Reference Architecture, Metamodel, and Modeling Principles for Architectural Knowledge Management in Information Technology Services

Olaf Zimmermann[1,2], Christoph Miksovic[1], Jochen M. Küster[1]

[1] IBM Research GmbH
IBM Research – Zurich, Säumerstrasse 4, 8803 Rüschlikon, Switzerland
{cmi,jku}@zurich.ibm.com

[2] ABB Corporate Research (since February 2012)
ABB Switzerland Ltd., Segelhofstrasse 1K, 5405 Baden-Dättwil, Switzerland
olaf.zimmermann@ch.abb.com

Abstract. Capturing and sharing design knowledge such as architectural decisions is becoming increasingly important in firms providing professional Information Technology (IT) services such as enterprise application development and strategic outsourcing. Methods, models, and tools supporting explicit knowledge management strategies have been proposed in recent years; however, several challenges remain unaddressed. In this paper, we extend our previous work to overcome these challenges and to satisfy the requirements of an additional user group, presales architects that are responsible for IT service solution proposals. In strategic outsourcing, such solution proposals require complex, contractually relevant design decisions concerning many different resources such as IT infrastructures, people, and real estate. To support both presales and project architects, we define a common reference architecture and a decision process-oriented metamodel. We also present a tool implementation of these concepts and discuss their application to outsourcing proposals and application development projects. Finally, we establish twelve decision modeling principles and practices that capture the practical experience gained and lessons learned during the application of our decision modeling concepts to both proposal development and architecture design work on projects.

Keywords: architectural decisions; architectural principles; DSL; knowledge management; model-driven engineering; outsourcing; SOA; workflow

1 Introduction

The Information Technology (IT) service business comprises a diverse set of practices such as IT consulting, enterprise application development, software package integration, and strategic outsourcing.

Our previous publications focused on modeling and reusing knowledge about the design of enterprise applications that apply Service-Oriented Architecture (SOA) principles and patterns on enterprise application development projects [41][43]. Since then, we have also applied these concepts to other IT services practices and technologies such as software package integration and cloud computing [44]. In this paper, we extend our focus in two ways: 1) we investigate an additional practice and sub-domain of IT services, strategic outsourcing [16] and 2) we transfer our concepts to the early stages of the solution lifecycle (i.e., from post-contract projects to presales proposal work).

Outsourcing solution design is related and similar to the design of software-intensive systems such as SOA-based enterprise applications; the fundamental decisions that are made during the solution design process of an outsourcing proposal are important determinants of complexity and uncertainty. Hence, outsourcing solution design can be viewed as a superset of software design. Not only software applications and software-intensive systems are in scope, but also the IT infrastructures supporting these applications and systems, as well as human resources and physical assets such as real estate. Due to the need to estimate costs accurately, an outsourcing solution design has to be detailed before contracts are signed and billable projects are initiated. The complexity of the outsourcing business would suggest to analyze requirements and client environment in great detail before quoting any exact pricing information; however, market forces dictate a highly agile design approach [7]. Hence, a lot of uncertainty has to be dealt with throughout the proposal work; business decisions and technical decisions go hand in hand. In addition to architects, stakeholders from other fields, including sales executives, project managers, and domain specialists (e.g., for legal matters) are also involved in the solution design (e.g., when deciding how to deal with existing software licenses, contracts, and buildings).

Architectural Knowledge Management (AKM) solutions for software design projects have been developed and successfully applied in recent years [20]. In this paper, we leverage and extend these AKM concepts so that the pre-contract designers of strategic outsourcing solutions are empowered to manage the complexity and uncertainty that is inherent to this business domain and stage of design evolution. Specifically, we present a knowledge management solution that aims at increasing design productivity and proposal quality. This solution combines previous work in the AKM community with concepts from business process management and Domain-Specific Languages (DSLs). Our approach to presenting the solution is:

1. First we compile a set of Architecturally Significant Requirements (ASRs) that characterize the AKM needs in IT services (both in presales and on billable projects). To frame the detailed design and development work, we derive a conceptual tool *reference architecture* (i.e., a set of logical building blocks with their responsibilities and collaborations) from these ASRs next. Our reference architecture is founded on business process management concepts [22], workflow patterns [36], and our previous work [40][43].
2. To refine the design of the novel components in the reference architecture, we present a decision process-oriented *metamodel*. This metamodel defines a

Domain-Specific Language (DSL) that structures the interfaces between (and shapes the internal design of) these components.
3. We feature an implementation of the reference architecture and the metamodel in the *Solution Decision Advisor (SDA)*, first introduced in [24].
4. We report how we validated concepts and implementation during presales usage (in strategic ou tsourcing) and on billable projects (involving enterprise application development).
5. To ease the task of creating architectural decision knowledge, we distill twelve *modeling principles and practices* from user feedback gathered and lessons learned during our knowledge engineering activities on two projects conducted from 2006 to 2011, SDA development and *SOA Decision Modeling (SOAD)* [41].

The combination of reference architecture, the metamodel, and the twelve modeling principles and practices for architectural knowledge management in IT services (addressing the extended scope of both presales design activities and architecture design on projects) is the core contribution of this paper.

The remainder of the paper is structured in the following way. Section 2 introduces the problem domain of knowledge management in IT services and establishes the challenges, functional requirements, and quality attributes that frame our design work. It also provides a solution outline. Section 3 then presents the reference architecture and Section 4 specifies the decision processing metamodel and DSL. Section 5 covers the implementation of our concepts in SDA, Section 6 their validation on real-world proposals/projects. Section 7 reports on the modeling practices and principles distilled from the SOAD and the SDA projects; Section 8 investigates related work. Finally, Section 9 presents our conclusions from this work and gives an outlook to future work.

2 Architectural Knowledge Management in IT Services: Domain Context, Challenges, and Requirements

In our work, we primarily target two particularly knowledge-intensive domains and practices in IT services, enterprise application development and Strategic Outsourcing (SO) solution design (a secondary design goal is that all concepts are general enough to be applicable for other domains and practices). Enterprise application development and integration is investigated in detail in our previous publications, with special emphasis on the SOA style [40][43]; we highlight the specific characteristics of the SO domain now. We also emphasize the importance of pre-contract design in this context.

Decision making for Strategic Outsourcing (SO) proposals. Our user community in SO are solution architects and business decision makers working for presales organizations of IT service providers offering SO services. Not only software applications and software-intensive systems are in scope of such services, but also the IT infrastructures (e.g., servers, storage and networking devices, operating systems, and middleware) supporting these applications and systems, as well as human resources (e.g., help desk staff and system administrators) and physical assets such as real estate (e.g., data centers and offices).

Outsourced IT services comprise a wide range of IT infrastructure such as servers and storage, but also include labor-intense tasks such as help desk and service management [27]. Designing such SO solutions is a complex undertaking; a large body

of knowledge is required to deal with this complexity [10]. Much of the required knowledge has to be gained and applied during the proposal phase (i.e., when scoping a solution prior to contract signature); deep experience in the domain is required in order to be competitive. Proposal teams may have few individual team members, but also dozens of them; many Subject Matter Experts (SMEs) participate only temporarily. SO contracts typically run for several years; the preparation of a contract proposal is a project in its own right [1]. An example of such a project is the formal response to an official Request for Proposal (RfP) [25]. Proposal parameters include technology platform types (e.g., PCs, UNIX, and mainframe), volume (e.g., number of help desk requests, servers, and business transactions) and Service Level Agreements (SLAs) [25]. Two examples of SO solutions from opposite ends of the complexity spectrum are a standardized service to host a software package for a local client and a one-of-a-kind service management solution involving a number of data centers and technology platforms that have to comply with regulatory compliance requirements in multiple countries [1].

In their proposal work, SO solution architects have to follow a rather complex design process that investigates many design concerns, such as country- and industry-specific legal requirements, skill transfer, real estate takeover, and IT system handover (to name just a few) [14]. To address these concerns, a number of fundamental solution design decisions have to be made throughout the project; many of these decisions are interdependent. While many outsourcing requirements and constraints are specified explicitly, e.g., in a Request for Proposal (RfP), other decision drivers are less tangible, e.g., company-internal policies about Intellectual Property Rights (IPR) and the desire to reuse standard offerings in order to be able to operate in a cost-efficient manner.

What is needed in this context is knowledge management support for presales (and project) solution architects, making the fundamental solution design decisions that are relevant for the following handover activities [7].[1]

2.1 Knowledge Management Challenges (Proposals/Projects) and Example

Solution design both in enterprise application development and in SO requires numerous interrelated business decisions as well as technical decisions to be made. Due to market dynamics and the buying power of clients, many of these decisions have to be made before any contract is signed and any payment is made.

To scope our research in this broad and complex industrial setting, we surveyed the literature and interviewed members of the target audience regarding their decision making habits and methods and tools used [2][7][10][14][26][31]. We then filtered/consolidated the answers and grouped them by (business) relevance and coverage (or lack thereof) in existing work. The result of this scoping activity was the following list of challenges:

1. *Scope and scale* challenge: Numerous decisions potentially have to be made for each proposal. These decisions deal with the SO solution scope and the transformation of existing IT infrastructure (i.e., how to leverage or repurpose

[1] In SO, handover means that the IT service provider takes over legal responsibility and daily operations from the IT service requestor after a contract has been signed; this is followed by a series of infrastructural and organizational changes that are applied so that enterprise architecture guidelines are met and the contracted services are delivered in a cost-efficient manner.

resources such as servers and offices). Regulatory compliance requirements are particularly relevant in the scoping and transformation design work. Two sources of such requirements are the Sarbanes-Oxley Act [5] as well as the specifications issued by the US Food and Drug Administration [35].

2. *Priority and order* challenge: It is not always clear in which order these decisions should be made. Such insight might only be available in tacit form (e.g., in the tribal memory of a solution architect community). Focusing on less relevant decisions unnecessarily wastes resources and increases risk.
3. *Data quality and uncertainty* challenge: The decision making input is often incomplete, for instance when IT service requestors do not specify certain technical details early enough. Intellectual property management issues often arise – e.g., both IT service requestors and decision makers on proposal teams may be concerned about amount and scope of information divulged to other stakeholders. Furthermore, it is not always clear how to express uncertainty regarding decision outcomes and when to revisit earlier decisions as more information becomes available or when requirements change.
4. *Consistency and efficiency* challenge: As a corollary to the scope and scale challenge (i.e., large number of required decisions and design options that recur in the domain), the choices for a specific decision (instance) can and should be constrained by the outcome of predecessor decisions already made. Such approach helps to confine the number of selectable options to those that are still eligible and leads to consistent, workable designs.
5. *Reuse and education* challenge: During decision identification and decision making, decision makers welcome guidance. This helps to increase the confidence that a design is adequate and to ensure that industry- or enterprise-wide standards including architectural principles [39] are adhered to. Such guidance is particularly useful when mentoring less experienced IT service professionals (e.g., when practicing training-on-the-job).

As motivated in our previous work, the architects of enterprise applications face similar decision identification, making, and enforcement challenges [40].

Example. The following example is realistic and complex enough to justify and illustrate our design, but simple enough that it can be followed without domain-specific knowledge and experience: An early Decision Point (DP) is to determine whether the solution has to adhere to industry-specific regulations such as the Sarbanes-Oxley Act [5] in the finance industry or the specifications issued by the US Food and Drug Administration [35] in retail and in pharmaceutical businesses (DP-01). Another DP deals with the question whether client-specific Service-Level Agreements (SLAs) have to be adhered to (DP-02). These two decisions have an impact on the decision DP-03 regarding the selection of a service management tool; a wide range of options exist, from simple operating system scripts to commercial products. Depending on the outcome of this fundamental solution design decision, it may be possible to use a shared solution (e.g., a software-as-a-service offering) or not (DP-04). DP-01 and DP-02 always have to be considered (in any order) before DP-03 and then DP-04 can be investigated. DP-04 can be removed from a DP graph when DP-03 decides for a custom service management tool for which no shared solution exists. Figure 1 illustrates this DP graph example:

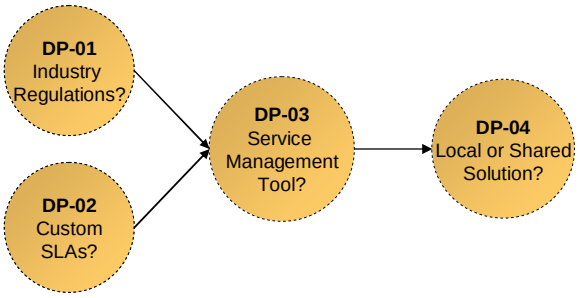

Fig. 1. Example SO/SOA scenario with connected Decision Points (DPs)

The example is general enough to also apply to SOA design (when taking a physical viewpoint, e.g., dealing with the deployment infrastructure of a SOA-based enterprise application and the management of this infrastructure [20]).

2.2 Research Questions and Solution Outline

To overcome the five knowledge management challenges from above, we envision a *Decision Knowledge Processing (DKP)* solution that aims at increasing design quality and project team productivity. Three research questions arise from the challenges:

1. *Content and structure (data and metadata)*: Which information about the decision points should be shared and when/how should it be used? This question is derived from two of the five challenges from Section 2.1, scope/scale (1) and reuse/education (5).
2. *Presentation*: How should this information be displayed to the decision maker so that he/she is able to see both the big picture and all relevant details? This question results from the priority/order challenge (2), and also the consistency/efficiency challenge (4) from Section 2.1.
3. *Processing*: How to deal with uncertainty and incomplete input? We derived this question from the data quality and uncertainty challenge (3) from Section 2.1.

