to be reached and implemented.
- Measure: It is important that the progress of implementation, the ability of the actors and the success or failure of the goals and tasks will be measurable through appropriate performance indicators.
- Control: If the defined control and management functions are measurable, it is possible to recognized error-sources and potential for new development-possibilities. The results can be converted into a new governance cycle plan.

Service Lifecycle Pattern

How can the development of enterprise-wide services be targeted, standardized, successfully used and implemented?

Classification: Reference Architecture - Management

Problem: If external and internal services are developed without any structured and specified orientation to a consistent approach, there can be no assurance that the developed services are without redundancy and that they are goal-oriented and following the suggested strategy plan.

Solution: ITIL describes a continuous improvement process which is called the service-oriented lifecycle. Based on the strategic direction of the company (Service Strategy) the services are being developed (Service Design) and transferred to the operating mode (Service Transition). Then, the services are operated and so they are available for the overlying business processes (Service Operation). These phases are surrounded by a continuous improvement process (Continual Service Improvement) [5].

Description: The detailed description of the service lifecycle can be read in the five publications [13, 14, 15, 16, 17] of ITIL.

- Service Strategy - Service Design - Service Transition
- Service Operation - Continual Service Improvement

These five core documents form and detail the phases of an iterative and multidimensional service-oriented lifecycle for the company's existing- or to be created services.

RA - Cloud Service Model Pattern

*How can enterprise-resources be represented as services (*aaS) in a cloud-environment?*

Classification: Reference Architecture – Cloud Services

Problem: Due to the currently and not adequate existing service-models (IaaS, PaaS, SaaS) is it not possible to represent a whole and extensive architecture within an enterprise.

Solution: The previously existing service models must be expanded. The service models must be based on the enterprise structures. These services within all different levels of an enterprise architecture, have to be transferred and mapped into a wide range of enterprise service-models (*aaS) in the cloud-environment.

Description: The below illustrated subdivision of different services-models (*aaS) illustrates a potential way to structure and represent all enterprise resources and artifacts in a cloud-environment.

Examples of current discussed cloud services can be found in [18]: Testing-as-a-Service, Management-as-a-Service, Governance-as-a-Service, Application-as-a-Service, Process-as-a-Service, Information-as-a-Service, Database-as-a-Service, Storage-as-a-Service, Infrastructure-as-a-Service, Platform-as-a-Service, Integration-as-a-Service, Security-as-a-Service, Software-as-a-Service. Further *-as-a-Services are likely to follow.

In the section above, we described the so called “Reference Architecture Patterns” (RA-Patterns). These patterns provide design models and important artifacts which have to be fundamentally addressed by developing all eight Reference Architectures. These patterns can be used in the “**design**”-phase within the “Architecture Development Method Pattern” and its iteration-loop (see Section III, Architecture Development Method Patterns).

V. Pattern Evolution Process

The partially described core-pattern-catalog in the sections before can be centrally accessed via a web-application. The pattern-catalog can be used, enlarged, evaluated and new patterns can be easily added in a predefined canonical structure.

One continuative idea is to open the current pattern-catalog to a wide range of interested stakeholders and involve them to the pattern creation and evolution process and to adapt already available knowledge and findings from the project’s domain as early as possible. For that possibility has to be an implementation of a well-defined Pattern Evolution Process within the core pattern catalog.

René Reiners [19] already introduced a Pattern Evolution Process and in our vision that evolution process will be merged with the established Architecture Development Method. Reiners introduced the notion of a pattern’s state that is used to track the development of a pattern over time. The current implementation provides the following pattern status information:

- **Just created** patterns were recently submitted as a non-validated idea.
- **Under consideration** means that the pattern looks promising but needs further evaluation.
- **Pattern candidate** states that the pattern is close to being approved.
- **Approved** finalizes the pattern review process and settles the pattern as a design pattern.

The pattern-lifecycle process allows for continuously evaluating the design knowledge gathered during the project’s lifetime and makes patterns as well as pattern ideas available during the whole development process. Successful findings or surprisingly failing results will be communicated as anti-patterns. In addition patterns can originate both from the project itself and from external sources.

Reiners [19] distinguishes three different categories:

- **Derived from project:** The pattern derives directly from the work within the project and will automatically be assigned the state under consideration. The pattern will be reviewed, perhaps re-worked and finally validated through an approval process by a validator.
- **Adapted to project:** The pattern originates from external sources, but has been adapted for use in the context of the project.
- **External:** The pattern exists in other pattern collections (e.g., a standard UI pattern) and is implemented in the current project's products and services.

With this background, the pattern-oriented Architecture Development Method can be developed in a comprehensive, structured and extensive knowledge-based manner.

VI. Conclusion

The research approach provides an innovative and so far not available Architecture Development Method, which is based on the our EA-Framework ESARC. Our new introduced Architecture Development Method relies on current standards for IT Enterprise Architectures like the TOGAF-Architecture Development Method, the service-oriented lifecycle of ITIL and the Deming Cycle.

We note that the entire Architecture Development Method is based on canonical structured patterns which form a basic core-pattern-catalog. The vision that is pursued with the pattern-based Architecture Development Method and the core pattern catalog is a constantly growing catalog. Enterprise architects can use the core catalog adapt them for their own needs and requirements and expand the catalog with new self-developed canonical structured architecture patterns.

The whole pattern catalog is public and can be centrally accessed, increased and evaluated via a web-application. The core pattern catalog was applied in a research assessment workshop with students, researchers and professors. That provided new transparent results for subsequent changes on the Architecture Development Method, the patterns and the related processes. The results of these assessments need to be interpreted in the context of company specific strategies and use cases.

The outcome of our research is a first cut of a new core-pattern-catalog for the guided development and architecture modeling. Additional future work has to consider additional patterns, because the current pattern catalog does not contain "Reference Architecture Patterns" for all reference architectures of ESARC.

An additional improvement idea deals with patterns for visualization of architecture artifacts and architecture control information to be operable on an architecture management cockpit. To improve semantic-based navigation within the complex space of EAM-visualization and service-oriented enterprise architecture management we are currently working on ontology models [20] for the ESARC – The Enterprise Services Architecture Reference Cube.

Acknowledgment

Our paper extends current work from the SOA Innovation Lab (www.soa-lab.de) Germany, which is a major business innovation network of service-oriented software technology and application companies.

References

[1] A. Zimmermann and G. Zimmermann, "*ESARC - Enterprise Services Architecture Reference Cube for Capability Assessments of Service-oriented Systems.,* " The Third International Conferences on Advanced Service Computing, Rome, Italy, ISBN 978-1-61208-152-6, IARIA Proceedings of SERVICE COMPUTATION 2011, pp. 63-68, 25-30 September 2011.

[2] T. O. Group, "*The Open Group Architecture Framework Version 9.1,* " 2009. [Online]. Available: http://www.opengroup.org/togaf/. [Accessed 24 February 2012].

[3] P. R. Harrison and T. O. Group, "TOGAF 9 Foundation Study Guide", Wilco, Amersfoort: Van Haren Publishing, 2009.

[4] Essential, "*Essential Architecture Project*", [Online]. Available: http://www.enterprise-architecture.org. [Accessed 19 June 2011].

[5] OGC, Office of Governance Commerce, "*Introduction to ITIL*", UK London: Van Haren Publishing, 2005.

[6] C. MacKenzie, K. Laskey, F. McCabe, P. Brown and R. Metz, "*OASIS "Reference Model dor Service Oriented Architecture*" 1.0, OASIS Standard," 12 October 2006.

