**Technical Report, No. TR-SWA-20120601,**
**Faculty of Computer Science, University of Vienna,**
**June, 2012**

# Architectural Decision Making for Service-Based Platform Integration: A Qualitative Multi-Method Study

Ioanna Lytra[1], Stefan Sobernig[2], Uwe Zdun[1]

[1]Faculty of Computer Science
University of Vienna, Austria
Email:
*firstname.lastname@univie.ac.at*

[2]Institute for IS and New Media
WU Vienna, Austria
Email: *stefan.sobernig@wu.ac.at*

**Abstract**

Nowadays the software architecture of a system is often seen as a set of design decisions providing the rationale for the system design. When designing a software architecture multiple levels of design decisions need to be considered. For example, the service-based integration of heterogeneous platforms and the development of applications on top of those integration services requires high-level as well as technology-, domain-, and application-dependent architectural decisions. In this context, we performed a series of qualitative studies following a multi-method approach. First, we conducted a systematic literature review from which we derived a pattern language for platform integration featuring 40 patterns, as well as a pattern-based architectural decision model. Then, we performed interviews with 9 platform experts from 3 companies for revising the architectural knowledge captured by the pattern language and the decision model. Finally, we participated in a case study and observed the decision-making process to validate the results further. Our observations resulted in 1) a qualitatively validated, pattern-based architectural decision model and 2) a generalized model of the different levels and the different stages of architectural decision making for service-based platform integration.

## 1 Introduction

In recent years, software architecture is less and less seen as only the components and connectors constituting a system's principal design, and more and more as a set of principal design decisions governing a system [22, 31]. With this, the idea to gather the architectural knowledge about a software system became a focus of the software architecture community. Key in this context is to document not only the components and the connectors which constitute an architectural structure, but also the rationale for an architectural design using means such as architectural decision models. An architectural decision

model (ADM) documents the decision-making process leading to an architectural design in terms of the architectural design decisions (ADDs) made and their relations (e.g., follow-up decisions, implication dependency). The ADDs and their relations are collected in a structured form, for example, by using text templates and/or diagrammatic modeling based on a meta-model for ADDs [23, 32, 37].

In this paper, we address the decision making for a particular kind of software architectures: software architectures based on one or more software platforms. A *software platform* is a collection of software sub-systems, like communication middleware and databases, and interfaces which together form a reusable infrastructure for developing a set of related software applications. To build a concrete application by reusing software artifacts in a platform, the platform lays out a customization and configuration process on top of its interfaces [16]. A software platform, therefore, abstracts from details inside and underneath the platform and thereby facilitates developing, maintaining, and deploying domain-specific software applications. Today, many software systems are based on software platforms. Examples include SAP R/3 for enterprise resource planning, the Facebook Platform for social networks, Amazon's S3 for distributed data storage, Google's Android and Apple's iOS platforms for mobile applications, and Mobicents as a reusable VoIP infrastructure.

Architectural design decisions (ADDs) must be captured for each and every architecture when being designed. Documenting ADDs in a disciplined manner is tedious work consuming critical amounts of time. Besides, ADDs depend on the amount of knowledge available at a relatively early time in the software development process; and architects cannot leverage any scaffolding in ADD documentation processes [18]. At the same time, certain design decisions taken in a given technical domain, such as SOA and service-based platform integration, are found to be taken repetitively. This is due to certain design decisions reflecting established design knowledge in the field; or certain technology or development process choices being firm requirements in the field. It has been proposed to base the application-generic knowledge in decision models on software patterns [18, 38] by reusing the recurring knowledge embodied in the pattern descriptions. Lacking prior work, to the best of our knowledge, we addressed the following research question:

*What are recurring architectural design decisions on service-based platform integration documented by existing software patterns and pattern collections?*

Early works on architectural decision modelling (such as [32]) stressed application-specific architectural knowledge, while in more recent works (such as [37]) also decision models for application-generic architectural knowledge have been proposed [33]. When integrating heterogeneous service platforms for supporting domain-specific software applications, decisions ranging from high-level architectural decisions to implementation-technology-dependent and application-dependent decisions must be tackled to specify the system's architecture. Motivated by these findings on

multi-level decision-making, we investigated:

*What are the levels of decision making when designing an architecture for service-based platform integration?*

To address these two research questions, we conducted a series of three qualitatively-driven studies on architectural decision making. To identify patterns relevant for service-based platform integration solutions, as well as their potential relationships and the relevant forces and consequences of applying the patterns, we performed a qualitative systematic literature review [24]. We reviewed in depth 402 patterns from 11 pattern collections. The result was a candidate pattern language containing 29 patterns (plus 11 referenced patterns). From this pattern language we derived a pattern-based architectural decision model. Next, we interviewed platform experts [27, 29] to validate the architectural knowledge documented in the pattern language and the architectural decision model. Finally, we participated in an industry case study [29] on service-based platform integration in the context of a European research project to learn from the industry partners about the decision-making process.

By integrating the results of the three studies to answer the two research questions, we arrived at two major contributions: First, we documented a pattern language and a pattern-based architectural decision model for service-based platform integration [25], validated and refined through expert interviews and a case study. Second, the analysis of the decision-making process has led to a model identifying the levels and the stages of architectural decision making in service-based platform integration.

The remainder of this paper is structured as follows. In Section 2, we introduce a motivating example for service-based platform integration. Section 3 describes in detail the steps we followed in our multi-method empirical study, and Section 4 presents the results of this qualitative study. In Sections 5 and 6 we discuss the limitations of our approach and the learned lessons from our study respectively. Finally, we discuss the related work in Section 7 and summarize our conclusions in Section 8.

## 2 Motivating Example

To illustrate the research context of service-based platform integration, we give an example (see Figure 1) extracted from the industrial case study on industry automation reported in Section 4: Three heterogeneous platforms, a Warehouse Management System (WMS), a Yard Management System (YMS) and a Remote Maintenance System (RMS), are integrated to allow an operator application to utilize the services provided by these platforms. The YMS manages the scheduling and coordination of trucks in a yard, as well as the loading and unloading of the goods from these trucks. The WMS handles the storage of the goods (or storage bins) into racks via conveyor systems. The RMS system is connected to the warehouse to monitor every incident occurring in the warehouse and the yard. It also supports

remote communication of operators and workers in the warehouse and the yard.

An operator application uses the services of the three platforms via a domain-specific virtual service platform (VSP) which performs service-based platform integration. The overall architecture is schematically illustrated in Figure 1. The VSP must handle various integration aspects including interface adaptation between the platforms; integration of service-based and non-service-based solutions; routing, enriching, aggregation, splitting, etc. of messages and events; handling synchronization and concurrency issues, and so on. To design the details of such integration solutions and the applications on top of it, for all the integration aspects high-level architectural decisions as well as application-specific decisions need to be considered. Furthermore, decisions regarding the technologies employed in the platforms, the applications, and the VSP must be made.

