Architecturally Significant Requirements, Reference Architecture, and Metamodel for Knowledge Management in Information Technology Services

Christoph Miksovic
IBM Research – Zurich
Rüschlikon, Switzerland
e-mail: cmi@zurich.ibm.com

Olaf Zimmermann
IBM Research – Zurich
Rüschlikon, Switzerland
e-mail: olz@zurich.ibm.com

Abstract— **Capturing and sharing design knowledge such as architectural decisions is becoming increasingly important in professional Information Technology (IT) services firms. Methods, models, and tools supporting explicit knowledge management strategies have been proposed in recent years. In this paper, we extend previous work in the architectural knowledge management community to satisfy the requirements of an additional user group: the designers of IT infrastructure solutions that are outsourced from one company to another. Such strategic outsourcing solutions require complex, contractually relevant design decisions concerning many different resources such as IT infrastructures, people, and real estate. In this paper, we present a reference architecture and a decision process-oriented knowledge metamodel that we synthesized from the domain-specific functional requirements and quality attributes. We also present a tool implementation of these decision modeling concepts and discuss user feedback.**

Keywords-DSL;knowledge management; outsourcing;workflow

I. INTRODUCTION

Strategic Outsourcing (SO) [14] is an important trend in the Information Technology (IT) service business. 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 [21]. Designing such SO solutions is a complex undertaking and a knowledge-intensive process [8]. 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; hence, the preparation of a contract proposal is a project in its own right [1]. An example is the formal response to an official Request for Proposal (RfP) [19]. Proposal parameters include technology platform types (e.g., PCs, UNIX, and mainframe) and volumes (e.g., number of help desk requests, servers, and business transactions) as well as Service Level Agreements (SLAs) [19]. Two examples of SO solutions from opposite ends of the spectrum are short running standard services such as hosting a software package locally and long running, multi-country solutions involving a number of data centers and technology platforms that have to comply with regulatory compliance requirements [1].

The outlined business context of our work is related to the task of designing software-intense systems. Our practical application scenario comprises fundamental decisions that occur during the solution design process of an SO proposal. This application scenario differs from software design in several ways:

1. SO 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 (e.g., data centers and offices).
2. A lot of requirements are unknown during the early design phases. Due to the need to estimate costs accurately, the design has to be detailed before contracts are signed and billable projects are initiated. Hence, uncertainty about data quality has to be dealt with, as well as incompleteness of requirements. Scope and complexity of the SO business would suggest to analyze requirements and client environment in great detail before quoting any exact pricing information; however, market forces dictate a more agile design approach [6].
3. Business decisions and technical decisions go hand in hand. In addition to architects, sales executives, project managers, and domain SMEs (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 have been developed and successfully applied in recent years [17]. In this paper, we investigate whether AKM concepts are suitable for the domain of SO solution design and how these concepts have to be extended to satisfy domain-specific challenges and requirements.

We propose a knowledge management solution that aims at increasing design productivity and proposal quality. In this work, we leverage previous work from the AKM community and combine it with Business Process Management (BPM) concepts. In this paper, we present the following solution building blocks:

1. We compile a set of *architecturally significant requirements* that characterize the AKM needs in the SO solution design domain. These requirements drove our design work and we believe that they can

be beneficial for other researchers that target a similar problem domain or application scenario.

2. To frame the detailed design and development work in the problem/solution domain, we derive a conceptual *reference architecture* (i.e., a set of logical building blocks with their responsibilities and collaborations) from the requirements. Our reference architecture is based on business process management and workflow patterns [27] as well as previous work in the architectural knowledge management community [17][30].
3. To deliver one particular workable solution that can be deployed in practice, we design a decision process-oriented *metamodel*. This metamodel defines a Domain-Specific Language (DSL) that shapes and structures the interfaces and internal design of the components and connectors in the reference architecture.

The remainder of this paper is structured in the following way. Section 2 introduces the problem domain of knowledge management in SO solution design and establishes the challenges, functional requirements and quality attributes that frame our design work. Section 3 outlines the reference architecture, Section 4 the decision processing metamodel and DSL. Section 5 then covers the application of our concepts and their implementation to the SO solution design domain, including an evaluation of the user feedback; Section 6 investigates related work. Section 7 presents our conclusions from this work and an outlook to future work.

