Architectural Decision Models as Micro-Methodology for Service-Oriented Analysis and Design

Olaf Zimmermann[1] Jana Koehler[1] Frank Leymann[2]

[1] IBM Research GmbH
Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland
{olz,koe}@zurich.ibm.com

[2] Universität Stuttgart, Institute of Architecture of Application Systems
Universitätsstraße 38, 70569 Stuttgart, Germany
frank.leymann@iaas.uni-stuttgart.de

Abstract. During the construction of service-oriented architectures, service modelers concern themselves with the characteristics of good services and how such services can be designed. For instance, they look for advice regarding interface granularity and criteria to assess whether existing software assets are fit for reuse in service-oriented environments. There are no straightforward answers to such questions – service identification, specification and realization techniques are required. Service identification and specification are well covered by existing methodologies; for service realization, architectural decision models can be leveraged. At present, the construction of architectural decision models is an education- and labor-intensive undertaking; if such models exist at all, they often are isolated from other artifacts. In this paper, we propose a new engineering approach to service modeling that leverages reusable architectural decision models as its central service realization concept. We outline a multi-level decision tree and position it as a prescriptive service realization methodology for three engagement types observed in practice. The benefits of service engineering with reusable architectural decision models are semi-automatic decision identification in analysis models, improved decision making quality, and better decision enforcement and risk mitigation capabilities.

Keywords: Architectural decisions, methodology, MDA, service design, SOA

1 Introduction

During the early stages of the evolution of Service-Oriented Architecture (SOA) as an architectural style, Grady Booch stated that “the fundamentals of engineering like good abstractions, good separation of concerns never go out of style”, but he also pointed out that “there are real opportunities to raise the level of abstraction again” [6]. When designing large-scale enterprise applications, this raise of abstraction level concerns the *business domains* a company deals with, as well as already existing Information Technology (IT) assets. Business-aligned *software services* organized into an enterprise-scale SOA reside on this increased level of abstraction. However, implemented software services still have to meet domain-specific Non-Functional

Requirements (NFRs) and standard software quality criteria [11]; old and new design challenges arise, e.g., designing services for multiple invocation contexts.

Therefore, there is more to constructing service abstractions of quality and style than identifying abstract software services, specifying them with technical interface contracts such as Web Service Description Language (WSDL) port types, and then applying WSDL-to-code transformation wizards. No single SOA fits all purposes; many architecture design issues and tradeoffs arise. There are no straightforward answers to the *service modeling* questions that arise – *service identification*, *specification*, and *realization* techniques are required.

For service identification and specification, several techniques such as Service-Oriented Modeling and Architecture (SOMA) [2] and others [9][24] exist; for service realization, *architectural decisions* [26] are a promising complementary abstraction. In [34], we defined architectural decisions as "conscious design decisions concerning a software system as a whole, or one or more of its core components. These decisions determine the non-functional characteristics and quality factors of the system". *Architectural decision modeling* is an emerging field in software architecture research [18]. Unlike other approaches to document software, architectural decision models focus on the expert knowledge motivating certain designs rather than the resulting designs themselves. Model elements present architecture alternatives with pros and cons, as well as known uses, and they also capture the rationale behind and the justification for many different types of decisions.

In SOA construction, *service realization decisions* include strategic concerns such as technology and product selections (e.g., composition technology, workflow engine). Finer-grained modeling tasks such as finding the right service interface granularity (operation-to-service grouping, message signature shaping) form a second decision category. Numerous decisions deal with non-functional aspects such as operation transactionality (business-level compensation, system transactions). For instance, imagine a scenario in which a situational data warehouse application on a Personal Computer, an e-commerce software package on a Linux workstation and a custom developed inventory management solution on a central mainframe computer have to be integrated. Such systems typically differ in the way they manage human user access, balance load, synchronize concurrent requests, persist data, protect themselves against security threats, and so forth – their software architectures are different. When being integrated in an SOA, these systems provide services to each other. Even if the service interfaces can be specified in an abstract, business-driven and technology-independent way, the mapping of the abstract interfaces to implementation components during service realization differs substantially in the three outlined environments. As a consequence, the service realization decisions for the three systems differ. For example, using a workflow engine might not be possible for the situational application, the software package might impose interface granularity constraints, and the mainframe might realize its own transaction monitor.

In this paper, we propose an *engineering approach to service modeling*. We treat service realization decisions as first-class entities that guide the service modeler through the design process. We capture these decisions in machine-readable models. This SOA knowledge is organized in a reusable *multi-level SOA decision tree*, including a conceptual, a technology, and an asset level. The tree organization follows Model-Driven Architecture (MDA) principles, separating rapidly changing platform-

specific concerns from longer-lasting platform-independent decisions. Architecture alternatives in the conceptual level are expressed as SOA patterns. An underlying meta model facilitates automation of service realization decision *identification*, *making*, and *enforcement*: Meta model instances (models) can be created from requirements models and reference architectures, and shared across project boundaries. The meta model also enables decision *dependency modeling* and *tree pruning* – making one decision has an impact on many other decisions. For example, a workflow engine is only required if process-enabled service composition has been decided for. In the resulting process-enabled SOA [15], transaction management settings must be defined consistently for various architecture elements such as process invoke activities, service operations and database drivers.