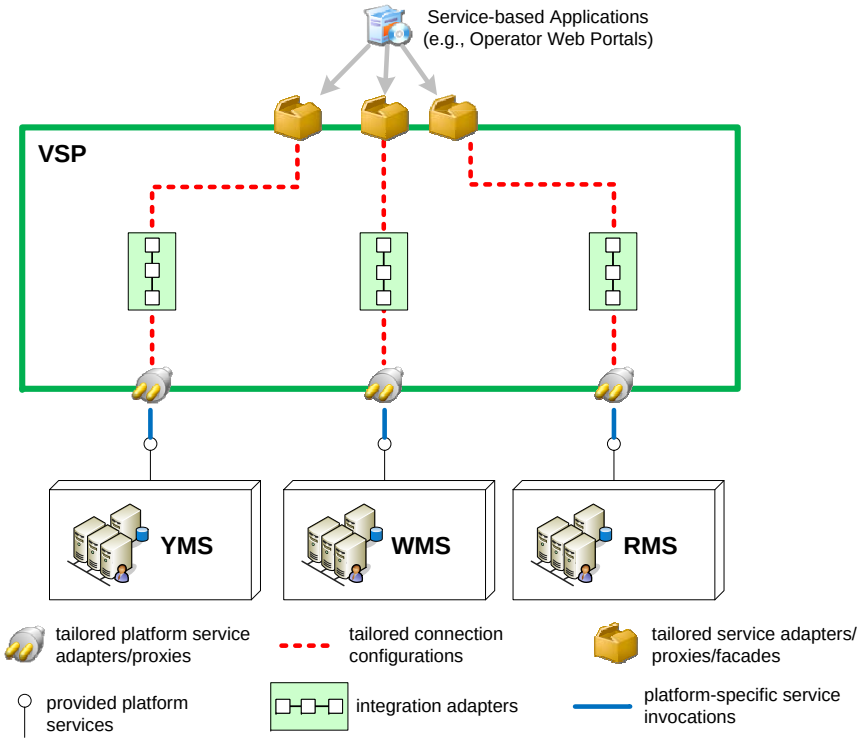


Figure 1: Service-based Platform Integration

## 3 Research Study Design

When designing our study, we faced the problem of the two research questions being substantially different in terms of their views on architectural decision making; while pattern descriptions relate to architectural decisions

and decision drivers content-wise, the issue of decision-making levels requires a process view. A single research method could not accommodate both views. Also, the research questions required both exploratory *and* confirmatory approaches. For example, after having identified candidate pattern material and architectural design decisions as initial results, their *recurring* character should be explained based on practitioners' experience. At the same time, the research questions are closely related, given that different patterns can address different decision-making levels. Consequently, we adopted a multi-method design [30], following a sequential timing for three qualitative study projects.

Figure 2 gives an overview of our research study design: The first method applied was a systematic literature review from which we distilled an initial pattern language and a preliminary architectural decision model according to the procedures in [18, 38]. The systematic review step was performed by the authors in cooperation with a forth postdoc researcher (i.e., the authors in [25]; referred to as *reviewers* hereafter). The second instrument was intended (1) to validate and to improve our pattern language and decision model, as well as (2) to explore how decision making in platform integration architectures is performed. For this, we performed semi-structured interviews [27] with experts on three platforms used in industry. The interviews' data were synthesized using the constant comparison method [29] and used to revise the pattern language and the decision model. Based on the integrated interview findings, we designed a case study (e.g., the study proposition) which was then performed as the third and final inquiry step. With this, we were able to document specific design decisions required in the decision-making process. The second and the third research steps were taken by the three authors. In the subsequent sections, we elaborate on the individual study parts in greater detail.

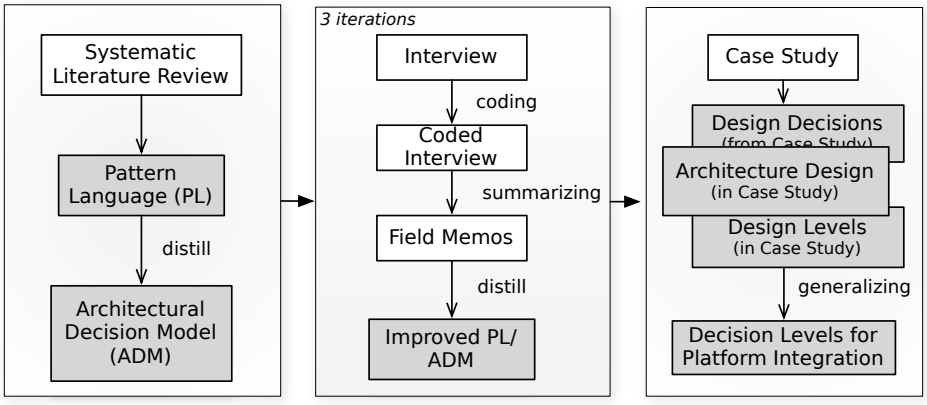


Figure 2: An Exploratory, Sequential, Qualitative Multimethod Design

### 3.1 Systematic Literature Review

As an initial step, a systematic review of the pattern literature was conducted to identify, gather, and filter relevant pattern descriptions from pattern sources on distributed system design [6], enterprise application architecture [14], messaging [20], remoting middleware [35], service design [10] and process-driven SOA [19]. In addition, software design [15] and software architecture [1, 4] patterns have been consulted. The goal of this systematic review was to derive a solution space for the technical domain of service-based platform integration, by integrating selected subsets of these pattern collections into a dedicated pattern language. By studying the forces and consequences of the patterns of the resulting pattern language, an architectural decision model was derived with certain patterns entering the decision model as decision alternatives and options (see [18, 38]).

The practice of systematically selecting and integrating parts of existing pattern languages while adding new, context-specific relations between the integrated patterns has been reported before [1, 5]. While pattern catalogues, pattern collections, and pattern languages have been used before to extract and populate decision models [18, 38], our approach based on a systematic pattern literature review (following the guidelines by [24]) is novel. The steps performed in the systematic review process are documented below.

#### 3.1.1 Review Questions

The questions guiding the systematic review were the following:

$RQ_1$ *Which patterns or pattern collections exist that support designing service-based software system?*

$RQ_2$ *Which patterns or pattern collections document integration of service-based software systems?*

$RQ_3$ *Which patterns and pattern collections describe the architecture and design of an integration platform?*

#### 3.1.2 Pattern Search

The process for searching relevant pattern descriptions and pattern collections was performed manually by the authors in a coordinated manner. In a first step, the authors held a brainstorming session and identified initial pattern candidates based on their experience in the field. Second, conferences, journals, and book series specific to the pattern community were selected. This selection corresponds to the established stages of publishing pattern material, with pattern descriptions and pattern collections first being disclosed to a pattern audience at a PLoP conference. The PLoP conference series guarantees that the pattern material has been reviewed and edited under the guidance of a shepherd, reviewed by a program committee, and discussed in a writer's workshop. From there, the pattern material may

be submitted to a pattern journal, such as LNCS Transactions on Pattern Languages of Programming (TPLoP), which subjects the material to the additional scientific review process of an academic, archival journal. Alternatively, pattern material may grow into a book publication. We considered papers originating from 33 conferences proceedings of PLoP and EuroPLoP, 2 issues of an archival journal (TPLoP), 5 pattern collection books, and 25 pattern books.

#### 3.1.3 Inclusion and Exclusion Criteria

Patterns and pattern collections, published till February 20th, 2012 were included, provided that the articles met certain minimum requirements: Firstly, the pattern or pattern collection concerns one of the review questions reported before. Secondly, the article presents one or more patterns, in one of the shapes established in the various pattern communities. In particular, the patterns are described in an established pattern form [8] as single patterns, pattern compounds, pattern stories, pattern catalogues, or pattern languages (see [5]). In our in-depth analysis we studied all patterns with regard to the review questions and excluded those that are not addressing one of the review questions. In total we selected 11 sources with 402 patterns for further in-depth analysis [1].

#### 3.1.4 Quality Assessment

Each pattern publication identified was evaluated according to the following criteria: (1) At least a single version of the pattern collection must have been reviewed by a pattern audience (e.g., a writers' workshop at a PLoP conference, TPLoP journal review); (2) At least three known uses are reported; (3) The pattern or pattern collection was considered by the related work [37] on documenting architectural decision in the SOA technical domain.