Since a large body of work on architectural knowledge management already exists and our work is conducted in an industrial setting, the following supporting questions arise:

4. *Reuse:* Which existing assets out of the vast array of related work (AKM and other research communities, commercial assets) should be leveraged? Do these assets have to be extended and, if so, how?
5. *Transfer and deployment*: What are the assumptions underlying the solution? E.g., do solution design decisions actually recur in practice? What are the resulting challenges for a production deployment in a commercial setting?

Solution outline. Both practical challenges and research questions framed our solution design and construction (engineering) activities towards the envisioned DKP solution. Figure 2 provides an overview of the key elements of this solution:

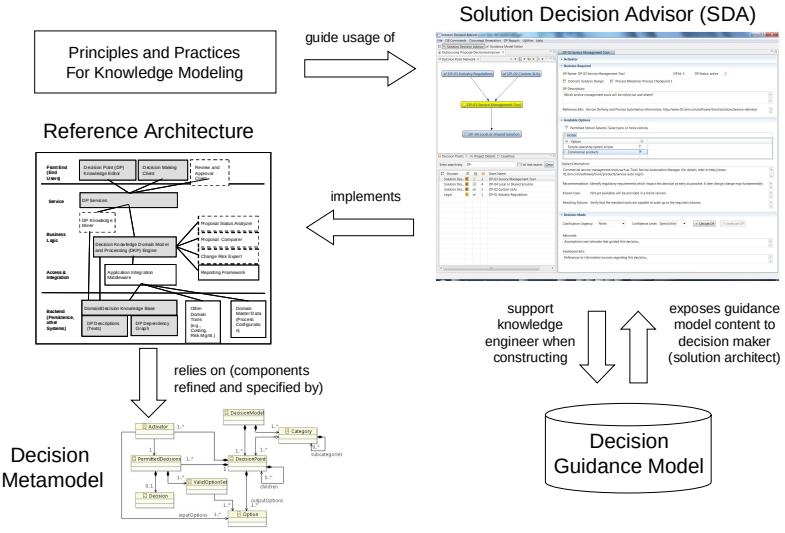

Fig. 2. Overview of decision knowledge processing solution (for proposal and project work)

A key element of our solution is the *reference architecture* (answering research questions 2 and 3) for decision knowledge processing which relies on a novel *decision metamodel* (answering research questions 1 and 4). This reference architecture is complemented by *principles and practices for knowledge modeling* (answering research question 5) that we harvested from the experience gained on our two knowledge management projects (SOAD and SDA). The SDA tool implements the reference architecture (answering research question 2); it is the tool to be used by *knowledge engineers* to build decision guidance models (i.e., knowledge bases) and by *solution architects* to receive guidance and capture decisions. In Sections 3, 4, and 5, we will explain these elements of our DKP solution in more detail; as an intermediate step, we state the most significant functional and non-functional requirements next.

2.3 Architecturally Significant Requirements

We gathered the functional and non-functional requirements driving our DKP solution design in [24]. Hence, we only recapitulate the most significant ones here (referencing both knowledge management challenges from Section 2.1 and quality attributes [17]):

1. Present those Decision Point (DPs) to *decision makers* (solution architects) that can be decided as a consequence of decisions already made (context-specific decision identification). This responds to several of the challenges from Section 2.1, including a) scope and scale and b) priority and order.
2. Allow certain DPs to be made multiple times (allowing different outcomes for each instance), e.g., in order to express that different designs are required in each country of an international SO solution, or that certain SOA decisions have to be made for each Web service provider. Such support for multiple DP instances also helps to overcome the scope and scale challenge.

3. Visualize related DPs as a DP graph with DPs as nodes and their connections (dependencies) as edges to overcome the priority and order challenge.
4. Propose suitable decision alternatives (options) in response to the reuse and education challenge; these can be company standards or solutions proven on previous projects.
5. Express confidence in a made decision and allow users to "undecide" one or more already decided DPs without losing consistency in the DP graph. The addressed challenges are a) data quality and uncertainty, and b) consistency and efficiency.
6. Support the display of decisions and their rationale to *decision reviewers* (e.g., senior solution architects) via the user interface and via report generation. The addressed challenges are a) data quality and uncertainty, and b) reuse and education.
7. The decision maker should not be able to choose design elements (i.e., combinations of decisions) that are contradictory, that cannot be delivered efficiently, or that create unacceptable amounts of risk for handover and operations. The related quality attributes are consistency and accuracy.
8. It must be possible to narrow the decision space down to the DPs that are relevant in a given design context; e.g., pruning of dead paths in the DP graph is desirable (challenge and quality attribute: efficiency).
9. No development work should be required when the DP texts are authored and edited (e.g., when updating a decision guidance model in response to changes in the business model or the technical context) or when the scope of a decision guidance model is extended (modifiability, maintainability).
10. It must be possible for *knowledge engineers* that do not have any programming skills to create, configure, and maintain DP graphs rapidly (modifiability, maintainability).
11. User cannot be assumed to be familiar with workflow concepts or general-purpose business process modeling languages (usability).
12. In anticipation of reuse and application in other domains and application scenarios, the solution should not be limited to (or even hard code) any knowledge domain such as SOA or SO design, but be extensible to cover other practices of IT services, e.g., package integration (modifiability).

We compiled these requirements and qualities iteratively by usage scenario/user story through requirements engineering activities that included interviews with members of the target audience, user storytelling [4], and use case modeling [25]; as we only summarize selected results of our requirements engineering activities here, details on our requirements gathering method (i.e., chosen process, notations, and techniques) remain out of scope. The linear structure of this paper might suggest a waterfall approach a.k.a. big design upfront; however, we actually followed agile practices and performed sprints (i.e., short, time-boxed development iterations).

In the next sections, we present a DKP solution that satisfies these functional and nonfunctional requirements, and apply it to several IT services practices/subdomains.

3 Reference Architecture for Decision Knowledge Processing

To overcome the knowledge management challenges and satisfy the requirements from Section 2, we combine domain-specific decision knowledge bases with a supporting workflow-oriented decision knowledge processing tool. In this section, we focus on the architecture of the knowledge processing tool; the metamodel for the domain-specific knowledge bases, which we also call *guidance models* [41], is introduced later in Section 4.

Figure 3 presents a layered view of our tool reference architecture. We position this architecture as reference architecture because it is based on the architecting experience from two earlier prototypes, the PHP-based Architectural Decision Knowledge Wiki [40] and its successor, the Java-based Architectural Decision Knowledge Web tool. This reference architecture steered the development of our third-generation tool, Solution Decision Advisor (SDA), which we will introduce in Section 5.

In Figure 3, the AKM-specific core components are displayed with a grey background, while general purpose components appear as white boxes with solid lines. Optional AKM components are indicated by dashed borders and white background (unlike the core components and the general purpose components, we have not implemented these optional components in the current version of SDA, see Section 5).

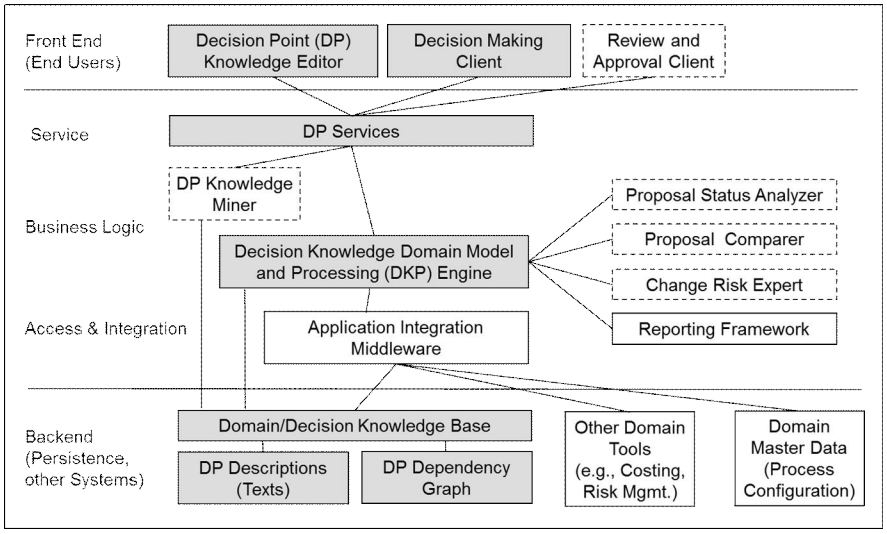

Fig. 3. Reference architecture for decision knowledge processing (presales/project work)

Following a layered architecture design style [12], the architecture is organized into 1) a *front end* serving end users, 2) a mid layer that is responsible for the processing (in turn organized into *service*, *business logic*, and *access and integration* sublayers), and 3) a *backend* persisting Decision Points (DPs) and integrating other tools and data.

The frontend contains one logical component for each user role we identified in user stories and use cases, a) *knowledge engineer*, (i.e., DP authors and maintainers), b) *solution architect (decision maker)*, and c) *decision reviewer*. DP editor, decision making client, and decision review and approval client can be realized with standard frontend design client patterns such as observer and model-view-controller. Such

patterns are supported in many contemporary user interface design frameworks. At the platform-independent level of refinement of the architecture in this section, we do not make any assumptions about physical distribution or client implementation strategies/technologies such as Eclipse Rich Client Platform (RCP) or thin Web client.

The middle layer features a *DP services* component (a straightforward instance of the Service Layer pattern [12], providing a unified interface decoupling the front end from the mid layer) and a particularly relevant component in our reference architecture, the *Decision Knowledge domain model and Processing (DKP) engine* encapsulating the core business logic. Unlike general-purpose workflow engines, this component implements a domain-specific metamodel that is optimized for our particular scenario; both knowledge engineer and decision maker operate on the same data so that no deployment or code generation step is required. The design of this component is detailed in the following Section 4. The *application integration middleware* appears in response to tool integration requirements. It applies standard integration patterns such as channel adapter and normalizer [15] and can easily be implemented with the help of commercial products and open source assets; hence, we do not feature it any further in this paper. The same holds true for the *reporting framework*. We also foresee a *design status analyzer* (calculating metrics regarding number of open and resolved DPs, degree of confidence in the chosen design, etc.), a *design comparer* (working with structural patterns in DP graphs), a *what-if predictor* (assessing probability and impact of the consequences of a design change such as switching to another option in a particular DP), and a *DP knowledge miner* that is responsible for identifying relevant knowledge, e.g., in design documents, and partially automating their conversion into formally modeled DPs.

The novel components in the backend are the *decision knowledge base* (persisting reusable decision guidance models as introduced in Figure 2, but also project-specific decision logs), the *DP descriptions* and the *DP dependency graph*. The backend also comprises already existing solution design *tools* in a domain that our DKP solution integrates with, as well as a *process configuration* database. This database contains reference information (master data), for instance the countries that might be in scope of an SO solution and the review boards that serve as process milestones in the solution design process. In enterprise application development, relevant subsets of the software engineering and project management methods could reside in this database (e.g., definitions of quality gates and project milestones).

The components can easily be traced back to the requirements from Section 2. For instance, all DP graph processing takes place in the DKP engine (e.g., in response to requirements 1, 3, and 5). From a quality attribute perspective, the components in the front end have to ensure usability, while the middle layer design is most critical for performance. Modifiability is ensured via several components in the backend, e.g., the configurable DP dependency graph that ensures that the DP processing flow is not fixed in the implementation code of any component in the front end or in the mid layer (e.g., in the decision making client or in the DKP engine).

Let us now walk through two key use cases to specify the core components in more detail, a) knowledge engineering (guidance model creation) and b) decision making (on proposal or development projects). In use case a), knowledge engineering:

1. The knowledge engineer opens the DP editor.
2. The knowledge engineer creates an empty guidance model in the DP editor.
3. The knowledge engineer creates DPs in the DP editor (like DP-01 to DP-04 from Section 2) and populates their attributes (see Section 4 for details).

4. The DP editor creates a draft DP graph (in the service sublayer).
5. The knowledge engineer creates a DP graph in the DP editor by connecting DPs via their options (see Section 4 for details).
6. The DP editor updates the DP graph accordingly.
7. The knowledge engineer saves the DP graph and the DPs in the DP editor.
8. The DP editor forwards the save request to the DP services.
9. Via the DKP engine that manages the state of the DP graph and individual DPs, the DPs and their connections are made persistent in the decision knowledge base.
10. The knowledge engineer continues to create, update, and connect DPs through steps 3 to 9 until the new guidance model is ready for review or release.

At any point during the knowledge engineering, the knowledge engineer may validate the resulting guidance model. Validation requests are accepted by the DP editor, forwarded to the DP services and, in turn, to the DKP engine. The validation checks syntactical and semantic correctness with respect to the metamodel (Section 4).

