The Role of Architectural Decisions in Model-Driven SOA Construction

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. On Service-Oriented Architecture (SOA) delivery projects, practitioners 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 SOA environments. In this paper, we position architectural decision modeling as a prescriptive service realization technique. We propose a multi-dimensional SOA decision catalog, separating platform-independent from platform-specific concerns and supporting dependency management. The catalog is positioned in a three-stage model transformation chain for SOA.

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

1 Introduction

There is more to constructing service abstractions of quality and style than creating Web Service Description Language (WSDL) files or applying simple WSDL-to-code transformation wizards. Enterprise-scale service abstractions have to meet many ever-changing functional and non-functional requirements, including environmental factors and regulatory compliance requirements; existing assets reside on various platforms. Consequently, no single software architecture fits all purposes; many architecture design issues arise. Service modelers juggle with design tradeoffs at many decision points [1]. There are no straightforward answers to the modeling questions that arise – *service identification*, *specification*, and *realization* techniques are required [2].

For service identification and specification, several techniques have been proposed [3] [4]; for service realization, *architectural decisions* [5] are a promising complementary abstraction. We define 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 [6]. Unlike other approaches to document software, architectural decision models focus on the expert knowledge behind certain designs rather than the actual designs; decision points present *architecture alternatives* with pros and cons [7].

In a SOA construction, service realization decision points include strategic concerns such as technology and product selections (composition technology, business process 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, 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 might 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. Services these systems provide to others have to acknowledge that: 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 reality differs substantially in the three outlined environments. As a consequence, the service realization decisions for the three systems differ. For example, using a business process 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-oriented design*. We treat service realization decisions as first-class entities that guide the service modeler through the design process. In our approach, we explicitly capture SOA decisions in machine-readable models. The SOA knowledge is organized in a *three-level SOA decision tree*, comprising of a conceptual, a technology, and a product/open source asset level. The tree organization follows Model-Driven Architecture (MDA) principles, separating rapidly changing platform-specific concerns from longer-lasting conceptual decisions. Architecture alternatives in the conceptual level are expressed as patterns. A meta model allows developing tools for automation of decision identification, analysis, and enforcement; meta model element instances are created from requirements models. Decision dependency and constraint propagation mechanisms are supported in the tree – making one decision has an impact on many remaining decisions. For example, a business process engine is only required if process-centric service composition has been decided for. Transaction management settings must be defined consistently for various architecture elements (service components, database drivers, etc.).

The remainder of this paper is organized in the following way: Section 2 discusses current architectural decision making practices and their shortcomings. Section 3 then presents our position, outlining a proposal for explicit SOA decision modeling. Section 4 sketches an action plan for further research and concludes.

2 Current Architectural Decision Making Practices

Three common practices on today's SOA delivery projects are to use general purpose software engineering methodologies without decision modeling support, to delegate architectural decision making responsibilities to developers or tools, and to hard wire the architecture inside transformation. These practices have significant shortcomings:

General purpose software engineering methodologies do not model decisions. Existing methodologies focus on *design* rather than *decision* models. There are many design processes and notations such as the Rational Unified Process (RUP) [8] and the Unified Modeling Language (UML) [9]; service modeling techniques are available as extensions [2] [10]. These methodologies and languages focus on defining structural and behavioral service design models, not the dynamics and details of the decision making process. For instance, RUP mentions architectural decisions in its supporting documentation, even in the definition of the term software architecture [8]. However, there is no artifact or meta model element for architectural decisions.

On the other hand, consulting firms recommend documenting all architecturally relevant decisions; structured text templates are available [11]. Without model support, *documenting architectural decisions is time consuming*; keeping the decision log up to date is even more challenging. The identification of architectural decision points is a labor-intensive process with a steep learning curve; decision making often is based on gut feel rather than informed judgment. Enforcing decisions is a manual and error-prone process involving official code reviews and unpleasant auditing efforts.

Developers and/or SOA tools do the architectural thinking. SOA raises the level of abstraction and is about bridging the business-IT gap, so indeed business analysts and developers should communicate directly, and as much routine work as possible should be automated. Therefore, one might think that architectural decision making could be deferred to lower levels of abstraction, e.g. development or deployment roles, or even be automated; MDA transformation tools [12] rather than architects could be seen as the architectural authorities in control of the development process.