#### 3.1.5 Pattern Extraction

The data extracted from the search phase were the publication sources, a categorisation of the pattern material (e.g., single pattern, pattern story, etc.), short summarizing descriptions, pattern forces and consequences and other referenced patterns. In accordance with the review guidelines in [24], the extraction was performed by the authors independently from each other.

#### 3.1.6 Synthesis

We selected 29 out of the 402 patterns in our in-depth analysis for inclusion in the pattern language, plus 11 patterns that are referenced by the pattern language. Detailed results are presented in Section 4.1.

[1] More technical details are included in the technical report [26].

Table 1: Excerpt of interview instrument

| Question | Type |
|---|---|
| *Adaptation and Integration* | |
| 1.1 Can the services from the source platform be directly used in the VSP platform? | closed |
| *Interface Design* | |
| 2.1 Have the platform services been exposed as services using standard interfaces/technologies? | closed |
| *Communication Style* | |
| 3.1 How important is performance for the connection? | open |
| *Communication Flow* | |
| 4.1 What kinds of data are/will be exchanged between the integrating platforms? | open |

### 3.2 Interview Data Collection and Analysis

The interview instrument was carefully designed using the guidelines by Hove and Anda [21]. We performed 3 interviews with 9 experts from 3 different companies that offer the 3 platforms introduced in Section 2. The platform experience of the interviewees varied from 1 to 4 years, and most of them had more than 3 years of industry experience. Almost all of them had experience with services and were either software designers or developers for the platforms. The interview instrument was mainly based on the decision model and consisted of 4 categories (*Adaptation and Integration*, *Interface Design*, *Communication Style* and *Communication Flow*) and 29 questions. This questionnaire comprised open-ended, as well as close-ended questions. Out of the 29 questions, 24 questions were based on general architectural knowledge and 5 concerned technical platform details with focus on the preparation of the case study. Table 1 contains an excerpt of the interview instrument.

The interviews varied between 100 and 150 min. in length. The interview data was recorded during the interviews in field notes which were then transformed into a structured format through a process of coding. That is, we used a coding scheme to categorize decisions, decision alternatives, and levels/stages of decision making to map the interviewees' answers to concepts and terms like patterns in our pattern language, relationships between patterns and correlations between application-generic and application-specific decisions. For example an interviewee's statement that *"the interface can be accessed remotely without any changes"* implies the pattern REMOTE PROXY as a code.

After each interview data analysis, the pattern language and the decision model were reviewed through the process of *data saturation* [17]. In particular, new patterns and new connections between the patterns were added to the pattern language and decision categories and levels/stages of decision making were documented. Apart from that, patterns and pattern connections that were considered to be either irrelevant or superfluous were selected as candidates for removal in the next iterations.

### 3.3 Case Study Research

Our integration case study had both a confirmatory and an exploratory nature. Our task was to discuss and design together with the platform experts architectural views to reflect the integrated system architecture based on four integration scenarios. First of all, we studied to which extent the pattern language corresponds to the platform integration domain and the architectural decision model helps to navigate in the design space. We evaluated the applicability and appropriateness of the pattern language and the decision model by making decisions in the context of the case study using these assets as our main guidance. Apart from that, we observed the case study design systematically to gain insight into the decision-making process and derive new hypotheses. The design of a common case study for the integration of the three platforms allowed us to define and explore new decision levels, correlations between application-generic and application-specific design decisions, and different stages of the decision- making process.

## 4 Results

### 4.1 Pattern Language and Architectural Decision Model

Having synthesized a set of 29 patterns (plus 11 referenced patterns), the authors distilled a pattern language by assigning each pattern to one (or several) of the previously identified thematic categories, resulting in 6 *Adaptation and Integration* patterns, 6 *Interface Design* patterns, 8 *Communication Style* patterns and 9 *Communication Flow* patterns. The full pattern language is reported in a patterns publication [25]. The pattern language documents relations between the patterns, within and between the four categories. In Figure 3, the 6 *Integration and Adaptation* patterns and their relationships are depicted. The patterns are related in four ways: First, a pattern can represent a *variant* of a more generic pattern description (e.g., REMOTE PROXY) as a variant of PROXY). Second, patterns can play the role of alternative (exclusive-or) or complementary (inclusive-or) design practices when applied at the same design level (e.g., for integration and adaptation, or denoting communication styles). As an example in Figure 3, the REMOTE PROXY and the INTEGRATION ADAPTER pattern are alternatives for each other, each realizing an integration platform with different capabilities (i.e., direct proxy vs. interface adaptation). Third, adopting one particular pattern (i.e., INVOCATION ADAPTER) *leads to* evaluating another pattern at the same level of abstraction under certain conditions or requirements (e.g., the COMPONENT CONFIGURATOR if runtime adaptability is needed). Finally, some patterns (e.g., PROTOCOL PLUGIN) are used as part of the solution of a higher-level pattern (e.g., REMOTE PROXY) to realize a certain aspect of the solution (i.e., multi-protocol support).

In the drafting phase of our architectural decision model, we considered such identified pattern relations as indicators of architectural decisions points

to be documented for a single concrete (or even multiple) integration platform architectures. For capturing such recurring, pattern-based architectural design decisions (ADDs), we adopted the following description scheme (see also Figure 3):

- *Decision context*: Arriving at a decision point is motivated by previous decisions taken. Also, the same decision point may be reached several times while constructing an architecture; yet, depending on the given context, different decision options and drivers become relevant. For instance, while Decision 1 in Figure 3 for a given architecture refers to the REMOTE PROXY pattern in the context of the overall integration platform design, the REMOTE PROXY is also relevant in the context of designing the communication flow (e.g., to bridge to a backend platform service).

- *Decision point*: The point of decision making documents the essence of the decision problem. Depending on the decision context (e.g., proxying with or without adaptation), the decision description can refer to the problem and solution statements of decision-related patterns (e.g., REMOTE PROXY and INTEGRATION ADAPTER).

- *Options*: In our pattern-based ADD model, the range of adoptable patterns represent the options space at a given decision point in a decision context. For Decision 1 in Figure 3, applying either the REMOTE PROXY or the INTEGRATION ADAPTER are alternative options.

- *Decision drivers*: The actual drivers for selecting one of the available options can partly be mined from the forces and consequences documented for the decision-related patterns.

Figure 3 illustrates how decisions of our pattern-based ADD model are derived from the pattern language. Table 2 shows 3 exemplary decisions that have been derived from the excerpt of the pattern language shown in the figure.

### 4.2 Interviews and Improvement Iterations

The results from performing the interviews with the platform experts were manifold. The importance of key patterns of our pattern language has been confirmed from the point of view of the independent experts. For example, either PROXIES or ADAPTERS are needed in all cases for invoking the platform services. Some patterns, like SERVICE ABSTRACTION LAYER and EXTENSION INTERFACE were regarded to be useful, but in fact not necessary for the needs of platform integration of the concrete platforms. The industry experts agreed that those patterns are rather needed in the field of enterprise applications. In addition, we found out that patterns omitted in the first version of the pattern language (e.g., PUBLISH-SUBSCRIBER) were used very often by the platform experts. These patterns were added to the