Explicit dependency modeling has another key advantage: the decision tree can serve as a *micro-methodology* during service design, operating on a more detailed level of abstraction than general purpose methods such as the Rational Unified Process (RUP) [16] and the service modeling approaches described in the literature [2][9][24]. Our approach is complementary to these assets; e.g., the decisions can be organized along the RUP phases such as inception, elaboration and construction.

The remainder of this paper is organized in the following way: Section 2 derives the problem statement motivating our work from state of the art and the practice. Section 3 then presents structure and content of our SOA decision tree. Section 4 explains how this SOA decision tree can be used as a micro-methodology for SOA. Section 5 presents related work; Section 6 discusses the benefits of our approach and how we applied it in practice. Section 7 concludes and gives an outlook to future work.

2 The Service Modeling Problem

Methods like SOMA define the tasks within the service modeling process, SOA patterns present proven solution designs. However, the detailed technical design steps between business-level service identification and pattern instantiation on implementation and runtime platforms still are covered rudimentarily only. Detailed modeling guidance leading to a prescriptive modeling algorithm is desired that helps answering the following question:

How to design and develop "good", business-aligned service abstractions from analysis-level business process models and technical requirements?

As a corollary, the question arises how good services can be characterized. For example, what does business alignment mean from a technical standpoint, and what is the appropriate service granularity in a certain domain context? When trying to answer these questions, development projects start from vaguely articulated requirements, documented as high-level business process and/or use case models created by business analysts. In many cases, these models are defined informally or semi-formally only. However, eventually formal service descriptions have to be defined so that the realized services can be deployed to some IT infrastructure such as an application server or transaction monitor. We jointly refer to these issues as service modeling or *Service-Oriented Analysis and Design* (SOAD) [35]. Currently, these

issues are among the most frequently discussed topics in the industry and academia; we have not participated in a single SOA effort yet in which such service modeling aspects have not been a major concern.

Elements from several existing service modeling methodologies and techniques served us well when dealing with these issues. For example, SOMA covers top-down service identification in business process models and other business analysis artifacts. Service specification typically is also addressed well [13]. However, we noticed a gap between these two steps and detailed technical service realization aspects encountered on SOA construction projects. Existing patterns and general purpose method extensions are informative and educational, but often too coarse grained and incomplete. For example, advice regarding service-enablement of existing legacy systems typically is weak, and transaction management is not covered in detail. Technology and vendor recommendations (often called "best practices") are not integrated sufficiently, often causing quality problems and unnecessary duplication of efforts.

3 An Architectural Decision Model for SOA Construction

The actors involved in SOA construction are *business analysts*, *service architects*, and *service developers*. When Model-Driven Architecture (MDA) concepts are applied, these actors create a Platform-Independent Model (PIM) of the design based on a Computing-Independent Model (CIM) of requirements analysis results. They transform the PIM into one or more Platform-Specific Models (PSMs) and eventually into code. Therefore, it is natural to organize the architectural decision models according to MDA principles as well, applying principles such as layering and separation of concerns. Therefore, we propose three levels of decision model refinement, the *conceptual*, the *technology*, and the *asset level*. In addition, we see a need for an overarching *executive level*, comprising of decisions of strategic relevance. Executive decisions impact the project as a whole [18]. They influence all other decisions.

To harvest already gained knowledge, we synthesized an initial *SOA decision tree* from our own project experience [33][36] and the literature [10][15]. Each decision node describes a single, concrete design issue. We describe the decision nodes according to the following informal representation of an underlying meta model [26]:

- *Decision name* and unique *identifier*.
- *Problem statement*, either in question form or a single paragraph.
- *Scope* of the decision, linking the decision model to design model elements.
- *Decision drivers*, a list of key NFRs and software quality factors driving the design; the pattern community use the term *forces* synonymously.
- *Architecture alternatives* listing the available design options with their pros, cons, and known uses. On the conceptual level, these are architectural patterns. On the technology level, they represent technical choices such as protocol and design pattern selections; the asset level is concerned with open source and commercial product selection and configuration.
- *References* linking in literature such as short overviews, in depth tutorials.
- *Recommendation*, depending on the decision type either a simple "do/do not" rule of thumb, a weighted mapping of forces to alternatives, or a pointer to

more complex analysis process to be performed outside of the decision model. An example for a simple rule of thumb is a commonly agreed best practice: for example, WS-I recommends document/literal as SOAP communication style and bans rpc/encoded [29]. For the design of transaction management boundaries on the other hand, no simple recommendation can be given; a more sophisticated algorithm capturing decision making heuristics and proven alternatives as patterns is required. Decision drivers include business semantics, fault handling and resource protection needs, and NFRs.