[7] J. Estefan, K. Laskey, F. McCabe and D. Thomton, "*OASIS Reference Architecture for Service Oriented Architecture*" Version 1.0, OASIS Committee Draft 02," 14 October 2009.

[8] J. Estefan, K. Laskey, F. McCabe and D. Thomton, "*OASIS Reference Architecture for Service Oriented Architecture*" Version 1.0, OASIS Public Review Draft 1," 23 April 2008.

[9] T. Erl, "*SOA Design Patterns*", Boston: PRENTICE HALL / Pearson Education, Inc, 2009.

[10] T. Brunner, "*Ein patternbasiertes Vorgehensmodell für den Enterprise Services Architecture Reference Cube,"* Masetr-Thesis, Reutlingen University, 2012.

[11] The Open Group, "*The Open Group Architecture Development Method (ADM),"* [Online]. Available: http://pubs.opengroup.org/architecture/togaf8-doc/arch/chap03.html. [Accessed 24 February 2012].

[12] W. E. Deming, "*Out of the Crisis*", Massachusetts: Massachusetts Institute of Technology, 1982.

[13] M. Iqbal and M. Nieves, "*ITIL® V3 Service Strategy*", The Stationery Office , 2007.

[14] C. Rudd and V. Lloyd, "*Service Design*", The Stationery Office , 2007.

[15] S. Lacy and I. Macfarlane, "*Service Transition*", The Stationery Office , 2007.

[16] O. o. G. Commerce, "*Service Operation*", The Stationery Office, 2007.

[17] G. Spalding, "*Continual Service Improvement*", The Stationery Office, 2007.

[18] D. S. Linthicum, "*Cloud Computing and SOA Convergence in Your Enterprise*," 2009.

[19] R. Reiners, "*A Pattern Evolution Process – From Ideas to Patterns*", Proceeedings Informatiktage 2012 Bonn - Germany, in Lecture Notes in Informatics, Vol. S-11,, pp. 115-118, 2012.

[20] S. Bourscheidt, T. Breuer, T. Brunner, B. Fetler and G. Fogel, "*ESARC - Referenzmodell und Ontologie für Enterprise Architecture Management*", Hochschule Reutlingen, Fakultät Informatik, Reutlingen, February 2012.

Development of Graphical User Interfaces based on User Interface Patterns

Stefan Wendler, Danny Ammon, Teodora Kikova, Ilka Philippow
Software Systems / Process Informatics Department
Ilmenau University of Technology
Ilmenau, Germany
{stefan.wendler, danny.ammon, teodora.kikova, ilka.philippow}@tu-ilmenau.de

Abstract — **This paper addresses the research concerning possibilities for reducing the effort of adapting graphical user interfaces to requirements of individual customers. User interface patterns are promising artifacts for improvements in this regard. The details of graphical user interface transformations from user interface patterns into executable interface code are considered. We describe how reuse and automation within user interface transformation steps can be established. For this purpose, formal descriptions of user interface patterns are necessary. Today, however, most user interface patterns exist only in a verbal or graphical form of description. We use XML-based user interface description languages like UIML and UsiXML for the specification of user interface patterns. We experimentally investigated and analyzed strengths and weaknesses of two transformation approaches which were built on different software patterns. As a result, we show that formal user interface patterns can be transformed into executable interfaces, and that they assist in raising effectiveness and efficiency of the development process of a GUI system. Finally, we developed suggestions on how to apply these positive effects of user interface patterns for the development of pattern-based graphical user interfaces.**

Keywords — graphical user interface; model driven software development; user interface patterns; UIML; UsiXML

I. Introduction

Interactive systems. Interactive systems demand for a fast and efficient development of their graphical user interface (GUI), as well as its adaption to changing requirements throughout the software life cycle. In this paper, e-shops serve as a representative of these interactive systems. Currently, they are a fundamental asset of modern e-commerce business models. In many cases, such systems are offered as standard software, which allows several customization options after installation. In this context, they are differentiated into the application kernel and a GUI system.

The application kernel software architecture relies on well-proven and, partially, self-developed software patterns. Thus, it offers a consistent structure with defined and differentiated types of system elements. This has a positive effect on the understanding of the modular functional structures as well as their modification options.

Limited customizability of GUIs. Contrary to the application kernel, the customization of the GUI is possible only with rather high efforts. An important reason is that software patterns do not cover all aspects needed for GUIs. These patterns have been commonly applied for GUIs [1][2] but in most cases they are limited to functional and control related aspects [3]. The visual and interactive components of the GUI are not supported by software patterns yet. Furthermore, the reuse of GUI components, e.g., layout, navigation structures, choice of user interface controls (*UI-Controls*) and type of interaction, is only sparsely supported by current methods and tools. For each project with its varying context, those potentially reusable entities have to be implemented and customized anew leading to high efforts.

Moreover, the functional range of standard software does not allow a comprehensive customization of its GUI system. The GUI requirements are very customer-specific. In this regard, the customers want to apply the functionality of the standard software in their individual work processes along with customized dialogs. However, due to the characteristics of standard software, only basic variants or standard GUIs can be offered. So far, combinations of components of the application architecture with a GUI are too versatile for a customizable standard product.

UIPs. We propose an approach to this problem through the deployment of User Interface Patterns (UIPs). These patterns offer well-proven solutions for GUI designs [4], which embody a high quality of usability [5]. So far, UIPs have not been considered as source code artifacts, in contrast to software patterns. Current UIPs and their compilations mostly reside on an informal level of description [6].

A. Objectives

In this paper we show that formal UIPs can assist in raising effectiveness and efficiency of the development process of a GUI system. For a start, we describe, from a theoretical point of view, how reuse and automation within GUI transformation steps can be established by the deployment of UIPs. On the basis of formal UIPs, we discuss the possibilities of transformations into executable GUIs. For this purpose, two different transformation approaches have been experimentally investigated. These approaches will be assessed facing two different GUI dialogs. As a result, we develop suggestions, how the positive effects of UIPs for the development of GUIs can be applied. Finally, influences resulting from the use of UIPs in the development process are discussed.

B. Structure of the Paper

In Section II, state of the art and related work are presented and assessed according to our objectives. The theoretical influences of UIPs on the development process for GUIs are elaborated in Section III. Subsequently, Section IV presents our two approaches for the transformation of formal UIPs into source code. The findings of Sections III and IV are summarized in Section V. Finally, our conclusions and future research options are presented in Section VI.

II. Related Work

A. *GUI Development Process and Model Transformations*

Abstract GUI development model. The specification and development of GUI systems remains a challenge. To discuss the activities and potentials of UIPs independently from specific software development processes and requirement models, we refer to a generic model concept. In reference [7], the common steps of a GUI development process are elaborated. To master the complexity that occurs when deriving GUI specifications from requirement models, Ludolph proposes four model layers and corresponding transformations built on each other. Three of them, being relevant in our context, are depicted in Figure 1.