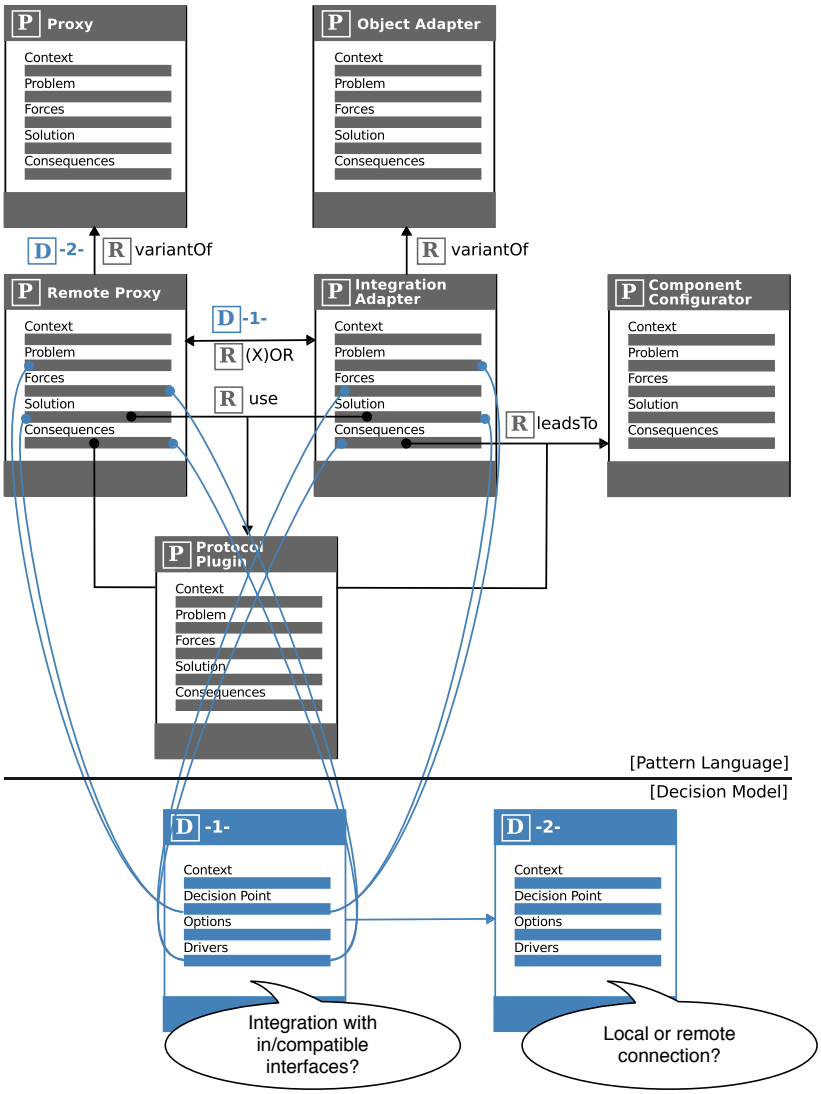


Figure 3: Example excerpt: Pattern Language and 2 Derived ADD Model Fragments

Table 2: Exemplary Decisions in the ADD Model

| Decision Context | Decision Point | Options and Patterns Dependencies |
|---|---|---|
| Integration of a platform service | *D*1 – Which kind of component will be used for integrating the platform service into the service-based integration platform? | • None (direct calls from application to platform)<br>• Integration component with same interface (select pattern PROXY or a PROXY variant)<br>• Integration component with a different interface (select pattern ADAPTER or an ADAPTER variant) |
| A integration component (such as a PROXY or ADAPTER) has been selected for integrating a platform service into the service-based integration platform. | *D*2 – Is the connection between platform and service-based integration platform a local or a remote connection? | • Local (Select local variant of PROXY or ADAPTER (as selected in other decisions))<br>• Remote (Select remote variant of PROXY or ADAPTER (as selected in other decisions)) |
| A integration component (such as a PROXY or ADAPTER) has been selected for integrating a platform service into the service-based integration platform. | *D*3 – Are extra functionalities (such as monitoring, logging, access control, etc.) needed for the invocation path controlled by the integration component? | • Integration component just forwards requests and replies<br>• Integration component triggers one or more extra functionalities and then forwards requests or replies |

pattern language for the next iterations. Apart from that, new connections between the patterns were introduced. Also, we identified some candidate patterns for removal from the pattern language. For example GATEWAY was not considered to be relevant as its functionality can be covered by SERVICE ABSTRACTION LAYER. Along with the improvements to our pattern language we updated our decision model respectively.

These interviews were also used as a preparation for the design of the case study. During this process we gathered existing platform services that were combined in four integration scenarios for the needs of an operator application. We noticed that during the interviews the interviewees could not avoid getting into technical details that mainly focused on the technologies used by the platforms and for their integration with other platforms. These technical details introduce constraints that exclude patterns from the pattern language and narrow down the design space. Hence, another important finding of our interviews was that in the domain of decision making for platform integration has to be performed in multiple application-generic and application-specific levels and in multiple stages (see Section 4.4).

### 4.3 Case Study Design

During the design of the case study the general architectural decisions were refined in stages according to the single platform technologies and the in-

tegration solutions and operator application requirements. The design decisions were documented as instances of the architectural decision model. The outcome of this process was an initial design of the integration solution, for which we used different architectural views to describe the adaptation and communication levels. The names and annotations of the design elements imply in many cases the use of a specific design pattern used. Due to space limitations we only present 2 very small excerpts of a component diagram and communication flow diagram (following the EIP notation; see `http://www.eaipatterns.com/`) in Figure 4 to illustrate the different views that have been developed. In the next section, we illustrate examples of decision making levels and stages in the context of the case study, which also provide more details about the case study design.