- *Lifecycle management* information such as decision owner, project phase, validity timestamp, modification log, and decision enforcement.
- *Outcome* and *justification* of made decisions per decision instance.

We have captured 130 such SOA decisions so far. Table 1 lists selected ones from all four levels of abstraction introduced above, including their scope attribute and some of the alternatives available. The decision naming indicates dependencies, e.g., between the various decisions dealing with transactions.

Table 1. Excerpt from initial SOA decision tree

Tree level	Decision node (scope)	Alternatives
Executive	Platform/language/tool preferences (global)	e.g. J2EE or LAMP
Conceptual (PIM)	Service composition technology (process)	Workflow vs. custom code
	Transaction management strategy (process)	System transaction vs. business transaction
	Transaction management pattern (process)	None vs. single transaction vs. several transactions
	Transaction attribute (operation)	None vs. new vs. join [31]
	Message exchange pattern (operation)	Request-reply vs. one way
	In and out message breadth (operation)	Single vs. multiple parts
	In and out message depth (operation)	Flat vs. nested payload
Technology (PIM/PSM)	Workflow language (process)	Business Process Execution Language (BPEL) vs. other
	Service container (service)	SCA vs. J2EE vs. CORBA vs. .NET vs. other
	Java service provider type (service)	EJB vs. plain Java object
	SCA transaction qualifiers (operation)	See SCE specifications [21]
	EJB transaction attribute (operation)	Defined in EJB specification
	Message exchange style and format (operation)	WS-*/SOAP vs. REST/JSON vs. other
	Transport protocol binding (operation)	HTTP vs. reliable messaging
	SOAP communication style (operation)	Document/literal vs. rpc/literal vs. rpc/encoded
Asset (PSM)	Workflow engine (process)	Vendor or open source (IBM WebSphere Process Server, ActiveBPEL, etc.)
	SOAP message exchange engine (service)	e.g. Apache Axis, Codehaus XFire, vendor engines such as IBM WebSphere engine
	Service provider sourcing (service)	Buy, build, adapt

The generic service granularity discussion leads to several decisions dealing with in and out message signature design, e.g. single vs. multiple message parts and flat vs. deeply nested payload (e.g., XML documents). On the technology level, there is a transport protocol binding decision with service operation scope – SOAP/HTTP or reliable messaging are the alternatives. About 40 other service operation realization decisions exist. One of about 20 process realization decisions is the choice of workflow language, e.g. Business Process Execution Language (BPEL) [20]. The service container decision is closely related; BPEL can only be used if supported by the selected container, for example Service Component Architecture (SCA) [21].

Other decisions not shown in the table deal with security: cryptographic algorithms as means of integrity preservation are available on the transport and on the messaging layer. Message-level XML and Web services security or transport-level HTTPS/SSL are two of the available candidate assets.

4 The SOA Decision Tree as SOA Micro-Methodology

Once the various SOA decisions have been captured in the tree, and the decision dependencies have been modeled explicitly, the decision model can guide the practitioner through the design process. This is a SOA domain-specific method engineering approach, which aims to provide finer grained guidance than traditional artifact- and activity workflow-centric methodologies. The individual SOA decision is the central metaphor. Figure 1 shows selected *decision topics* and atomic decisions on five levels of abstraction, and illustrates the guiding role of the SOA decision tree.[1]

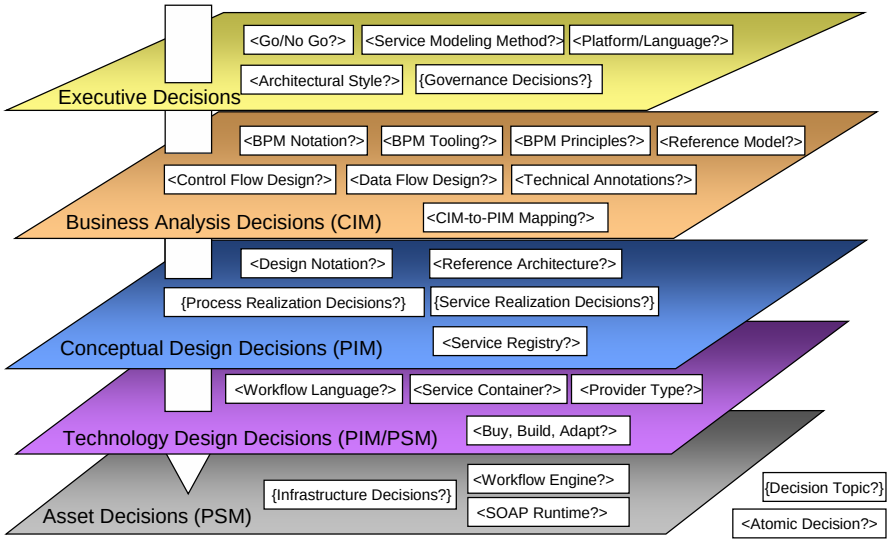

Fig. 1. SOA decision tree in guiding role (micro-methodology)

[1] In line with the design decision literature, we refer to the SOA knowledge structure as a tree; if in the formal sense it actually is a directed, not necessarily acyclic, graph.