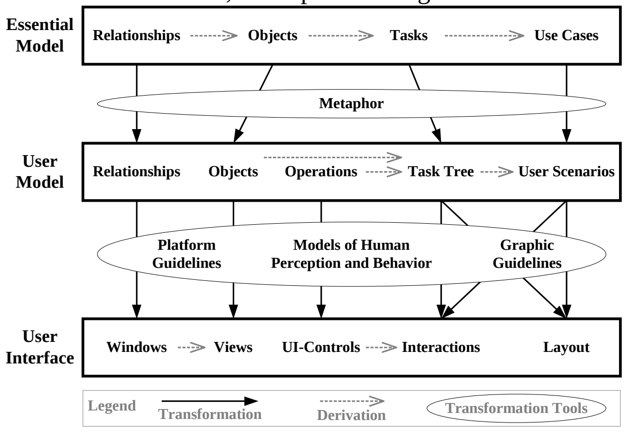


Figure 1. Model transformations in the GUI development process based on [7]

Essential model. By the e*ssential model*, all functional requirements and their structures are described. This information constitutes the core of the specification which is necessary for the development of the application kernel. Examples for respective artifacts are *use cases*, domain models and the specification of *tasks* or functional decompositions. These domain-specific requirements are abstracted from realization technology and thus from the GUI system [7]. Consequently, a GUI specification must be established to bridge the information gap between requirements and a GUI system.

User model. A first step in the direction of GUI specification is prepared by the *user model*. With this model the domain-specific information of the *essential model* is picked up and enhanced by so-called *metaphors*. They symbolize generic combinations of actions and suitable tools, which represent interactions with a GUI. Examples of *metaphors* would be indexes, catalogues, help wizards or table filters. The principal action performed by these examples is a search for objects, accompanied by the varying functionality embodied by the respective *metaphor*.

The *tasks* of the *essential model* have to be refined and structured in *task trees*. For each *task* of a certain refinement stage, *metaphors* are assigned, which will guide the GUI design for this part of the process. In the same manner, *use cases* can be supplemented with these new elements in their sequences to describe *user scenarios*.

User interface. This model is used for establishing the actual GUI specification. Through the three parts rough layout, interaction design and detailed design [7], the appearance and behavior of the GUI system are concretized. The aim is to set up a suitable mapping between the elements of the *user model* and *views*, *windows*, as well as *UI-Controls* of the *user interface*. For the *metaphors* chosen before, graphical representations are now to be developed. The *objects* to be displayed, their attributes and the relations between them are represented by *views*. Subsequently, the *views* are arranged in *windows* according to the activities of the *user scenarios*, or alternatively to the structure of the more detailed *task trees*. In these steps, there are often alternatives which are influenced by style guides or the used GUI library and especially by the provided *UI-Controls*. At the same time, generic interaction patterns are applied as transformation tools which also have an impact on the choice of *UI-Controls*.

Conclusion. Model transformations as stated by Ludolph show a detailed account of relevant model elements for the GUI specification. However, the occurring transformations are carried out manually. Besides that, no automation and only few options for reuse are mentioned.

B. *UIP Definition and Types*

Current research has been discussing patterns and especially User Interface Patterns (UIPs) for a longer period [8][9][6]. A UIP is defined as a universal and reusable solution for common interaction and visual structures of GUIs. UIPs are distinguished between two types:

Descriptive UIPs. Primarily, UIPs are provided by means of verbal and graphical descriptions. In this context, UIPs are commonly specified following a scheme similar to the one used for design patterns [10]. Reference [11] proposes a specialized language for UIPs and [6] shows its detailed sections. The verbal descriptions mainly serve for pure specification purposes and solely fulfill an informational function for the GUI developer. Being a guideline in this manner, they provide templates, points of variability and sketched examples for GUI elements. These UIPs named as descriptive UIPs [6] are informal. With their application, a developer receives aid when specifying a GUI, as he is able to express and hence operationalize usability requirements with UIPs. However, these informal patterns still have to be implemented manually.

UIP-Libraries. UIP libraries such as [12], [13] and [14] provide numerous examples for descriptive UIPs. Based on the presented categories, conceptions about possible UIP hierarchies and their collaborations can be imagined.

Formal UIPs. Rarely, generative UIPs [6] are presented. In contrast to descriptive UIPs, they feature a machinereadable form and are regarded as formal UIPs accordingly. Frequently, the formal format constitutes of a graphic notation, e.g., UML [8]. The formal UIPs are of great importance since they can be used within development environments, especially for automated transformations to certain GUI-implementations.

C. Formalization of UIPs

In order to permit the processing of descriptive UIPs, they have to be converted to formal UIPs. Possible means for this step can be provided by formal languages applied for specifying GUIs. These languages, however, have been designed for the specification of certain GUIs and were not intended for a pattern-based approach. Until now, there is no specialized language available for formalizing UIPs.

UsiXML and UIML. In our prior work, an extensive investigation on formal GUI specification languages and their applicability for UIPs was conducted. Intentionally the XML-based languages UsiXML [15] and UIML [16] were developed for specifying a GUI independently from technology and platform specifics. However, such languages may be applicable for UIPs since they offer elements like templates (UIML) and abstract as well as concrete models (UsiXML). Moreover, both have been developed further for a long period of time. Thus, the languages have reached a high maturity level.

IDEALXML. For efficient development environments tools are necessary that facilitate formal specifications of UIPs with regard to language definitions and rules. A widespread tool concept for UsiXML is presented with IDEALXML [6]. By using the various models defined by UsiXML, many aspects of a GUI and additionally the applied domain model of the application kernel are included in the specification. As a result, a detailed and comprehensive XML specification for the GUI is created. Many aspects of the *user model* from [7] are already included. However, it is not mentioned how UIPs are being expressed in models such as the „abstract user interface model“ (AUIM) [6] as reusable patterns or an hierarchy of these and consequently transformed to the „concrete user interface model“ (CUIM) [6].Furthermore, it has to be questioned, how a formal specification on the basis of UsiXML can be used for processing by code generators or other tools of a development environment.

D. GUI-Generators

Besides the formal specification of GUIs system concepts and frameworks exist which are able to generate complete GUI applications based on a partly specification of the application kernel. As representatives Naked Objects [17] and JANUS [18] can be mentioned. Both rely on an object-oriented domain model which has to be a part of the application kernel. Based on the information provided by this model, standard dialogs are being generated with appropriate *UI-Controls* for the respective *tasks*. For instance, in order to generate an *object* editor for entities like product or customer, certain text fields, lists or date pickers are selected as *UI-Controls* which match the domain data types of the selected domain *object* for editing.

In contrast to IDEALXML, which enables the extensive modeling of the GUI, GUI-generators may generate executable GUI code but they lack such a broad informational basis. Therefore, GUI-generators possess two essential weaknesses:

Limited functionality. The information for generating the GUI is restricted to a domain model and previously determined dialog templates along with their *UI-Controls*. Hence, their applicability is limited to operations and relations of single domain *objects*. When multiple and differing domain *objects* do play a role in complex *user scenarios* [7], the generators can no longer provide suitable dialogs for the GUI application. Moreover, extensive interaction flows require hierarchical decisions, which have to be realized, e. g., by using wizard dialogs. In this situation, GUI generators cannot be applied as well. The connection between dialogs and superordinate interaction design still has to be implemented manually.

Uniform visuals. A further weakness is related to the visual GUI design. Each dialog created by generators is based on the same template for the GUI-design. Solely the contents which are derived from the application kernel are variable. Both *layout* and possible *interactions* are fixed in order to permit the automatic generation. The uniformity and its corresponding usability have been criticized for Naked Objects [19]. Assuming the best case, the information for GUI design is founded on established UIPs and possesses their accepted usability for certain *tasks*. Nevertheless, the generated dialogs look very similar and there is no option to select or change the UIPs incorporated in the GUI design.

III. Influence of UIPs on GUI-Transformations

A. GUI Customization of Standard Software

On the basis of the customization of GUIs for standard software and the model transformations described in Section II.A the theoretical influences of UIPs are now considered.