Use case b), decision making, leads to the following high-level sequence of component interactions:

1. The solution architect opens the decision making client.
2. The solution architect searches for an available guidance model, selects one and opens it (many implementation-specific component interactions may take place in this step, but are not further detailed here for brevity).
3. The decision making client creates a project-specific instance (copy) of the selected guidance model, displays the DP graph and those individual DPs to the solution architect that are eligible for decision making.
4. The solution architect picks one of the eligible DPs and reviews its description, including option descriptions (see Section 4 for metamodel attributes).
5. The solution architect selects a combination of options that have been specified by the knowledge engineer to yield a working solution to the design problem represented by the DP.
6. The solution architect provides rationale for the decision expressed by the option selection, e.g., by referencing client requirements (from an RfP) or technical constraints such as enterprise-wide architectural principles regarding certain quality attributes (e.g., in the security space).
7. The decision making client receives the input from the solution architect and passes it on to the DP services component which, in turn, forwards it to the DKP engine.
8. The DKP engine updates the DP graph both locally and by adjusting the states of successor DPs (see Section 4 and Section 5 for details).
9. The status updates are reflected in the decision making client (via the DP services component).
10. The solution architect loops through steps 4 to 9 until all DPs are decided.

At any point in the processing, the solution architect may decide to save the decision model via the decision making client. In response to such request, the decision making client invokes the DP services, DKP engine, and, eventually, the persistent decision knowledge base/DP text/DP graph components.

In the following Section 4, we detail the design of the novel component(s) that are primarily responsible for persisting, processing, and presenting the DP graph.

4 Metamodel for Decision Knowledge Processing

In the reference architecture that we introduced in Section 3, the novel components are the decision making client, the DP services, the DKP engine, and the decision knowledge base (persisting DP description texts and DP graphs). The interfaces between these components as well as their internal design are shaped by a metamodel for decision knowledge processing that extends previous work in the AKM community [6][20][43]. In this section we introduce this extended metamodel both informally and formally.

Figure 4 illustrates the DP attributes informally by giving them a name and by providing rationale in question form (from a user's perspective):

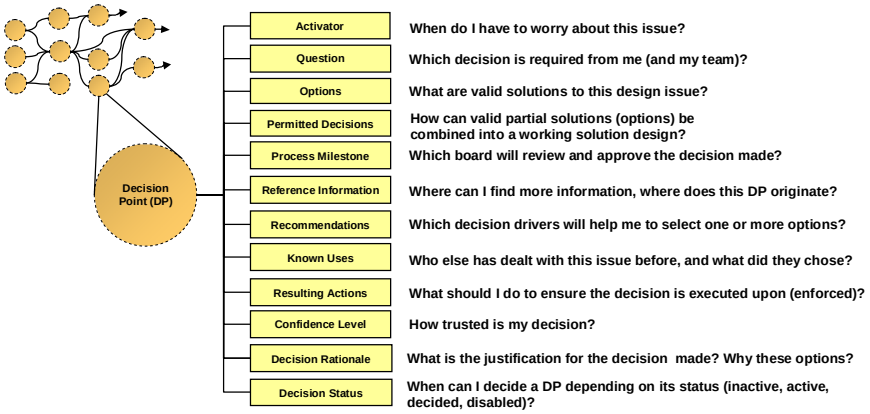

Fig. 4. Attributes of a DP embedded in a DP graph

In instances of the metamodel, individual DPs provide guidance in the form of *known uses* (i.e., selection of particular options on previous projects), decision making *recommendations*, and *resulting actions* (i.e., actions that have to be taken to ensure a design that has been decided for is actually implemented and becomes visible in the project deliverables and the contract with the client). DPs can be categorized by criteria such as functional domain and *process milestone*. Attributes such as *confidence level* and *decision status* may be used by the design status analyzer in Figure 3 (in Section 3) to calculate metrics which are then displayed and exported via the reporting framework component (e.g., a metric assessing the completeness of the design).

Decision Points (DPs) and their dependency relations are assembled into a directed graph consisting of nodes and connections such that the dependencies form a partial order [22]; individual DPs are modeled as finite state machines [28]. The overall state of the DP graph is defined by the aggregation of the states of the individual DPs.

The DP graph metamodel design is formalized in the class diagram in Figure 5:

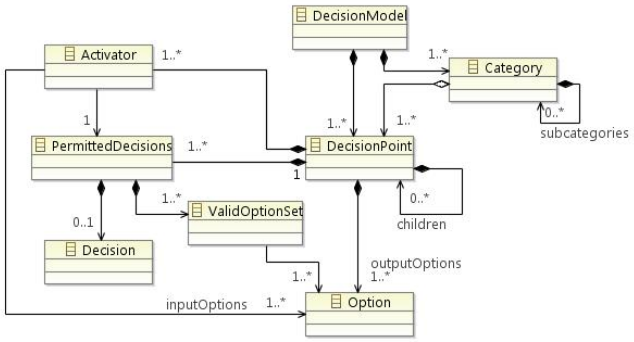

Fig. 5. Classes of the common metamodel

The entity types (i.e., classes) in the metamodel, e.g., *activator*, *decision point*, and *option*, jointly form a *decision model* which is organized into a discrete (non-intersecting) set of *category* instances. Instances of these entity types provide reusable decision guidance and are therefore entered by the knowledge engineer. During decision making, the solution architect then creates *decision* instances referring to specific instances of permitted decisions and providing rationale for their selection.

A DP represents a required decision. It acts as an aggregate root [9] for the other entity types and is a node in the DP graph (with the edges being defined by the control and data flow connections between DPs). The category concept allows to group DP by arbitrary topic areas; through subcategory links, hierarchies can be built. The resulting classification is orthogonal to the primary structuring means, the DP graph. In SO solution design, functional domains such as human resources, legal, and real estate may serve as categories; in enterprise application development and SOA design, the abstraction-refinement levels in software engineering methods as well as layers and components in reference architectures can supply structuring means for decisions that can be expressed as category hierarchies [43]. These categories allow solution architects and decision reviewers to search, filter, and select DPs by topic groups.

An option defines a design alternative that can be chosen or neglected, i.e., an option has a binary value (true or false). A DP typically contains multiple options. In each DP, the *permitted decisions* specify combinations of options that are compatible with each other and lead to working solution designs. Permitted decisions are specified by one or more *valid option sets*. Such valid option sets are constructed with logical expressions. We defined four construction rules for these logical expressions: *exactly one* option chosen and all others neglected, *at least one* option chosen and remaining ones neglected, and *zero or more (any)* option chosen or neglected; finally, fixed *selection patterns* can also define the chosen/neglected options in a valid option set. The concept of valid option sets allows the knowledge engineer to express design constraints, e.g., certain business rules that can be used to prune the DP graph as the design evolves and matures (in response to requirement 8 from Section 2).

The following Figure 6 instantiates the metamodel from Figure 5 for the service management example from Section 2, shortly after DP-03 has been decided:

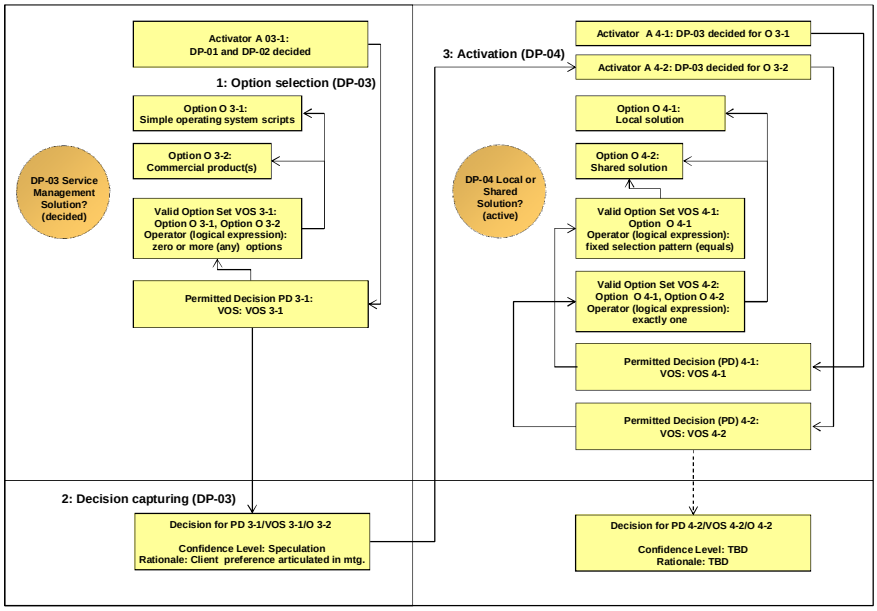

Fig. 6. Exemplary DP graph and metamodel instantiation

To specify control and data flow, the options in a DP may be connected to options in other DPs. More precisely, a DP contains one or more activators. Each of these activators references one or more options in predecessor DPs and exactly one permitted decision instance in the DP that contains it (as already explained, a permitted decision instance lists a set of valid options in its DP). In the example in Figure 6, activator A 4-1 in DP-04 lists and refers back to option O 3-1 in DP-03 and to permitted decision PD 4-1 in DP-04. Similarly, A 4-2 triggers PD 4-2 and references O 3-2. Hence, the decision for commercial products (O 3-2) in DP-03 triggers A 4-2 and not A 4-1; both a local solution (O 4-1) and a shared solution (O 4-2) are still possible.

The data flow is defined to run from the referenced options in predecessor DPs to the activator and from the activator to the permitted decision instance and its valid option set(s). The control flow (which induces state changes in DPs) is derived automatically from the data flow. DPs without predecessors (in the control and data flow) are *active* by default (i.e., eligible for decision making/ready to be made); a DP with one or more inbound dependencies from any active predecessor DP is *inactive* (i.e., pending/not ready to be made yet). An inactive DP changes its state to active or disabled when all inbound dependencies connect the DP to decided or disabled DPs only (i.e., no more dependencies from active or inactive DPs exist). In the example in Figure 6, this is the case for DP-04. When an architect makes a decision by selecting options and providing rationale, the corresponding DP changes its state from active to *decided*. In the example in Figure 6, this is the case for DP-03. An inactive DP changes its state to *disabled* when the option selections of its predecessors indicate that it is no longer relevant (the DKP engine finds this out when evaluating the data flow after a DP has gone into the decided state). To satisfy functional requirement 5, a decided DP can be undecided; in this case, its state changes back to active (or inactive, if a

predecessor DP is undecided). Consequently, the states of all dependent DPs are adjusted as well. This approach to data flow and state management addresses requirements 1, 5, 7, and 8 from Section 2.

The Multi-Instance Workflow pattern [36] allows us to represent multiple DP dimensions (e.g., SO countries, Web service provider instances in SOA) as DP instances to satisfy requirement 2 from Section 2. Each of these instances has its own state and participates in the data flow logic through its own activators.

The rationale for this declarative, graph-based approach to decision ordering can be found in the decision making challenges we motivated in Section 2 (e.g., priority and order, consistency and efficiency): the number of DPs that still have to be decided can be reduced based on information about decisions made. The knowledge engineer defines activators, permitted decisions and valid option sets (i.e., all links in Figure 6) during guidance model creation as outlined in use case a) in Section 3.

5 Implementation of Concepts: Solution Decision Advisor (SDA)

We used Eclipse components (e.g., Eclipse Rich Client Platform, Eclipse Modeling Framework, Graphical Editing Framework, ZEST, and BIRT reporting) to implement the concepts from Section 3 and Section 4 in a tool called *Solution Decision Advisor (SDA)* [24]. Figure 7 shows the user interface of the decision making client of SDA:

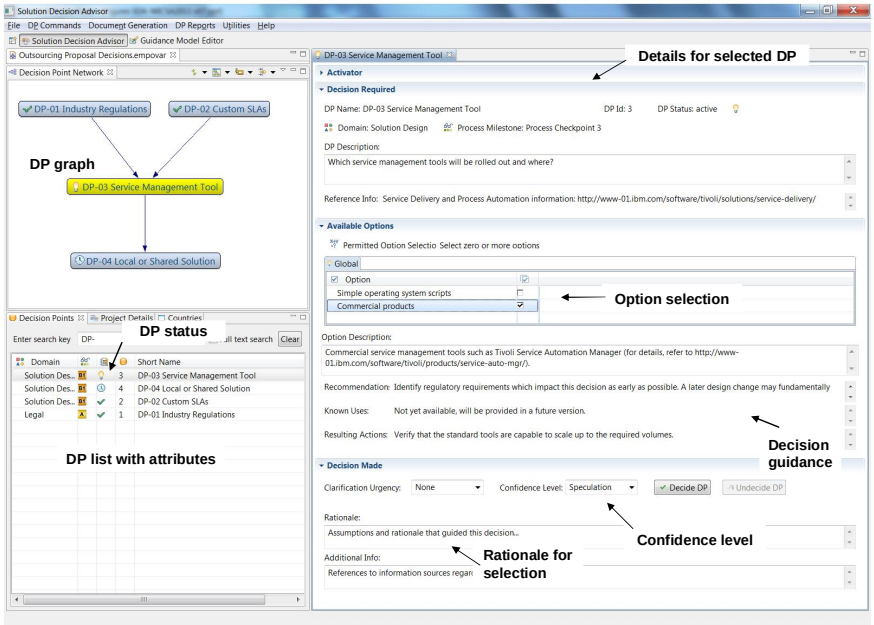

Fig. 7. User interface of the decision making client in Solution Decision Advisor (SDA)

The user interface is organized according to the master-details pattern. The master part of the pattern displays the DP graph, which is displayed both graphically (top left)

and in list form (bottom left). In the details part of the pattern, DP texts can be reviewed and decisions be made with a minimal amount of user actions (clicks). The attributes from the metamodel in the previous section (e.g., see Figures 4, 5, and 6) can easily be identified in this part of the user interface, e.g., options and recommendation.