II. SOLUTION DESIGN IN STRATEGIC OUTSOURCING (SO)

In our work, we target the domain of SO solution design; our user community comprises solution architects and business decision makers working for presales organizations of IT service providers offering SO services. As already motivated in the introduction, not only software applications and software-intense systems are in scope, but also the IT infrastructures (servers, storage and networking devices, operating systems, middleware, etc.) 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).

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, know how transfer, real estate takeover, and IT system handover (to name just a few) [12]. A number of fundamental decisions have to be made, which are interdependent. While many requirements driving the design 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. Solution design decisions are made during the entire proposal phase. Information about usage of certain design elements on previous deals (i.e., in previous solutions for other IT service requestors) may exist in the form of lessons learned reports, but is not always exchanged and analyzed systematically (e.g., due to time pressure and lack of tools).

The concrete problem that we solve is the design of a knowledge base and a tool that support SO solution architects when they make the fundamental solution design decisions that have a high relevance for the following handover activities [6].[1]

A. Practical Challenges and Research Questions

Knowledge management challenges in the domain. Solution design in SO requires numerous interrelated business decisions as well as technical decisions to be made. According to interviews with members of the target audience and the literature [2][6][8][12][20][24], solution architects are confronted with the following 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 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 [26].
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).
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. Furthermore, it is not well understood how to express uncertainty regarding decision outcomes and when to revisit earlier decisions as more information becomes available or when requirements change. 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.
4. *Consistency and efficiency* challenge: As a corollary of the scope and scale challenge (i.e., large number of required decisions and design options that recur in the domain), the valid option set of a specific decision (instance) must be based on the status of predecessor decisions already made. This is required to confine the number of selectable options to those that are still eligible and lead to consistent, workable designs.

[1] Handover means that the IT service provider takes over legal responsibility and daily operations from the IT service requestor when 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.

5. *Reuse and education* challenge: During decision identification and decision making, decision makers welcome design guidance. This helps to increase the confidence that a design is adequate and to ensure that industry- or enterprise-wide standards including architectural principles [29] are adhered to. Such guidance is particularly useful when mentoring less experienced IT service professionals (e.g., training on the job).

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 [26] 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 decided for a custom service management tool for which no shared solution exists. Figure 1 illustrates the DP graph example:

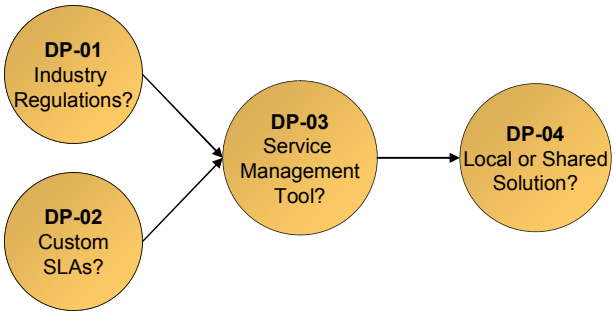

Figure 1. Example scenario with connected DPs

Generalization into research problem and questions. To overcome the five challenges described above, we envision a *Decision Knowledge Processing (DKP)* solution that aims at increasing proposal quality and team productivity. Several research questions arise from the challenges and the overall research problem of designing a DKP solution:

- *Scoping*: Which information about the decision points should be shared and when should it be used?
- *Presentation*: How should this information be displayed to the decision maker (e.g., ordered lists of required decisions, graphical visualizations of decision dependencies)?
- *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?
- *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?

Both practical challenges and research questions framed our solution construction (engineering) activities towards the envisioned DKP solution. The first step was to state the requirements explicitly. We report on this step next.

B. *Architecturally Significant Requirements for Decision Process-Oriented Knowledge Management Solution*

We now gather the architecturally significant requirements. These requirements cover both functional and non-functional concerns.[2]

Functional requirements. Through requirements engineering activities such as interviews with members of the target audience, user storytelling [4], and use case modeling [22], we identified the following functional requirements:

1. Activate relevant Decision Points (DPs) that can be decided as a consequence of decisions already made (decision identification support). Such functionality responds to the scope and scale challenge from Section 1.
2. Allow certain DPs to have multiple dimensions and instances, e.g., in order to express that different designs are required in each country in an international or global SO solution. Such support for multiple DP dimensions also helps to overcome the scope and scale challenge.
3. Visualize a DP graph with its DP nodes and their connections to overcome the priority and order challenge.
4. Propose suitable decision alternatives (options) in response to the reuse and education challenge.
5. Reference context-specific cases in which similar or the same decisions were made and provide best practices regarding decision making and decision enforcement. Such additional guidance also helps to overcome the reuse and education challenge.
6. Express confidence in a made decision and allow users to "undecide" one or more already decided DPs without loosing consistency in the DP graph. The addressed challenges are the priority and order challenge, but also the consistency and efficiency challenge.