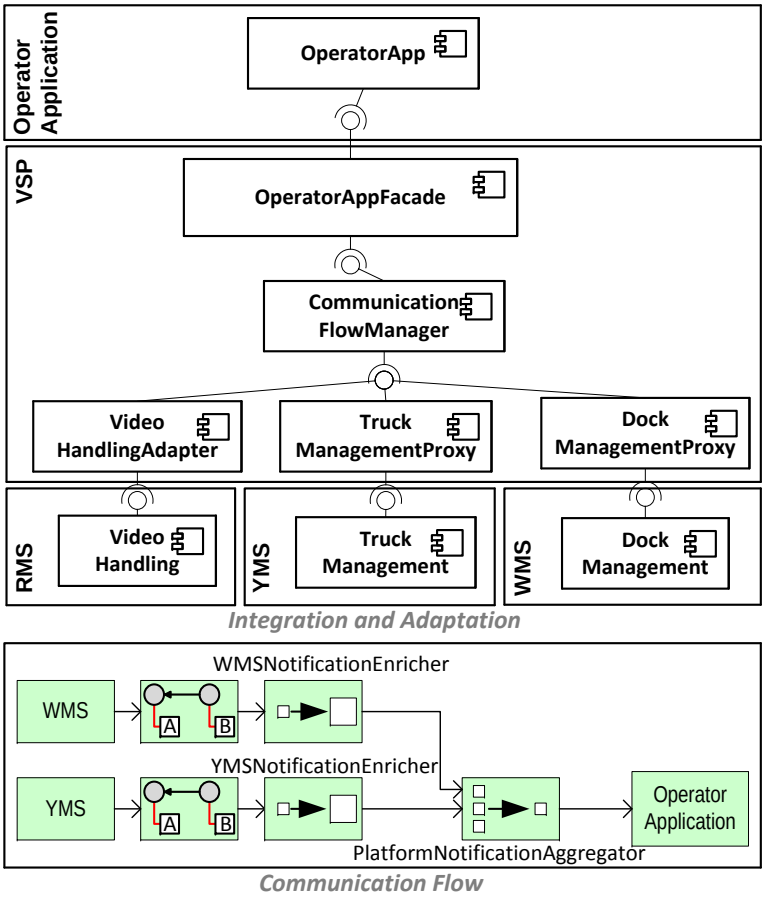


Figure 4: Excerpts of the Case Study Designs

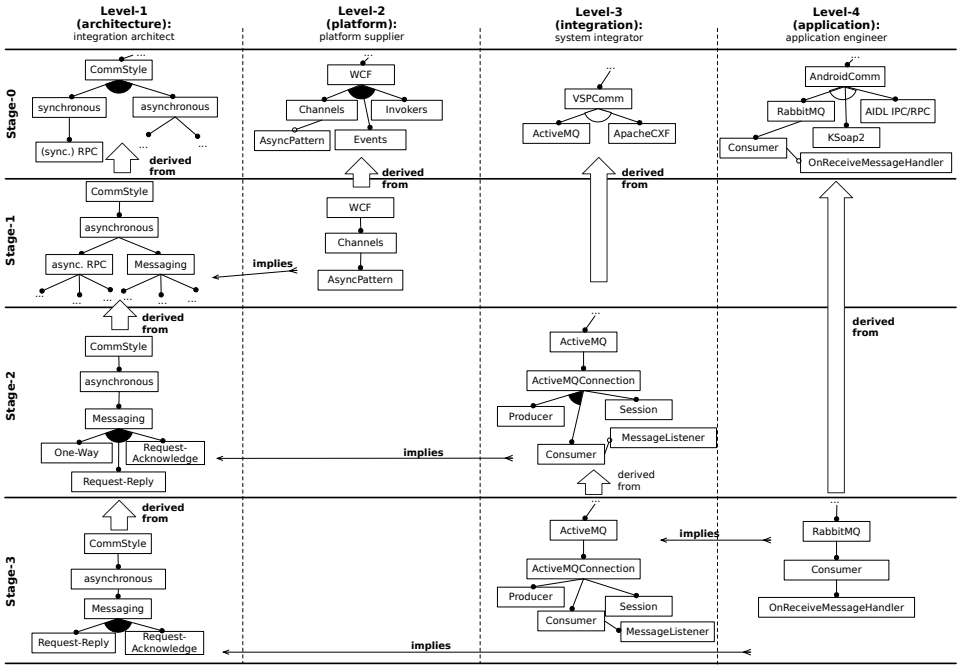


Figure 5: Exemplary Levels and Stages of Decision Making

## 4.4 Decision-Making Levels and Stages

Approaches to modelling architectural decision making usually focus only on the one or two levels of decision making and knowledge acquisition (i.e., the application-generic and application-specific levels; see [33]). From analyzing our interviews, we found that in platform-based software development, multiple *levels* of decisions and multiple decision times (i.e., *stages*) must be considered. Each decision level is commonly performed by a different stakeholder or stakeholder group (e.g., platform supplier, system integrator), often working in different organizations. At each stage, the decision space (e.g., the decisions to be taken and the available options) derives from the decisions taken at prior stages. In addition, at each stage the results of decision making at different levels must be taken into account. That is, starting from generic knowledge on a specific form of software platform integration (i.e., reusable ADDs; *Level-1*), architectural and technical knowledge about each of the platforms integrated (*Level-2*), about each of the integration technologies and solutions used (*Level-3*), and about the applications developed based on top of the platforms (*Level-4*) must be considered.

Inspired by the guidelines on multi-stage, multi-level transformations on feature models by Czarnecki et al. [9], we documented this characteristic decision-making process as observed in our case study. Figure 5 depicts an abbreviated example on decisions related to the communication style in our case study. Here, an architectural decision is illustrated using a transforma-

tion between two feature models. In our case study, and in the example in Figure 5, the four levels represent decisions to be taken by different stakeholder roles (i.e., integration architect, platform supplier, system integrator, and application engineer). The reusable, pattern-based ADDs form the basis of *Level-1* decisions. The original decision space for each role is modelled as an initial feature model at *Stage-0*. Each of subsequent stages sets a discrete decision time, in which decisions at different levels of abstractions are addressed at *Level-1* by the integration architect(s) through reviews of historical decisions taken at the other levels. Levels of abstraction are represented by, e.g., appending increasingly specific branches to the decision outcomes (i.e., here: the feature trees) for each level.

The flow of decisions in Figure 5 illustrates the process of narrowing down the communication style to be adopted by the service-based integration platform. For instance, at *Stage-1* (in *Level-2*) the architects of the WMS platform supplier have opted for the Windows Communication Foundation (WCF) technology for building asynchronous services (i.e., the `AsyncPattern ChannelFactory` approach). As a result, the integration architects at *Level-1* can refute all the synchronous communication alternatives from the initial architectural decision model. Next, at *Stage-2*, the decisions about the integration middleware (*Level-3*) were considered. Given the predominantly asynchronous communication style adopted by the integrated platforms, the platform integrator chose a message-oriented middleware (ActiveMQ) to drive the integration platform. For the *Level-1* decision space, this meant to refute all the communication style patterns except those related to MESSAGING. Reviewing the decisions taken for the operator application (*Level-4*) at the final stage (*Stage-3*), it turned out that the engineers of the Android-driven application had decided for the RabbitMQ message-oriented middlware and that the application had been built around required notifications about the status of its requests (i.e., using the special callback technique: a `OnReceiveMessageHandler`). Reviewing this decision step meant to specialize the decisions available to the integration architects for the operator application connection even further. That is, any one-way MESSAGING can be excluded for decisions. Also, this application-level decision demanded a particular configuration of the message connection in the VSP (*Level-3*), i.e., mandatory message producers and message consumers, to deliver the notifications to the client application.

We have observed this decision scheme in our case study. The dimensions (levels and stages) result from the nature of distributed decision making (e.g., given that the developers of platforms, applications, and integration solutions are often not working in the same organization). However, still the decisions they make influence the design space left open for later decisions, and each additional stage introduced, excludes or refines decisions that must be made in the subsequent steps. We hence propose to extend the existing architectural decision making concepts with support for multiple levels and stages of decision making along those lines. Our general model of the artifacts at the different decision-making levels and their architectural

knowledge derivation relationships is shown in Figure 6.

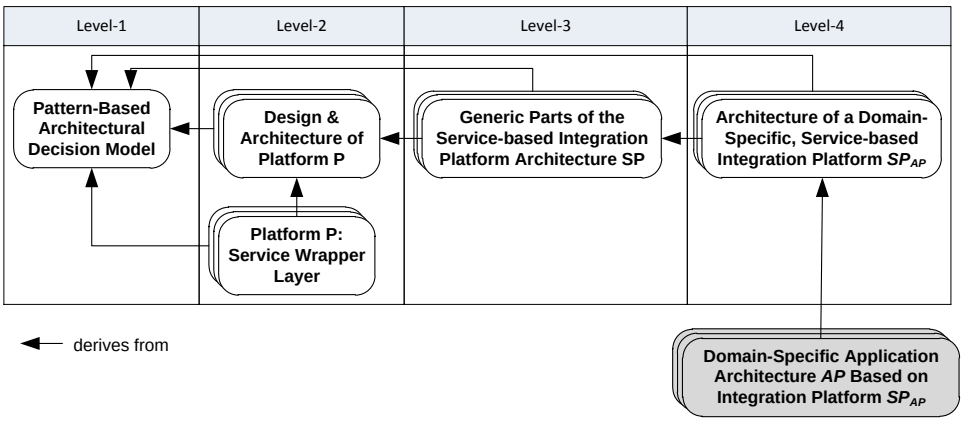


Figure 6: Artifacts and their Architectural Knowledge Derivation Relationships in Different Levels of Decision Making

# 5 Limitations and Threats

**Systematic Literature Review** In conducting the systematic review of pattern literature, we deviated from the original guidelines [24] in important ways: Most importantly, we conducted a purely manual search over a confined selection of conference proceedings, journals, and book series; rather than a semi-automated search process supported by (scientific) search engines. While this practice is consistent with closely related work on pattern-based architecture decision modeling [33, 38], there is the risk that we have missed relevant pattern material. It simply might be the case that relevant architectural knowledge regarding service-based platform integration has not yet been documented in pattern form. We think this threat is a small threat, as we have found pattern material for all aspects of the considered cases and found no evidence in our interviews that relevant architectural knowledge is missing. As for the manual search, we explicitly excluded pattern materials which had not been published at any of the pattern community venues.

