An Approach for Pattern Mining through Grounded Theory Techniques and its Applications to Process-Driven SOA Patterns

CARSTEN HENTRICH, PricewaterhouseCoopers AG WPG
UWE ZDUN, University of Vienna
VLATKA HLUPIC, University of Westminster
FEFIE DOTSIKA, University of Westminster

Pattern mining is a term used in the pattern community to describe the process of identifying or discovering patterns. To identify new patterns, usually an informal or ad hoc process of finding patterns (e.g., in existing software systems) is used. This paper reflects on lessons learned and methods used regarding the integration of software pattern mining with qualitative research methods during our work on a pattern language for process-driven and service-oriented architectures (SOAs). This pattern language aims at closing the conceptual gap between business architecture and software architecture with a focus on process-driven solutions. In this paper we emphasize the notion of understanding patterns as sociological phenomena of problem solving behavior. We further introduce a systematic approach for pattern mining based Glaserian Grounded Theory techniques. This approach has been applied for mining the pattern language for process-driven SOAs. This work may also contribute to a better empirical grounding of software pattern mining. We will illustrate our approach using the pattern language for process-driven SOAs as a pattern mining case study.

1 INTRODUCTION

A software pattern is a technology independent conceptual solution to a generic software design problem (Buschmann et al., 2007; Gamma et al., 1994). The basic idea of patterns is that problems arise due to conflicting forces in a given context, and solutions resolve these conflicting forces (Alexander, 1977). *Pattern mining* is a term used in the pattern community to describe the process of identifying or discovering patterns in existing software systems. That is, as patterns describe established knowledge rather than original solutions, each software pattern is associated with a number of known uses where the pattern is used in an existing software system (Coplien, 1996). To identify new patterns, usually an informal or ad hoc process of finding patterns in software systems is used by the pattern author. In particular, often the pattern author identifies one or more patterns from his own experiences, and then broadens the scope by searching for the identified patterns in other contexts (e.g., other related software systems) and looking for related other patterns in the contexts or systems under consideration.

Because patterns are sociological phenomena of human problem solving behavior, we introduce a pattern mining approach based on Glaserian Grounded Theory (Glaser, 1992). Grounded Theory is a systematic scientific method for the discovery of theory through the analysis of data, originally coming from the social sciences. It is mainly used in qualitative research, but is also applicable for quantitative data. We discuss how Grounded Theory can help to systematize the process of software pattern mining and base it on (qualitative) empirical evidence. In addition, we have applied the typical community-based review process of the pattern community for pattern validation.

In this paper, we use our process-driven SOA pattern language as a pattern mining and validation case study. This paper reflects on lessons learned and methods used regarding the integration of software pattern mining with qualitative research methods during our work on that pattern language. This pattern language aims at closing the conceptual gap between business architecture and software architecture

Author's address: Carsten Hentrich, PricewaterhouseCoopers AG WPG, Friedrich-Ebert-Anlage 35-37, 60327 Frankfurt am Main, Germany, carsten.hentrich@de.pwc.com. Uwe Zdun, Software Architecture Group, University of Vienna, Vienna, Austria, uwe.zdun@univie.ac.at. Vlatka Hlupic, University of Westminster, Westminster Business School, 35 Marylebone Road, London NW1 5LS, United Kingdom, hlupicv@wmin.ac.uk. Fefie Dotsika, University of Westminster, Westminster Business School, 35 Marylebone Road, London NW1 5LS, United Kingdom, f.e.dotsika@westminster.ac.uk.

model building following business process-driven and service-oriented principles (see Hentrich and Zdun, 2012). During the work on this pattern language we followed an inductive empirical approach based on software patterns grounded in empirical data to understand how organizational and software architecture structures can be integrated and how this integration enables organizational flexibility of information systems (IS).

This paper is structured as follows: In Section 2 we first introduce the pattern language for process-driven SOAs as a case study for pattern mining. In Section 3 we describe background on Grounded Theory. Next, in Section 4 we introduce our approach for pattern mining based on Grounded Theory, and in Section 5 we present the details of the approach. Section 6 provides a pattern mining example including an extract from the interviews we conducted and an example for the Grounded Theory based interview analysis. In Section 7, we revisit the case study and discuss the results mined from the empirical data in terms of patterns. Finally, in Section 8 we conclude.

2 PATTERN MINING CASE SYNOPSIS: PROCESS-DRIVEN SOA PATTERN LANGUAGE

In our previous work, we have performed empirical research to investigate software patterns that integrate business processes and Service-Oriented Architectures (SOAs). The resulting integrated systems are called *process-driven SOAs*. The pattern language resulting from this work is reported in various PLoP and EuroPLoP papers and a textbook (Hentrich and Zdun, 2012). In this paper, we will discuss the empirical methods used during the pattern mining efforts for this pattern language. Before that, in this section, we will give a synopsis on the area of process-driven SOAs, i.e. the case in which we have performed pattern mining and validation. In Section 7 we will discuss the results of this pattern mining and validation case study.

2.1 Process-aware Information Systems

The latest definitions of the term Business Process Management (BPM) illustrate that workflow technology has become an important conceptual artifact that brings formerly separate worlds of organizational and technical design into an interdependent context (Prior 2003). Conceptually, BPM implies, on a technical level, the usage of technologies that allow organizational flexibility (Dumas *et al.*, 2005). For this reason, BPM has technical consequences and requires corresponding architecture design concepts.

The promise of BPM to enable organizational flexibility leads in many industries to a strong demand for BPM platforms, as the time to react on organizational change requirements is becoming shorter and shorter. Organizationally inflexible technology implies higher efforts in implementing the changes and thus higher costs are involved than using IT architecture concepts that consider organizational flexibility (Sauer and Willcocks, 2003). As many organizations are shifting towards process-oriented organizations, IT platforms have to consider this process approach conceptually.

In the resulting *process-aware information systems* (Dumas *et al.*, 2005), the notion of process represents a linking element between IT, organizational, and social issues, as business processes are strongly involved in the definition of an organizational structure and its culture on the one hand, and its technology on the other hand. Process-aware information systems depict organizational structure more or less directly with technology. For this reason, the corresponding process technology is very important for the future of businesses because this technology conceptually supports the interdependence between IT and organizational structures and provides organizational flexibility by technically decoupling business process logic.

2.2 Process-driven SOAs

In the context of process-aware information systems, *Service-Oriented Architectures* (SOAs) play an important role. In particular, recent trends towards SOA indicate that paradigms are required that conceptually support organizational flexibility and that support the interdependence between IT and organizational issues (Cherbakov *et al.*, 2005; Dumas *et al.*, 2005; Scheer *et al.*, 2007).