[2] The linear structure of this paper might suggest a waterfall approach a.k.a. big design upfront; however, we followed agile practices and performed nine sprints (i.e., short, time-boxed development iterations). In this paper, we are only able to present the results of our requirements engineering activities; detailed information about our requirements gathering method (i.e., chosen process, notations, and techniques) remains out of scope.

7. Calculate various statistics, e.g., confidence level of decisions, risk implications, decisions already made vs. remaining decisions. This is in response to the priority and order challenge, as well as the consistency and efficiency challenge.
8. Leverage the information in the DP graph to generate reports, allowing reuse of this information in artifacts such as proposal overview presentations, approval forms, and review documents (decision enforcement support). Such features help to overcome the consistency and efficiency challenge, but also the quality and uncertainty challenge.

Nonfunctional requirements (quality attributes). In order to be able to be accepted by the user community, a DKP solution has to deliver its functionality with certain qualities (which we compiled iteratively by usage scenario):

1. The solution should prevent the decision maker from creating design elements that are contradictory, that can not be delivered efficiently, or that create unacceptable amounts of risk for handover and operations. The related general quality attributes (attribute types) are consistency and accuracy [15].
2. 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 is desirable (quality attribute: efficiency).
3. The status of the entire DP graph must be updated in real time when a DP is changed (accuracy and efficiency/performance).
4. The perceived response time is less than two seconds for user events (efficiency/ performance).
5. It must be possible to work in a disconnected mode (i.e., offline capability, no Internet connection) so that solution design-specific decision information can be captured anytime, e.g., when traveling to and returning from remote locations (usability).
6. Multiple design versions must be supported to allow iterative and parallel (concurrent) population of a decision model (flexibility/usability).
7. No development work should be required when the DP texts are authored and edited (e.g., when updating the knowledge base in response to changes to business model and technical context) or when the scope of the knowledge base is extended (modifiability, maintainability).
8. It must be possible for knowledge engineers that do not have any programming skills to create, configure, and maintain a DP graph rapidly (modifiability, maintainability).
9. User can not be assumed to be familiar with workflow concepts or general-purpose Business Process Modeling (BPM) languages (usability).
10. The installation experience must resemble that of standard personal productivity software, e.g., in the form of a standalone image that auto-installs itself instantly and hence does not require a system administrator (usability).
11. It must be possible to integrate the solution with other processes and tools according architectural principles such as loose coupling so that unreasonable development effort can be avoided (flexibility/adaptability).
12. The solution must adhere to enterprise-wide executive decisions [16], e.g., not to use any 3rd party software that has to be licensed and not to use certain open source assets that have problematic licensing schemes (modifiability, maintainability). The rationale for this executive decision and examples of problematic licenses can be found in [11].

In the next section, we present a DKP solution that satisfies these functional and nonfunctional requirements.

III. Decision Knowledge Processing Solution and Domain-Specific Knowledge

To overcome the knowledge management challenges and satisfy the requirements from Section 2, we combine a *domain-specific knowledge base* and a supporting *workflow-oriented decision knowledge processing tool* that is largely independent of the SO domain (anticipating reuse and application in other domains and application scenarios). In this section, we focus on the knowledge processing tool; we then discuss structure and content of the domain knowledge base in Section 4 and Section 5.

Figure 2 presents a layered view of our reference architecture. The novel components are displayed with a grey background, standard components appear in a box with solid lines and white background, future components are indicated by dashed borders and white background (unlike the novel components and the standard components, we have not implemented these future components yet).

Following a layered architecture design style [10], 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 and integrating other tools and data.