While there is a strong bias towards pattern-specific literature sources, this is reduced by the established quality assurance procedures in the pattern community (e.g., shepherding, writers' workshops, maturing pattern formats from single patterns to pattern languages).

To reduce the overall bias of personal judgement (e.g., due to individual research interests, experiences, time constraints), the search step, the quality-assessment step, and the extraction step were performed by each of the reviewers independently from each other. The results were then synthesized in joint re-iterations. In addition, in the overall sequential research design, we validated our resulting pattern language and decision model through interviews with 9 independent industry experts. Despite these efforts, an authors

bias cannot be completely excluded.

**Interviews** To ensure that our conclusions from the interviews are valid we discuss how we dealt with the threats to external and internal validity [27, 36]. As with all qualitative studies, there is the threat to validity with regard to the generalizability of results (external validity) that the small size of considered cases is not enough to generalize the results to other cases of service-based platform integration. We have tried to limit this threat by focussing on patterns as sources of architectural knowledge. In our point of view, it is more likely that similar established best practices are used for other cases of service-based platform integration than if we would have analysed novel, innovative approaches. Apart from that, our interviewees come from three different broad domains (warehouse management, yard management, telecommunications) which of course does not permit us to do any statistical generalization but it offers a broad span of domains in our sample.

As with all qualitative studies, there is the threat to validity that the authors themselves have been an instrument in the study (internal validity) and might have accidentally influenced the study's outcomes or made misinterpretations e.g., because of limited knowledge, wrong prior assumptions, or personal bias. We tried to avoid this threat by keeping an open mind, encouraging the interview participants to talk and rather observed than steered the discussion, and tried to design open-ended and close-ended questions to get the widest range of feedback. However, it cannot be fully excluded that we somehow influenced the results by our own participation in the qualitative studies or our own interpretation.

Regarding the threats to construct and interpretation validity we used *observer triangulation* [27], i.e., multiple observers and interviewers, so that three different researchers were involved.

**Generalizability** The limitations of the literature search in our systematic-review setup imply that our findings, namely the pattern language and the derived architectural decisions, cannot be reused for similar architecting activities in related SOA settings *with different emphasis*. With the focus on the integration and adaptation view, our review questions and the subsequent pattern search did not cover related SOA concerns such as monitoring or middleware framework design. Pattern material, and patterns, even if identified partly, have not been incorporated. Due to the systematic literature filtering and quality assessment, possible connection points to such concerns might have been missed.

Another important threat to generalizing the findings of our study is the level of knowledge of the people involved in different phases. As for the systematic review and pattern extraction, the reviewer group consisted of a pattern expert and senior researcher, two experienced postdoc researchers and a doctoral student focussing on pattern-related research [25]. This heterogeneous distribution of pattern knowledge among the authors has certainly affected the preparation of the review protocol, the quality assessment of the

pattern material, and the extraction of pattern data for drafting the pattern language. By continuously switching the roles of extractor and checker [3], and multiple iterations over the selected pattern material, this effect might have been substantially lowered, yet not neutralized. The expert opinions collected from the interviews are directly dependent on the 9 experts' experience with the platforms, service-oriented architecting, middleware technologies and software patterns. However, the opinions were collected from a relatively mature expert pool with 7 out of 9 interviewees having more than 3 years of industry experience.

While the aforementioned limitations and risks threaten the generalizability of our results, our methodological approach and the sequential multi-method research design [11] can be applied for comparable research activities on pattern-based architecting.

# 6 Lessons Learned

Empirical methods in software engineering are marked by a high level of investigation costs [11]. The time effort required by the authors for preparing and conducting the systematic review was considerable, caused by face-to-face coordination meetings, numerous phone conferences over a period of two months, the role shifting during pattern extraction etc. Identifying and approaching experienced subjects for the interview and case study steps was facilitated by the involvement of the authors in a large-scale European research project with major industry partners.

In our study, we have learnt that using software patterns facilitates iterative architectural decision making [18]. The actual forces and consequences in pattern descriptions document abstracted choices at a given decision point. While our pattern language does not document ADDs for a concrete integration platform architecture, it identifies observed designs for a family of integration platform architectures. In the case study, the pattern language was reused by the architects for sketching a concrete architecture by referencing pattern descriptions from within ADD descriptions. In addition, the pattern form complemented the qualitative research methods used in our study. The structured presentation of patterns and pattern collections facilitated the systematic review. Patterns proved to be an important communication vehicle between the interviewers and the interviewees to bridge their different technology backgrounds.

At the same time, when preparing our study, we became aware of documented limitations of software patterns as empirical research objects. Especially forcing subjects in experiments into using and applying architectural patterns (e.g., the MVC pattern [11]) and design patterns (i.e., VISITOR) resulted in observations indicating an *increased* level of development and maintenance effort. The complexity of learning and comprehending pattern material affected the observations. Such effects have more systematically been reported for research designs involving patterns and artificial software

artifacts of lower complexity (toy applications). While the earlier studies adopted experimental designs, we learnt two lessons from these. First, our research design should not impose design decisions (in terms of patterns) onto our subjects [11]. Second, the architecting process must be observed in the context of a real, not artificially constructed development project. The first risk was mitigated by confronting the experts not directly with the selected SOA patterns, but rather with design decisions derived from our pattern language. With this, pattern played only an indirect role for the observed architecting process. As for the second risk, the architecture designed in our case study applies to a large-scale software prototype which is to be delivered in the context of a European research project; as such, the architecture is not specific to or was not created for the purpose of our study alone.

# 7 Related Work

In this work we create reusable ADDs in the context of platform integration based on design patterns. Many ADD approaches propose prescriptive ADD meta-models [7, 32, 39] that introduce relationships between decision alternatives and activities related to them. By reusing and linking ADDs to patterns as design artifacts [33, 38], documenting architecture decisions and design rationale can be substantially facilitated. For instance, Capilla et al. [7] consider architectural patterns as concrete decision alternatives using a structured Wiki as an ADD documentation tool and a SOA-centric industry case study. Patterns on adapting and integrating platform services [4, 19], on enterprise application integration [14] and on remoting middleware and messaging systems [20, 35] have been documented in the literature, but not with focus on service-based platform integration as in our pattern language.

In the design decision literature many approaches consider multiple levels of decision making and distinguish between application-generic and application-specific knowledge. Examples of generic knowledge are software patterns [4, 15], architectural styles and reusable architectural decisions [38]. Decision models [22, 32] and models created in architecture description languages (ADLs) are examples of application-specific knowledge. A number of approaches (e.g., [32, 38]) capture technology-specific knowledge in the fields of decision meta-models or decision templates. Zimmermann et al. [39] introduce four levels of the decision making process: the executive decision level (process and requirement analysis patterns), the conceptual decision level (high-level architectural patterns), the technological decision level (design and remoting patterns) and the implementation decision level (concrete technology options). So far, however, the integration of multiple levels of application-generic and application-specific architectural decisions has not been studied systematically. In our work, we study the inter-decision connections and observe the decision making process in the context of service-based platform integration and we abstract the different

levels and stages of this process.

Our multi-level and staged decision making approach is inspired by the staged configuration of feature models [9] which is used for product line configurations. In contrast to this approach we support the decision making on the architectural level rather than on the concrete software characteristics. In addition, not all required knowledge is known in the early stages, but the decision models get concretized and decision models from previous stages get refined through iteration cycles. To the best of our knowledge, an approach for defining stages in architectural decision making does not exist in the literature so far.