As specified in the reference architecture and metamodel, DPs are represented as state machines. SDA currently supports four states, *inactive* (visualized by a stop watch icon), *active* (light bulb), *decided* (green check mark), and *disabled* (red cross). State transitions are triggered by decide-undecide events triggered by the decision maker, e.g., a SO presales architect or a SOA project architect.

The dependencies between DPs are used to check the consistency of decisions made. Key figures (e.g., assumptions versus facts, number of deviations from standard solution(s) by functional domain or by process milestone) of the DP graph can be gathered, displayed, and exported using the reporting component (from Figure 3). In addition, an artifact generation framework supports automatic generation or population of configurable output artifacts such as business documents (e.g., approval forms and board presentations).

For the knowledge engineer, there is a separate user interface (implementing the DP editor component from the reference architecture). Both user interfaces are implemented as Eclipse perspectives. Hence, it is easy to switch between the two perspectives and roles (from knowledge engineer to decision maker role and back). This is useful for testing and demonstration purposes, but can also be disabled to protect content. For instance, certain decision makers might only be entitled to make decisions, but not to update DP texts and DP graph. An additional advantage of implementing the SDA frontend with Eclipse perspectives is that SDA can easily be integrated with other Eclipse tools this way, e.g., many Unified Modeling Language (UML) tools.

During decision making, DPs are activated by configurable rules that depend on state changes of connected predecessor DPs. As explained in Section 4, these rules create a partial order; the decision maker is free to prioritize (i.e., select, review, and decide) any active DP. Process milestones assist the decision maker in this prioritization effort; for SO proposals, the proposal review boards serve as such milestones; in the user interface, they appear as color-coded numbers in the list view.

Figure 8 shows a UML sequence diagram that illustrates the user activities and component interactions during decision making (including status update management):

1. The solution architect works with a DP that is in status “active”.
2. The solution architect selects the option(s), sets the clarification urgency as well as the confidence level and adds a rationale for the decision (see Figure 4 in Section 4 for more information about these DP attributes).
3. The solution architect decides the DP (by selecting the decide command).
4. The DP loops over all direct successor DPs and invokes a method for each to update its decision status.
5. Each successor DP loops over its activator objects to evaluate the activation rules. The activators encapsulate the rules that define the status transition conditions for the containing DP. They refer to options of direct predecessor DPs. The activators also calculate the permitted decisions that define which options are meaningful for their containing DP based on the option values of the predecessor DPs (details of this data flow logic are described in Section 4, e.g., in the example in Figure 6).
6. The status of the corresponding DP is updated.

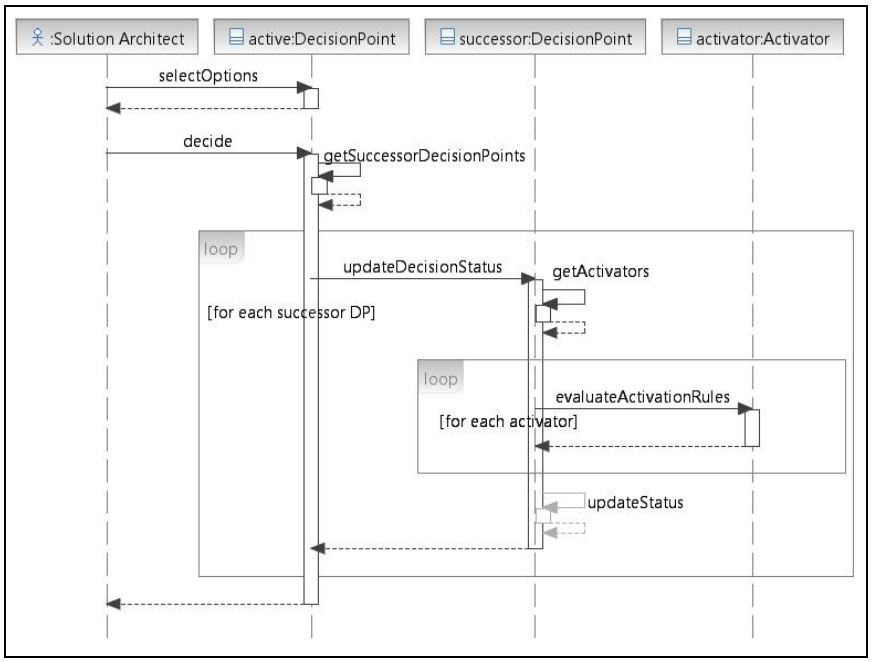

Fig. 8. Sequence diagram for decision making use case scenario in SDA

Note that there is no automated support for "simulating" paths associated to certain decisions at present; such functionality would reside in an implementation of the what-if predictor component from the reference architecture (Figure 3 in Section 3).

The DP graph content can be imported and exported so that the content management can optionally be performed in other tools.

6 Research Approach and Validation

In this section, we first discuss our overall research approach and validation strategy. We then feature the application of SDA to SO design in depth and outline additional usage of SDA in other IT services practices and application domains such as enterprise application development, SOA design, and cloud computing.[2] We also discuss threats to validity as well as general benefits and liabilities of our approach.

[2] Some of the details of the applications can only be provided in abstract form to protect client confidentiality and company-internal intellectual property rights.

6.1 Research Approach, Design Techniques, and Validation Types

The problem that we solve with SDA is the creation of a decision-centric knowledge processing solution. This problem has a design nature; hence, the literature suggests validation by experience as an adequate validation type, as opposed to analysis with formal proofs or other validation types [30]. Our validation approach is in line with these recommendations: We decided for "experience" as our primary validation type, with the objective to show "correctness, usefulness, and effectiveness" of our concepts [30].

Overview of approach. Our overall research and development approach combines software engineering research techniques with general software engineering practices proven in practice. It can be summarized in the following principles (guidelines):

- Use well-established software engineering concepts both from academia and from practice for solution design (e.g., object-oriented analysis and design, agile planning, quality attribute-driven design). Since these techniques are well documented in the literature, we do not discuss them in detail in this paper.
- Select a diverse set of users and other stakeholders for requirements elicitation (e.g., both junior and senior architects, decision makers from multiple countries and from multiple business units, with expertise in all affected knowledge domains); approach both active practitioners and those that are no longer active, but have been promoted into technical leadership roles (rationale: such practitioners have deep, long standing experience; they are familiar both with the past and with the present state of the practice).
- Validate/evaluate the emerging SDA tool and domain-specific knowledge bases iteratively and incrementally. Involve users and other stakeholders from requirements elicitation, but also reach out to additional parts of the target audience that have not been involved with the SDA project so far (e.g., quality assurance specialists and business unit managers).
- Apply a mix of validation techniques to balance investment needed and to mitigate project risk and counter threats to validity; these validation techniques include prototyping (implementation), controlled experiments, action research, formal reviews, and structured interviews (feedback gathering).

Depending on the validation type, feedback was a) gathered orally, then logged and shared after the session (for corrections) or b) obtained in writing. The feedback was analyzed and interpreted by mapping it back to the evaluation questions and updating the product backlog if needed (both for tool and knowledge bases).[3]

Selection of participants and interview questions (validation objectives). The interview participants were a) officially nominated evaluators from business units and b) volunteers that had heard about the solution or were already known to the research team to be members of the target audience.

We had to find compromises (make tradeoffs) between the ideal setup from a scientific and from an economical and feasibility point of view. For instance, IT services practitioners in technical leadership positions often have many simultaneous

[3] Product backlog is a term from Scrum, one of the agile methods selected; we maintained our backlog in the form of work items using the Jazz tool platform.

high priority tasks, while members of commercial project teams have to account for their working hours; both study groups therefore have difficulties to find time to participate in internal evaluation activities. Hence, their involvement had to be prepared and managed carefully to minimize effort and maximize output.

When eliciting requirements and when presenting our solution to practitioners, we followed a three-level approach and confronted these practitioners with the practical challenges (from Section 2.1), the research questions 1 to 3 (from Section 2.2), as well as more detailed follow-up questions. Key follow-up questions were:

1. Which design decisions are you responsible for today (or were in the past)?
2. How do/did you capture your decisions (notation, tool, level of detail)?
3. How do/did you cooperate with other members of the proposal/project team during the preparation and execution of decision making?
4. How do/did you follow-up on decisions made, both with project internal and external stakeholders (e.g., design reviewers)?
5. Would you agree that a subset of the key decisions have to be made for each proposal/project; if so, would it be worth having checklists for them and sharing knowledge about them across proposal teams and projects?

Selected validation details. Most of our validation types are well documented in the literature (e.g., prototype development, action research, and user surveys); we applied them in a straightforward way. However, in response to the validation challenges outlined above (e.g., limited availability of some members of the target audience), we designed an additional validation type: a dedicated series of interactive feedback gathering sessions with a fixed setup that included a role play.

We conducted these feedback gathering sessions consistently each time, using face-to-face meetings or Web conferencing (screen sharing). The agenda of these one-hour sessions was: 1) present problem statement and solution outline 2) give SDA tool overview, 3) walk through DP texts (reading sample content out loud), 4) decide DP based on what was read, 5) compare decision made with expert input/decisions made on real projects, 6) conduct blind test (content probing), 7) ask for general comment and correctness, usefulness, effectiveness feedback (e.g., amount of DPs, depth of DP descriptions).

On these sessions, one researcher took the SDA user/decision maker role. Another researcher took the knowledge engineer role (i.e., creator of guidance model). A third researcher did not participate actively, but acted as timekeeper and observer (he was only allowed to asking clarification questions and request feedback). A fourth session participant, a prospective user of SDA (i.e., not a member of the research team) acted as decision reviewer (design authority), commenting whether he/she agreed with the action taken by the decision maker (researcher 1) and the recommendations given by guidance model and knowledge engineer (researcher 2).

A T-shaped approach was followed during these walkthroughs and role plays to be able to assess the depth and the breath of the guidance model without having to read it in its entirety: after the initial SDA tool orientation (decision maker client), two to three DPs were reviewed in detail with texts and dependencies and then decided (causing status changes in the DP graph). Next the list-style GUI was looked at, to give the user an impression of the breadth, mostly only reading DP names (grouped by process milestones). After that, the remaining SDA features were presented (e.g.,

reporting). Finally, a blind test was conducted: the prospective user was asked to name an important decision in his tacit knowledge (e.g., a decision that was difficult to make on a previous project); this knowledge was then searched for using SDA search and filter capabilities. If found, DP content was reviewed; if not found, is was discussed whether it should have been found (it could not be present for good reasons, e.g., scope control for current release); if missing, a related knowledge engineering activity was added to the backlog.

The observer/feedback gatherer asked the following specific questions:

1. Is the DP that was reviewed in detail relevant according to your experience?
2. Is its DP text (attribute values) understandable? Is the amount of information right (amount/writing style)? Are any types of information missing?
3. Are the options complete or is any option missing? Do the option names work? Would you have organized/modeled the options differently?
4. Do the effects on the DP graph get clear, e.g., did you notice the status changes and do you understand why they happened?
5. Is the SDA user interface intuitive and efficient to use, e.g., the status icons?
6. Are the DP names expressive enough (e.g., length, terminology used)?
7. Are there too few or too many DPs? Are you missing any DP?
8. Do the generated reports have adequate breadth and depth?

We conducted more than 20 of these agile/instant user feedback sessions after having released Version 1.0 of SDA [24] and, later on, throughout 2011. The AKM principles from Section 7 were incrementally developed and refined during these sessions; they were used to train knowledge engineers and to steer content reviews.

6.2 Validation in SO Domain

SDA was applied in the following tests, each involving multiple SO client cases:[4]

1. Deal replays in two European countries (controlled experiments).
2. Live pilot series 1 with real users and data in two other European countries, partially involving action research.
3. Interactive tool and content walkthrough/role plays (see Section 6.1).
4. Live pilot series 2 outside of Europe, not involving action research.
5. Rapid construction and review of additional guidance model content, e.g., about decisions for data center relocation (the input to these controlled experiments were a text document and a spreadsheet, the output was a draft guidance model adhering to the SDA metamodel from Section 4).

Having conducted the early tests to evaluate value and usability, we released several versions of SDA to early adopters on live proposal teams. Version 1.0 shipped with a reusable DP guidance model comprising 109 fundamental solution design

[4] The tests were conducted between August 2010 to December 2011 and were a primary responsibility of all project team members; one researcher was in charge of coordination. More details about the cases would exceed the scope of this paper and cannot be disclosed due to confidentiality clauses in contracts.

decisions. After two months, SDA had about a dozen active users [24]. Two additional versions of SDA tool and guidance model content were released afterwards.