The frontend contains one logical component per user role, *knowledge engineer*, (i.e., decision point authors and maintainers) and *solution designer (decision maker)*, and *decision reviewer*. At this platform-independent level of refinement of the architecture, we do not make any assumptions about physical distribution or client implementation strategies and technologies such as Eclipse Rich Client Platform (RCP) or thin Web client.

The middle layer features *DP services* (a straightforward instance of the service layer pattern [10]) and a particularly relevant component in our reference architecture, the *Decision Knowledge Processing (DKP) engine and Domain Model* supporting all front end components. 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 and no deployment or code

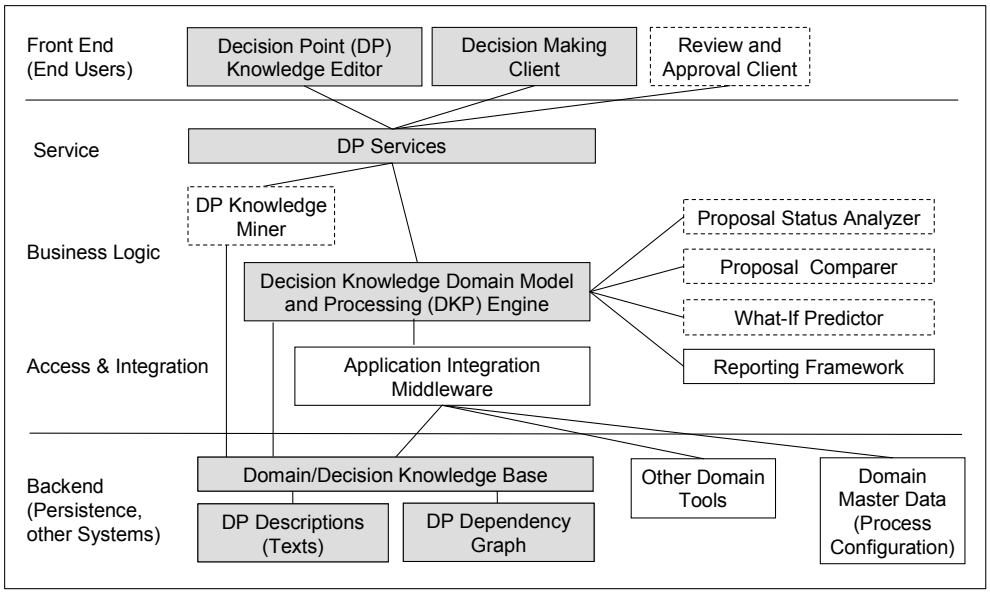

Figure 2. Reference Architecture for Decision Knowledge Processing

generation step is required. The design of this component is detailed in the following Section 5. The *application integration middleware* appears in response to the integration requirements; as it implements standard integration patterns [13] and can easily be implemented with the help of commercial products and open source assets, we do not feature it any further in this paper. The same holds true for the *reporting framework*. We also foresee a *proposal status analyzer* (calculating metrics regarding number of open and resolved DPs, degree of confidence in the chosen design, etc.), a *proposal comparer* (working with structural patterns in DP graphs), a *what-if predictor* (assessing probability and impact of negative consequences of a design change such as switching to another option in a particular DP), and a *knowledge miner* that is responsible for identifying relevant knowledge in proposal project documents and partially automating their conversion into formally modeled DPs.

The novel components in the backend are the *domain/decision knowledge base*, the *DP descriptions* and the *DP dependency graph*. The backend also comprises already existing solution design tools that our DKP solution integrates with, as well as a proposal process configuration database that contains information about countries potentially being in scope and names of review boards (process milestones).

The components can easily be traced back to the functional requirements we established in Section 2. For instance, all DP graph processing takes place in the DKP engine. 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 DP graph configuration that ensures that the DP processing flow is not fixed anywhere in the code (e.g., in the DKP engine).

The components cooperate in the following way. The knowledge engineer enters DPs (like DP-01 to DP-04 from Section 2) and connects them in the DP knowledge editor. Via the DP services and 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. The knowledge engineer then validates this knowledge base with the help of the validation features in the DKP engine, as exposed through the DP knowledge editor. Next, a test user (acting as solution designer and decision maker) works with the features that are available in the decision making client, e.g., reads DP texts, clicks on links to reference information, decides DPs using different option expressions (observing how the status information changes), and generates reports. Several visualizations support him or her during these decision identification, making, and enforcement activities.