Atomic decisions such as <ReferenceArchitecture?> appear in paired lower/greater-than signs. Decisions can be grouped into decision topics, which are embedded in curly braces. For example, topic area {Process Realization Decisions?} contains the process-scoped decisions that Table 1 assigned to the conceptual level. The selection of a <ServiceCompositionTechnology?> is an example.

Section 3 introduced the global executive decisions. In Figure 1, we also added *business analysis decisions* as another level of our micro-methodology. This level deals with decisions about the various Business Process Modeling (BPM) notations, tools, and techniques. While our main focus is technical service realization, we show this layer here to illustrate that our micro-methodology can work with different business modeling approaches, accepting the output of them as analysis input to the technical design.

The vertical arrow does not imply that our micro-methodology is a strict top-down waterfall process; the ability to backtrack and revisit higher-level decisions because of feedback from the lower layers is a key concept in our approach. For example, a certain asset might not support a pattern selected on the conceptual level.

A topic area on the asset decisions level is {Infrastructure Decisions?}: Once the logical design has reached a reasonable level of detail, the physical layout of the solution can be designed, including service deployment onto hardware, and network topology layout. Naturally, the detailed decisions in this group depend on many decisions on higher levels. For example, if stateful services exist, a session handover concept is required, at least in clustered environments.

Navigating through the tree. We provide a single point of entry into our SOA decision tree, a global project <Go/No Go?> decision. Having passed this entry point, executive decisions like selection of <Architectural Style?>, <Service Modeling Method?> and <Engagement Type?> are the next decisions that have to be made. The outcome of these decisions defines the detailed path through the tree. So far, we have predefined the three ways through the tree for three engagement types, which are roughly equivalent to the maturity levels in the Service Integration Maturity Model (SIMM) in [3] and the stages in [15].

Web services enablement of a single component to achieve cross-platform interoperability is a simple engagement type with many known uses, but limited strategic importance. Therefore, many executive decisions, e.g. regarding governance and service lifecycle management, are not required and can be removed from the tree. About 40 technical decisions remain to be taken per service.

A second, emerging engagement type is the introduction of a *service choreography layer implementing an end-to-end business process.* Such engagements address increased pressure from business in the areas of operational efficiency. For example, a process might have to complete within 24 hours. Enforcing such a business rule requires active process instance tracking, which can be achieved by workflow technology. This engagement type is a superset of the Web service enablement; decisions about BPEL usage also have to be made in this engagement type. The scenario from Section 1 is an example; traversing all process and service realization decisions including those in Table 1 comprises a single iteration through our micro-methodology.

Enterprise-wide SOA enablement requires a full traversal of all tree levels. All decision nodes are applicable, and extensions are likely to be required. For example, an enterprise-wide <Service Registry?> must be selected; many *governance decisions* and more executive decisions have to be made.

MDA positioning. If MDA principles are followed, architectural decisions drive *model transformations* between the levels. Our conceptual level is a PIM, the technology level has both PIM and PSM characteristics (depending on the viewpoint), and the asset level is a PSM. In a MDA transformation chain for SOA, decision models can be created and transformed just like design models. Figure 2 shows the resulting three-step MDA transformation chain:

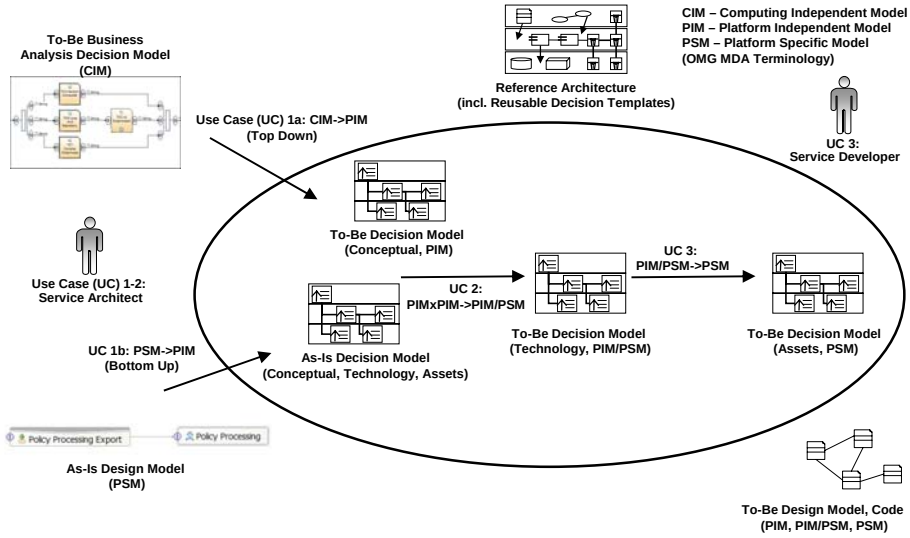

Fig. 2. Architectural decision models in three-step model transformation chain for SOA

Step 1 consists of two transformations: Step 1a transforms to-be requirements business analysis decision models into an initial conceptual design decision model; Step 1b is a reverse transformation from as-is design models describing existing assets to a full instance of the SOA decision tree. Steps 2 and 3 correspond to the conceptual to technology and technology to asset level transitions in Figure 1. One pass through these steps comprises a single iteration in the micro-methodology.