EShop standard software fulfils the functional requirements of a multitude of users at the same time. Therefore, these systems share a well-defined *essential model* that specifies their functional range and has many commonalities along existing installations. Standard software implements the *essential model* through different components of the application kernel as shown in Figure 2. Each installation consists of a configuration for the application kernel which includes many already available and little custom components in most cases. In this context, the *User Interface* acts as a compositional layer that combines *Core* and *Custom Services* together with suitable dialogs for the user.

Individual GUIs for eShops. Concerning eShops, the visual design of the GUI is of special relevance since the user interface is defined as a major product feature that differentiates the competitors on the market. Hence, the needs of customers and users are vitally important in order to provide them with the suitable individual dialogs. In this regard, the proportions of components related to the whole system are symbolized by their size in Figure 2. In comparison to the *Custom Components* of the application kernel the *Custom Dialogs* represent the greater part of the *User Interface* and the customization accordingly. Along with the customization of the application kernel there is a high demand for an easy and vast adaptability of the GUI.

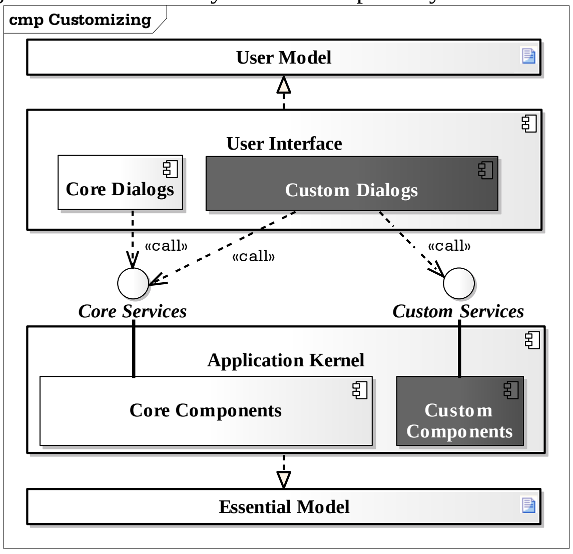


Figure 2. Components for the customization of standard software

Moreover, the customization of the GUI system is needed, as elements of the *essential model* tend to be very specific after extensive customization or maintenance processes. Thus, the standard *user model* as well as the *user interface* can no longer be used for the customized services. In this case, models have to be developed from scratch and after this, a suitable solution for the GUI has to be implemented.

Usability. The development of GUIs is caught in a field of tension between an efficient design and an easy but extensive customization. High budgets for the emerging efforts have to be planned. Additional efforts are needed for important non-functional requirements such as high usability and uniformity in interaction concepts and an eased learning curve during the customization process of GUIs. For realizing these requirements, extensive style guides and corresponding *user interface* models often need to be developed prior to the manual adaption of the GUI. These specifications will quickly lose their validity as soon as the GUI-framework and essential functions of the application kernel change.

B. *Model Aspects of UIPs*

With the aid of UIPs the time-consuming process of GUI development and customizing can be increased in efficiency. To prove this statement, the influences of UIPs on the common model transformations from Section II.A are examined in the next step. In Section III.C potentials for improvements are derived from these influences.

Metaphors and UIPs. *Metaphors* act as the sole transformation tool between *essential model* and *user model*. Since they lack visual appearances as well as concrete interactions, the mapping of *metaphors* to the elements of the *essential model* is very demanding. *Metaphors* will not be visualized by GUI sketches prior to the transformation of the *user model*.

Since UIPs are defined more extensively and concrete, they can be applied as a transformation tool instead of using *metaphors*. Descriptive UIPs feature a pattern-like description scheme that is provided in the catalogues in [12] and [13], for example. Thus, they offer much more information as well as assessments which can inspire the GUI specification. In addition, descriptive UIPs do already possess visual designs that may be exemplary, or in the worst-case, abstract.

With the *user model*, operations on *objects* have to be specified. The *metaphors* do not provide enough hints for this step. In contrast, UIPs are definitely clearer concerning these operations because they group *UI-Controls* according to their *tasks* and do operationalize them in this way. Interaction designs and appropriate visuals are presented along with UIPs. These aspects would have to be defined by oneself using only the *metaphor*.

When UIPs are used in place of *metaphors* for formalization, these new entities can be integrated in the tools for specifications. Concerning UsiXML, UIPs could describe the AUIM. *Task-Trees* are already present in UsiXML, so this concept of specification partly follows the modeling concepts in [7] and thus may be generically applicable.

User model and UIPs. With regard to the *user model*, the numerous modeling steps no longer need to be performed with the introduction of UIPs. Instead, it is sufficient to derive the *tasks* from the *use cases* within the *essential model* and allocate UIPs for these. Detailed *task-trees* no longer have to be created since UIPs already contain these operations within their interaction design. Interactions can already be specified in formal UIPs, and later this information can directly be used for parts of the presentation control of *views* or *windows*. As a result, an extensive *user scenario* also is obsolete, as it was originally needed for deriving the more detailed *task-tree*. Now it is sufficient to lay emphasis on expressing the features of UIPs and their connection to the *tasks* defined by the *essential model*. The *objects* are also represented within the UIPs in an abstract way. With the aid of placeholders for certain domain data types adaptable *views* for *object* data can already be prepared in formal UIPs. Finally, much of the afore-mentioned information of the *user model* now will be explicitly or implicitly provided by completely specified UIPs.

User interface and UIPs. UIPs provide the following information for the *user interface*: *Layout* and *interaction* of the GUI will be described by a composition of a hierarchy of UIPs that is settled on the level of *views* and *windows*. When creating the UIP-hierarchy, a prior categorization is helpful which features the distinction between *relationship*, *object* and *task* related UIPs. This eases the mapping to the corresponding model entities.

For *interactions*, the originally applied *Models of Human Perception and Behavior* from Figure 1 are no longer explicitly needed since they are implicitly incorporated in the interaction designs of the UIPs. In this context, suitable types of *UI-Controls* are already determined by UIPs. Nevertheless, a complete and concrete GUI-design will not be provided by UIPs since the number and contents of *UI-Controls* depend on the context and have to be specified by the developer with parameters accordingly. In the same way *Platform* and *Graphic Guidelines* act as essential policies to adapt the UIPs to the available GUI-framework and its available *UI-Controls*.

Conclusion. We explained that UIPs might cover most parts of the *user model* as well as numerous aspects of the *user interface*. By using UIPs in the modeling process, these specification contents can be compiled based on the respective context without actually performing the two transformations from Figure 1 explicitly. Basically, the transformation to the target platform remains as depicted in Figure 3.

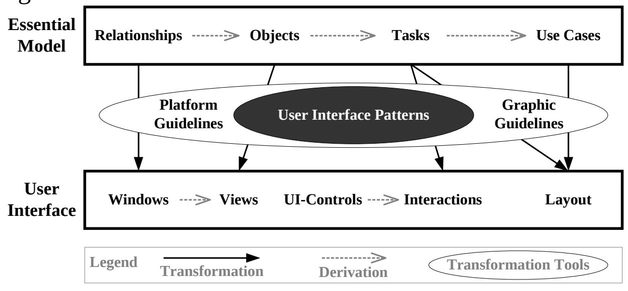


Figure 3. GUI transformations with the aid of UIPs and automation

C. Potentials of UIPs for Improvements

In this section, the potentials of UIPs related to the GUI development process are summarized from a theoretical perspective. The implications resulting from the application of UIPs in experimental transformations are presented in Section IV.