SOA is an architectural concept in which all functions, or services, are defined using a description language and have invokable, platform-independent interfaces that are called to perform business processes (Channabasavaiah *et al.*, 2003; Barry, 2003). Each service is the endpoint of a connection, which can be used to access the service, and each interaction is independent of each and every other interaction. Communication among services can involve simple invocations and data passing, or complex activities of two or more services. Though built on similar principles, SOA is not the same as Web Services,

which is a collection of technologies, such as SOAP and XML. SOA is an architectural paradigm, and hence it is independent of any specific technologies.

A SOA is typically organized as a layered architecture (see Figure 1), both on client and server side (Zdun *et al.*, 2006). At the lowest Communication Layer, low-level communication issues are handled. On top of this layer, a Remoting Layer is responsible for all aspects of sending and receiving of remote service invocations. The Remoting Layer consists of several sub-layers that handle request creation, request transport, and marshaling in a Request Handling Layer, request adaptation in an Adaptation Layer, and request invocation in an Invocation Layer. Above the Remoting Layer is a Client Application/Service Provider Layer of service clients on the client side and a layer of service providers on server side. The top-level layer is the Service Composition Layer in which the service clients and providers from the layer below are used to implement higher-level tasks, such as service orchestration, coordination, federation, and business processes based on services.

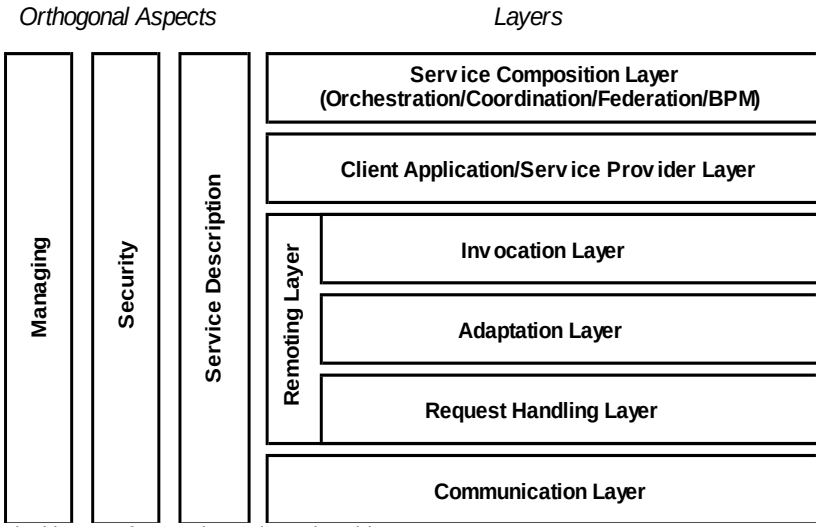

Figure 1: Overview of typical layers of a Service-Oriented Architecture

In this layered architecture, we can see a decoupling of process logic by introducing a dedicated Service Composition Layer. The process approach is a junction between organizational design/change and technical flexibility. This Service Composition Layer is technically represented by process engines that enact long running business processes (macroflows) and also short running integration flows (microflows) (Zdun *et al.,* 2006; Hentrich and Zdun, 2012).

We have investigated the integration between business processes and SOA to foster higher business agility, following an interdisciplinary approach based on software patterns. In the following sections, we discuss the empirical methods applied in this context and the lessons learned.

3 BACKGROUND: GROUNDED THEORY

Grounded Theory (Glaser and Strauss, 1967) is a systematic research methodology in the social sciences following the discovery of theory through the analysis of empirical data. It is considered very useful in qualitative research. As a result, it is a research method in which one begins with data collection methods rather than developing a hypothesis first. In that sense it operates almost in reverse order compared to traditional research methods. From the collected data, the key issues are identified and marked as *codes* with a series of coding procedures. The codes are grouped into *concepts* and *categories* are shaped from the concepts. These concepts and categories are the foundation for the emergence of a theory, i.e. a reverse engineered hypothesis. Grounded Theory (Glaser, 1992) basically deals with four analytical elements that also refer to stages of coding:

- *Codes* are key points extracted from empirical data
- *Concepts* are groupings of codes of similar content that allows the codes to be grouped
- *Categories* are groups of similar concepts

- A *theory* is a collection of categories, concepts, and codes that explain the subject under investigation

Validity in Grounded Theory is judged by the characteristics of fit, relevance, workability, and modifiability (Glaser and Strauss 1967):

- *Fit* means to evaluate how closely concepts fit with the observations they represent. It shows how well constant comparison was conducted. *Constant comparison* in this context means to constantly compare the codes, concepts, and categories as they emerge.
- *Relevance* means to evaluate whether one deals with a real concern of participants.
- Workability is satisfied if the theory explains how the problem is being solved with much variation.
- *Modifiability* means that a theory can be altered when new relevant data is compared to existing data. An elaborated theory is not right or wrong, it rather has more or less fit, relevance, workability and modifiability.

Theoretical memoing is *"the core stage of grounded theory methodology"* (Glaser, 1998a). *"Memos are the theorizing write-up of ideas about substantive codes and their theoretically coded relationships as they emerge during coding, collecting and analyzing data, and during memoing"* (Glaser, 1998a). Memos are important to refine and to keep track of ideas that develop in an emerging theory. In memos you develop ideas about naming concepts and setting them in relation to other concepts.

Grounded Theory is based on inductive principles and anchored in empirical data. This idea maps to the concept of emergent software patterns forming a theory of conceptualizations of problems with corresponding solutions. For this reason, we consider Grounded Theory as a suitable approach for pattern mining.

4 AN APPROACH BASED ON GROUNDED SOFTWARE PATTERNS

A software pattern describes an abstraction or conceptualization of a concrete, complex, and reoccurring problem that software designers have faced in the context of real software development projects and a successful solution they have implemented multiple times to resolve this problem (Gamma et al., 1994). A software pattern may thus resolve many different influential forces that constitute a complex problem, i.e. it is not just technical issues that a software pattern may deal with. The original concept of a design pattern has been invented by Alexander (1977) in the context of civil architecture and has been successfully applied in software development projects for many years (Buschmann et al., 2007; Coplien, 1996).

As a software pattern basically tells “how to” solve a (design) problem instead of prescribing exactly “what” to do, it abstracts away from the very specific situation one might be facing. It captures a general rule that is not dependent on the actor and/or the specifics of a certain situation. In addition, patterns capture the rationale behind a solution, i.e. “why” a problem is solved in the way described by the pattern. A software pattern language consists of a set of related software patterns (Alexander, 1977).