# 8 Conclusions and Future Work

In this empirical work, we employed multiple qualitative methods to explore the practise of architectural decision making in service-based platform integration. First, we performed a systematic literature review from which we distilled a pattern language and a pattern-based architectural decision model. In the next stage, we performed interviews with platform experts and conducted a case study. Our intention was to validate both the pattern language and the decision model and to investigate the decision making process in this context. The results of these research steps were a refined pattern language and a reusable architectural decision model. In addition, we propose a model capturing the four levels of decision making for platform integration, as identified in our case study. Based on this model, we documented design decisions for all levels and stages of decision making.

The evidence presented in this paper indicates that the notion of decision levels and decision stages applies to architectural decision making in more general. As for platform-driven software development, the four decision levels (e.g., architecture, platform, integration, application; see Section 4.4) could be characteristic and are likely to apply to other, platform-like software development approaches (e.g., software product lines). This motivates us to follow up on this conjecture in our future research. Our finding of multiple levels and multiple stages in architecture decision making also relates to the way architectural decision-making techniques are designed and how these decision-making techniques are classified [12]. For example, our finding converges with the recently proposed decision-making technique by Zimmermann et al. [39], also signalling the need for integrating leveled decision making. In systematic reviews on decision-making techniques (such as [12]), leveled and staged decision making have not yet been considered, for example, in terms of discriminating between domain-specific groups of decision makers (as represented by our four levels, e.g., integration architect, platform supplier).

As for our reusable, pattern-based architectural decision model (ADM), on the one hand, we plan to refine it further by performing further qualitative studies (e.g., interviews, case studies) with an increased number of

participants. On the other hand, and based on the refined ADM, we want to assess its cost-benefit balance [13] for documenting architectural design rationale empirically. For example, while benefits from pattern-based decision documentation have been indicated [18, 33, 38], the use of patterns can also cause a documentation overhead due to the extra (or, initial) learning effort caused by the pattern form.

**Acknowledgements** This work was partially supported by the EU FP7 project INDENICA, grant no. 257483.

# References

[1] P. Avgeriou and U. Zdun. Architectural patterns revisited – A pattern language. In *Proceedings of EuroPLoP 2005*, pages 1–39, Irsee, Germany, July 2005.

[2] Paris Avgeriou and Uwe Zdun. Architectural Patterns Revisited – A Pattern Language. In *Proc. 10th Annu. Europ. Conf. on Pattern Languages of Programs*, pages 1–39, Jul. 2005.

[3] P. Brereton, B. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *J. of Syst. and Softw.*, 80(4):571–583, 2007.

[4] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal, editors. *Pattern-Oriented Software Architecture – A System of Patterns*. Wiley, 2000.

[5] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern-Oriented Software Architecture – On Patterns and Pattern Languages*. Wiley, April 2007.

[6] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern-Oriented Software Architecture – A Pattern Language for Distributed Computing*. Wiley, 2007.

[7] Rafael Capilla, Olaf Zimmermann, Uwe Zdun, Paris Avgeriou, and Jochen Küster. An Enhanced Architectural Knowledge Metamodel Linking Architectural Design Decisions to other Artifacts in the Software Engineering Lifecycle. In *Proc. 5th Europ. Conf. Softw. Architecture*, volume 6903 of *LNCS*, pages 303–318. Springer, 2011.

[8] James O. Coplien. *Software Patterns*. SIGS Books, 1996.

[9] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged Configuration Through Specialization and Multi-Level – Configuration of Feature Models. *Softw. Process: Improvement & Practice*, 10(2):143–169, 2005.

[10] Robert Daigneau. *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and Restful Web Services*. Addison-Wesley, 2012.

[11] Davide Falessi, Muhammad Babar, Giovanni Cantone, and Philippe Kruchten. Applying empirical software engineering to software architecture: challenges and lessons learned. *Emp. Softw. Eng.*, 15:250–276, 2010.

[12] Davide Falessi, Giovanni Cantone, Rick Kazman, and Philippe Kruchten. Decision-making techniques for software architecture design: A comparative survey. *ACM Comput. Surv.*, 43(4):33:1–33:28, 2011.

[13] Davide Falessi, Lionel Briand, Giovanni Cantone, Rafael Capilla, and Philippe Kruchten. The Value of Design Rationale Information. *ACM Trans. Softw. Eng. Method.*, 2012. to be published.

[14] Martin Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[15] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994.

[16] Yaser Ghanam, Frank Maurer, and Pekka Abrahamsson. Making the leap to a software platform strategy: Issues and challenges. *Inform. Software Tech.*, 54(9):968–984, 2012.

[17] B. Glaser and A. Strauss. *The Discovery of Grounded Theory.* Aldin, 1967.

[18] N. Harrison, P. Avgeriou, and Uwe Zdun. Using Patterns to Capture Architectural Decisions. *IEEE Softw.*, 24(4):38–45, Jul. 2007.

[19] Carsten Hentrich and Uwe Zdun. *Process-Driven SOA: Patterns for Aligning Business and IT.* Infosys Press, 2012.

[20] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley, 2004.

[21] Siw Elisabeth Hove and Bente Anda. Experiences from Conducting Semi-structured Interviews in Empirical Software Engineering Research. In *Proc. 11th IEEE Int. Software Metrics Symp.*, pages 23–32. IEEE, 2005.

[22] A. Jansen and J. Bosch. Software Architecture as a Set of Architectural Design Decisions. In *Proc. 5th Work. IEEE/IFIP Conf. Softw. Architecture*, pages 109–120. IEEE, 2005.

[23] A. Jansen, J. van der Ven, P. Avgeriou, and D.K. Hammer. Tool support for architectural decisions. In *The 3rd Working IEEE/IFIP Conference on Software Architecture*, page 4, 2007.

[24] B.A. Kitchenham, O.P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering – a systematic literature review. *Inform. Software Tech.*, 51(1):7–15, 2009.

[25] Ioanna Lytra, Stefan Sobernig, Huy Tran, and Uwe Zdun. A Pattern Language for Service-Based Platform Integration and Adaptation. In *Proc. 17th Annu. Europ. Conf. on Pattern Languages of Programs.* Hillside, Jul. 2012.

[26] Ioanna Lytra, Stefan Sobernig, and Uwe Zdun. Architectural Decision Making for Service-Based Platform Integration: A Qualitative Multi-Method Study. Technical Report TR-SWA-20120601, Faculty of Computer Science, University of Vienna, 2012.

[27] C. Robson. *Real World Research - A Resource for Social Scientists and Practitioner-Researchers.* Blackwell Publishing, 2002.

[28] Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture – Patterns for Concurrent and Networked Objects.* Wiley, 2000.

[29] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Trans. Softw. Eng.*, 25(4):557–572, July 1999. ISSN 0098-5589.

[30] Abbas Tashakkori and Charles Teddlie. *Handbook of Mixed Methods in Social & Behavioral Research*. Sage Publications, Inc., 2nd edition, 2010.

[31] Richard N. Taylor and Andre van der Hoek. Software Design and Architecture: The once and future focus of software engineering. In *Proc. 2007 Future of Softw. Eng. Conf.*, pages 226–243. IEEE, 2007.

[32] J. Tyree and A. Ackerman. Architecture Decisions: Demystifying Architecture. *IEEE Softw.*, 22(19-27), 2005.

[33] Uwe van Heesch and Paris Avgeriou. A Pattern-based Approach Against Architectural Knowledge Vaporization. In *Proc. 14th Annu. Europ. Conf. on Pattern Languages of Programs*, volume 566 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[34] Oliver Vogel. Service Abstraction Layer. In *Proceedings of EuroPLoP 2001*, Irsee, Germany, 2001.