However, addressing conflicting non-functional requirements and general software quality factors such as data integrity, performance, scalability, and reliability is a complex, holistic endeavor. The *architectural thinking capabilities of experienced architects are required* to make related decisions, e.g. about transaction boundaries.

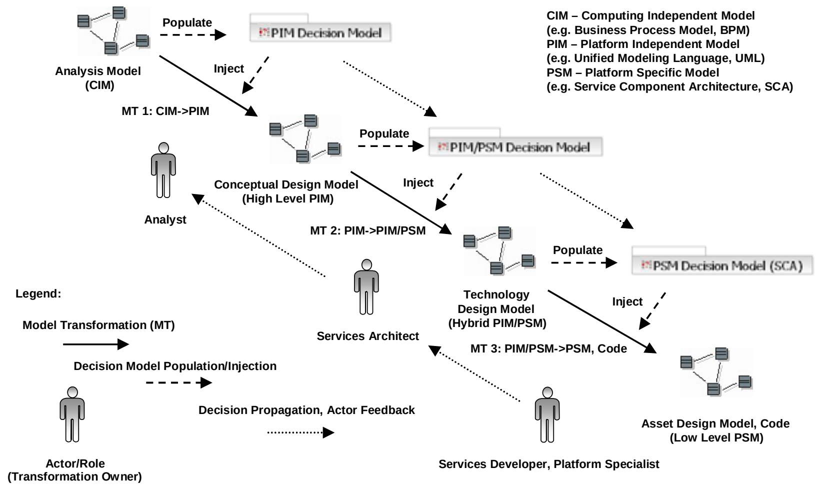
Model transformations hard wire architecture. At present, design-to-implementation transformations typically do not enforce architectural decisions such as defining a service as transactional; neither do they apply appropriate default values. As a result, much design work is left to the developer, who encounters *reconciliation problems* when modifying the generated models according to the architectural decisions and then re-executing the design-to-implementation transformation due to requirement changes. Hence, model transformations can not be regarded as appropriate architectural authorities in most real-world cases (e.g. when agile development is practiced).

In SOA construction, flexibility is a key success factor for the applicability of model transformations. Modern transformations and code generators are configurable e.g. via patterns and templates [13] [14]. However, without access to machine-readable representations of architectural decisions, *configurable transformations remain disconnected from the architectural decisions*, which we view as the primary transformation configuration input. This is a fundamental governance and control issue.

We believe that due to the limitations and shortcomings of these practices, more emphasis on architectural decision modeling is required during SOA construction. In the next section, we outline our proposal and present its advantages.

3 Support for Architectural Decision Models in SOA and MDA

In MDA transformation chains for SOA construction, actors such as analysts, service architects, and service developers typically use three subsequent model transformations to construct platform-specific design models and code from computing-independent requirements analysis models. In such a chain, architectural decisions drive the transformations on all levels. In our work, we position genuinely modeled decisions as *a separate track in the transformation chain* (Figure 1):



Analysis Model (CIM)
Populate
PIM Decision Model
Inject
MT 1: CIM->PIM
CIM – Computing Independent Model (e.g. Business Process Model, BPM)
PIM – Platform Independent Model (e.g. Unified Modeling Language, UML)
PSM – Platform Specific Model (e.g. Service Component Architecture, SCA)
Populate
PIM/PSM Decision Model
Analyst
Conceptual Design Model (High Level PIM)
Inject
MT 2: PIM->PIM/PSM
Populate
PSM Decision Model (SCA)
Legend:
Model Transformation (MT)
Services Architect
Technology Design Model (Hybrid PIM/PSM)
Inject
MT 3: PIM/PSM->PSM, Code
Decision Model Population/Injection
Decision Propagation, Actor Feedback
Asset Design Model, Code (Low Level PSM)
Actor/Role (Transformation Owner)
Services Developer, Platform Specialist

Fig. 1. Positioning of architectural decision models in three-stage transformation chain for SOA

We propose three stages of design and decision model refinement – *Conceptual* (High Level PIM), *Technology* (Hybrid PIM/PSM), and *Asset* (Low Level PSM). On all levels, a design model captures the result of the decision making; there is a corresponding decision model. Decision models are populated from the higher levels and injected into the transformations, starting from analysis model and finishing with code. Dependency propagation relationships exist between the levels. For instance, transaction management can be discussed as an abstract pattern, but has to map to technology-specific attributes, e.g. Enterprise JavaBean (EJB) deployment descriptor entries.