In this section we explain how to apply Glaserian Grounded Theory (Glaser, 1992) as a research method for pattern mining and the grounding of the mined patterns in empirical data by presenting the linkages between software patterns and Grounded Theory. We do not intend to explain concepts of Grounded Theory in detail, as this would go beyond the scope of this paper. We would refer the interested reader to existing Grounded Theory publications on Glaserian Grounded Theory itself (Glaser, 1992; Glaser, 1998a; Glaser, 1998b; Urquhart, 2001).

In the pattern mining problem, in focus of this paper, the concrete process used for pattern mining is largely up to the pattern author and rarely evidence beyond references to known uses and experiences of the pattern author is given in pattern texts. That is, today pattern mining is a rather informal or ad hoc process of finding patterns in software systems. In particular, often the pattern author identifies one or more patterns in his own experiences, and then broadens the scope by searching for the identified patterns in related systems and/or looking for related other patterns in the systems under consideration.

Our approach is based on the observation that software patterns are sociological phenomena. As a consequence, they can be discovered following a Grounded Theory (Glaser, 1992) approach. One important aspect which is well accepted in the practical software pattern community, but so far not in focus of many research approaches about patterns, is that software patterns have a significant evolutionary and human aspect in them (Seaman, 1999). They represent successful problem solving behavior, which corresponds to the concept that a pattern describes “how to” and “why” to solve a problem rather than

concretely dealing with “what” to solve. Thus, they deal with people’s behavior to solve complex design problems, and they describe how and why conflicting forces arise and how and why these forces are resolved by a (human) software designer. For this reason, patterns are (also) a sociological phenomenon. From a sociological perspective, software patterns are successful ways of thinking that have emerged in an evolutionary context and that capture the generic problem solving behavior of people.

Patterns thus represent a conceptualization of problem solving behavior achieved by abstracting from implementations and the explicit decisions software designers have made in software development projects when dealing with many influential forces that constitute complex problems. That is, software patterns to be newly discovered are successful mental design constructs, which have not yet been discovered as such, but are rather represented as unexplored and unconscious knowledge of expert software designers that is hidden in the software implementations. A pattern represents a rule that is applicable to explain a broad variety of instantiations.

Grounded Theory deals with theory development based on inductive principles following concepts of abstraction and conceptualization (Glaser, 1992).The discovery (or mining) of a software pattern language corresponds to the theory building process in Grounded Theory. In Grounded Theory, the primary method of data analysis is continuous *coding* of data. That is, the existing empirical data is examined, data item by data item, to find interesting categories of e.g. elements, actions, or events in the data. Then, it is tried to make conceptual connections between a category and its subcategories and between different categories. Codes and categories will be sorted, compared, and contrasted until all the data is accounted for. After each coding step, new data is added, e.g. by looking at additional systems or performing more interviews with developers. Through recurring coding steps, a theory is developed in a process of *constant comparison*. In Grounded Theory coding means to create a theoretical model for the observed data. This model is not created beforehand, but it emerges during the comparative process of Grounded Theory.

The pattern language is thus discovered by constant comparison of grounded data to work out the patterns and their relationships in conjunction with Glaserian coding procedures. Open coding procedures are applied during the initial stage of constant comparison in order to discover software pattern categories and their properties. Theoretical coding is applied to develop conceptual linkages between (software) categories and their properties as they emerge (Glaser, 1992). The concept of writing *memos* in Grounded Theory can be represented by the process of writing the actual patterns in several iterative stages. Developing a pattern language represents a theory building process that is elaborated by abstraction and conceptualization from empirically grounded qualitative data. In summary, Grounded Theory can be used as a systematic method for pattern mining that bases the mined patterns on empirical data.

As Grounded Theory thus provides a suitable method to address software patterns from a social perspective in terms of inductive analysis of human problem solving behavior, our approach has been to systematically investigate successful SOA implementations following a Grounded Theory approach and discover the patterns being used to form a grounded pattern language for this class of architectures from the empirical data being collected. SOA has been an unexplored domain as far as software patterns are concerned and no substantial and coherent software pattern language has been available. An initial survey of available patterns in this domain has led to this conclusion (Zdun *et al*., 2006). This served as the primary motivation to develop a software pattern language for this class of software architectures.

5 GROUNDED PATTERN MINING AND VALIDATION

Based on the approach discussed in the previous section, our research has been conducted in various professional software engineering projects that have taken place in various industries, such as telecommunications, automotive, transportation, insurance, and banking. The investigations have included project documentations, design specifications, analysis of existing running systems and applications, as well as discussions and interviews with the people involved in design and programming activities. We have conducted 43 semi-structured interviews in ten projects. Five of these projects were large projects and the other five were medium sized and small projects. The average duration of an interview was approx. 2 hours. The research described in this article has been conducted between 2003 and 2007 with the aim to discover a software pattern language for SOA. We have focused on emerging patterns that address the integration between software and business architecture to allow easy implementing of changes to the business and thus to achieve higher business agility.

Using the approach discussed in the previous section we have developed a procedure for systematic pattern mining based on principles of Grounded Theory. The general procedure for grounded pattern mining and validation consists of three phases:

- Phase A: Fieldwork preparation
- Phase B: Pattern mining (fieldwork)
- Phase C: Refinement and validation

The ordering of the phases and the steps in the phases map to the coding procedures of Grounded Theory as introduced in the previous section. The phases will be explained in more detail in the following.

5.1 Phase A: Fieldwork Preparation

First, a general conceptual approach must be developed to frame the investigation based on the domain in which the patterns are mined. For instance, in our process-driven SOA case study we have developed a conceptual approach based on theoretical foundations on enterprise agility (Hentrich, 2006). The conceptual approach forms an equivalent to Glaser's (1992) coding families for the specific requirements of pattern mining and validation. That means the conceptual approach guides what areas of interest to investigate in terms of patterns and the potential relationships between them.

5.2 Phase B: Pattern Mining

In the next step, (software development) projects need to be selected that cover a broad range of expertize in the domain in which the patterns are to be mined. For instance, in our process-driven SOA case study we have selected process-driven SOA projects that span across different industries and different sizes. Internal company networks have been used to identify the projects. For each identified project the following steps are executed:

- *Step 1:* Relevant representatives are identified for interviews. The following roles are considered as most relevant: business analysts, designers, programmers, architects, and project managers.
- *Step 2:* First of all, available documentation is scanned as to draw first conclusions and to scope the first areas of work. The documentation is mapped to the conceptual areas of the guiding framework for the investigation (see Phase A).
- *Step 3:* A defined pattern format is used as a documentation framework. Based on the available documentation, assumptions about relevant pattern contexts are made and interview partners are selected in a prioritized order according to the identified contexts. Interviews are scheduled and prepared to address the identified contexts.
- *Step 4:* Each interview is conducted in a semi-structured format using a defined pattern specification format (addressing pattern context, problem, solution, consequences, examples, and known uses) as a documentation framework. The interview structure is as follows:
    - An introduction to the pattern mining work is given. Interview partners are informed in advance via e-mail. First interviews are scheduled for 2 hours. Further interviews with the same partner are scheduled according to topics that need to be addressed as a result of previous interviews.
    - From the implementations the person has been involved with, abstract solutions are conceptualized to abstract away from the implementation and to capture the generic solution. The abstract solution is briefly documented during the session and agreed with the interviewee, e.g. by drawing some conceptual pictures on a white board with some annotations.
    - Possible problems that have been addressed by the solution are identified with the interview partner, questioning the key issues (depending on the person's role this might vary). This discussion is led by identifying the conflicting forces that a solution resolves and which form the actual problem. Thus problems are identified documenting the conflicting forces that have led to the conceptual solution. The problem-solution relation thus evolves.
    - The pattern is discovered giving a unique name to the problem-solution relation (open and theoretical coding).
    - Consequences that result from the solution are discussed, both positive and negative consequences.
    - Relationships to other patterns are identified and discussed. Patterns that have already been elaborated in other interviews are briefly introduced and relationships are discussed.
    - The results are documented in a first draft of a pattern specification. Note that it is possible to elaborate several pattern candidates in a single session. The procedure needs to be

performed for each pattern. This is an iterative process of exploration, and usually more than one session is expected to be necessary.
    - After the interview, the examples are documented for the patterns based on the project's implementation that has been discussed and known uses are identified. Thus, a version of the pattern specification is created. The pattern specification is communicated via e-mail to the interviewee and if further issues for discussion are identified (from the interviewer or the interviewee) then a new interview is scheduled.
- *Step 5:* Through the concept of constant comparison with evolving patterns from different interviews the pattern specifications and relationships are refined (theoretical coding). Pattern categories emerge through logically related patterns in logical domains. The pattern relationships evolve using concepts from Stream Analysis (Porras, 1987). New contexts that point to unaddressed areas emerge and the whole process is repeated for these new contexts starting with Step 1. From interviews in different projects the same patterns might emerge and new known uses for a pattern may evolve. In this case the discussion is based on the existing pattern documentation to investigate whether the existing pattern specification is sufficient, needs to be improved, and new known uses and examples can be added.

5.3 Phase C: Refinement and Validation

For refinement and validation of the mined patterns we suggest following the established process of the pattern community. Of course, this community-driven process can be replaced by an equivalent review process with internal reviewers e.g. in the context of a company. For our process-driven SOA, the pattern specifications and relationships between patterns that have evolved to a satisfactory degree have been submitted to pattern conferences for further refinement and validation. The process goes as follows:

- *Step 1:* The paper is submitted to a pattern conference. The PLoP[1] and EuroPLoP[2] conferences have been used for paper submission.
- *Step 2:* The paper is accepted or rejected. If the paper is rejected the reviewer's comments are used to refine the paper. Possibly another interview is scheduled to discuss the review comments with the original interviewees. The paper is re-submitted until it is accepted. If the paper is accepted the process moves on as follows.
- *Step 3:* The paper moves through a so-called "shepherding" process, where the paper is improved to achieve a conference ready version. Shepherding is a process of giving and receiving feedback on the paper and incorporating the feedback in the paper. A "shepherd" i.e. a reviewer, is assigned by the conference committee. The reviewer is an experienced pattern author and usually has expertise in the area the paper addresses. This phase lasts about three months and three iterations to improve the paper are planned.
- *Step 4:* After the shepherding process is finished the reviewer speaks out a recommendation whether the paper is accepted for the conference. If the paper is rejected, the process as described above will be repeated, i.e. improvement of the paper using the reviewer's comments with interviewees and re-submission. If the paper is accepted for the conference, the paper will go into a conference workshop and is published in the pre-conference proceedings.
- *Step 5:* In the conference workshop, four to five experts (not having been involved in the process before) discuss the paper giving comments for improvements. The author is listening and takes notes on the comments. The idea is that a pattern paper needs to speak for itself and should be understandable simply by reading the paper. For this reason, the author is not allowed to justify the statements in the workshop. The workshops follow a structured format.
- *Step 6:* After the conference, the paper is either accepted for publication or needs to be improved, based on the comments (same procedure via re-submission).
- *Step 7:* If the paper is accepted for publication it can be considered as validated and is published. Comments from the workshop are incorporated before publication. There is a span of about six months between the conference and the publication to improve the paper based on the workshop comments.

[1] http://www.hillside.net/plop/
[2] http://www.hillside.net/europlop/

All elaborated patterns have run through this process of publication to allow an acceptable degree of validity.

All patterns have been mined, refined, and validated via this procedure. The results of this research have been published with detailed pattern descriptions including examples in our prior work; this work has been reported in various PLoP and EuroPLoP papers and a textbook (Hentrich and Zdun, 2012). In the next section we summarize the main outcomes of this research.

6 PATTERN MINING EXAMPLE

In this section we provide an example of the software patterns analysis based on grounded data from an interview. The original analysis procedure for mining patterns has been described in the previous section. The following example demonstrates how this analysis procedure has been applied to mine SOA patterns.

This example essentially is an excerpt of a transcript from an interview that has been conducted with an IT architect that has been responsible for parts of an SOA solution. This architect has been specialized on the business modeling methods and techniques to be used to model the business architecture that had to be implemented on an SOA platform. The interviewee has been informed in advance about the questions that are going to be addressed during the interview. The questions ask for the specific solution, the problem the solution addresses, the conflicting forces that constitute the problem, and the context where the problem occurs. During the interview these questions are first focused on the very specific situation and implementation scenario of the project. This serves first to gather grounded data and then to abstract away towards codes using conceptualization based on that grounded data.

The flow of the interview questions isolate and discuss a specific solution the architect has designed, the problem he has solved, the specific forces that constitute this problem, and the specific implementation context where the problem occurs. The example mainly illustrates how significant elements of the context-problem-solution relation of the MACRO-MICROFLOW (Hentrich and Zdun, 2012) pattern evolve (in reverse order). Aspects of other patterns like the MACROFLOW ENGINE and MICROFLOW EINGINE (Hentrich and Zdun, 2012) pattern are also touched. The following text illustrates an excerpt from the original transcript of the interview. The text has been changed slightly to make the data anonymous and to make a few corrections on the Grammar. Some irrelevant details have been left out. A tape recorder was used to record the interview.

6.1 Interview Extract

Interviewer: Thank you very much for taking the time to do this interview today, it is very much appreciated. So [...] thanks a lot!

Interviewee: You're welcome. Hope I can help you a bit with my answers?