Dependency management and tree pruning. Dependency propagation relationships exist between and within the levels. For instance, the decision to introduce a workflow engine leads to the need for a user management subsystem, because the engine has to manage the status and progress of the process instance execution. This includes assigning activities to users. Users can be systems, if parts of the business process are automated and Web services technology is used to connect an automated client to a process instance. Transaction management is a second example. It can be discussed as

an abstract pattern, which has to be mapped to technology-specific attributes, e.g., SCA qualifiers [21] and Enterprise JavaBean (EJB) deployment descriptor elements.

The number of decision nodes is a major challenge for a broad applicability of our approach; usability and scalability are key concerns. Regarding volume metrics, consider a mid-size SOA construction project automating five business processes with 20 atomic activities each. Let us assume realistic figures: 20 global executive decisions, 25 decisions per process and 40 per activity might have to be taken. If this is the case, already close to 1000 decisions have to be made.

Decisions support systems can provide solutions to such problems. We can leverage the explicit dependency management to actively remove unnecessary nodes as soon as possible. Several opportunities for doing so exist; one of them is to disable decisions and alternatives based on previous decisions. For instance, .NET details are no longer relevant if Java is the language of choice. If SOAP/HTTP has been chosen for the first ten activity services in a process, the next 15 probably will probably use it as well. Developing a more general tree pruning strategy is ongoing and future research work – in the now completed first project phase, our main focus was on structuring and populating the tree.

5 Related Work

In this section, we position our work relative to service modeling, patterns and pattern languages, Object-Oriented Analysis and Design (OOAD), software engineering methodologies, and design decision rationale research.

Service modeling. Service modeling methodologies are subject to current research [2][9][24]. These methodologies cover all phases of service-oriented analysis and design; they are particularly strong in early such as business modeling and service identification. Typically, they reside on higher levels of abstraction than our SOA decision tree; therefore they provide less detailed technical advice than we do. The relationship between these methodologies and our approach is complementary and synergetic; e.g., a candidate service model created with SOMA can service as a starting point for the detailed technical decision making based on our approach.

Patterns and pattern languages. The patterns movement has been highly successful in the past decade [14]. Architecture and design patterns go a long way in supporting practitioners during design and development of enterprise applications. SOA as an architectural style refines many abstract patterns such as `Proxy` and `Broker` [8]. In enterprise application architecture literature, we find service layer patterns and general coverage of transaction management issues, but no specific coverage of SOA. The "Putting it all together" chapter in [10] has inspired parts of our overall SOA design space structure. SOA patterns have emerged over recent years. For example, Zdun [32] defines a pattern language for process-driven SOA.

Practitioners often report difficulties in seeing the big picture when looking at individual patterns and pattern catalogs. Pattern *catalogs* do not discuss how the various patterns are connected. Pattern *languages* address this concern, describing an entire

domain as a consistent and comprehensive set of related patterns, and providing orientation within the solution space via intent, context, and forces discussions. However, most pattern languages have a technology-centric nature; the transition from business-level requirement and NFR analysis to pattern application is described informally if at all. Cross-domain relationships between patterns are discussed rarely.

Patterns have educational character. By definition, patterns reside on a conceptual and/or technical level; none of the existing SOA pattern languages map the patterns to an asset level. For example, transactional workflow patterns [5] do not provide BPEL or SCA mappings, even if these technologies appear as known uses. We believe that such mappings are required. In practice, many architectural decisions have to be taken on the asset level because vendor-specific extensions and limitations exist.

In summary, patterns and pattern languages do not cover service modeling issues such as service granularity or transactionality design aspects with enough detail. Advice from the referenced sources still provides valuable background information in our approach; patterns appear as architectural alternatives on the conceptual level.

OOAD. During our early adoption SOA projects, we employed many OOAD techniques, which inspired the design of our SOAD framework [35]. For example, we often used a combination of system context, use case, and collaboration diagrams during early project scoping workshops. Design-by-contract [23] and responsibility-driven design [30] are two principles that apply to services just as well as to objects. The Classes, Responsibilities and Collaborations (CRC) cards technique [4] is not limited to specifying classes; services and service components can be conceptualized similarly. However, it is key to take service design specific principles into account. For instance, services are invoked via messages, should not have any identity, and preserve as little conversational state as possible. We further discuss the OOAD usage for SOA construction in [37].