During the validation tests, the questions listed in Section 6.1 were answered; these answers were analyzed and consolidated afterwards. For instance, all 109 modeled DPs were confirmed to be relevant and recurring (validation question 1 in Section 6.1). The names of DPs were updated several times based on early adopter feedback; eventually, they were commented to be rich and specific enough to be understandable, but also compact and digestible (question 6). Only a few requests for additional DPs were made, e.g., for DPs dealing with industry-specific regulations and cloud computing (question 7). DP texts were considered to be concise, i.e., not too verbose and meaningful, i.e., not obvious/trivial; all attributes featured in the metamodel (e.g., recommendations and known uses) were seen to convey useful information; users considered it to be important to separate objective, official recommendations from subjective, personal statements (question 2). The modeling of most options had to be simplified based on user feedback; in a few cases, missing options had to be added (question 3);

From an SDA user interface design standpoint, the search and filter capabilities in the DP list view and the explicit status and dependency management were greatly appreciated (questions 4 and 5). Ordering DPs by process milestone was seen to be a valid approach because many proposal activities require reviews and approvals. The reporting and artifact generation features were commented to be powerful features and critical success factors; several layout and content assembly changes were requested and implemented, e.g., verbosity modes were introduced, as well as multidimensional filtering capabilities leveraging the properties defined in the metamodel (question 8).

To review and make a single decision is a matter of minutes (excluding the time needed to obtain the required deal context information from client and RfP). Novice users (i.e., users that had not been exposed to the knowledge in the DP model previously) reported that it takes them between two to three hours to investigate, make and capture all 109 decisions. If the same knowledge was exposed to these users in a simple spreadsheet, the processing time could assumed to be longer, but still in a similar order of magnitude (we verified this assumption in several small experiments). However, in such setting it would be more difficult to order DPs according to their dependencies and to disable DPs that are no longer relevant. Furthermore, it would be more difficult to satisfy all requirements from Section 2 and to support the additional capabilities introduced in Sections 3 and 4 (e.g., reporting, proposal status analysis, and integration with other tools).

To satisfy their auditing and archiving needs, users requested that knowledge provenance information should be preserved in decision guidance models. These suggestions lead to opportunities for usability improvements, e.g., allowing the SDA tool to show a relevant subset of attributes only (in a particular view). They also demonstrate the need for modeling principles such as those established in Section 7.

6.3 Application of SDA to other Domains/Practices in IT Services

The SDA metamodel and tool are designed to be applicable to other domains. They provide a generic Architectural Knowledge Management (AKM) solution that, with additional guidance model content, was used in the following cases throughout 2011:

1. *SOA design usage.* For this test, the SOA guidance model from the SOAD project was imported into the SDA tool (which required the design and development of an additional integration component as the SOAD and SDA metamodels are not identical). The objective of this test was to prove that SDA is at least as comprehensive in its support for the SOA architect as previous tools that we had developed, e.g., Architectural Decision Knowledge Wiki [40] and Architectural Decision Knowledge Web tools.[5] The test duration was two days; development took less than one person day.
2. *Integration of existing non-SOA content.* To demonstrate that the knowledge base in SOAD and SDA is extensible, existing knowledge about Enterprise Resource Planning (ERP) package integration was integrated into SOAD and, due to the positive results from test 1, made available in the SDA tool. This knowledge had been captured in the form of dynamic HTML tables previously (in a company-internal wiki). The effort for re-modeling about 30 existing decisions was two person days (copy-paste plus refactoring).
3. *Application to infrastructure and systems management design.* To gain modeling experience in yet another technical domain and demonstrate that our concepts indeed provide a generic AKM solution (that is not limited to SO and SOA), knowledge about cloud design was also mined and integrated into SOAD/SDA. A subset of these recurring architectural decisions was presented in the form of conference tutorials [44]; the results of this work were also transferred into a company-internal reference architecture that later on was submitted to The Open Group for standardization.
4. *Multi-company exposure.* To gather feedback from architects with very different responsibilities and professional background, interactive SOAD/SDA content and tool walkthroughs (duration: one hour each) were conducted with enterprise and solution architects from three Swiss organizations (i.e., an IT unit of a major financial institution, a government agency, and a globally operating insurance company) and seven German insurance companies (during an insurance architect community event). The setup of these sessions was similar to that described in Section 6.1.
5. *Evaluation and education project with a major car manufacturer*. To receive user feedback in a commercial professional services setting, a three-step client engagement was conducted. First, an interactive SOAD/SDA content and tool walkthrough similar to those in test 4 was performed (project initiation). Next, a small commercial consulting project was performed upon request (including a two-day onsite workshop with seven participants); existing SOAD was reviewed, but also sample data from the client modeled (i.e., upgraded from unstructured to structured representation) to demonstrate flexibility and extensibility of concepts and implementation. Finally, a concluding interactive content/tool walkthrough was performed together with a senior architect from the client (using additional sample knowledge from the client); this one-day walkthrough used results from the previous steps, but also fresh decisions that were rapidly identified and modeled.

[5] Both of these tools had to be sunset due to the amount of technical debt that had been accumulated, e.g., dependencies to obsolete software prerequisites.

The evaluation questions for these five tests resembled those from Section 6.1. The tests delivered the desired results; the test objectives were met and the feedback was similar to that presented in Section 6.2 (further details cannot be disclosed due to confidentiality agreements). In total, more than 20 architects from ten companies were involved in addition to the research team. We conclude that the validation results resemble those from previous SOAD validation activities, see for instance Chapter 12 of [20] and Chapter 9 of [40]. On the one hand, collaboration features as in an application wiki or Web tool are missing in SDA, e.g., commenting (the attribute "additional information" has been used as a workaround by pilot users); on the other hand, many other architect tools are also Eclipse-based, which leads to synergies. In comparison to our earlier tools, SDA improves the decision control and data flow management significantly. For instance, pilot users reported the graphical display of the DP graph to be superior to the explorer-style trees used in our earlier tools.

From a knowledge engineering and content management perspective, an updated version 3.0 of the SOA guidance model was released to a company-internal architect community via the official knowledge management system based on the product IBM Rational Asset Manager (RAM); 500 issues with more than 2000 alternatives appear in this version.

For the future, we consider to author additional guidance models for other application genres and architectural styles: an initial qualification and metamodel review has already been done for distributed control systems and controller design in industrial IT (as part of a software development improvement program).

6.4 Threats to Validity (of Validation)

While we strived for a comprehensive evaluation and broad applicability of our concepts and guidance model content, several threats to validity can be identified. Inherently, the assessment of knowledge sharing and decision making methods has to be different from measurements of algorithmic complexity or system response times; human users with beliefs and opinions are involved, which poses a number of validation challenges.

In our setting, it was not possible to work with randomly selected users or large multi-company populations. Root causes include protection of intellectual property rights, goal conflicts, costs, and both internal and external competition. We responded to this threat by working with practitioners from more than five countries in different geographies, with various levels of seniority; some of these users were able to bring in multi-faceted experiences e.g., from previous job assignments.

Another threat is the requirement for industrial research projects to have a positive impact on the business of the project sponsors. This leads to situations where business development and education activities cannot always be clearly decoupled from evaluation work; e.g., a falsification experiment (intentionally exposing poorly modeled decision content to users) might leave a negative impression with funding decision makers. We mitigated this risk by encouraging users to give objective feedback; our structured approach with predefined evaluation questions (see Section

6.1) supported this approach.[6] We already commented on the availability issues that we encountered with some of the key stakeholders (see Section 6.1).

One advantage of the industrial setting was that we were able to achieve software product quality in the SDA development work (e.g., with respect to bug fixing, change management, and release/version management).

6.5 Discussion

In this subsection we present a critical evaluation of the proposed approach in the context of the research questions from Section 2: 1) content and structure (data and metadata), 2) presentation, 3) processing, 4) reuse, 5) transfer and deployment.

Content and structure. Section 4 defined the structure of the knowledge base. The SO guidance model (knowledge base content) cannot be disclosed for confidentiality reasons; see Section 7 for some guidance and insight regarding its content.

Presentation. Section 3 (reference architecture) and Section 5 (SDA tool design) responded to this question. One key feature of SDA is that solution decisions are documented in one place. Decision making guidance is delivered to the user (e.g., a presales or project-level solution architect) as specified in Section 4 (recommendations, known uses, etc.). Both ordered lists of required decisions and graphical visualizations of decision dependencies are made available via the master-details pattern in SDA (Section 5).

Processing. This question was partially answered in Section 4. Ideally, all decisions should be made consistently. However, the confidence of consistency cannot be judged objectively since designers have an inclination to believe their work is good. This may affect the ideal situation suggested in requirements 5 and 7 (in Section 2.3). Our solution cannot prevent a decision maker from choosing contradictory designs in all cases; the SDA tool can only enforce business rules that were modeled as data flows in the DP graph (e.g., no globally shared solution possible if local data export and data privacy laws prevent it, see example in Section 2.1). The knowledge engineer has to find a balance between pruning as many undesired or incorrect paths as possible and not constraining the user in ways that would render the tool unusable, e.g., in situations where clients are strong enough to insist on certain specific designs that might not be desired from a service provider point of view. According to user feedback, such soft dependencies should not be modeled as DP graph dependencies, but commented upon in the recommendations attributes of the affected DPs. We adjusted our modeling practices accordingly.

Reuse. We leverage and extend work from several fields, see Section 4 and Section 8.

Transfer and deployment. An important liability is the need to create, review, and maintain a guidance model within a community of architects or an organizational unit (e.g., IT service practice in a certain region). At the early stages of the SO guidance model development, it became clear that metamodel and tool alone are not sufficient to get knowledge engineers started, they require some training as well as examples and modeling guidance (as delivered in Section 7). In response, agile practices were

[6] Bias is a large influence factor during decision making; see for instance the related working session at WICSA 2011. Tool selection decisions also qualify as architectural decisions.

applied during the population of the knowledge base (i.e., decision guidance model creation). For instance, we worked with senior subject matter experts to conduct *pair knowledge engineering* sessions. This novel form of technical writing took inspiration from the pair programming practice in eXtreme Programming (XP).

To review the entire DP graph with all texts and provide comments in writing took less than one person day. Due to this positive experience, we assess the knowledge base to be maintainable under economic constraints. Clear ownership and agreed upon funding models for maintenance and support remain to be critical success factors to make any explicit (or hybrid) AKM approach sustainable in practice.

In summary, we consider the development of concept, tooling, and guidance models a success. According to the user feedback, our research contributions and their implementation responded well to the five practical challenges from Section 2: 1) scope and scale (of decision models), 2) (decision making) priority and order, 3) data quality and uncertainty, 4) consistency and efficiency, and 5) reuse and education. We could demonstrate value and technical feasibility of our approach in several domains, including SO solution design, enterprise application development with SOA principles and patterns, and cloud computing.

7 Architectural Knowledge Modeling Principles and Practices

This section captures quality attributes, principles, and practices from both of our industrial knowledge management projects, a) *SOA Decision Modeling (SOAD)* (see [40], [43]) and b) *Solution Decision Advisor (SDA)* (first presented in [24]).

While the two projects target different audiences (i.e., SOA solution architects and outsourcing solution architects) and apply different terminologies, they share a common vision and approach. From 2006 to 2011, we cooperated with many business and technical leaders from several services firms and software vendors to harvest their knowledge and make it explicit. Overall, more than 750 DPs/issues with more than 3000 alternatives/options were identified, modelled, reviewed, released, and/or consumed by the stakeholders of these projects. During this work, we identified twelve *Modeling Principles and Practices (MPPs)*. Earlier versions of these MPPs have been used in practice since May 2010.

Target audience and purpose. The target audience for the MPPs comprises Subject Matter Experts (SMEs) for particular business areas and technical domains. When following the SOAD/SDA approach, these SMEs harvest architectural decision knowledge, create SOAD/SDA *guidance models* [41] and maintain these models in the role of a knowledge engineer. As decision makers on projects, these SMEs also consume the guidance models when creating and updating their decision models.

The objective of the MPPs is to distil and share the essence of our model management experiences (i.e., results from team discussions and other insight gained during the modeling activities, as well as responses to review findings, frequently asked questions, and usage feedback):

1. Establish quality attributes and review criteria regarding architectural knowledge management and decision modeling (primarily for guidance models as reusable assets, but also for project-level decision models).
2. Give actionable advice to SMEs and knowledge engineers that speeds up model creation and maintenance and assures the quality of these activities as well as the produced guidance models.

Quality attributes. To have a chance to be adopted in practice, guidance models have to meet the following overall criteria:

A high-quality guidance model comprises decision making advice that a) is relevant and worth sharing, b) is both comprehensive and comprehensible.

Relevant means that identifying, making, and enforcing the modeled decisions has the potential to accelerate design work and reduce project risk and that failing to do so has negative consequences for the decision maker's project. For example, superordinate standards could be unintentionally neglected or execution effectiveness might be reduced. Sharing implies the need for a knowledge maintenance approach; e.g., the required effort can often be justified if relevant knowledge that has not been published elsewhere yet (at least not in explicit, structured form).

Modeling principles and practices overview. The twelve MPPs established in this section aim at instructing knowledge engineers how to achieve the quality criteria:

1. *Naming principle:* Use expressive, self-explaining problem and solution names to foster quick orientation.
2. *Q+A principle:* Characterize problems in question form to whet consumers' appetite (and don't forget to answer the question in the solution description).
3. *Diversity principle:* Explain solution both to senior and to junior audiences to ensure broad applicability.
4. *Decisiveness principle:* Be assertive to make the given advice (guidance) actionable.
5. *Objectivity principle:* Separate facts from opinions to ensure accuracy and acceptance.
6. *Annotation and structuring principle:* Provide context information and organize the model coherently to support decision ordering and clustering.
7. *Mentoring and management principle:* Facilitate knowledge exchange to support collaboration between knowledge engineers and to stimulate discussions between decision makers.
8. *Provenance principle:* Acknowledge sources and status of knowledge to ensure authenticity and actuality.
9. *Consistency principle:* Establish naming conventions and structuring heuristics to ensure readability (model orientation) and repeatability (during model updates).
10. *Rigor and accuracy principle:* Perform peer reviews and professional editing steps (e.g., copy editing) to achieve publication quality.
11. *Dependency management principle:* Be careful and diligent when modeling and modifying a network of decision points and options.
12. *Rationale principle:* As a decision maker, justify decisions properly.