Interviewer: I am pretty sure you can [...] as you have already been informed about the whole idea of this research I would like not to waste too much of your valuable time and go into the details quickly. Hope you don't mind if we do so?

[...]

Interviewee: No, not at all – please go ahead.

Interviewer: So [...] you know the questions already - I understand you are an IT architect and sort of take the role of a business architect. Could you explain your role in this project?

Interviewee: Yes, sure [...] I am an IT architect that's correct. In this project I have been responsible for the modeling conventions to model the business architecture. The team then had to follow these conventions when they did their modeling in ARIS. ARIS is the design tool we are using, by the way.

[...]

Interviewer: OK, that's interesting. Let's say you could pick one aspect of modelling business architecture, which is the most important one, which one would you pick? I mean is there some key issue

or key solution you have incorporated in your modelling conventions or the techniques you have defined for the design tool.

Interviewee: Well, we have been using ARIS mainly to model the business processes [...] so if we talk about business architecture I more or less mean business process architecture or simply business processes. I did a lot of conventions on modelling the business processes as those are not really obvious [...] or, better to say, we found some issues that are not obvious. Getting the processes right is possibly the most important issue, here.

Interviewer: That's interesting [...] what do you mean, when you say "getting the processes right is possibly the most important issue"? Why is it so important and what are these key issues about it. Please try to describe the solutions you have implemented that are so important to you.

Interviewee: First of all it's the business processes [... or rather] modelling the business processes. ARIS has a notation, [...] it actually offers quite a lot of notations but we have used the EPC models, which is pretty much standard in ARIS. I think the problem was that the EPC notation is just a notation, not more, but it does not tell you enough about the content of the models. Hard to explain maybe [...] we made some first attempts with the business analysts and identified that they modelled processes in an insufficient way [...] some knew a bit about the SOA stuff so they also modelled some technical things and others modelled very detailed issues which we rather saw as being part of use cases.

Interviewer: You say people have modelled in an "insufficient way" and you mention technical things and use cases. What was the insufficient part about modelling technical things and use cases? I understand you consider this as some core element of your solution [...] maybe the modelling conventions you defined?

Interviewee: Yes [...] I think the most important conventions we defined were about how to model the business processes. It was not the notation; [...] you get all that from the EPC but how you model the EPC is quite a challenge. We have defined some conventions that give guidance on how an EPC should look like, what should be in it and what not. These conventions are about separating the technical concerns from the business processes and also the things that we want to put in the use cases. So I think it is about the granularity of the models [...] we want to them to model the right information that we can then later use to refine the business processes in more technical transactions and into use cases. The use cases make GUI interactions later on, so this is also application design stuff. The business processes should be independent of application design and also independent from the technical processes that the IT guys do model later. They start when the business processes are finished. Nevertheless, they need to understand each other [...]; that the models link in later on and we can more or less directly map them on the SOA platform for implementation. If you ask me about that core solution again, it is possibly about providing them with the conventions to model the EPCs at a level that does not yet consider technical processes, nor things at a use case level. We use different technologies for implementation of those things – that's another reason why to separate this. So the conventions we defined are about what to consider in the models and what not to consider. They are also about how things will move on from their models ... how the technical designers refine the business steps in the business processes with their technical, transactional micro-processes.

Interviewer: Now, here is the challenge: If you had to put everything you have said in you last answer into a few precise sentences that constitute this solution – some kind of short solution statement – what would that be? Take your time, you can write it down on the white board or on this sheet of paper if you like.

Interviewee: Good question [...] the solution is to define modeling conventions, guidelines for the BAs in this case that strictly separate the business concerns of the business processes from the technical concerns and the GUI concerns of the application in the use cases. This is to provide that the business processes remains more stable to changes as they are not depended on the technical details and the screen-flows in the GUIs – these may change more often. On the other hand those guidelines provide an understanding for the GUI designers and the technical designer how their models can be directly linked in

the EPC models. They are linked in by refining the single steps in more fine grained technical micro-processes and also screen-flow processes in the use cases. I think the solution is probably about separating the concerns while on the other hand linking the models effectively to not do much rework in the later design phases. [...] Ah, I forgot that we also did this make the models easily implementable on MQ Workflow. On [MQ] Workflow we implement the EPCs and they are more or less directly implementable when the BAs follow the conventions.

[...]

Interviewer: So, let me summarize: we got three levels of process, transactional micro-processes, screen-flows in use cases and the EPCs which are the actual business processes. All three models link in and your conventions tell how to model that they can link in. Is that right?

Interviewee: Right.

Interviewer: You mentioned MQ Workflow and that you implement the EPCs on MQ. Your conventions for EPCs also tell how to model to make implementation on MQ Workflow easy, right?

Interviewee: Right.

Interviewer: Why did you choose to use MQ Workflow and not another tool or why do you need such a technology?

Interviewee: MQ [Workflow] is good for the long running processes. Those are the EPCs they are usually long running and we got humans interacting in the business processes [...] so you need an engine that can do this. It needs to be able to model your organisation and the roles you got in the processes. This is sort of the long running businesses processes in the macro-world, while the other processes are finer grained transactional in the micro-world. We called them micro-processes as well. We use the message broker to implement the transactional micro-processes and not MQ Workflow.

Interviewer: So we got a separation in three levels of processes: GUI screen-flows in use cases, EPC business processes, and transactional micro-processes. Then, we say that EPCs are long-running and implemented on MQ Workflow – this is the macro world. Then we got transactional micro-processes, which are short running implemented on a message broker [...]

Interviewee: Sorry, when I say “message broker” I mean the IBM broker tool.

Interviewer: Ah, OK that’s the [IBM] WebSphere Message Broker I suppose?

Interviewee: Yes.

Interviewer: OK, seems that we got some sense of a solution you designed. I would now like to move to the question why this is a solution? I mean, what does it actually solve? We have talked about some of this already but I would like to work this out a little better. What problem does this solution solve?

Interviewee: Ok now – the solution is the conventions or guidelines that tell you how to model the business processes in EPCs. [...] We have discussed why we did it this way. The problem [... thinks]? As indicated in the beginning already, we need to separate the concerns of these different models on the one hand but we also see relationships between them. So, this is about [...] the problem to separate concerns, while at the same time linking different models done by different people to deal with the relationships. It’s an integration issue. You need to understand that this was quite a big project and there were quite a few people modeling at the same time. People had to know how to use the models and to be on the same page. Kind of a common paradigm we had to establish.

Interviewer: Sounds to me that this resolves a kind of conflict. You say on the one hand it has to do this and on the other hand it also has to do that at the same time. Is it a conflict that we see here and which is resolved by your modeling conventions?