Once these testing activities succeed, the DP graph then is made available to target audience via a released tool implementation, one of which will be featured in Section 5. 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.

IV. A 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 DP domain model with the DKP engine, and the domain/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 [17][30]. In this section we introduce this metamodel both informally and formally.

In the metamodel, individual DPs provide decision 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 certain design is actually implemented and becomes visible in proposal documents such as the contract). DPs can be categorized by criteria such as functional domain or process milestone (review board linkage).

Figure 3 illustrates these and other DP attributes informally by giving them a name, providing rationale in question form (from a user's perspective), and relating them to the DP graph. The confidence level and decision status, for instance, may be used by the proposal status analyzer

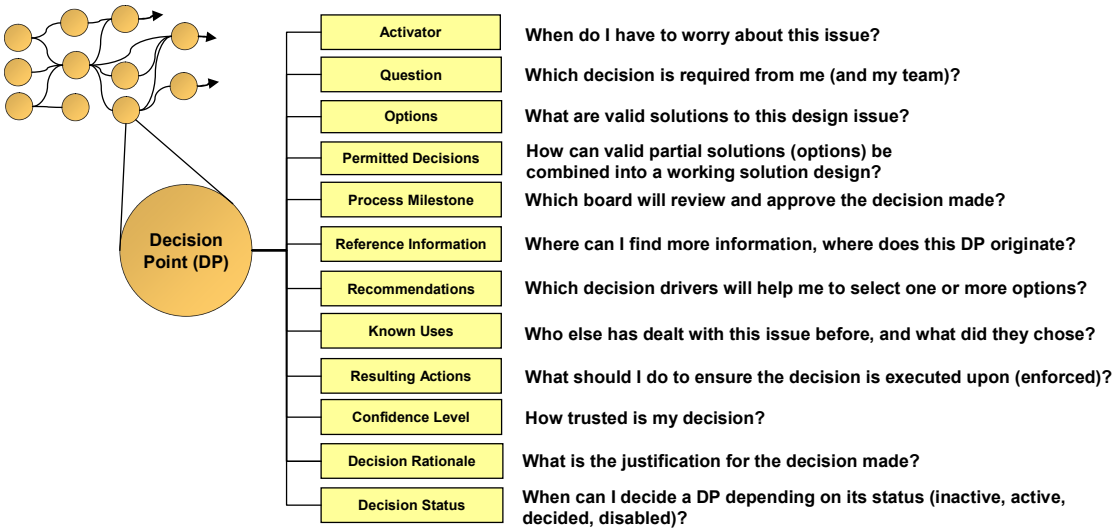

Figure 3. Attributes of a DP that is embedded in a DP graph

shown in Figure 2 to calculate metrics which are then displayed and exported via the reporting framework. Two examples of such metrics were given in Section 3.

Decision Points (DPs) and their dependency relations are modeled as a directed graph consisting of nodes and connections such that the dependencies form a partial order [18]; individual DPs are modeled as finite state machines [22]. The number of decisions to be made is minimized depending on the state of the individual DPs in the DP graph. This overall state of the DP graph is defined by the aggregation of the states of the individual DPs, which allows reducing the number of DPs that still have to be decided.[3] The calculation of meaningful selectable options based on predecessor decisions is based on previous metamodels suggested by the AKM community [17][30]. The DP graph metamodel design is formalized in the class diagram shown in Figure 4 (attributes not shown).

The key entity types (i.e., classes) in the metamodel are *activator*, *Decision Point (DP)*, and *option*. They jointly form a *decision model* which is organized into a discrete (non-intersecting) set of *category* instances. All this information is recurring and therefore entered into the knowledge base by the knowledge engineer; during decision making, the solution architect then creates *decision* instances referring to specific instances of permitted decisions.

A DP represents a required decision. It acts as an aggregate root [7] 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, as explained previously).

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 and constrained by one or more alternative *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. Furthermore, specific *selection patterns* can also define the chosen options in a valid option set.