Reuse. By means of UIPs the transformational gap between *essential model* and *user interface* can be bridged more easily since reuse will be enhanced significantly. Thereby UIPs are not the starting point of model transformations; they rather serve as a medium for conducting needed information for the transformations. The information originally included in the *user model* and parts of the *user interface* are now extracted from the selection and composition of UIPs.

Layout and *interaction* of *windows* as well as the interaction paradigm of many parts of the GUI can be determined by a single UIP configuration on a high level in hierarchy. This superordinate GUI design can be inherited by a number of single dialogs without the need for deciding about these aspects for each dialog in particular.

Many interaction designs can be derived from initial thoughts about GUI design for the most important *use cases* and their corresponding *tasks*. When a first UIP configuration has been created, the realization of the *Graphic* and *Platform Guidelines* therein can be adopted for other UIP-applications since the target platform is the same for each dialog of a system. Especially when *user scenarios* overlap, meaning they partly use the same *views* or *windows* as well as *object* data, UIPs enable a high grade of reuse. UIP assignments, already established for other *tasks,* can be reused with the appropriate changes. eShops tend to use many application components together although they offer them by different dialogs as illustrated in Figure 2 UIPs can contribute to a higher level of reuse in this context. Depending on the possible mapping between application kernel components and UIP-hierarchy, new dialogs can be formed by combining the views of certain services which are determined by their assigned UIPs.

Reuse and usability. Besides reuse, UIPs assure multiple non-functional requirements. As proven solutions for GUI designs their essential function is to enable a high usability by the application of best-practices. In this context, they facilitate the adherence of style guides by means of their hierarchical composition.

Technically independent essential model. It is a common goal to keep elements of the *essential model* free or abstract from technical issues. Following this way, the *essential model* has no reference to the GUI specification. Therefore, it is not subject to changes related to new requirements which the user may incorporate for the GUI during the lifecycle of the system. User preferences often tend to change in terms of the visuals and interactions of the GUI. Concerning *use cases*, this rule is elaborated in [20] and [21]. Technical aspects and in particular the GUI specification are addressed in separate models such as *user model* and *user interface* according to [7]. After changes, these models have to be kept consistent what results in high efforts. For instance, a new or modified step within a *use case* scenario has to be considered in the corresponding *user scenario*, too.

By assigning UIPs to elements of the *essential model*, explicit *user models* and the prior checking of consistency between these models both become obsolete. Instead, *user models* will be created dynamically as well as implicitly by an actual configuration of UIPs and *essential model* mapping. A technical transformation to the source code of the GUI that relies on the concrete appearances of the UIPs remains as shown in Figure 3. By modeling assignments between UIP and *task* or between UIP and *object*, the number of *UI-Controls*, the hierarchy and *layout* of UIPs, sufficient structured information on the GUI system is provided. Subsequently, a generator will be able to compile the GUI suited for the chosen target platform. These theoretical influences enable an increased independence from the technical infrastructure since the generator can be supplied with an appropriate configuration to instantiate the UIPs compatible to the target platform and its specifics.

Modular structuring of windows and views. Common to software patterns, UIPs reside on different model hierarchies. Dialog navigation, frame and detailed *layout* of a dialog can be characterized by separate UIPs. The *views* of a *window* can be structured by different UIPs on varying hierarchy levels. In this way, a modular structure of dialogs is enabled. In addition, versatile combinations, adaptability and extensibility of building blocks of a GUI will be promoted.

IV. Experimental Application of UIPs in GUI-Model-Transformations

Up to now there have been no reports about experiences in the practical application of formal UIPs. The particular steps to be performed for a model-to-code-transformation and the shape as well as the outline of a formalization of UIPs have to be examined in detail. In order to gain further insights about UIPs, they have been experimentally applied by two different prototypes. Similar to the probing of software patterns, selected UIPs were instantiated for simple example dialogs. These are illustrated in Figure 4.

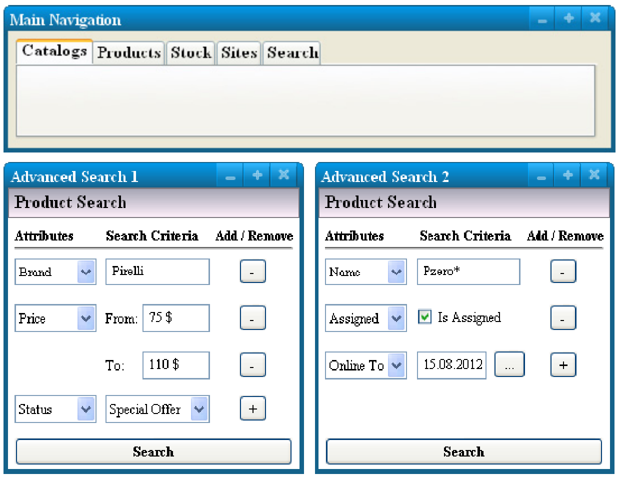


Figure 4. Example dialogs used for prototypes

On the one hand, the examples consisted of a *view* fixed in shape that contained the UIP „Main Navigation“ [12] on the upper part. On the other hand, the lower part shows two variants for a *view* whose visuals are dependent on the input of the user. Thereby, the UIP „Advanced Search“ [12] was applied. This UIP demands for a complex presentation control and is characteristic for eShops. Depending on the choice of the user, the *view* and *interactions* are altered. The search criteria can be changed, deleted and added as depicted in Figure 4 by two variants. Both example dialogs should have been realized by formalized UIPs and one prototype.

A. Generation at Design Time

Scope. Firstly, generating code for the GUI based on previously specified UIPs was probed. In general, the possibility to generate an executable GUI with the aid of UIPs had to be proven. The UIPs had to be completely defined at design time. Testing of the prototype had to be conducted after the GUI system was fully generated.

Approach. Foremost, the simple UIP *Main Navigation* was realized. This informally specified UIP was formalized after a language for formalization had been chosen. By means of a self developed generator, a model-to-code-transformation was performed to create an executable dialog. Subsequently, the complete GUI system was started without any manual adaptions to the code.

Choice of formalization language. A comparative study of UIML and UsiXML was conducted.

Regarding the structure of a GUI-specification, UsiXML proposes numerous models in order to separate the different information concerns *domain objects*, *tasks* and *user interface*. Not all the models were mandatory in terms of the example because no explicit *essential model* was given. On the contrary, UIML operates with few sections within one XML-document. This is because the UIML format was easier to handle and learn with respect to the simple example.

According to *UI-Controls*, UsiXML defines precisely which types of *UI-Controls* are available and what properties they can possess. An additional mapping model would have to be created in order to assign these elements to the entities of the target platform. In contrast, with UIML and its peer-section this mapping can easily be specified. The mapping to the GUI-framework can be altered afterwards without the need for changing the already defined UIPs. Moreover, UIML offers a more flexible definition of *UI-Controls* since custom *UI-Controls* can be declared in the structure-section as well as their properties in the style-section [22]. In addition, UIML provides templates for integration and reuse of already defined UIPs in other UIP formalizations.

Concerning *layout*, UsiXML uses special language elements to set up a GridBagLayout. UIML offers two variants: Firstly, it is possible to use containers as structuring elements along with their properties. The containers have information attached that governs the arrangement of their constituent parts. Secondly, UIML provides special tags that are committed for layout definition. UIML has a more flexible solution by defining *layouts* with containers that can be nested arbitrarily.