Software engineering methodologies. RUP [16] provides a business modeling discipline, which uses UML activity diagrams for process modeling both on analysis and design level. There is a RUP SOA plugin [13], which defines a Service Model artifact. SOA-specific design advice is given informally in technique papers and method extensions. The given advice is helpful, but in our opinion also not detailed and prescriptive enough. For instance, process guidance in RUP workflows stops at *Design Software Architecture* and *Design Service Model* level of granularity. There is some integration of design patterns via recipes; however, detailed architectural decisions to be made are not captured and modeled systematically. Service interface design, communication protocol selection, and transactional runtime configuration issues are examples for such decisions that are typically not covered detailed enough.

Design decision rationale, architectural decision research. Architectural decision capturing [18] is an emerging field in software architecture research, which emerged from work in design decision rationale research [19]. We use several techniques from both fields as part of our SOAD micro-methodology, filling gaps where needed. One popular form of knowledge capturing are *Questions, Options, Criteria (QOC)* diagrams [22]. QOC Diagrams raise a design question which points to the available

options for it; decision criteria are associated with the options. Option selection can lead to follow-on questions. QOC triples are similar to our decisions, alternatives, and decision drivers. Existing work typically focuses on documenting already made decisions, an additional, time consuming obligation even with tool support.[2] With such an approach, the decision viewpoint remains isolated and disconnected from the 4+1 logical, process, development, physical and scenario views on software architecture defined in [17]. In the industry, templates for architectural decision capturing exist [7]. There are cases where predefined decision documents are part of reference architectures, for example in the IBM e-business reference architecture [26]. These assets mainly have documentation character; they do not provide active guidance as our micro-methodology does. Copy-and-paste of static documents is the only way to customize and reuse the assets on SOA construction projects.

6 Discussion

As the examples in this paper have shown, successful service modeling is as not as easy as it might appear at first glance. Much more than simple drill-down from business-level process flow to IT realization is required; many SOA-specific architectural decisions have to be made. Almost all but the most trivial cases require a meet-in-the-middle modeling approach as opposed to a top-down process; existing system reality and software packages constrain the modeling choices. With our multi-level SOA decision modeling approach, we capture the corresponding design advice.

We use the architectural decision metaphor in a more dynamic fashion than existing work. In our approach, architectural decisions do no not just have passive reference character, but serve as a micro-methodology. Decisions are identified from business requirement models, legacy system descriptions and earlier decisions. Because SOA is specified and standardized openly, it is possible to leverage domain-specific knowledge captured in SOA reference architectures, principles, and patterns. Standardization and openness have another welcome side effect: service modelers speak the same language – it becomes possible to share architectural knowledge between projects, thus helping to develop economies of scale.

Unlike any other modeling approach or methodology we are aware of, we *push* a detailed initial technical to-do list including available alternatives, pros and cons, known uses, and literature references to the responsible service modeler, bringing in experience from previous projects. This is a significant advantage of our approach; state-of-the-art so far has been that the service modeler had to *pull* the required decision points and possible designs from the literature, personal experience, and personal networks. Some of the required information is available in method browsers nowadays; however, practitioners still have to perform a pull operation.

Our SOA decision tree also serves as a communication vehicle between the actors; feedback regarding practicability and enforcement of decisions can be exchanged this way. Another benefit of this approach is that it facilitates the knowledge exchange across project boundaries. The SOA decision tree also can serve as an analysis tool

[2] Tool support for capturing design decision rationale has been a research topic in the 1990s, but has failed to accomplish industry adoption so far. There is no SOA specific support yet.

during bottom-up service modeling if it captures the architectural decisions once made for a legacy asset that is currently being service enabled. By helping to assess whether an available service operation is suited to implement a certain process activity, a decision model provides buy vs. build decision support. The decision catalog can also serve as governance and risk mitigation instrument; compliance with industry and company guidelines can be ensured. Additional usage scenarios for the decision tree are quality assurance reviews and best practices benchmarks.

While the presented approach is generic enough to be applicable to many architectural styles, it is particularly useful, or even required, in SOA. Decision modeling and SOA share many design goals such as applicability in heterogeneous, complex domains. NFRs of shared services usually are more challenging than those of standalone applications. Services have to handle multiple usage contexts, and clients compete for shared resources. A software service is not just a reusable code fragment, but an enterprise asset, much like a product; service lifecycle management is required on enterprise-wide SOA initiatives, which is simplified by capturing and preserving the rationale behind the original service design in a SOA decision tree.