DPs are connected via options of predecessor DPs. More precisely, a DP contains one or more (disjoint) activators, each of which is connected to options of predecessor DPs. The control flow (which induces state changes in DPs) is calculated internally/derived automatically from the data flow. This data flow is expressed by connections between one or more options of predecessor DPs that are referenced in an activator and the permitted decision of this activator, which lists a set of valid options for the DP which contains the activator. The rationale for this graph-based approach to decision ordering can be found in the second decision making challenge we motivated in Section 2, the ordering and prioritization challenge. In Section 5, we will discuss state management and DP ordering in more detail in the context of our tool implementation.

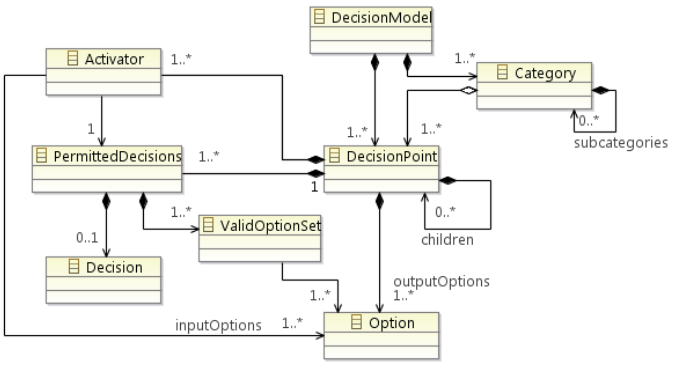

Figure 4. Key classes of the metamodel

The multi-instance workflow pattern [27] allows us to represent multiple DP dimensions (e.g., countries) as DP

[3] With respect to the data flow between decided DP options, our DP concept takes inspiration from the signal transmission domain in electronics.

instances. Each of these instances has its own state and participates in the data flow logic through its own activators.

V. Implementation of Concepts and Application to Strategic Outsourcing Domain

We validated the feasibility and practicality of our concepts through a tool implementation and the creation of an actual decision model. Tool implementation and decision model were applied first using test data from past proposals and then on a series of full scope proposal projects with live users.

A. Tool Implementation

We used the Eclipse platform (e.g., Eclipse Rich Client Platform, Eclipse Modeling Framework, Graphical Editing Framework, and BIRT reporting) to implement the concepts from this paper. The current implementation supports about 50 use cases; its domain model comprises about 15 entity types (Figure 4 shows a simplified version of the actually implemented metamodel). To manage decision dependencies, the tool has workflow semantics; the sole activity type is the Decision Point (DP). The implementation consists of 11 physical components that are packaged as Eclipse/OSGi plugins. We refer to this implementation as *Solution Decision Advisor (SDA)* in this paper.

Figure 5 shows the user interface (decision making client) of SDA.