Related to behavior, both languages define own constructs. Nevertheless, complex behavior is difficult to master without clear guidelines for both. Concerning the examples, the behavior was limited to the technical presentation control within a *view*.

Choice of UIML. We decided to apply UIML for the example dialogs. Firstly, UIML is more compact in structure and enables a higher flexibility for shaping the formalization. Secondly, many of the language elements and models from UsiXML were not actually needed for the UIP „Main Navigation“. Thirdly, even the „Advanced Search“ example could not profit from the vast language range of UsiXML since all possible variants for search criteria could not have been formalized. At least UIML offered the possibility to rely on templates in order to define all possible lines of search criteria composed of simple UIPs. UsiXML turned out to be too complex for these simple UIPs. In addition, it was not clear whether UsiXML permits the reuse of already specified UIPs.

Realization of „Main Navigation“. Java Swing was chosen as target platform. For the peer-section we decided to map the elements of „Main Navigation“ to horizontal JButtons instead of tabs. In the formalization the mandatory parameters for number, order and naming of *UI-Controls* were specified. As result, the UIP was described concretely. The architecture was structured following the MVC-pattern [1]. The sections of UIML were assigned to components like in Figure 5.

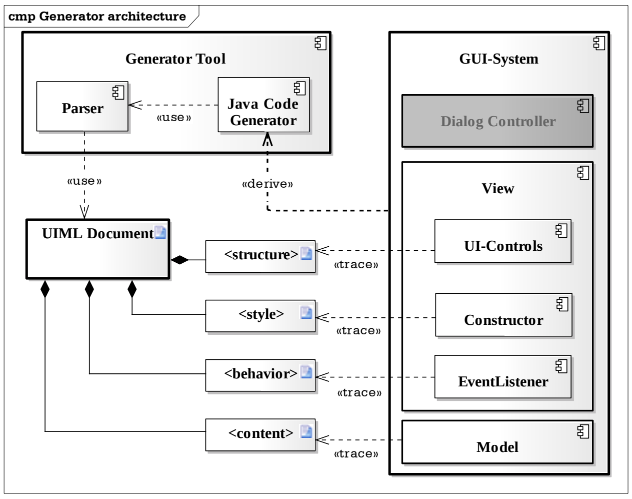


Figure 5. Architecture applied for code generation

Structure and style were processed within the object declarations (*UI-Controls*) of the *View* and its constructor. Based on the behavior-section, *EventListeners* were generated acting as presentation controllers. For the *Model* the content-section was assigned. Hence, the UIP "Main Navigation" formalized with UIML was transformed to source code.

Realization of „Advanced Search". Even by using the UIML templates, this complex dialog could not be realized by a generation at design time. It was not possible to instantiate the formalized UIPs that were depending on the choice of attributes at runtime.

Results. The prototype primarily was intended to prove feasibility. This is because we chose a simple architecture and did not incorporate a *Dialog Controller* for controlling the flow of dialogs. The control was restricted to the scope of the *UI-Controls* of the respective UIP. Thus, the behavior only covered simple actions like the deactivation of *UI-Controls* or changing the text of a label. Complex decisions during the interaction process like the further processing of input data and the navigation control amongst dialogs could not be implemented. A corresponding superordinate control could have been realized through a UIP-hierarchy in combination with appropriate guidelines for the formalization of control information. Despite the simplicity of the prototype, the following insights could be gathered:

Informal UIPs could be converted to formal UIPs by using UIML as a formal language. There was the need to define certain guidelines for this initial step. The *layout* of the example was specified by using containers for the main *window* and their properties. As a result, the *UI-Controls* were arranged according to these presets. Nested containers and complex *layouts* have not yet been used for the experiment in this way. The style also was described concretely within the UIML-document as well as the number and order of *UI-Controls*. The mapping of a formal UIP to a software pattern was simply performed by the scheme in Figure 5.

Concerning the example *Advanced Search*, only fixed variants or a default choice of criteria could have been formalized. The generator could have created static GUIs accordingly without realizing the actual dynamics of this particular UIP.

B. Generation at Runtime

Scope. The dynamic dialog *Advanced Search* could not be realized by the first approach. Thus, a solution had to be found that enables the instantiation of UIPs at runtime. Thereby, it was of importance to keep the platform independency of the UIML specification. The formal UIPs had to be processed directly during runtime without binding them to a certain GUI-framework.

Approach. Since the *Advanced Search* UIP was very versatile and could not be formalized with all its variants, the *layout* of the dialogs was fragmented. By the means of a superordinate UIP the framing *layout* of the *view* was specified in a fixed manner at design time. In detail, the headline, labels and the three-column structure of the *view* appropriate to a table with the rows of search criteria were defined.

The mandatory but unknown parameters that determine the current choice of criteria and UIPs had to be processed at runtime. Accordingly, a software pattern had to be chosen that is able to instantiate UIP representations along with their behavior. This pattern had to act similarly to the builder design pattern [10] which enables the creation and configuration of complex aggregates. In [23] a suitable software pattern was described which is explained shortly and illustrated in Figure 6:

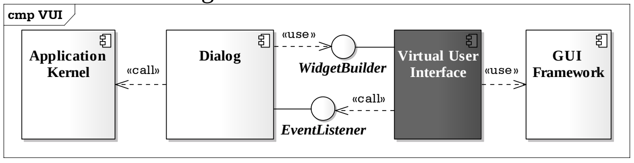


Figure 6. Virtual user interface architecture derived from [23]

Quasar VUI. The Virtual User Interface (VUI) of Quasar (quality software architecture) follows the intention of programming dialogs in a generic way. This means that the dialog and its events are implemented via the technical independent, abstract interfaces *WidgetBuilder* and *EventListener* rather than using certain interfaces and objects of a GUI-framework directly. By means of this concept, the GUI-framework is interchangeable without affecting existing dialog implementations. Solely the component *Virtual User Interface (VUI)* depends on technological changes. Upon such changes, its interfaces would have to be re-implemented. By using the interface *WidgetBuilder*, a dialog dynamically can adapt its *view* at runtime. For instance, the *Dialog* delegates the *VUI* to create and configure a new *window* containing certain *UI-Controls*. The *VUI* notifies the *Dialog* via the interface *EventListener* when events have been induced by *UI-Controls*. Both interfaces have to be standardized for a GUI system of a certain domain in order to enable the reuse of reoccurring functionality such as the building of *views* and association of *UI-Controls* with events without regarding the certain technology or platform specifics being used.

VUI for UIPs. The concept, the *VUI* is based on, can be adapted to the requirements of the UIP *Advanced Search*. The idea is to instantiate complete *view* components with UIP definitions besides simple *UI-Controls*. The *Dialog* is implemented by using generic interfaces which enable the instantiation of UIPs, changing their *layout* and their association with events. In Figure 7 our refinement of the original *VUI* is presented.

The *VUI* for UIPs is based on our previously described generator solution. Each possible variation of *UI-Controls* matching the attributes of the domain *objects* for *Advanced Search* has been formalized before. Hence, the rows of the dialog were visualized by different UIPs. Concerning the formal UIPs, the proper implementations for the chosen GUI-framework were generated as stated in Section IV.A. The previously mentioned generator was integrated in the component *UIP Implementations*. These implementations of UIPs located within *VUI* are based on the interfaces and objects of the GUI-framework. In analogy to the *UI-Controls* already implemented in the GUI-framework, the available UIP instances were provided via the interface *UIPBuilder* and could be positioned with certain parameters.

