Architectural Decision Modeling with Reuse: Challenges and Opportunities

Marcin Nowak, Cesare Pautasso
Faculty of Informatics, University of Lugano (USI)
via Buffi 13, 6900 Lugano, Switzerland
http://www.pautasso.info/
marcin.nowak@usi.ch, c.pautasso@ieee.org

Olaf Zimmermann
IBM Zurich Research Laboratory
Säumerstrasse 4, 8803 Rüschlikon, Switzerland
http://www.zurich.ibm.com/
olz@zurich.ibm.com

ABSTRACT

Architectural decision modeling aims at supporting the software architecture design process by capturing a reusable body of architectural knowledge. Whereas significant progress has been made towards this vision, there still remains a number of open problems. This paper outlines selected research challenges and opportunities related to knowledge capturing and sharing, model evolution and verification, and the integration of the architectural design process with existing software development methodologies. Our goal is to start a discussion on a roadmap for future research on reusable modeling of architectural decisions.

General Terms

Design, Documentation, Measurement, Verification

Keywords

Software Architecture, Architectural Decision Modeling, Visualization

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; D.2.2 [**Software Engineering**]: Design

1 Introduction

Architectural decision modeling is an emerging research area in software engineering [7]. It puts decisions at the center of the software design process and uses principal decisions as the main conceptual element to describe a software architecture [10, 11, 20, 37]. The process of taking decisions practically embodies the transformation of the rationale and requirements into the architecture of a system. Software architects need to be supported in their decision making by architectural design tools which provide the following features: 1) Capture a body of architectural knowledge [20, 29] for future reuse [26]; 2) Support the design process and the decision making of the software architect [8, 40]; 3) Improve architectural knowledge propagation within and among project teams [6, 19]; 4) Monitor the evolution of a project decision space over time [32].

For each of these features, we identify a set of open research challenges and requirements for the design of future architectural decision modeling tools and environments. For example, we propose to enrich the captured architectural knowledge with metadata and tags to open up new ways of traversing, analyzing, and measuring the design space. We suggest a new, fuzzy approach for making architectural decisions and identify the need for verifying the consistency of a design as the underlying architectural knowledge evolves. Since architectural decisions need to be shared among a potentially distributed design and development team, we discuss the opportunities related to collaborative decision making and the need for role-based access control to the architectural knowledge. In this context, it is also important to understand the relationship between the architectural decision making process and the chosen software development methodology as this will affect the integration of architectural decision modeling tools with existing architectural specification and modeling tools [13].

The main contribution of this paper is to outline selected research opportunities. These opportunities have been identified based on our collective experience in industry projects and by surveying the existing scientific literature. We do not lay claim to the completeness of the set of identified research challenges, as our goal is to start a discussion on a possible roadmap for future research in the area of architectural decision modeling for reuse.

This paper is structured as follows: in Section 2 we introduce and define terms and concepts used throughout the paper. Section 3 contains a case study which is going to serve as a running example. Sections 4 to 7 present a collection of research ideas, outlining their potential benefits with several examples in the context of the case study. Section 8 contains references to related work. In Section 9 we present our conclusions.

2 Background

In order to store efficiently the architectural knowledge, it is important to define an expressive conceptual model. In this section we do not aim at proposing yet another architec-

SHARK '10, May 2-8 2010, Cape Town, South Africa

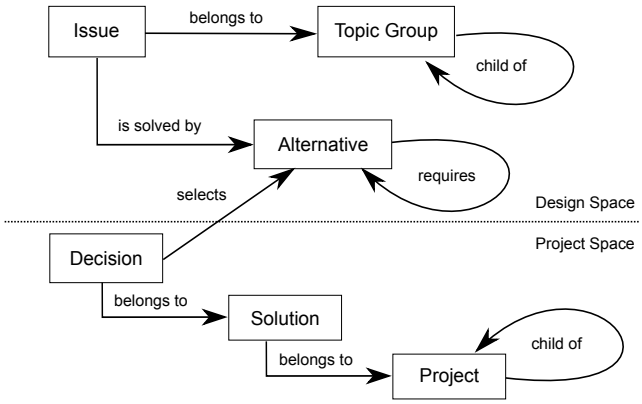

Figure 1: Relations between the design and project space artifacts

tural knowledge metamodel, but rather to distill a common set of core concepts agreed upon in the literature [3, 8, 15, 16, 17, 23, 25, 40]. Our purpose is to introduce the terminology that will be elaborated upon throughout the paper.