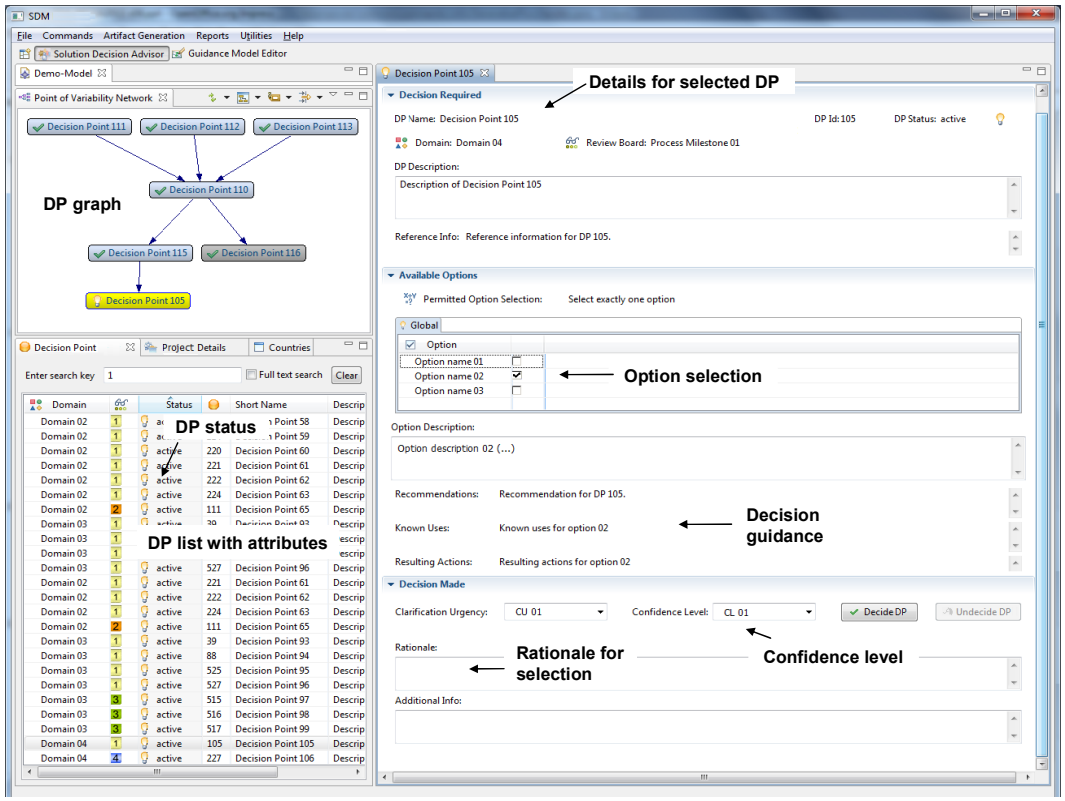

Figure 5. User Interface of the Solution Decision Advisor (SDA)

As specified in the reference architecture and metamodel, DPs are represented as state machines. SDA currently supports four states, *inactive*, *active*, *decided*, and *disabled*. State transitions are triggered by events. Any new DPs is active by default; a DP with one or more inbound dependencies from any active predecessor DP is inactive. An inactive DP changes its state to active when all inbound dependencies connect the DP only to decided or disabled DPs (i.e., no more dependencies from active DPs exist). An active DP changes its state to deactivated 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 6, decided DPs can be undecided; in this case, their states change to active. Consequently, the states of all dependent DPs are adjusted accordingly as well.

In SDA, DPs are categorized by arbitrary criteria. Currently three criteria are supported: *functional domain*, *business process milestone*, and *editorial state*. One key feature is that solution decisions are now documented in one place. Decision making guidance is delivered to the user (e.g., an SO solution architect) as specified in Section 4 (recommendations, known uses, etc.). DPs are activated by configurable rules that depend on state changes of connected predecessor DPs. These rules create a partial order; the decision maker is free to prioritize (i.e., select, review, and decide) any active DP. The process milestone criterion (and metamodel attribute from Section 4) aims at assisting the decision maker in this prioritization effort.

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

content management can optionally be performed in other tools.

B. Live Pilots

Having conducted user acceptance tests with real users and data in September and October 2010 (to evaluate value and usability), we released SDA Version 1.0 to early adopters on real proposal projects. This version ships with a reusable DP model comprising 109 fundamental solution design decisions. At the time of writing, SDA had about a dozen active users who already provided valuable feedback regarding value, usability, and missing features. One of these users had contributed substantially to the creation of the DP model. More dissemination and exploitation activities have already been scheduled.

During project initiation in the beginning of 2010, we had to address certain stakeholder concerns. For instance, several solution architects stated that enough processes and tools exist already and that any additional learning and processing effort would not be welcome. These workload concerns could be overcome by integrating these existing assets. Moreover, we created a demo version of tool and knowledge base that included a realistic example. Our tool demonstrations had a 100% success rate, i.e., the involved prospective users requested to become members of the early adoption program or asked for other follow up activities.

The members of they early adoption program came from several countries and had different amounts of professional experience in their current job role (i.e., from two to more than 20 years). We received rather detailed feedback both in writing and in official feedback meetings. For instance, we received and answered many detailed questions about the semantics and business rules justifying the introduction of individual DPs and DP connections (showing that test users indeed worked with the SDA tool and DP knowledge base).

All modeled DPs were confirmed to be relevant and recurring. Only a few requests for additional DPs were made (e.g., for DPs dealing with industry-specific regulations). DP names and 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 consider it to be important to separate objective, official recommendations from subjective, personal statements. Preserving knowledge provenance information is required for auditing and archiving purposes.[4]

From a user interface design standpoint, the search and filter capabilities in the DP list view (see Figure 5) and the explicit status and dependency management were greatly appreciated. Ordering by process milestone (review board) was seen to be a valid approach as many of the proposal activities are driven by review and approval needs. The reporting and artifact generation features were seen to be powerful and critical success factors. To review and make a single decision is a matter or 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; however, in such setting it would be more difficult to prune (remove) DPs and DP options that are no longer relevant. Furthermore, it would be more difficult to support the various features motivated and outlined in Sections 2 to 4 (e.g., reporting, proposal status analysis, and integration with other tools).