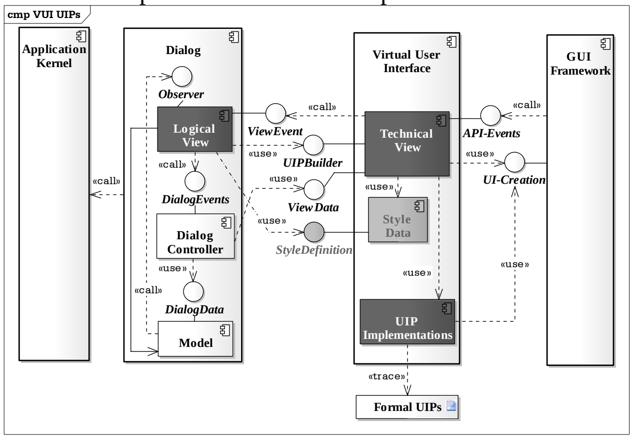


Figure 7. Virtual user interface architecture for UIPs

The *VUI* builds the *view* or a complete *window* as requested by the *Logical View*. Furthermore, the *VUI* provides information about the current composition and the *layout* of the *Dialog*. This information can be used by the *Logical View* for parameters to adapt the current *view* by delegating the *VUI* respectively. The *Dialog* coordinates the structuring of the *view* with the component *Logical View* and implements the application specific control in the *Dialog Controller* as well as dialog data in the *Model*.

Initially, events are reported to the *VUI* via *API-Events*. The *VUI* only forwards relevant events to the *Logical View*. When the respective event is only related to properties of a *UI-Control* or a UIP instance, it is directly processed by the *Logical View* which delegates the *VUI* when necessary. If the *Logical View* cannot process the particular event on its own, it will be forwarded to the *Dialog Controller*. For instance, this occurs when the user presses the button *Search* and a new *view* with the search results has to be loaded. The *Dialog Controller* collects the search criteria via the interface *ViewData* and sends an appropriate query to the *Application Kernel*. The result of the query will be stored as dialog data in the *Model*.

Results. For realizing *Advanced Search* with UIPs, a complex architecture had to be invented. Details like the connection of UIP instances to the *Dialog* data model as well as the automation potentials of the *Dialog Controller* could not be investigated extensively, yet.

The UIPs had to be specified in a concrete manner like in Section IV.A. The prototype was not mature enough to handle abstract UIP specifications. The style of the *UI-Controls* was also described concretely, so the control of style by a component of the *VUI*, as depicted in Figure 7, has not yet been realized.

Through the *VUI*, the versatile combinations of *Advanced Search* could be realized according to the example at runtime. The VUI constitutes of a component-oriented structure related to the software categories of Quasar [24]. Accordingly, it possesses its virtues like the division of application and technology, separation of concerns and encapsulation by interfaces. Despite its challenging complexity, a flexible and maintainable architecture for dynamic GUI systems has been created.

V. DISCUSSION

The theoretical reflection of the influence UIPs have on GUI transformations and the results of our experimental prototypes led us to the following findings.

A. *Formalization of UIPs*

Reflection of results. By experimentally probing the model-to-code-transformation of formal UIPs, we came to the conclusion that the generation of a GUI is not the complicated part of the process. Instead, the formalization and the occurring options in this step lead to the main problem. Primarily, the preconditions to benefit from the positive influences of the UIPs on the GUI development process have to be established by the formalization:

The generator solution was well suited for stereotype and statically defined UIML contents. In this context, *layout*, number and order as well as style of UIPs have been specified concretely. This led us to a static solution that can be applied at design time. But the UIP *Advanced Search* could not be realized by following this approach.

Parameters for UIPs. In order to overcome this static solution, a parameterization of formal UIPs has to be considered. Via parameters the number, order, ID, *layout* and style of *UI-Controls* within UIPs specifications have to be determined to provide a more flexible solution. Especially the number and order of *UI-Controls* have to be abstractly specified in the first place. In this way UIPs will be kept applicable for varying contexts. In place of a concrete declaration of style for each UIP, a global style template has to be kept in mind. By using this template, dialogs could be created with uniform visuals and deviations are avoided. For this purpose, the *VUI* incorporated the *Style Data* component. It is intended to configure the visuals of UIP instances and *UI-Controls* globally. The configuration is used for the instantiation of these entities by the *Technical View*. Consequently, style information from single UIP specifications could be avoided and the UIPs would receive a more universal format.

B. *Generation at Design Time*

In principle, complex UIPs or UIP-hierarchies can be realized with the generation at design time. The easiest cases are elementary or invariant UIPs like calendar, fixed forms or message windows. These examples can be generated with ease since they do not need parameters besides a data model. For UIPs, which require parameters such as hierarchical UIP structures, an additional transformation is needed prior to the generation of source code:

Transformation of abstract UIPs. Firstly, the UIP is abstractly specified along with all parameter declarations needed and placeholders for nested UIPs. Subsequently, these parameters have to be specified via a context model which adapts the UIP to a certain application. Based on the abstract UIP specification and the context model, a model-to-model-transformation is performed in order to generate concrete UIP specifications like they were used in our examples. In this state all required information is available for the generation of the GUI system. The described model-to-code-transformation can be performed as a follow-up step. It has to be examined whether a suitable format is given to realize this approach, by means of UsiXML or IDEALXML and their models *AUIM* and *CUIM*.

C. *Generation at Runtime*

Regarding the UIP *Advanced Search*, it is clear that a large gap has to be bridged between the *essential model* and the *user interface*. A *use case* which demands for such dynamic UIPs hides a whole variety of different GUI-designs. Consequently, one static *user interface* cannot always be established for the elements of the *essential model*. However, even for these dynamic GUIs UIPs can serve as media to enable reuse of numerous aspects directly by generation along with a composition at runtime. The combined application of both our approaches can provide a feasible solution. Concerning the example from Figure 4, the previously generated *layouts* actually were reused for the *Advanced Search window* and the *views* of search criteria. By instantiation of matching UIPs, even the interactions respectively the presentation control was reused as well.

Generation of dialogs. As shown with our example, the current VUI is capable of the instantiation and composition of single parts of a certain *Logical View*. The generation of complete *Logical Views* on the basis of formal UIPs and their hierarchy could possibly be realized with the VUI architecture. The model describing the *Logical View* has to refer to the standardized interfaces of the VUI and a common UIP catalog. To formally specify the UIPs to be used in this environment, only UIML currently seems to be suitable. Firstly, an analysis of the required and reused elementary UIPs as well as the relevant *UI-Controls* has to be conducted in order to populate the basic level in the hierarchy of UIPs. Next, these UIPs have to be formalized with UIML along with their required data types and invariant behavior that acts as a basis for presentation control within the *VUI*. Furthermore, the interaction and *layout* within the *Logical View* have to be specified using UIML as well. This is because UIML already offers templates that can be parameterized and thus used for the composition of several UIP-documents into one master document establishing a UIP of higher level. Concerning UsiXML, one dialog can only be specified by a single AUIM respective CUIM document.

To complete the *Dialog*, meaning *Dialog Controller* and *Model*, relevant information on *tasks* and data *objects* has to be incorporated into a formal model. The research on the collaboration between adaptable UIPs and these logical aspects has just begun.

D. *Limitations through the Application of UIPs*

Individualization. Using UIPs instead of time-consuming manual transformations, a compromise is being contracted: A full individualization of the GUI is not possible with UIPs since the customizing is conducted within the limits of available and formalized UIPs. The UIPs can embody a further building block of standard software. Customization will be facilitated by defined parameters and automation.