For the purpose of clarity we have decided to divide the entities of the architectural knowledge into design space and project space (Fig. 1). The design space collects reusable design entities intended for sharing, such as issues and alternatives together with a number of relations defined between them. The project space consists of the entities specific for the particular application, such as decisions and solutions.

Within the design space, issues represent a design concern and alternatives render a potential choice to address the corresponding design concerns. An *is solved by* relation binding an issue with an alternative indicates that a particular alternative is suitable to address a given issue. Further relations such as *refined by*, *decomposes into*, *conflicts with*, *enables*, *subsumes*, *overrides* or *forces* were proposed in the literature [8, 39]. Two alternatives can bound by the *requires* relation indicating that when one is selected, also the other should be chosen within the corresponding decision. The purpose of topic group is to logically group issues. In order to hierarchically structure a large set of topic groups, these can be related to another topic group with the *child of/parent* relation.

In the project space, the central role is played by the decision entity. A decision is made to *select* the alternative most suitable to address a certain issue. Decisions can be grouped into a solution. As a next aggregation step, multiple solutions can be grouped in a project. Finally projects can be hierarchically structured.

3 Case Study

We illustrate the ideas collected in this paper with a case study related to SOA. In particular we focus on the choice between RESTful and SOAP/WS-* Web services [33].

The artifacts presented in Fig. 2 are depicted with rectangles to represent various technological and conceptual issues. Ellipses represent architecture alternatives. The *is solved by* relation between issues and alternatives is illustrated with solid arrows. Alternatives bound by the *requires* relation are connected with dashed lines.

Overall, this design space is concentrated around the two main alternatives (*REST* vs. *SOAP/WS**) of the issue *Web Service Paradigm*. Most of the alternatives are related to a single, specific issue (for example *Design Contract*, or *Web Service Paradigm*). However, there exists also an alternative (*HTTPS*) which is related to multiple issues (*Security* and *Transport Protocol*). More details about the case study will be provided as we introduce each of the challenges in the rest of the paper.

4 Knowledge Capturing

Knowledge is of essential value for every modern organization. If not captured in the right moment it evaporates easily [35]. From the perspective of the software design process the following three methods of knowledge acquisition were proposed:

1. research and development of greenfield projects [20],
2. decisions recovery of existing projects either undergoing refactoring or documentation [21, 22],
3. harvesting pattern repositories [30].

In order to organize the broad variety of the knowledge artifacts, we introduce the concept of of *knowledge domain*. A knowledge domain consists of the architectural knowledge and metadata describing the content of a related set of design issues.

4.1 Tags

Capturing a large amount of knowledge is not enough, unless the knowledge is properly organized. In order to make the design space knowledge accessible and reusable, a classification and categorization strategy is needed.

Our idea is to organize architectural knowledge with tags. Considering the complexity of the captured architectural knowledge, providing a multi-dimensional perspective over the design space could help to improve the classification and the navigation over the artifacts and their relations. Tags appear to be a perfect candidate for this purpose [34]. It is reasonable to use typed tags because of the potential they provide for systematic analysis, however we are also considering to offer free-form tags for giving users the ability to quickly introduce custom classifications. By introducing a rich set of tags we expect to provide architects with a flexible way to browse through architectural knowledge by following the hierarchical structure of nested topic groups [40] or the cross-cutting perspective provided by other tags.

A good example are tags describing the *Phase* (e.g., *Micro design*, *Macro design*, *Solution Outline*) in which a decision should be made [39]. Also, the *Role* within the development team can be represented using tags (e.g., *Architect*, *Modeler*, *Integrator*) associated with the issues that fall under the responsibility of the corresponding role. Another interesting tagging application are user satisfaction and rating tags. This Web 2.0 concept should promote the accumulation of content with good quality by encouraging its continuous improvement and revision.