The decision models serve as a communication vehicle between the actors; feedback regarding decision practicability and enforcement can be exchanged this way. An additional use case for the decision models is to assist with the resolution of reconciliation problems. As already explained, model-driven software development approaches encounter this problem when higher-level models are updated manually and subsequent re-executions of generated low-level models overwrite these changes.

We are in the process of synthesizing a *SOA decision tree* from project experience [15] [16] and literature [4] [17]. Table 1 on the following page elaborates on the decisions outlined in the introduction to this paper, and assigns them to the three MDA levels introduced in Fig.1. So far, we have captured about one hundred such decisions and key dependencies between them (e.g. in the transaction management domain).

Table 1. Excerpt from initial SOA decision tree (decision naming indicating dependencies)

MDA Level	Decision point (affected design model element)	Alternatives (subset)
Global	Platform/language/tool preferences (all)	e.g. J2EE or Linux Apache MySQL PHP (LAMP)
PIM (Conceptual)	Service composition technology (process)	Business process engine vs. custom code
	Transaction management strategy (process)	System transaction vs. business transaction [17]
	Transaction attribute (service operation)	None/new/join [18]
	Message exchange pattern (service operation)	Request-reply vs. one way
	Operation-to-service grouping (service)	Single/multiple operations
PIM/PSM (Technology)	Business process engine language (process)	Business Process Execution Language (BPEL) [19], other
	Transaction management pattern (process)	None vs. single distributed transaction vs. several atomic transactions
	EJB transaction attribute (service)	requires, supports, etc. [20]
	Message exchange format and transport protocol binding (service operation)	SOAP/HTTP vs. JMS messaging vs. other
	In and out message signature design (service operation)	Single/multiple parameters; flat/deep XML structures
PSM (Asset)	Business process engine (process)	Vendor or open source (IBM WebSphere Process Server, ActiveBPEL, etc.)
	Transaction attribute (service operation)	e.g. Service Component Architecture (SCA) qualifiers [21]
	SOAP message exchange engine (service)	e.g. Apache Axis, IBM

Existing methodologies must be enhanced with architectural decision support. SOA practitioners desire prescriptive advice for service realization. Such advice must be grounded in project experience and reusable assets, e.g., reference architectures. Method artifacts defining decision models can frame such advice. Architectural decision model instances can become part of SOA reference architectures.

SOAs must be modeled taking 4+1+1 viewpoints. As already explained, no one-size-fits-all SOA exists. The justifications for architectural decisions form a new viewpoint as important as the traditional logical, process, development, physical and scenario ("4+1") views on architecture design defined in RUP [22]. Capturing architectural decisions is a recommended practice independent of the architectural style of choice – in SOA/MDA, modeling decisions becomes a key success factor for delivering values such as increased flexibility, agility, and productivity.

Architects identify and make architectural decisions, not tools. Making architectural decisions is a primary responsibility of the SOA solution architect on a project, not something to be delegated to business analysts, developers, model transformations, or any other roles or tools. One key example is the *buy vs. build decision* (a.k.a. design time enterprise service composition) – should a new service be implemented, or an existing one reused, possibly with adaptations? This decision requires deep technical knowledge, e.g. the transaction attributes of a realizing service component candidate have to be analyzed before the decision is made. It is not acceptable to defer such key decisions until development time.[1]

Model transformations must be configurable and enforce architectural decisions. Flexibility is a major argument for SOA. Hard wiring certain architecture designs in model transformations or letting integration specialists reconfigure the system on every build cycle according to the architectural decisions can not be regarded agile enough if any heterogeneity, integration, and productivity challenges have to be met.

Automation requires formalization and/or meta modeling. An *architectural decision meta model* is therefore required. On top of a decision meta model, powerful tool support can be developed. For example, project-specific instances of the model can be created from business process analysis and other requirements models. Constraint management can reveal architectural mismatches. Marked as resolved, decisions can steer subsequent model transformations via *decision injection* (see Figure 1).