MPP 1 to 11 target knowledge engineers creating or updating guidance models; MPP 12 targets decision makers such as solution architects.

Notation. We use a common structure to document the MPPs, beginning with the *principle name*, an imperative *statement* (which best practice rule should be followed?), a *motivation* (why should the principle be followed?), *scope and rules* information (what does the principle govern?), an *example*, and *enforcement practices* (how to adhere to the principle?).[7] This structure is inspired by templates used to capture patterns [12], [15] and to document architectural principles [34].

7.1 Naming Principle (and Practices)

Use expressive, self-explaining problem and solution names to foster quick orientation.

Motivation. Concise, expressive names help the users (e.g., decision makers consuming guidance models) to get started and orient themselves rapidly. They also have a positive effect on the readability of the knowledge model and help with referencing model elements (e.g., in dependency links, in graphical user interfaces, and in reports). However, experience from the patterns community reports that finding good names is hard [23].

Scope and rules. This modeling practice pertains to both problems (i.e., DPs and issues) and solutions (i.e., options and alternatives, but also decision outcomes). We now describe three key tactics (rules) to achieve quick orientation and readability.

1. All names must be easily understandable by members of the target audience. The names should convey the semantics of the problem or solution. Domain-specific terms may be used as DP/issue names; if done so, the naming approach should be consistent, e.g., a single term per domain concept be used (i.e., avoiding synonyms). If possible, a domain glossary should be referenced (if already existing, e.g., de-jure and de-facto standards in the knowledge domain) or created (if not existing yet). The meaningful metaphor pattern [23] may be applied.

2. All names must be self-explaining. Knowledge engineers and subject matter experts should be aware of the usage scenarios for the names they chose in tools (e.g., in lists, tables, and visualizations of collections of interdependent problem-solution sets (i.e., guidance models subsets). The names should make sense to tool users without any further explanation. For instance, a SOAD issue name might be the only information available to a solution architect when deciding whether an issue modeled in an enterprise-wide guidance model is applicable in a particular SOA project. This decision is typically made during method tailoring and candidate asset screening, two of many activities to be performed early in a project (which means that often not much time is allocated to conduct these activities, although getting these tailoring and asset selection decisions right is a critical success factor for any SOA project [40]).

[7] Note that the examples are taken from the SOA domain (not from SO) because a number of excerpts from our SOA guidance model have been published already [40][44]. No loss of generality arises from this selection of examples; where needed, we comment on domain-specific modeling practices.

3. A DP/issue name must be engaging and should communicate a sense of urgency to the reader ("an action is required, I have to decide something now") and scope the problem tackled ("what do I have to decide"?). The name of options and alternatives should allow the reader to recover the solved problem without further information and also indicate how the problem can be solved: Why is the option/the alternative a solution? How can this solution be realized in a given context?

Example. Three names from the SOAD guidance model that users perceived/reported to be understandable are MESSAGE EXCHANGE PATTERN, TRANSPORT PROTOCOL CHOICE, and ENTERPRISE SERVICE BUS (ESB) ASSET SELECTION.

Less suited names for these issues would be SYNCHRONY VS. ASYNCHRONY (because this name squeezes two option/alternative names into the DP/issue name and therefore duplicates information and does not indicate that this is a pattern selection decision), TRANSPORT LAYER (because this sounds like a broad, multi-issue topic group and no sense of urgency is established) and ESB PRODUCT (because not all available assets are commercial products).

Enforcement practices. Related advice has been documented in pattern form, e.g., there is an authoring pattern called evocative pattern name [23].

Additional enforcement advice for this MPP is to distinguish decision types so that the names reveal the nature and location of a decision /guidance model element. In SOAD, issue types are defined by the refinement levels (such as conceptual level, technology level, and vendor asset level); in addition to that, pattern selection decisions are distinguished from pattern adoption decisions. In SDA, two DP types are distinguished, deal context analysis decision and actual solution design decision.

It is good practice to use a single naming scheme and consistent naming conventions (see MPP 9 for details). Note that this may become difficult when knowledge from multiple, diverse sources is compiled in a single model; however, it is still worth striving for this homogeneity.

We also recommend to model incrementally and iteratively and not to hesitate to rename or restructure guidance model content if needed. Achieving quality requires time and effort, e.g., several modeling cycles and the incorporation of review feedback (from multiple sources, e.g., writer's workshops). Renaming issues and alternatives, structural refactorings, and other edits may cause rather tedious rewriting tasks for the knowledge engineer. Such investments in quality pay off in the mid term when the usage of a guidance model increases and in the long term if the guidance model is considered a strategically important reusable asset under change and maintenance management.

7.2 Q+A Principle (and Practices)

> *Characterize problems in question form to whet consumers' appetite (and don't forget to answer the question in the solution description).*

Motivation. Once knowledge consumers (such as presales or realization project architects) have identified a DP as being relevant (by having read its name), they have

to be motivated to read on. Only curious readers will do – and they will only be curious if they get the impression that they will gain relevant information that they have not had before. Hence, this second MPP requests the DP/issue description to be engaging; it aims at improving the readability of guidance models.

Scope and rules. Three rules define this MPP.

1. The problem description of DPs and issues should be articulated in question form: if the asked question is a relevant one that is often brought up by clients or other external stakeholders (e.g., in meetings or in requests for proposals), then the reader can easily decide whether the described problem is relevant. Ideally, the reader reacts like "I have often asked that question myself, and was struggling to find a good answer despite all my experience" or "I am an expert on this topic, so I am curious whether the author of the guidance model has anything new to say" and therefore is eager to read on.

2. The question style should be chosen consciously and determined by the DP/issue type. Binary yes/no questions and multiple choice checklists narrow the solution space and leave little room for creativity in the design; they work well for DPs/issues that determine the scope of a solution. Open questions starting with "how to" or "where/what/when" work better when the user should contemplate the issue and reason about the solution (starting with an analysis of the requirements).

3. The names and descriptions of options/alternatives must answer the question raised by the problem description of the corresponding DP/issue.

Example. An example from the SOAD project is the question raised for the issue IN MESSAGE GRANULARITY: "How many message parts should be defined in the service contract and how deep should the part elements be structured?" The alternative names answer this question by recommending using the DOT, BAR, DOTTED LINE, or COMB pattern (which are explained in the solution description, e.g., the DOT pattern uses a single scalar parameter, whereas COMB uses multiple complex parameters).

Enforcement practices. Knowledge engineers should think carefully about the level of detail of the questions asked. A simple question that is straightforward to answer might be easy to process, but runs the risk create a reaction like "I know this already" or even "this is a trivial truism". As a rule of thumb, it is appropriate to assume an intermediate level of technical background (i.e., skills and experience).

It is possible to ask questions that are particularly broad and difficult to answer and order possible solutions from not working to acceptable to well suited (or from partial solution to full solution according to certain quality attributes); this creates a certain tension and excitement (also known as cliff hangers or aha effects). For instance, Hohpe and Woolf apply this writing style to present some of their enterprise integration patterns [15].

7.3 Diversity Principle (and Practices)

Explain solution both to senior and to junior audiences to ensure broad applicability.

Motivation. To justify an investment in knowledge management, the knowledge gathered in a guidance model has to serve a large enough user community; a critical mass needs to be reached. This user community cannot be expected to be homogeneous in terms of education and professional experience. For instance, senior users may only use DP/issue names while junior users tend to depend on detailed explanations to be able to appreciate the model content (and to apply it successfully).

Scope and rules. This modeling practice pertains to both problems (DPs, issues) and solutions (options, alternatives). Three key points have to be addressed:

1. It must be possible to understand the essence of a problem within a few moments; e.g., a decision to use a reusable piece of knowledge (as captured by a DP/by an issue) may be made instantly. Elaborate descriptions should not be included, but referenced.

2. Experienced (senior) users should be able to comprehend and fully understand a solution (option/alternative) by looking at its name alone.

3. Less experienced (junior) users should find enough information in the solution description to make an informed decision (possibly with the help of the knowledge found or linked under reference information, known uses, and recommendations).

Example. For instance, the INVOCATION TRANSACTIONALITY PATTERN issue in the guidance model created in the SOAD project refers to a short online article as recommended reading, but also to a text book on workflow management that has a detailed chapter on transaction processing [22]. As one of the most elaborate issues, this issue consists of about 900 words (occupying 2-3 pages in a model report) [40].

Enforcement practices. The enforcement advice for this MPP is to know and articulate the target audience, like book authors do in prefaces (forewords). Knowledge engineers should bring in – but not solely rely – on their personal experience. There is no need to go to the level of depth of a text book (or an auto biography); the style of an executive summary or a technical white paper likely is more adequate as reading speed and consumability are important quality attributes.

URIs to Web resources like short articles and blogs should be provided (if they provide permalinks) for medium experienced readers who would like to refresh their memory. Links to classical articles and seminal books should be added for inexperienced readers; these links can take the form of official citations (e.g., DOIs) or come as links to the Websites of the authors (if these provide sufficient free information online in addition to publication ordering information).

7.4 Decisiveness Principle (and Practices)

Be assertive to make the given advice (guidance) actionable.

Motivation. The reader of a guidance model (knowledge consumer) should not react like “this is generally interesting, but how does it help to perform my day job?”. He or she should be empowered to make, document, and enforce design decisions; a guidance model should provide tangible and actionable advice for doing so.

Scope and rules. This MPP comprises three scoping rules.

1. Language should be clear: Fuzzy or vague words statements like “perhaps”, “sometimes”, or “almost all” should be avoided (or at least qualified properly, e.g., with additional information about the assumed context). The same holds for rather generic technical terms such as “process” or “component”; the guidance model should define what is meant by the terms in the context of the issue, and give examples.

2. Recommendations should be tangible and concrete: To support decisiveness, knowledge engineers should state clear criteria when to select which solution in the recommendations attribute, but also in the form of issue-level decision drivers and pros/cons of alternatives (in SOAD) or as part of the DP and option descriptions (in SDA). They can also report which options/alternatives others decision makers have chosen on successful projects (in the known uses attribute). An additional option is to provide sample guidance model walkthroughs (decision making patterns).

3. Next steps should be specified: the enforcement recommendation should give advice on how to execute a decision that has just been just made, e.g.: Fill out a form and submit for review? Generate some artifacts? Schedule a meeting, phone somebody, send an email? Plan and initiate a (sub)project? Making use of dependency modeling, the user can be informed about decisions that can now be made (because activators have been fired or because pending decisions become eligible).

Example. An example from SOAD is that according to a specification from the Web Services Interoperability (WS-I) initiative, the SOAP COMMUNICATION STYLE of RPC/ENCODED should be avoided and DOCUMENT/LITERAL should be preferred because of better interoperability characteristics. The SOA guidance model reports that this style has been used many times in practice; it can be chosen by selecting the corresponding tool (code generation) options, e.g., in the Java Web Services Wizard available in the Eclipse Web Tools project.

Enforcement practices. Enforcement advice for this MPP is to avoid ambiguities in solution modeling in particular (e.g., options/alternatives should be disjoint) and to add explicit references to the requirements addressed by certain options/alternatives. In enterprise application development, Non-Functional Requirements (NFRs) including software quality attributes fall in this category. In SO design, many regulatory compliance rules have to be met.

We advise not to push too hard and not to give the impression that a recommendation or corporate standard fits all client environments and requirements. Conditional statements like “use option A if requirement X has high priority; prefer option A over option B in case of NFR Y” can be used to do so.

7.5 Objectivity Principle (and Practices)

Separate facts from opinions to ensure accuracy and acceptance.

Motivation. To become broadly accepted and have sustainable value, a guidance model must be correct and credible. Some thought provokers are required to get readers interested, to keep them engaged, and to stimulate discussions; however, a knowledge engineer runs the risk of being misunderstood (or even rejected) if his/her guidance model contains overly aggressive statements, brags, or oversimplifies.

Scope and rules. A key point that helps to achieve credibility is to clearly mark subjective information as such. Subjective information such as real-life road stories adds to the value of a knowledge exchange solution; however, the reader should be able to distinguish facts from opinions easily and unambiguously. Hence, factual descriptions/definitions should be separated from biased context information like personal motivations and the consequences of a decision made on a particular project.

Example. In SOAD guidance models, only the recommendation attribute contains personal opinions. For instance, the one for the INTEGRATION STYLE issue reads "introduce the ESB pattern if loose coupling (i.e., location, format, protocol, and implementation transparency) is a valued strategic architectural principle". The one for MESSAGE EXCHANGE PATTERN reads "do not follow an Message-Oriented Middleware (MOM) hype – decoupling in time is just one of several dimensions of loose coupling. The equation (reliable messaging is not used, so this is not an SOA) does not hold true" [44].

Enforcement practices. The most important enforcement advice for this MPP is to use the recommendation attributes (in SOAD and in SDA) for opinions, and keep problem and solution descriptions factual and free from subjective information.