Interviewee: Yeah [...] it is some kind of conflict; because this is actually separate things but not really separate. It is completely different people doing those different kinds of models with different skill-sets. Nevertheless they build on each other and need to integrate, so we can't treat them as completely separate. The separate models need to be treated as separate but need to form a greater whole, fit together at some point in the development stage. They also need to be implementable on the technologies we use, like MQ [Workflow] and the [IBM] message broker. That's why we need conventions and guidelines to make things fit together in the end. Ah yes, important thing: the separation is necessary to encapsulate change requirements later on. You can't change the whole thing. If you mix things you got a problem. When you separate [the concerns] you got more reuse and limit the changes. You can't manage the models any more when you mix it all up and your architecture is too inflexible. SOA is about easier change though, but it's not just the technologies it's about how you design your business architecture. It's more about business architecture than about technical architecture.

Interviewer: OK, that's great [...] I think we got an idea of the problem as well – also the conflict that this problem addresses. For now, I would finally like to talk about context. What do you think is the context this problem occurs?

Interviewee: The context is probably quite simply when you model your business processes for SOA implementation using workflow systems – that's when you run into this.
[...]

Interviewer: Thanks a lot. It has been a great pleasure talking to you. I think I got enough material to come up with a draft document. There are probably several aspects on quite a few patterns involved in this discussion. I'll return to you as soon as I got a draft, so we can further discuss this.

6.2 Grounded Theory Based Interview Analysis Example

We now illustrate how codes have been analyzed from this data according to Grounded Theory (Glaser, 1992). Those codes are grouped into a grounded concept that constitutes the frame of a software pattern by capturing a context-problem-solution relation. The concept represents a first draft of a pattern specification, which is later on refined in several iterative stages. In this example, a first initial draft of the MACRO-MICROFLOW pattern evolves from the data. Table 1 shows the data captured from the interview and the codes developed from that data to frame the concept of the MACRO-MICROFLOW pattern.

Table 1: Coding example of codes and concept

Concept (pattern): MACRO-MICROFLOW	
Examples of grounded data	**Developed code**
"strictly separate the business concerns of the business processes from the technical concerns and the GUI concerns of the application in the use cases. This is to provide that the business processes remains more stable to changes as they are not depended on the technical details and the screen-flows in the GUIs – these may change more often. On the other hand those guidelines provide an understanding for the GUI designers and the technical designer how their models can be directly linked in the EPC models. They are linked in by refining the single steps in more fine grained technical micro-processes and also screen-flow processes in the use cases. I think the solution is probably about separating the concerns while on the other hand linking the models effectively to not do much rework in the later design phases."	*Solution:* Structure a process model into two kinds of processes, macroflow and microflow. Strictly separate the macroflow from the microflow, and use the microflow only for refinements of the macroflow activities. The macroflow represents the long-running, interruptible process flow which depicts the business-oriented process perspective. The microflow represents the short-running transactional flow which depicts the IT-oriented process perspective.
"As indicated in the beginning already, we need to separate the concerns of these different models on the one hand but we also see relationships between them. So, this is about [...] the problem to separate concerns, while at the same time linking different models done by different people to deal with the relationships. It's an integration issue. You need to understand that this was quite a big project and there were quite a few people modelling at the same time. People had to know how to	*Problem:* Models of business processes must be developed considering the relationships and interdependencies to technical concerns. If technical concerns are tangled in the business process models, however, business analysts are forced to understand the technical details, and the technical experts must cope with the business issues when they are realising technical solutions. This should be avoided. On the other hand, to create executable process models, somehow the

use the models and to be on the same page. Kind of a common paradigm we had to establish." *"it is some kind of conflict; because this is actually separate things but not really separate. It is completely different people doing those different kinds of models with different skill-sets. Nevertheless they build on each other and need to integrate, so we can't treat them as completely separate. The separate models need to be treated as separate but need to form a greater whole, fit together at some point in the development stage."*	two independent views need to be integrated into a coherent system.
"The context is probably quite simply when you model your business processes for SOA implementation using workflow systems – that's when you run into this."	*Context:* Business processes shall be implemented using process (workflow) technology.

7 CASE STUDY RESULTS: PROCESS-DRIVEN SOA PATTERNS

In this section, we summarize the results of our pattern mining effort for the process-driven SOA pattern language, to illustrate in the context of this case the potential outcomes of our approach. The general pattern relationship overview diagram in Figure 2 provides a consolidated overview on the actual relationships of the patterns in the process-driven SOA pattern language (Hentrich and Zdun, 2012) resulting from our pattern mining effort. The grey areas in Figure 2 show the pattern categories with the patterns associated to a category inside them. This figure does actually not display all possible relationships but only outlines the major relationships between the pattern categories in the pattern language. The annotations on the arrows in Figure 2 between the single patterns and the pattern categories indicate how to navigate between the different contexts associated to the patterns and pattern categories.

The identified patterns in this pattern language resolve interdisciplinary conflicting forces that influence architecture design. The empirical data coded in the patterns shows that in SOA sociological, business, and technical perspectives are integrated in a common design paradigm. People are an integral part of a larger organizational system and interact with IT systems. The systemic perspective reflected by SOA works at a higher level of abstraction than other recent architectural paradigms. The pattern language has evolved around the concern how this integration and convergence of technical, human, and business aspects is reflected by problem solving behavior.

As previously mentioned, in the SOA context, the notion of business process has become a linking element between work practices, business drivers, and technical aspects in problem solving behavior. Data design is linked to process design, as well as designing functions (services) that fulfill tasks within processes. The human perspective is integrated by people playing an essential and active part in the business processes when fulfilling tasks in processes and interacting with each other. The business processes reflect the interaction principles of an organization's working culture. When following the pattern language, business processes are directly implemented using special technologies that allow configuring process models rather than implementing the processes in program code written by software developers. As a result, process-orientation has become a leading problem solving paradigm in SOAs that follow the process-driven SOA approach. The patterns thus deal with designing software architectures from a business-process-centric perspective.