Metamodels. The application of UIPs demands for clear guidelines for modeling of the *essential model* which result in a second limitation. The rules for this model need to define stereotype element types and their delimitations. The definition of the *essential model* is governed by a metamodel in the best case. Based on the metamodel, the elements can be defined uniformly and as stereotypes. For instance, it will be defined what types and refinements of *tasks*, *domain objects* and domain data types do exist in order to assign them homogenously to certain UIP categories. This concept is essential for the proposal of suitable UIPs for the partly automated development of GUI systems. The proposing system needs to work in two ways: On the one hand, the GUI developer asks for a suitable selection of UIPs for a certain part of the *essential model* at design time. On the other hand, users need to be provided with suitable UIPs in dynamic dialogs at runtime based on their current inputs. The mechanisms can only work if a uniform *essential model* with clear defined abstractions derived from fixed guidelines is available as fundamental information.

VI. CONCLUSION AND FUTURE WORK

A. *Conclusion*

We theoretically and experimentally elaborated that UIPs do have numerous positive influences on the GUI development process. UIPs integrate well in the common GUI transformations. Therefore, our findings are not restricted to the domain of eShops but rather can be adapted to other standard software such as enterprise resource planning systems. Even for individual software systems, UIPs can be of interest in case that numerous GUI aspects are similar and their reuse appears reasonable.

Currently, adaptability and reuse of UIPs is limited to their invariant formalizations. UIP compositions could only be created by manual implementation. We pointed to the limitations of current UIP specification format options and presented architectural solutions for their practical application. Above all, the upstream transformation of the abstract UIP description into UsiXML or UIML is worth to be considered since one could use their strength in concretely specifying user interfaces. Afterwards, the generation of GUIs based on this information would pose a minor issue.

B. *Future Work*

Formalization. For future work, we primarily see the research in formalizing UIPs. An important goal is to enable UIPs to act as real patterns that are adaptable to various contexts. The synthesis of a UIP-description model is the next step to determine properties and parameters of UIPs exactly and independently from GUI specification languages. Consequently, it can be more accurately assessed whether UIML or UsiXML are able to express the description model and thus UIPs completely. The independence from the platform can be achieved by both languages. However, it was not possible to specify context independent UIPs besides invariant or concrete UIPs. In this regard, the composition of UIPs, to form structured and modular specifications, remains unsolved, too.

Paradigm. Another open issue exists in the field of interaction paradigms [7] and the applicability of UIPs. With respect to the procedural paradigm, processes are defined which exactly define the single steps of a *use case* scenario. To provide a matching *user interface* for this case, additional information needs to be included in the formalization of UIPs. For instance, the process or *task* structures have to be specified by UIPs on a high level of hierarchy. These UIPs possess little visual content, maybe a framing *layout* for *windows*, and mainly act as entities for controlling the application flow. The *Dialog Controller* from Figure 5 and Figure 7 could be based on such a UIP. In this paper, no information for these components was integrated in the formal UIPs. So these components had to be implemented manually. For example, the *Dialog Controller* opens a new *window* with search results for the *Advanced Search,* controls the further navigation and delegates the structuring of the next or previous *windows*. In this context, our *VUI* solution is a compromise between automation and the reuse of elementary and invariant UIPs through manual configuration of the *Dialog Controller* and the delegated *Logical View*. A full automation needs further research.

References

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stahl, A System of Patterns, New York: Wiley, 1996.

[2] M. Fowler. Patterns of Enterprise Application Architecture, Addison-Wesley, Boston, 2003.

[3] M. Haft, B. Olleck, “Komponentenbasierte Client-Architektur,” in Informatik Spektrum, 30(3), 2007, pp. 143-158, doi: 10.1007/s00287-007-0153-9

[4] M. van Welie, G. C. van der Veer, A. Eliëns, “Patterns as Tools for User Interface Design,”. in Tools for Working with Guidelines, Springer, London, Eds.: Ch. Farenc, J. Vanderdonckt, 2000, pp. 313-324.

[5] M. J. Mahemoff, L. J. Johnston, “Pattern languages for usability: an investigation of alternative approaches,” Proc. Computer Human Interaction, pp.25-30, 15-17 July 1998, doi: 10.1109/APCHI.1998.704138

[6] J. Vanderdonckt and F.M. Simarro, “Generative pattern-based Design of User Interfaces,” Proc. 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems (PEICS '10), ACM, June 2012, pp. 12-19, doi: 10.1145/1824749.1824753.

[7] M. Ludolph, “Model-based User Interface Design: Successive Transformations of a Task/Object Model,” in User Interface Design: Bridging the Gap from User Requirements to Design, CRC Press, Boca Raton, Ed.: L.E. Wood, 1998, pp. 81-108.

[8] N. J. Nunes, “Representing User-Interface Patterns in UML,” in International Conference on Object-Oriented Information Systems (OOIS 2003), LNCS 2817, D. Konstantas, M. Léonard, Y. Pigneur, S. Patel, Eds. Heidelberg: Springer, 2003, pp. 142–151, doi: 10.1007/978-3-540-45242-3_14.

[9] A. Dearden and J. Finlay, “Pattern Languages in HCI; A critical Review,” Human-Computer Interaction, 21, 2006.

[10] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software, Reading: Addison-Wesley, 1995.

[11] S. Fincher, J. Finlay, S. Greene, L. Jones, P. Matchen, J. Thomas, and P. J. Molina, “Perspectives on HCI Patterns: Concepts and Tools (Introducing PLML),” Ext. Proc. Computer-Human Interaction (CHI’2003). Workshop Report, ACM Press, 2003, pp. 1044–1045.

[12] M. van Welie, “A pattern library for interaction design,” http://www.welie.com 10.05.2012.

[13] Open UI Pattern Library, http://www.patternry.com 10.05.2012.

[14] A. Toxboe, “User Interface Design Pattern Library,” http://www.ui-patterns.com 10.05.2012.

[15] J. Vanderdonckt, Q. Limbourg, B. Michotte, L. Bouillon, D. Trevisan, and M. Florins, “UsiXML: a User Interface Description Language for Specifying multimodal User Interfaces,” Proc. W3C Workshop on Multimodal Interaction (WMI'2004), 19-20 July 2004.

[16] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster, “UIML: An Appliance-Independent XML User Interface Language,” Proc. Eighth International World Wide Web Conference (WWW’8), Elsevier Science Pub., May 1999.

[17] R. Pawson and R. Matthews, Naked Objects, Chichester: John Wiley & Sons, 2002.

[18] H. Balzert, “From OOA to GUIs: The Janus system,” IEEE Software, 8(9), February 1996, pp. 43-47.

[19] L. Constantine, “The Emperor Has No Clothes: Naked Objects Meet the Interface”, http://www.foruse.com/articles 10.05.2012.

[20] D. Kulak, E. Guiney, Use Cases: Requirements in Context, New York: Addison-Wesley, ACM Press, 2000.

[21] K. Bittner, I. Spence, Use Case Modeling, New York: Addison-Wesley, 2003

[22] UIML 4.0 specification, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uiml 10.05.2012.

[23] E. Denert, J. Siedersleben, „Wie baut man Informations-systeme? Überlegungen zur Standardarchitektur,“. in Informatik Spektrum, 23(4), 2000, pp. 247-257

[24] J. Siedersleben, Moderne Softwarearchitektur - Umsichtig planen, robust bauen mit Quasar, 1st ed. 2004, corrected reprint, Heidelberg: dpunkt, 2006