Established argumentation techniques should be used, e.g., citing authoritative references such as technology standards or de facto authorities such as widely respected and recognized opinion leaders. Just like in other technical publications, evidence for any claims made and non-obvious facts should be provided.

Model reviewers can be asked to focus on hot spots (i.e., potentially controversial model content) and on the "political correctness" of the reviewed model content.

7.6 Annotation and Structuring Principle (and Practices)

Provide context information and organize the model coherently to simplify guidance model tailoring and to support decision ordering and clustering.

Motivation. Guidance models can become large and cover hundreds of DPs/issues and thousands of solution options/alternatives. If a user is forced to read an entire guidance model from beginning to end, the benefits of knowledge sharing (e.g., design acceleration and risk reduction) become hard to realize. Hence, it is required to tag the knowledge model content and organize it in such a way that relevant content (e.g., sub-models) can be searched for and extracted rapidly (e.g., via tool-supported filtering).

Scope and rules. Two rules apply to and define this modeling principle.

1. We advise to define a hierarchical overall guidance model structure that allows users to cut off entire subtrees that contain knowledge that currently is not relevant. For instance, such organizing principles can mirror those defined in architecture frameworks and methods (e.g., viewpoints and abstraction/realization levels).

2. Meaningful, expressive, and standardized values should be assigned to/for metamodel attributes such as scope, phase, and role in SOAD. The same holds for the

process milestone in SDA. The naming principles and practices from MPP 1 are also applicable to these value sets.

Example. In SOAD, we use recognized sub professions as role annotations (e.g., application architect, infrastructure architect); UML stereotypes from SOA profiles/patterns and other domain-specific reference material supply the values for the scope attribute. Additionally, free form keywords such as "workflow" and "security" are used as topic tags [40][44]. In SDA, guidance models contain process milestone annotations to structure DPs by business process phase. Categorization (grouping) schemes like (sub-)domain allocations by functional area, i.e., the legal and the HR domains, also support the clustering of DPs and guidance models in SDA.

Enforcement practices. The enforcement advice for this MPP is to use a recognized design method such as the Rational Unified Process (RUP) as source of values for the phase and role attributes in SOAD. UML profiles or reference architecture components are candidate sources of values for the scope attribute in SOAD; some of these assets also provide viewpoints and other structuring means.

In SDA, we recommend to group the decisions required by review boards, which take the role of quality gates per project milestone. Links to engagement process models tie the DPs to these process milestones; backward links may also be provided.

The supporting documentation, e.g., process descriptions, should encourage the knowledge consumer (e.g., decision maker), to tailor any incoming, shared guidance model according to the proposal or project team needs ("if in doubt, leave it out").

7.7 Mentoring and Management Principle (and Practices)

Facilitate knowledge exchange to support collaboration between knowledge engineers and to stimulate discussions between decision makers.

Motivation. We all learn well from mistakes; however, these mistakes do not always have to be our own ones. Learning from other people's mistakes, however, requires a certain culture of candor and trust that supports an open, honest approach to experience sharing.

Stakeholder (reader) expectations must be managed. A guidance model should not be expected to automate decision making; however, it should rather be positioned as an education instrument that lets senior technical leaders share their experience with their colleagues. However, experience sharing does not imply a responsibility transfer.

Scope and rules. Three rules define this MPP.

1. Knowledge engineers should choose the tone (writing style) a technical advisor or a career mentor would use; according to our experience, such consultative tone works better in this context than any official voice (e.g., of a design authority or of a technical standard). Hence, guidance models should be written in a way that creates a positive atmosphere of mutual respect in which it is ok to offer and to accept help.

2. It is acceptable to leave gaps in a guidance model; such gaps should be called out. The knowledge engineer must know what (s)he does not know (yet) and encourage

readers (knowledge consumers) to contribute additional or refined problem and solution descriptions to future versions of guidance models (leading to a give-and-take strategy for guidance model maintenance).

3. The guidance model text should not give the impression that architectural thinking is no longer required; the responsibility to make the right decision and the ownership of this decision (and its consequences) remain with the user (e.g., SOA or SO solution architect). Guidance models do not intend to unemploy their consumers.

Example. The SOAD recommendation about the SERVICE COMPOSITION PARADIGM issue reads "introduce process layer and realize it with workflow pattern and technologies if business scenario is a long running multi-actor scenario with advanced resource coordination/protection requirements. Use object-oriented composition in other cases" [44]. The editorial to do for an issue may state "review annually, update alternatives if needed, incorporate more SME input about alternative A".

Enforcement practices. Guidance models should not give commands and not threaten users to be punished when not following the given advice. In most cultures, modest phrases such as "consider solution A if quality attribute X has high priority" or "your colleagues on previous projects had to resolve issue B in situation C" work better than pompous statements such as "always select A, a great technology, unlike B" or pessimistic ones such as "you must resolve B or your project is bound to fail".

Decision makers may disagree with the content of a guidance model (i.e., overwrite the existing solution options/alternatives with engagement-specific ones); the courage to take conscious risks and a sense of pragmatism continue to be key elements of any decision making as projects and stakeholders are different. Blames should be avoided.

If a guidance model is incomplete in terms of problems and solutions presented, this should be disclosed, e.g., via placeholders or by maintaining a parking lot that collects incompletely modeled DPs and issues as candidates for inclusion in future versions.

7.8 Provenance Principle (and Practices)

Acknowledge sources and status of knowledge to ensure authenticity and actuality.

Motivation. This MPP helps to satisfy the general quality attributes of maintainability and credibility; the more meta information there is, the easier it becomes to assess the quality of the knowledge and to update it as needed.

Scope and rules. Again, three rules define this MPP.

1. We recommend providing author and editorial status information, e.g., how complete is the modeling? How trustable and trusted is the source? Do any Intellectual Property Rights (IPR) apply (for reusable asset, for raw project input)?

2. Timestamps or version numbers can be used to ensure models are kept up to date (quality attribute: actuality).

3. All contributors should be acknowledged, preferably by name (if they agree).

Example. We do not include an example here due to space constraints; Appendix A of [40] features a fully modelled issue that contains editorial information and links to reference material.

Enforcement practices. We recommend to leverage Uniform Resource Identifiers (URIs), used on the Web and many other places, as a universal, globally unique referencing technique.

Names of projects (known uses) and commercial/technical leadership contacts should also be provided, along with release and version numbers of artefacts in which the knowledge was harvested from (if needed). Sanitized names are ok to use, e.g., code names of engagements, as long as all authorized users can unveil the real names rapidly (without violating with data privacy regulations such as not to share sensitive personal information).

7.9 Consistency Principle (and Practices)

Establish naming conventions and structuring heuristics to ensure readability (model orientation) and repeatability (during model updates).

Motivation. The larger a model grows, the more important it becomes to be consistent in the writing and in the model organization. Naming conventions and a recognizable, balanced model structure help users to understand the type of a model element (e.g., problem or solution) rapidly; they also speed up model tailoring.

Scope and rules. Naming conventions for software program elements (e.g., Java class and methods names) and pattern languages have been established in the past and can provide inspiration [23].

The heuristics in [43] define *balanced* guidance model trees (note that in [43] the term guidance model is not used, but the term reusable architectural decision model).

Example. Two examples of naming conventions are:

- Name problems and solutions with nouns or verbs, depending on decision type (see MPP 1). SOAD follows this convention with issues like MESSAGE EXCHANGE PATTERN and alternatives like REQUEST REPLY and ONE WAY.
- [Action] [On Object] [Decision Type]. ASSESS LEVEL OF ARCHITECTURE CONTROL BY THE CLIENT (DEAL CONTEXT) is an example for this convention.

Enforcement practices. See [23] for related advice from the patterns community (e.g., noun phrase name pattern).

Extremes should be avoided, e.g., excessive number of solutions per problem and very long problem and solution names (usually a few words are sufficient). The same holds for overly deep topic trees (aiming for perfection) and overly broad topic trees (trying to "boil the ocean", but only scratching the surface). Also see MPPs 1 and 6.

Decision modeling patterns can be mined over time; see [43] for first examples.

7.10 Rigor and Accuracy Principle (and Practices)

Perform peer reviews and professional editing steps (e.g., copy editing) to achieve publication quality.

Motivation. Readers' judgement of the quality of the content will be influenced by the editorial quality – if the knowledge engineer did not spend the time to debug his/her text, does it mean he/she does not care about his/her readership and the success of the guidance model? If so, why should the reader trust the given advice? As everywhere, first impressions last, hence readability is a key success factor and quality attribute for any guidance model. Bias and Halo effects should be minimized.

Scope and rules. This modeling practice pertains to both problems (DPs/issues) and solutions (options/alternatives). The guidance model should be viewed as a technical publication in a practitioner magazine or research journal. The same level of editorial quality should be strived for, just like in commercial proposals and nomination packages for career promotions and professional certification.

Example. We do not include an example here due to space constraints; see Appendix A of [40] for an example of a fully modelled issue.

Enforcement practices. The enforcement advice for this MPP is not specific to guidance model authoring, but similar to general advice on technical writing.

Spell checkers should be used regularly (e.g., prior to a release) and all reported writing and grammatical errors fixed; layout issues should be resolved as well. Online dictionaries can be consulted. It is important to write full sentences in descriptions of problems and solutions, but to keep problem and solution names rather short (see MPPs 1 and 9). For instance, "stacking" nouns should be avoided. Moreover, long and complex sentences should be broken up into simple ones.

Reviews from different angles and perspectives should be conducted, e.g., syntax only, local content consistency, global content consistency; junior practitioner, senior practitioner, knowledge engineer, project sponsor (owner of reusable asset). Guidance model reports should be printed out and reviewed offline from time to time; according to our experience, not all editorial flaws are found when reading content only online.

7.11 Dependency Management Principle (and Practices)

Be careful and diligent when modeling and modifying a network of decision points and options (e.g., activator logic in SDA, issue/alternative dependencies in SOAD).

Motivation. The DP network spawns a graph (like business process models); creating and maintaining such graphs is a non-trivial design/development assignment [22].

Scope and rules. See SOAD heuristics in [43] about balanced guidance model trees. Also see the additional decision modeling advice in Appendix B of [40].

Example. We do not include an example here due to space constraints.

Enforcement practices. The enforcement advice for this MPP is:

- Couple DPs loosely, but strive for high coupling within one DP (e.g., all solution options solve the same problem). Inspiration can be drawn from database modeling, Object-Oriented Analysis and Design (OOAD), and Component-Based-Development (CBD) [21].
- Do not over-specify; if more than three to five outgoing decision dependencies are defined, consider refactoring the issue at hand into several related ones.
- Start the dependency modeling with the rather weak *influences* relations and refine into the stronger *decomposesInto* and other relations as a guidance model evolves (SOAD).
- Model problems and solutions as stateless as possible (SDA).
- Do not assume a certain DP network state when modeling options (SDA).

7.12 Rationale Principle (and Practices)

As a decision maker, justify decisions properly.

Motivation. Answering „why“ questions to provide design decision rationale is at the heart of architectural decision modeling. Unlike the previous MPPs, this one targets the knowledge consumer (i.e., the SOA or SO architect) applying one or more guidance models on projects.

Scope and rules. Decisions should be justified by backward referencing project goals and NFRs such as quality attributes. The literature suggests many supporting concepts such as metamodels for decision capturing and documentation viewpoints [38].

It is not sufficient to say “this solution is the best practice recommendation in SOAD/SDA” as a decision justification. A guidance model user may disagree with guidance model content and choose a different solution if this is justified by the project requirements; sometimes, this is even mandatory (imperative) to do.

Example. Several examples of good and bad justifications for architectural decisions are available online [42]. Appendix A of [40] identifies and groups common decision drivers.

Enforcement practices. It is recommended to refer to the requirements explicitly and as fine grained as possible (being conscious of effort). Killer phrases must be avoided (e.g., “we have always done it like that”, “other firms do that too”). The advice given by MPP 5 is valid in the decision capturing context as well. MPP 7 shares additional advice about writing style, knowledge exchange, and collaboration.

8 Related Work

With the concepts introduced in this paper, we extend and complement previous work in the Architectural Knowledge Management (AKM) field. The AKM community has published a significant body of work on design decision and design rationale models since about 2004 [6][8][18][19]; see [20] for overview articles and selected

contributions. For instance, SOA Decision Modeling (SOAD) [41] suggested a similar metamodel (with issues, alternatives, and outcomes as core entity types). In this paper, we added attributes that are relevant for pre-contract design work (see Section 2 for requirements and rationale) and optimized the decision dependency management For instance, we give a more precise, data flow-oriented definition of the relations that activate decision points based on earlier decisions, and we improve the expressivity and flexibility of the option modeling by introducing permitted decisions and valid option sets. Our previous publications on SOAD contain detailed metamodel comparisons [40][43].

Previous work in strategic outsourcing and services science highlighted similar challenges [29][31][32], but did not pursue a decision-centric approach to overcoming them. To the best of our knowledge, no integration of AKM concepts and workflow patterns has been attempted in the SO solution design domain yet.