[35] Markus Völter, Michael Kircher, and Uwe Zdun. *Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware*. Wiley, 2005.

[36] C. Wohlin, P. Runeson, M. Host, C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering – An Introduction*. Kluwer Academic Publishers, 2000.

[37] O. Zimmermann, T. Gschwind, J. Kuester, F. Leymann, and N. Schuster. Reusable Architectural Decision Models for Enterprise Application Development. In *Proc. 3rd Int. Conf. Quality of Softw. Architecture*, volume 4880 of *LNCS*, pages 15–32. Springer, Jul. 2007.

[38] Olaf Zimmermann, Uwe Zdun, Thomas Gschwind, and Frank Leymann. Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method. In *Proc. 7th Work. IEEE/IFIP Conf. Softw. Architecture*, pages 157–166. IEEE, 2008.

[39] Olaf Zimmermann, Jana Koehler, Frank Leymann, Ronny Polley, and Nelly Schuster. Managing architectural decision models with dependency relations, integrity constraints, and production rules. *J. of Syst. and Softw.*, 82(8):1249–1267, 2009.

# A Appendix

Table 3: Pattern collections studied in the systematic literature review

| Authors | Title | Pub. Year | Source | Category | Nr. of patterns selected | documented |
|---|---|---|---|---|---|---|
| Gamma et al. [15] | Design Patterns – Elements of Reusable Object-Oriented Software | 1994 | Addison-Wesley Book | PL | 3 | 23 |
| Buschmann et al. [4] | Pattern-Oriented Software Architecture – A System of Patterns | 2000 | Wiley Book | PL | 1 | 16 |
| Avgeriou and Zdun [2] | Architectural Patterns Revisited – A Pattern Language | 2005 | EuroPLoP | PL | 1 | 24 |
| Buschmann et al. [6] | Pattern-Oriented Software Architecture – A Pattern Language for Distributed Computing | 2007 | Wiley Book | PL | 3 | 114 |
| Fowler [14] | Patterns of Enterprise Application Architecture | 2003 | Addison-Wesley Book | PL | 4 | 50 |
| Hohpe and Woolf [20] | Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions | 2004 | Addison-Wesley Book | PL | 11 | 65 |
| Völter et al. [35] | Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware | 2005 | Wiley Book | PL | 5 | 31 |
| Daigneau [10] | Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and Restful Web Services | 2012 | Addison-Wesley Book | PL | 1 | 28 |
| Hentrich and Zdun [19] | Process-Driven SOA: Patterns for Aligning Business and IT | 2012 | CRC Press Book | PL | 1 | 33 |
| Schmidt et al. [28] | Pattern-Oriented Software Architecture – Patterns for Concurrent and Networked Objects | 2000 | Wiley Book | PL | 2 | 17 |
| Vogel [34] | Service Abstraction Layer | 2001 | EuroPLoP | SP | 1 | 1 |

[a] ADAPTER$^{IA}$, FACADE$^{ID}$, PROXY$^{IA}$, OBJECT ADAPTER$^{IA}$

[b] PIPES AND FILTERS$^{CF}$, PUBLISH&SUBSCRIBE$^{CS}$

[c] REMOTE PROXY$^{IA}$, PUBLISH&SUBSCRIBE$^{CS}$, ASYNCHRONOUS COMPLETION TOKEN$^{CS}$, OBJECT ADAPTER$^{IA}$

[d] DATA TRANSFER OBJECT$^{ID}$, REMOTE FACADE$^{ID}$, GATEWAY$^{ID}$, DATA MAPPER$^{CF}$

[e] REMOTE PROCEDURE CALL$^{CS}$, REQUEST-REPLY$^{CS}$, CORRELATION IDENTIFIER$^{CF}$, MESSAGING$^{CS}$, MESSAGE$^{CS}$, MESSAGE CHANNEL$^{CS}$, POINT-TO-POINT CHANNEL$^{CS}$, PUBLISH-SUBSCRIBE CHANNEL$^{CS}$, MESSAGE ROUTER$^{CF}$, CONTENT-BASED ROUTER$^{CF}$, SPLITTER$^{CF}$, AGGREGATOR$^{CF}$, CONTENT FILTER$^{CF}$, CONTENT ENRICHER$^{CF}$, MESSAGE SEQUENCE$^{CF}$, MESSAGE TRANSLATOR$^{CF}$

[f] PROTOCOL PLUG-IN$^{IA}$, POLL OBJECT$^{CS}$, RESULT CALLBACK$^{CS}$, FIRE AND FORGET$^{CS}$, SYNC WITH SERVER$^{CS}$, MARSHALLER$^{CF}$

[g] REQUEST/ACKNOWLEDGE$^{CS}$, REQUEST/ACKNOWLEDGE/POLL$^{CS}$, REQUEST/ACKNOWLEDGE/CALLBACK$^{CS}$

[h] INTEGRATION ADAPTER$^{IA}$

[i] COMPONENT CONFIGURATOR$^{IA}$, EXTENSION INTERFACE$^{ID}$

[j] SERVICE ABSTRACTION LAYER$^{ID}$

[k] (SP) single pattern, (PS) pattern story/sequence, (PCom) pattern compound, (PCat) pattern catalogue, (PL) pattern language

Table 4: Interview guide

| Questions on service adaptation and service integration |
|---|
| 1. Can the services from the source platform be directly used in the Virtual Service Platform (VSP)? |
| 2. Can the same service interfaces, as provided by the remote source platform, be used for contracting the VSP services? Do the service interfaces need to be adapted in order to call the services from the VSP? |
| 3. Is there a need for supporting additional functions such as access control, logging, or monitoring through the VSP? |
| 4. Do you want the VSP to support the management of interfaces change (e.g., by starting, stopping, suspending components)? |
| 5. Is it tolerable to stop the system for maintenance? Is it tolerable to lose messages when the service adapters are updated? |
| 6. Do you need to manage the service adapters in a controlled and centralized way? |
| 7. Does the target platform need to support multiple remoting and transport protocols at the same time? |
| **Questions on the service interface design** |
| 8. Have the platform services been exposed as services using standard interfaces or standard technologies? |
| 9. Is a domain-specific application interface needed? |
| 10. Should any services be combined/ integrated into a composite service by the VSP? |
| 11. Are there platform clients requiring different kinds of transport channels (e.g., JMS, SOAP/HTTP, REST) for accessing the platform services? Is there a need for offering a common interface over these different channels? |
| 12. Does the target platform provide a common interface for platform administration? |
| 13. Must versioning and extending service interfaces be supported? |
| 14. What is the expected format of the exchanged messages? Is there any common, canonical message format provided? |
| **Questions on the communication style** |
| 15. Does the platform support asynchronous communication (e.g., messaging, queuing)? |
| 16. Can the platform service be invoked synchronously and/or asynchronously? Do we need to translate from asynchronous to synchronous calls, and vice versa? Is there a batch mode? Are correlation identifiers supported by the platform? |
| 17. Does the platform service expect a result or an acknowledgement from the platform client? |
| 18. How are the messages emitted by the platform service delivered (e.g., using remote callbacks, local callbacks, or polling)? Do you want to use an event-driven or an imperative programming model in the VSP? |
| 19. Must the messages sent by the platform service in a reliable manner? |
| 20. How critical is the message delivery performance from the perspective of the platform service? |
| **Questions on the communication flow** |
| 21. What kinds of data are/ will be exchanged between the integrated platforms? |
| 22. Which data characteristics are critical for routing the data to the correct receivers? |
| 23. What kinds of data are sent by the source platform? What kinds of data are expected by the VSP? |
| 24. Is it necessary to have the data, which is returned by the platform services, processed further before being delivered by the VSP? |