4 Conclusion and Outlook

In this paper, we have positioned architectural decision models as the control center of model-driven SOA construction. Our key messages can be summarized as follows:

- Architectural decisions provide an additional view on software architecture complementary to the traditional 4+1 RUP views. Decision models for this sixth view on software architecture are a required MDA extension.
- The enterprise computing domain is complex, no SOA fits all purposes. Therefore, SOA methodologies and tools should be enhanced with genuine modeling and meta modeling support for SOA decision identification, elaboration, and enforcement. To isolate platform-independent from platform-specific service realization decisions, SOA decision models should adhere to MDA principles. Dependency and constraint management is required, as there are many SOA decisions influencing each other.
- Making SOA decisions is a primary responsibility of services architects.
- Model transformations must enforce architectural decisions and therefore be parameterized. Decision injection is one option to do so.

Our action plan resulting from these considerations is to continue to harden our SOA decision tree, to further extend already existing architectural decision meta models for a SOA and MDA context and to develop a decision model population and injection tool. We also plan to investigate several advanced usage scenarios for our SOA decision tree, for example SOA governance and software package evaluation.

[1] This also is a governance and accountability issue – due to their wide impact, architectural decisions have contractual relevance on professional services engagements.

Acknowledgments. We would like to appreciate the input and constructive feedback from numerous members of IBM practitioner and research communities, including Steve Abrams, Jim Amsden, Jonas Grundler, Matthias Grützner, Markus Kessler, Doug Kimelman, Jochen Küster, Albert Maier, Christoph Miksovic, Karl Schindler, Ian Simmonds, Carl Spencer, Martin Strenge, and Jussi Vanhatalo.

References

[1] 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
[2] Dijkman R., Dumas M., Service-Oriented Design: A Multi-Viewpoint Approach, International Journal of Cooperative Information Systems Vol. 13, No. 4 (2004), 337-368
[3] Arsanjani, A.: Service-Oriented Modeling and Architecture (SOMA), IBM developer-Works 2004, http://www.ibm.com/developerworks/webservices/library/ws-soa-design1
[4] Papazoglou M., van den Heuvel W. J., Service-Oriented Design and Development Methodology, International Journal of Web Engineering and Technology (IJWET), 2006
[5] Tyree, J., Ackerman, A., Architecture Decisions: Demystifying Architecture. IEEE Software, 22 (2005) 19-27
[6] Kruchten P., An Ontology of Architectural Design Decisions, in: Jan Bosch (ed.), Proc. of the 2nd Workshop on Software Variability Management, Groningen, NL, Dec. 3-4, 2004
[7] 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
[8] Kruchten P., The Rational Unified Process: An Introduction, Addison-Wesley, 2003
[9] Booch, G.; Rumbaugh, J.; Jacobson, I.: Unified Modeling Language User Guide, 1999
[10] Johnston S., SOA extension for RUP, IBM developerWorks
[11] Bredemeyer Consulting, Key Architecture Decisions Template, available from http://www.bredemeyer.com/papers.htm
[12] Frankel D. S., Model Driven Architecture: Applying MDA to Enterprise Computing, OMG Press, 2003
[13] Czarnecki K., Helsen S., Classification of Model Transformation Approaches, in Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, Anaheim, October 2003
[14] openArchitectureWare, http://www.openarchitectureware.org
[15] Zimmermann, O.; Doubrovski, V.; Grundler, J.; Hogg, K.: Service-Oriented Architecture and Business Process Choreography in an Order Management Scenario, OOPSLA Conference Companion, 2005
[16] Zimmermann O., Milinski M., Craes M., Oellermann F., Second Generation Web Services-Oriented Architecture in Production in the Finance Industry, OOPSLA Conference Companion, 2004
[17] Fowler M., Patterns of Enterprise Application Architecture, Addison Wesley 2003
[18] 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)
[19] Web Services Business Process Execution Language (BPEL), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
[20] Enterprise JavaBeans (EJB) specification, http://java.sun.com/products/ejb
[21] Service Component Architecture (SCA) specification http://www.ibm.com/developerworks/library/specification/ws-sca
[22] Kruchten P., The 4+1 View Model of Architecture, IEEE Software, vol. 12, no. 6, pp. 42-50, Nov., 1995