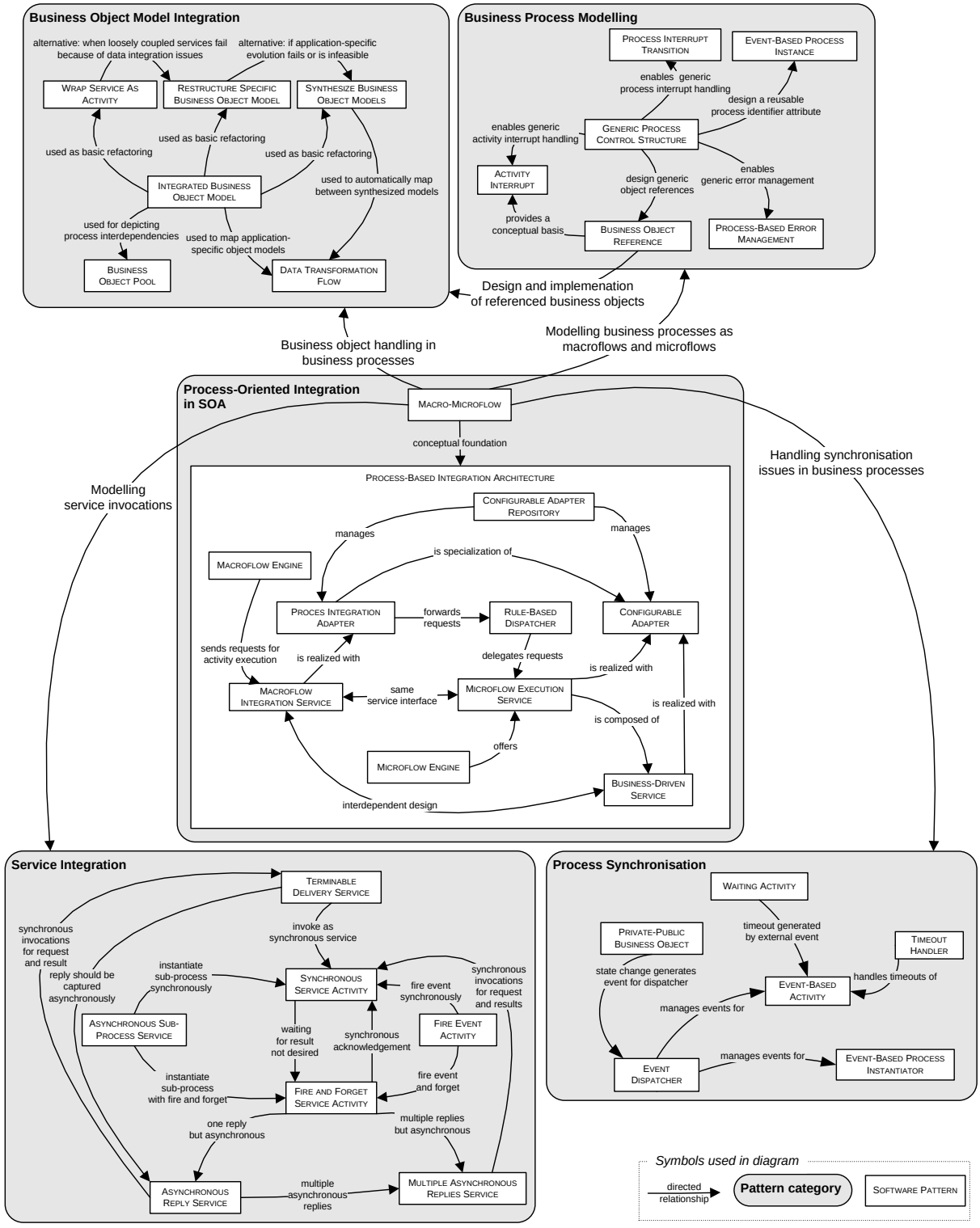

Figure 2: Pattern overview diagram for the process-driven SOA pattern language

Interestingly, in the investigated domain of SOA implementations, where process-orientation in business IS is strategically important to enable flexibility (see Section 2.1), it emerged during the mining of the pattern language that the guiding design concept captured by the pattern language is a business process focused approach to software architecture design and not primarily a service-centric (i.e. function-centric) approach. This is interesting as one might actually not expect this in an architectural paradigm

named “service-oriented” architecture. One would rather expect that a primary service-centric design paradigm is followed. This is not the case according to our observations in our empirical pattern research. This can be concluded from the real world examples and case studies of the patterns we have found. Examples and case studies can be found in our published pattern documentation (Hentrich and Zdun, 2012).

A service is basically a functional interface to some system, and service-oriented means that all systems are flexibly accessible by functional interfaces (services) no matter who or what wants to call that interface. The services aim to hide the internal complexity of a system being integrated in a business process focusing on offering functionality in a standard way. The idea is that functionality can be flexibly assembled using those services. The term “SOA” actually indicates that all design activities might be driven by the concept of service, but our empirical data shows that this is actually not the case from a problem solving point of view. Rather the problem solving behavior is business-driven, following a business-process-centric problem solving approach.

In particular, the central pattern in the pattern language is the MACRO-MICROFLOW pattern. This pattern helps in structuring and modeling business processes that later on invoke services. According to this pattern, the designer distinguishes between business and IT concerns when modeling business processes by putting these different concerns in different conceptual process layers. *Macroflows* reflect the process layer for the business concerns, and *microflows* represent the layer for the IT concerns. As a consequence of following the MACRO-MICROFLOW pattern, the overall architecture design activity is driven by a business-process-centric approach. That is, all design constructs are related to business processes, which need to be modeled and implemented.

Our research has shown that the business-process-centric design paradigm is the leading principle for all problem solving activities in the pattern language, as all patterns basically relate to business processes. Consequently, the core architecture design, represented by the PROCESS-BASED INTEGRATION ARCHITECTURE pattern, is also driven by this process-centric approach. The problems are created out of this process-based approach and the solutions at higher design abstraction levels create more detailed problems that are solved at more detailed design levels. Figure 3 shows a typical layered design following the PROCESS-BASED INTEGRATION ARCHITECTURE pattern.

Consequently, the more detailed patterns in the pattern language address more specific issues for dealing with data issues, designing services being invoked from processes, or synchronizing and coordinating the business processes. Processes are understood as reusable components that can be flexibly assembled. The processes themselves represent functionality that can be offered as services. The activities in the processes coordinate more fine grained service invocations as to flexibly assemble more fine grained functionality (Zdun *et al*., 2006).

The patterns address that service design, i.e. the reusable functional assets to compose the overall business functionality, is driven by business processes and the problem solving behavior for designing the business and corresponding software architecture is aligned with that. This approach actually allows identifying the relevant functionality for the business. Adaptability is achieved by directly mapping organizational structures of business processes on the technology platform represented by process engines. Changes to processes result in changed process models on the process engines where functionality in terms of services can be reused and only those parts of the functionality need to be newly developed that are not available yet.