Project results and action plan. This paper presents both results of our work and an action plan to further enhance the presented concepts and ensure practical applicability. So far, we have validated the presented approach in the following ways:

- We applied the micro-methodology retrospectively to two of our own projects [33][36]. The results of that step led to one of several refactoring iterations, in which we added the lifecycle management attributes to the meta model, as well as explicit support for capturing decision drivers.
- We have implemented tool support for the presented concepts in an Eclipse-based Architect's Workbench [1], as well as a Web 2.0 front end [25]. At the time of writing, two industry projects work with the tool.
- The content of the SOA decision tree could be reused successfully on these projects. In one case 13 of 15 required decisions could be anticipated, in the other 45 of 50.
- We have applied the micro-methodology to areas in which we do not have deep subject matter expertise. For example, we coached two information integration architects so that they could capture their expertise in decision tree form. The study succeeded.
- We are in the process of studying the design decisions and drivers in transactional workflows in SOA in more detail. A draft version of that content is currently under evaluation in our target community.

During these activities and workshops with more than 100 practicing software architects and service modelers, several benefits of the outlined micro-methodology have already become apparent. The approach serves well as an education and knowledge exchange vehicle; subject matter expertise becomes available to less experienced architects. It also increases productivity during the initial project setup activities such as team orientation and candidate asset screening. Numerous SOA case studies exist; therefore, the decision tree can be populated from experience. The MDA positioning and dependency modeling concepts provide traceability between analysis

and design, as well as between design and code. The feedback loops between the levels improve team communication.

For a broader adoption, several challenges have to be overcome. The complexity of the enterprise computing domain leads to a rather large decision tree structure. There are many business domains to be supported, and on the technology and asset levels, thousands of architecture alternatives exist. The change dynamics of the solution space are another challenge: new architecture alternatives arise almost daily. At least the asset level of our decision tree has to be updated whenever a vendor releases a new product version with enhanced features or different non-functional characteristics. Due to these challenges, usability and scalability are key success factors for our micro-methodology. Tree organization and tools under development take these challenges into account. Further validation work is required to assess whether these challenges can be addressed in such a way that a broader practitioner community can employ our micro-methodology successfully on complex SOA construction projects.

7 Conclusion and Outlook

In this paper, we have positioned reusable architectural decision models as a micro-methodology for model-driven service realization. The enterprise computing domain is complex; no SOA fits all purposes. Therefore, service modeling activities always have to be customized for particular project environments; combining elements from several methodologies is a valid option. Methodologies such as SOMA and SOA patterns can and should be leveraged during SOA construction, but must be further enhanced to ensure repeatability, and support quality assurance and reuse strategies.

Architectural decisions provide an additional view on software architecture complementary to the traditional 4+1 logical, process, development, physical and scenario views defined by Kruchten [17]. Decision models for this sixth view on software architecture can be organized according to MDA principles, separating executive, business analysis, conceptual, technology and asset design concerns.

According to our experience, such structured hierarchical decision models can serve as a service realization micro-methodology. In such as micro-methodology, genuine modeling and meta modeling support for SOA decision identification, making, and enforcement are required, as well as dependency and constraint management; many SOA decisions influence each other. In this paper, we introduced such concepts and pre-configured SOA decision trees for three engagement types, simple Web services enablement, process-driven SOA solution, and enterprise-wide SOA. We applied the presented micro-methodology retrospectively to our own SOA projects, and are in the process of validating our concepts together with solution architects and service modelers engaged in industry projects. Web 2.0 and Eclipse-based tool support is available.

Future work includes continuing to harden our SOA decision tree, to further extend the meta model and to develop a decision model population, dependency management and tree pruning tool. We also plan to investigate several advanced usage scenarios for our SOA decision tree, for example project management assistance, software package evaluation and software configuration.

Acknowledgments. We would like to appreciate the input and constructive feedback from numerous members of IBM practitioner and research communities, including David Janson, Bob Johnson, Petra Kopp, Jochen Küster, Christoph Miksovic, Sven Milinski, Frank Müller, Cesare Pautasso, Nelly Schuster, and Jussi Vanhatalo.