While the DKP engine in our reference architecture (Section 3) resembles a general purpose workflow engine and realizes a subset of the workflow patterns [36], its design differs from previous work in the business process management field. In comparison to the Business Process Modeling Language (BPMN) and other general-purpose process languages, our metamodel has been optimized for decision processing. BPMN 2.0 is coarser grained; it has the concept of a business rule task artifact, which specifically indicates the invocation of a business rule, i.e., a decision or rule set. While an individual decision point could conceptually be modeled as a business rule task, we consider BPMN not to be ideally suited for modeling fine grained "working instructions". In contrast, we provide a specific decision making solution that allows knowledge engineers to model detailed design variations and their relationships.

Our metamodel (Section 4) can be viewed as the syntax of a Domain-Specific Language (DSL) [37] for decision identification, making, and enforcement practices. In accordance with the DSL design philosophy, it provides concepts that allow knowledge engineers to model decision making guidance with a minimal set of syntactical elements; the user interface of SDA can directly expose the DSL. An instance of the metamodel (e.g., a decision guidance model for strategic outsourcing or SOA) can be viewed as a semantic model [11] for a particular application domain.

We are not aware of any work on quality attributes or design principles for guidance models collecting decisions that recur in a particular domain. The quality attributes for architectural principles in The Open Group Architecture Framework (TOGAF) [34] come close, but fail to deliver concrete advice for knowledge engineers that are in charge of creation knowledge guidance models. The patterns community only provides partial solutions for a related problem, pattern authoring [23].

9 Conclusions and Summary

Designing IT services solutions, e.g., in strategic outsourcing and service-oriented architecture design, are complex and challenging tasks. Many managerial and technical decisions must be made, both during presales and on billable projects.

In this paper, we introduced a reference architecture and a decision process-oriented knowledge metamodel synthesized from the functional requirements and quality attributes both of presales proposal architects and project architects. We also presented a tool implementation of these decision modeling concepts, discussed their

validation in several IT services domains, and established twelve decision modeling principles and practices that capture the lessons learned and feedback gained.

We first identified five knowledge management challenges: 1) scope and scale (of decision models), 2) (decision making) priority and order, 3) data quality and uncertainty, 4) consistency and efficiency, and 5) reuse and education. To overcome these challenges and create a decision knowledge processing solution, we contributed a reference architecture and a metamodel defining a workflow language and DSL optimized for decision making. Reference architecture and metamodel are implemented in the Solution Decision Advisor (SDA) tool.

We validated the feasibility and practicality of our extended architectural knowledge management concepts through the SDA tool implementation, which was released to real users, and the creation of several decision models for different IT services practices, including strategic outsourcing. In the strategic outsourcing context, tool implementation and decision model were validated using test data from past proposal projects first; applications to full scope proposal projects with live users followed, as well as a series of tool/content walkthroughs with additional stakeholders (role plays). We also used SDA on service-oriented architecture design workshops with clients and captured knowledge about cloud computing and data center relocations. Overall, the SDA validation involved more than 50 practicing architects, including presales and projects architects working in multiple IT services domains.

During our architectural knowledge management projects from 2006 to 2011, we identified and documented twelve modeling practices that knowledge engineers should follow to ensure that the produced guidance models have a high quality and remain maintainable: 1) naming principle, 2) Q+A principle, 3) diversity principle, 4) decisiveness principle, 5) objectivity principle, 6) annotation and structuring principle, 7) mentoring and management principle, 8) provenance principle, 9) consistency principle, 10) rigor and accuracy principle, 11) dependency management principle, and 12) rationale principle.

For the future, we consider applying our approach to business domains outside IT services. Other areas that require additional investigations and research are collaboration and tool integration; both presales and project architects interface with many other practitioner roles, e.g., project managers, developers, and enterprise architects. Each of these stakeholder has (one or more) own views on the architecture of the solution/system under construction; the decision-centric view presented/taken by SDA should be aligned and integrated with these views.

Acknowledgments

We would like to thank Thomas Diethelm, Ivan Kuraj, Ronny Polley, and Nelly Schuster for their contributions to the SOAD and SDA projects and the knowledge management solution presented in this paper.

References

[1] Blackmore D., Young A., Notardonato S., Outsourcing Contracts Annual Review, 2009: Outsourcing Uptake Continued, but Megadeals Declined. Gartner, Inc., ID Number: G00174751, 2009.

[2] Brown D., Wilson S., The Black Book of Outsourcing: How to Manage the Changes, Challenges, and Opportunities. John Wiley & Sons, 2005.

[3] Center for Global Outsourcing, http://www.outsourceglobal.org/

[4] Cohn D., User Stories Applied, Addison Welsey, 2004.

[5] Congress of the United States of America. Sarbanes-Oxley Act of 2002, H.R. 3763

[6] de Boer R.C., Farenhorst, R., Lago P., van Vliet H., Clerc V., and Jansen A. Architectural Knowledge: Getting to the Core. Proceedings of QoSA 2007, Springer LNCS Volume 4880/2008. Pages 197-214.

[7] Dhar S., Balakrishnan B., Risks, Benefits, and Challenges in Global IT Outsourcing: Perspectives and Practices. Journal of Global Information Management, Volume 14, Issue 3, 2006.

[8] Duenas, J. C., Capilla R., The Decision View of Software Architec-ture. Proceedings of 2nd European Workshop on Software Architecture (EWSA), Springer LNCS Volume 3527/2005, Pages 222-230

[9] Evans E., Domain-Driven Design. Addison Wesley, 2003.

[10] Everest Group, Global Sourcing Market Update: December 2006–Preview DeckTopic: Evaluating Risks from Outsourcing and Global Sourcing, http://www.everestresearchinstitute.com/Downloads/ERI-2006-2-R-0130-preview

[11] Fowler M., Domain-Specific Languages. Addison Wesley, 2010.

[12] Fowler M., Patterns of Enterprise Application Architecture. Addison Wesley, 2003.

[13] Haischt D., Georg F., Get me approved, please! Lizenzkompatibilitaet von Open-Source Komponenten. Objektspektrum, Sonderbeilage Agilitaet, Winter 2010. SIGS Datacom, 2010.

[14] Halvey J., Murphy Melby B., Information Technology Outsourcing Transactions: Process, Strategies, and Contracts. John Wiley & Sons, 2005.

[15] Hohpe G., Woolf, B., Enterprise Integration Patterns. Addison Wesley, 2004.

[16] Holcomb T., Hitt M., Toward a Model of Strategic Outsourcing. Journal of Operations Management, Volume 25, Issue 2, March 2007. Pages 464-481.

[17] International Standards Organization (ISO), ISO/IEC 9126-1:2001, Software Quality Attributes, Software Engineering – Product Quality, Part 1: Quality Model, 2001.

[18] International Standards Organization (ISO), ISO/IEC/IEEE 42010:2011, Systems and Software Engineering – Architecture Description. 2011.

[19] Kruchten P., Lago P., van Vliet H., Building up and Reasoning about Architectural Knowledge. Proceedings of QoSA 2006, LNCS 4214, Springer 2006. Pages 43-58.

[20] Lago P., van Vliet H., Ali Babar M., Dingsoyr T. (eds.), Software Architecture Knowledge Management: Theory and Practice. Springer, 1st edition, August 2009

[21] Larman G., Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition). Addison Wesley, 2004.

[22] Leymann F., Roller D., Production Workflow – Concepts and Techniques. Prentice Hall, 2000

[23] Meszaros G., Doble J., A Pattern Language for Pattern Writing, http://hillside.net/index.php/a-pattern-language-for-pattern-writing

[24] Miksovic C., Zimmermann O., Architecturally Significant Requirements, Reference Architecture and Metamodel for Knowledge Management in Information Technology Services. Proc. of IEEE/IFIP WICSA 2011 (2011).

[25] Outsourcingsforum, Complete Outsourcing Request for Proposal (RFP) Checklist, http://www.outsourcingsforum.com/complete-outsourcing-request-proposal-RfP-checklist

[26] Power M., Desouza K., Bonifazi C., The Outsourcing Handbook: How to Implement a Successful Outsourcing Process. Kogan Page, 2006.

[27] Rowan L., Motsenigos A., Shuchat R., Dialani M., Tapper D., IT Outsourcing and Utility Services: Top 10 Predictions. IDC Research Paper, IDC #226609, Volume: 1, January 2011.

[28] Rumbaugh, J., Jacobson, I., Booch, G., The Unified Modeling Language Reference Manual. Addison-Wesley, 1999

[29] SEAFOOD: Software Engineering Approaches For Offshore and Outsourced Development, http://seafood.inf.ethz.ch

[30] Shaw M., Writing Good Software Engineering Research Papers: Minitutorial. Proceedings of the 25th International Conference on Software Engineering. International Conference on Software Engineering. IEEE Computer Society, 2003., Pages 726-736.

[31] Sood R., IT, Software and Services: Outsourcing and Offshoring. AiAiYo Books, 2005.

[32] Strategic Outsourcing Conference, http://www.conference-board.org/conferences/conferencedetail.cfm?conferenceid=2261

[33] Tang, A., Ali Babar, M. A., Gorton, I., and Han, J. 2005. A Survey of the Use and Documentation of Architecture Design Rationale. Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture. IEEE Computer Society, 2005. Pages 89-98

[34] The Open Group, The Open Group Architecture Framework, Version 9, 2009. Available online: http://www.opengroup.org/togaf

[35] U.S. Food and Drug Administration, http://www.fda.gov/ICECI/ComplianceManuals/default.htm

[36] van der Aalst, W., ter Hofstede A., Workflow Patterns initiative, http://www.workflowpatterns.com/patterns/index.php

[37] van Deursen A., Klint P., Visser J., Domain-specific Languages: an Annotated Bibliography. ACM SIGPLAN Notices, Volume 35, Issue 6, June 2000.

[38] van Heesch, U., et al., A Documentation Framework for Architecture Decisions. J. Syst. Software (2011), doi:10.1016/j.jss.2011.10.017

[39] Woods E., Using Design Principles to Unify Architecture and Design, Keynote, IEEE/IFIP WICSA 2009. Available from http://www.iso-architecture.org/wicsa2009/Eoin-Woods-WICSA.ECSA-Keynote.pdf

[40] Zimmermann O., An Architectural Decision Modeling Framework for Service-Oriented Architecture Design. Ph. D. thesis, Stuttgart University, 2009.

[41] Zimmermann O., Architectural Decisions as Reusable Design Assets. IEEE Software, vol. 28, no. 1, pp. 64-69, Jan./Feb. 2011

[42] Zimmermann O., Schuster N., Eeles P., Modeling and Sharing Architectural Decisions, Part 1: Concepts. IBM developerWorks, 2008

[43] Zimmermann O., et al., Managing Architectural Decision Models with Dependency Relations, Integrity Constraints, and Production Rules, J. Systems and Software and Services, vol. 82, no. 8, 2009, pp. 1246–1267

[44] Zimmermann O., Recurring Architectural Decisions, A Context-Specific Guide through SOA and Cloud Design, SEI SATURN 2010 tutorial, available from:
http://www.perspectivesonwebservices.de/download/ZRL-ADModelingTutorialMay2010v101.pdf

Author Biographies (Short)

Dr. Olaf Zimmermann is a senior principal scientist at ABB Corporate Research in Switzerland. His areas of interest include Web-based application and integration architectures, SOA design, and architectural knowledge management. Until January 2012, Olaf was a research staff member and executive IT architect at IBM Research, investigating the role of architectural decisions in the design process. Prior to that, Olaf worked as a solution architect and consultant, helping international clients in multiple industries build enterprise-scale SOA/Web services and Java Enterprise Edition solutions on professional services projects. In the beginning of his career, Olaf was a scientific consultant and developer in the IBM European Networking Center (ENC) in Heidelberg, Germany, focusing on industry-specific middleware frameworks for systems and network management. Olaf is a certified Open Group Distinguished (Chief/Lead) IT Architect and a member of the Advisory Board of IEEE Software. He is an author of Perspectives on Web Services (Springer, 2003) and

contributed to several IBM Redbooks including the first one on Eclipse and Web services (2001). Olaf received a PhD in computer science from the University of Stuttgart in 2009 and a “Diplom-Informatiker” (=MS) degree in Computer Science from the Technical University in Braunschweig (1993).

Christoph Miksovic is a Senior IT Architect and Software Engineer at IBM Research in Zurich, Switzerland. His focus areas are decision modeling for architectural decisions and strategic outsourcing, mobile computing, and business analytics. Prior to joining IBM Research, he was a solution architect at IBM Global Business Services working for clients in various industries on application development, application integration, and enterprise architecture consulting projects. Mr. Miksovic holds an M.Sc. (Dipl. Ing.) from the Swiss Federal Institute of Technology. He is an Open Group Master Certified IT Architect, TOGAF Certified, and a Certified Scrum Master .

Dr. Jochen Küster studied computer science with mathematics at the University of Paderborn, Paderborn, Germany, with stays at University College London, London, U.K., and Carleton University, Ottawa, ON, Canada. He received the degree of Diplom-Informatiker and the Ph.D. degree, with a dissertation on consistency management of object-oriented behavioral models, from the University of Paderborn in 2000 and March 2004, respectively. His thesis has received a Faculty Award from the University of Paderborn. Since July 2004 he has been with the Zurich Research Laboratory, IBM Research Division, Switzerland, where he is part of the Process Management Technologies Group. He is the author of more than 50 peer-reviewed publications in the area of model-driven development, model transformations, and business process modeling. His research interests include consistency management, model transformations, business process modeling, and model-driven development of service-oriented applications.