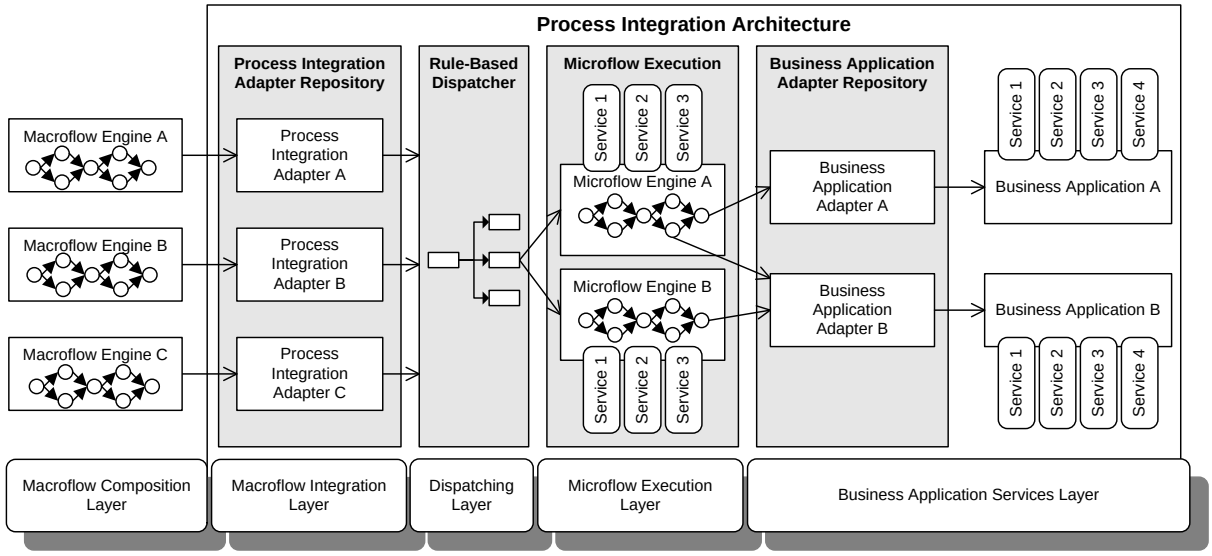

Figure 3: Layers and boundaries of a Process-Based Integration Architecture

8 CONCLUSION

In this paper we have emphasized the notion of understanding patterns as sociological phenomena of problem solving behavior. That is, a strong focus of patterns is on the notion that software design is a creative human problem solving activity (Seaman, 1999). This notion allows better understanding that patterns are successful social concepts of thinking being tacitly present until they are discovered as patterns. Researching software patterns means to make these tacit thought models (Nonaka and Takeuchi, 1995) transparent and to point out the complex relationships between them.

We have introduced a systematic approach for pattern mining using Glaserian Grounded Theory (Glaser, 1992) and pattern validation through the pattern community's review process, and we have illustrated this approach using a pattern mining and validation case in the SOA domain. That is, by adapting a method from the social sciences we have demonstrated that software design research benefits from interdisciplinary approaches following different and complementary methodologies. The patterns are explicitly captured in an instructive writing style to make this design knowledge accessible for inexperienced people. These people may apply the patterns in their daily work and thus significantly speed-up their learning curve. As a result, the pattern language serves as a way of transferring expert design knowledge. Having provided a scientific approach for mining software patterns, this work may contribute (1) to a better empirical grounding of software patterns and (2) to better understanding the sociological aspects of software design.

REFERENCES

Alexander, C. (1977). A pattern language – Towns, buildings, construction. Oxford University Press, Oxford.

Buschmann, F. & Henney, K. & Schmidt, D.C. (2007). Pattern-Oriented Software Architecture Volume 5: On Patterns and Pattern Languages. Wiley & Sons. Chichester, UK.

Barry, D.K. (2003). Web Services and Service-oriented Architectures. Morgan Kaufmann Publishers. San Francisco, CA

Channabasavaiah, K. & Holley, K. & Tuggle, E.M. (2003). Migrating to Service-oriented architecture – part 1. IBM Developer Works, from http://www-106.ibm.com/developerworks/webservices/library/ws-migratesoa/.

Cherbakov, L. & Galambos, G. & Harishankar, R. & Kalyana, S. & Rackham, G. (2005). Impact of service-orientation at the business level. IBM Systems Journal, Vol. 44 No. 4, pp. 653—668.

Coplien, J. (1996). Software Patterns. New York. SIGS Books.

Dumas, M. & van der Aalst, W.M. & ter Hofstede, A.M. (2005). Process-Aware Information Systems: Bridging People and Software Through Process Technology. Wiley. Hoboken, New Jersey.

Gamma, E. & Helm, R. & Johnson, R. & Vlissides, J. (1994). Design patterns – elements of reusable object oriented software. Addison Wesley, New Jersey.

Glaser, B.G. & Strauss, A. L. (1967). The discovery of grounded theory: strategies for qualitative research. Chicago.: Aldine.

Glaser, B.G. (1992). Emergence vs. Forcing: Basics of Grounded Theory Analysis, Mill Valley, Ca.: Sociology Press.

Glaser, B.G. (1998a). Doing Grounded Theory. Issues and Discussions. Mill Valley, Ca.: Sociology Press, from http://www.groundedtheory.com/soc13.html.

Glaser, B.G. (1998b). Gerund Grounded Theory: The Basic Social Process Dissertation. Mill Valley, Ca.: Sociology Press, from http://www.groundedtheory.com/soc8.html

Hentrich, C. (2006). A Language of Analytical Patterns for the Agile Enterprise. International Journal of Agile Systems and Management. Inderscience, Vol. 1 No. 2, pp. 146—165.
Hentrich, C. & Zdun, U. (2012). Process-Driven SOA - Proven Patterns for Business-IT Alignment. CRC Press, Taylor and Francis, Boca Raton.
Nonaka, I. and Takeuchi, H. (1995). The knowledge-creating company. Oxford, UK: University Press.
Porras, J. I. (1987). Stream analysis – a powerful way to diagnose and manage organizational change. Prentice Hall.
Prior, C. (2003). Workflow and Process Management, Maestro BPE Pty Ltd, Australia.
Sauer, C., & Willcocks, L. (2003). Establishing the Business of the Future: The Role of Organizational Architecture and Information Technologies, European Management Journal, Vol. 21 No. 4, pp. 497—508.
Scheer, A. W. and Kruppke, H. and Jost, W. (2007). Agility by ARIS business process management. Yearbook Business Process Excellence 2006/2007. Springer, Berlin.
Seaman, B. C. (1999). Qualitative Methods in Empirical Studies of Software Engineering. IEEE Transactions on Software Engineering. Volume 25 , No. 4. ISSN:0098-5589, pp. 557 – 572.
Urquhart, C. (2001). An Enounter with Grounded Theory: Tackling the Practical and Philosophical Issues. in Trauth, E. M. Qualitative Research in IS: Issues and Trends, Idea Group Publishing, London, pp. 104 – 140.
Zdun, U. & Hentrich, C. & van der Aalst, W.M.P. (2006). A Survey of Patterns for Service-Oriented Architectures. International Journal of Internet Protocol Technology. Inderscience, Vol 1 No.3, pp. 132—143.