References

[1] Abrams S. et al, Architectural thinking and modeling with the Architects' Workbench, IBM Systems Journal Vol. 45, 3/2006
[2] Arsanjani, A.: Service-Oriented Modeling and Architecture (SOMA), IBM developer-Works 2004, http://www.ibm.com/developerworks/webservices/library/ws-soa-design
[3] Arsanjani A., Holley K., Increase flexibility with the Service Integration Maturity Model (SIMM), http://www-128.ibm.com/developerworks/webservices/library/ws-soa-simm/
[4] Beck K., Cunningham W., A Laboratory For Teaching Object-Oriented Thinking, OOPSLA'89 Conference Proceedings, October 1-6, 1989, New Orleans, Louisiana
[5] Bhiri S., Gaaloul K., Perrin O., and Godart C., Overview of Transactional Patterns: Combining Workflow Flexibility and Transactional Reliability for Composite Web Services, in: van der Aalst W.M.P. et al. (Eds.): BPM 2005, LNCS 3649, pp. 440–445
[6] Booch, G.: InfoWorld interview 2004, http://www.infoworld.com/article/04/02/02/HNboochint_3.html
[7] Bredemeyer Consulting, Key Architecture Decisions Template, available from http://www.bredemeyer.com/papers.htm
[8] Buschmann F., Meunier R., Rohnert H., Sommerlad P., and Stal M., Pattern-Oriented Software Architecture – a System of Patterns. Wiley, 1996
[9] Dijkman R., Dumas M., Service-Oriented Design: A Multi-Viewpoint Approach, International Journal of Cooperative Information Systems Vol. 13, No. 4 (2004), 337-368
[10] Fowler M., Patterns of Enterprise Application Architecture, Addison Wesley 2003
[11] International Standards Organization (ISO), ISO/IEC 9126-1:2001, Software Quality Attributes, Software engineering – Product quality, Part 1: Quality model, 2001
[12] Jansen A., Bosch, J., Software Architecture as a Set of Architectural Design Decisions, Proceedings of the 5th Working IEE/IFP Conference on Software Architecture, WICSA'05
[13] Johnston, S., RUP Plug-In for SOA V1.0, http://www.ibm.com/developerworks/rational/library/05/510_soaplug
[14] Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995
[15] Krafzig D., Banke K., Slama D., Enterprise SOA, Prentice Hall, 2005
[16] Kruchten P., The Rational Unified Process: An Introduction, Addison-Wesley, 2003
[17] Kruchten P., The 4+1 View Model of Architecture, IEEE Software, vol. 12, no. 6, pp. 42-50, Nov., 1995
[18] Kruchten P., Lago P., van Vliet H, Building up and reasoning about architectural knowledge. In C. Hofmeister (Ed.), QoSA 2006 (Vol. LNCS 4214, pp. 43-58), 2006
[19] Lee J., Lai, K, What's in Design Rationale?, Human-Computer Interaction, 6(3&4), 251-280,1991.
[20] OASIS. Web Services Business Process Execution Language (WSBPEL), Version 1.1, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel, May 2003
[21] Open SOA Alliance. Service Component Architecture. http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications
[22] MacLean A., Young R., Bellotti V., and Moran T., Questions, Options, and Criteria: Elements of Design Space Analysis, Human-Computer Interaction, 6 (3&4), 1991.
[23] Meyer, B., Object-oriented software construction, Prentice Hall, 2 edition, 2000

[24] Papazoglou M., van den Heuvel W. J., Service-Oriented Design and Development Methodology, International Journal of Web Engineering and Technology (IJWET), 2006
[25] Schuster N., Zimmermann O., Pautasso C., ADkwik: Web 2.0 Collaboration System for Architectural Decision Engineering, Proceedings of the Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'2007)
[26] Tyree, J., Akerman, A., Architecture Decisions: Demystifying Architecture. IEEE Software, 22 (2005) 19-27
[27] W3C. SOAP Version 1.2, W3C Recommendation 24 June 2003, published online at http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, June 2003
[28] W3C. Web Services Description Language (WSDL) 1.1. Published online at http://www.w3.org/TR/wsdl, March 2001
[29] Web Services Interoperability. WS-I Basic Profile 1.1, http://www.ws-i.org/Profiles/BasicProfile-1.1.html, April 2006.
[30] Wirfs-Brock R., Object Design: Roles, Responsibilities, and Collaborations, Addison-Wesley 2002
[31] Witthawaskul W., Johnson R., Transaction Support Using Unit of Work Modeling in the Context of MDA,, Proceedings of the 2005 Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05
[32] Zdun, U., Dustdar, S., Model-Driven and Pattern-Based Integration of Process-Driven SOA Models, http://drops.dagstuhl.de/opus/volltexte/2006/820
[33] Zimmermann, O.; Doubrovski, V.; Grundler, J.; Hogg, K.: Service-Oriented Architecture and Business Process Choreography in an Order Management Scenario, OOPSLA Conference Companion, 2005
[34] Zimmermann O., Koehler J., Leymann F., The Role of Architectural Decisions in Model-Driven Service-Oriented Architecture Construction. In: Skar, L.A., Bjerkestrand A.A. (eds.), Best Practices and Methodologies in Service-Oriented Architectures, OOPSLA 2006 workshop proceedings
[35] Zimmermann, O.; Krogdahl, P.; Gee, C.: Elements of Service-Oriented Analysis and Design, IBM developerWorks 2004, http://www.ibm.com/developerworks/webservices/library/ws-soad1/index.html
[36] Zimmermann O., Milinski M., Craes M., Oellermann F., Second Generation Web Services-Oriented Architecture in Production in the Finance Industry, OOPSLA Conference Companion, 2004
[37] Zimmermann O., Schlimm N., Waller G. and Pestel M., Analysis and Design Techniques for Service-Oriented Development and Integration, pages 606-611 in INFORMATIK 2005 – Informatik LIVE! Band 2, Beiträge der 35. Jahrestagung der Gesellschaft für Informatik