Agile practices were applied during the population of the knowledge base (guidance model creation). We worked with senior subject matter experts to conduct *pair knowledge engineering* sessions. This novel form of technical writing took inspiration from eXtreme Programming (XP). To review the entire DP graph with all texts and provide comments in writing takes less than one person day. We already performed three major and several more editing cycles to get to version 1.0. Due to this positive experience, we assess the knowledge base to be maintainable.

In summary, we consider the development of concept, tooling, and knowledge base a success as we could demonstrate both value and technical feasibility.

VI. Related Work

While the DKP engine in our reference architecture (Section 4) resembles a general purpose workflow engine and realizes a subset of the workflow patterns [27], its design differs from previous work in the BPM 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 DP instance 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, our approach provides a domain-specific method and tool for decision making that allows knowledge engineers to model detailed design variations and the relationships between them.

Our metamodel (Section 4) can be viewed as the syntax of a Domain Specific Language (DSL) [28] for decision identification, making, and enforcement. In accordance with the DSL design philosophy, it provides concepts that allow knowledge engineers to model and configure a decision guidance system with a minimal set of syntactical elements. An instance of this metamodel (e.g., the decision model outlined in Section 5) can be viewed as a *semantic model* [9] for a particular domain (here: SO solution design).

We extend and complement previous work in the Architectural Knowledge Management field [16][17]. For instance, in our SOA Decision Modeling (SOAD) work [30], we 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 Section 5 for examples) and

[4] These suggestions lead to usability improvements, e.g., allowing the tool to only show a relevant subset of attributes in a particular view.

optimized the decision dependency management (e.g., we give a more precise definition of temporal relations activating decision points based on earlier decisions).

Previous work in strategic outsourcing and services science highlighted similar challenges [23][24][25], 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.

VII. Conclusions and Summary

This paper dealt with the domain of decision guidance for IT service professionals. It extended existing work in the architectural knowledge management community (i.e., work on architectural decision modeling with reuse) to cover the distinct requirements of another application domain, the design of complex strategic outsourcing solutions.

We identified five knowledge management challenges in the domain: 1) scope and scale (of decision models), 2) (decision making) priorities and order, 3) data quality and uncertainty, 4) consistency and efficiency, and 5) reuse and education. To overcome these challenges and create a domain-specific decision knowledge processing solution, this paper contributed a compilation of domain-specific, architecturally significant requirements, a reference architecture, and a metamodel (i.e., a workflow language optimized for decision processing). Reference architecture and metamodel are implemented in a tool and knowledge base, which have been released to early adopters and validated in a series of live pilots (i.e., real proposal development projects).

In the previous state of the art, technical solution design was done in office products such as presentation tools, spreadsheet editors, and word processors; our solution models the solution design decisions explicitly so that they can be managed effectively. Hence, solution designs become comparable; deviations from standards and best practices can be detected and it becomes easier to evolve designs when requirements and other constraints change. Moreover, routine tasks as well as difficult steps such as those required for moving from the proposal phase to the handover activities can be quality assured or even partially automated. As far as drawbacks and liabilities of our approach are concerned, a major success factor for a sustainable production use of our solution is the need for a support and maintenance organization. From our own knowledge engineering experience gained throughout 2010, we consider the maintenance effort to be justified and adequate.

We also have already collected several enhancement requests from the early adopters involved in the pilots. A future research and development direction that is particularly relevant is a risk and project management interface: The information in the decision model (e.g., confidence levels and rationales) can be leveraged to compare the current design with previous ones so that problematic situations requiring risk mitigation actions can be identified and reported upon semi-automatically. We also consider applying our approach to business domains outside IT.

Acknowledgment

We would like to thank Thomas Diethelm and Ivan Kuraj for their contributions to the project 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] 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.

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

[8] 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

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

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

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

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

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

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

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

[16] 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.

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

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

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

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

[21] 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.

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

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

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

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

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

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

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

[29] 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

[30] 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.