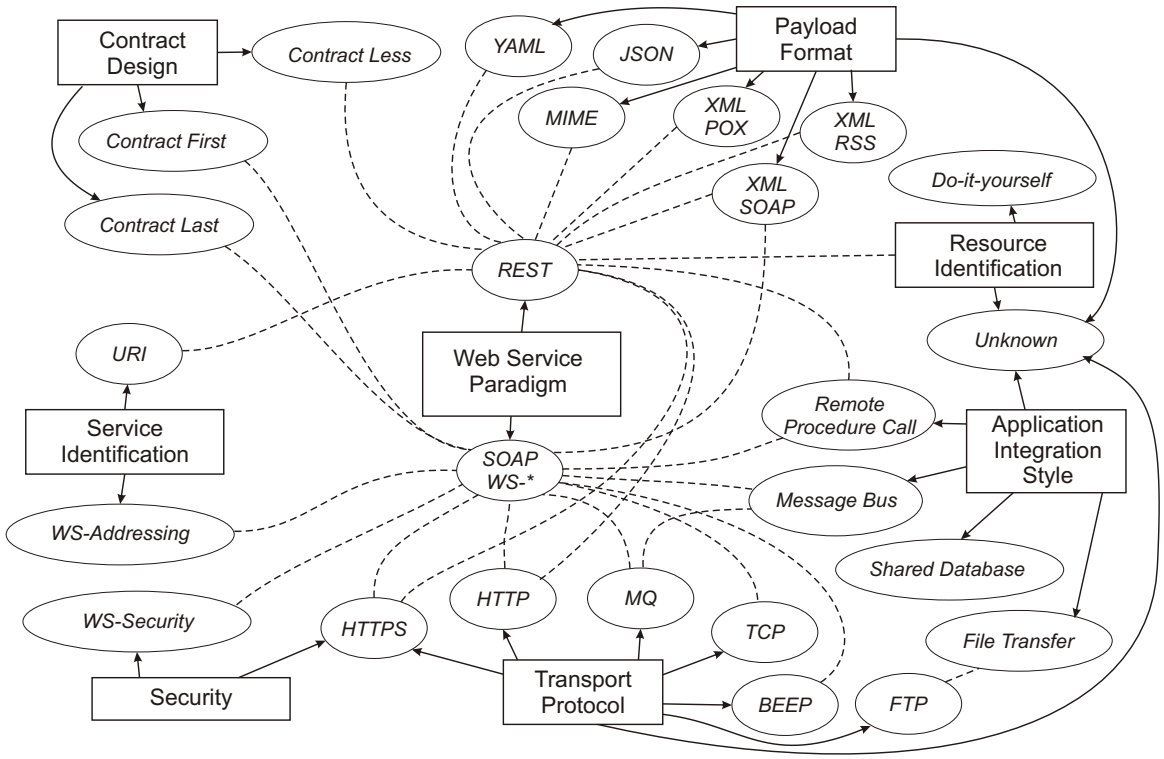

Figure 2: Case study design space on the REST vs. WS-* topic group

4.2 Metrics

We propose to apply a set of quantitative metrics to analyze the knowledge stored in an architectural design space. For example we want to measure completeness, complexity, descriptiveness and entanglement level. The benefit of expressing these attributes quantitatively is a clear indication of the effort needed to address particular issues of the design space.

In order to measure completeness we introduce the notion of the *Unknown* architecture alternative. This alternative acts as placeholder for an unspecified set of alternatives, which have not yet been captured. By counting the number of references to the *Unknown* alternative we could estimate the expected uncertainty of a decision. Indeed, this makes it possible to pin point only the unknown that we are aware of. We nevertheless believe that making unknown issues and alternatives explicit in the model can contribute to increasing the accuracy of uncertainty estimation and risk management activities.

Taking an example from the case study, a decision over the *Contract Design* issue, which does not reference the *Unknown* alternative, is different than deciding over the *Transport Protocol* which indeed references *Unknown*. Another example would be *Resource Identification* and *Payload Format*. In the first case *Unknown* is one of two alternatives, in the other it is only one of seven.

With the goal of encouraging a uniform content distribution over the design space, we suggest to measure the descriptiveness of the artifacts by counting the number of attributes and tags attached to them.

As suggested in [33], a possible complexity metric could be obtained by simply counting the issues and their related alternatives found in a specific scope of the design space.

In order to measure the level of entanglement of the artifacts, some form of distance between issues and alternatives linked by a given relation could be introduced. For example, the *Web Service Paradigm* issue is the most entangled as it is related (through its two alternatives) to all other issues in the case study design space.

All of metrics that can be applied to the architectural knowledge are supposed to be calculated within a given scope. The scope of applicability of a metric may range from global (where the entire domain is assessed), through a subset of the artifacts filtered by a given tag, topic group, or project, all the way down to individual artifacts.

By providing quantitative measures over the design space we expect to discover patterns emerging in specific areas. We plan to codify recognition of disharmonies [38] with use of an approach similar to detection strategies [31]. This way, we will locate elements of the design space which may have fallen into stagnation or are expected to be unclear and volatile. Likewise, complexity and descriptiveness metrics can give an indication on which artifacts may require refinement because their representation is not sufficiently developed or require simplification, decomposition and clarification because they are too complex.

4.3 Visualization

Storage, analysis and processing of the knowledge accumulated in the design and project spaces is very important, but

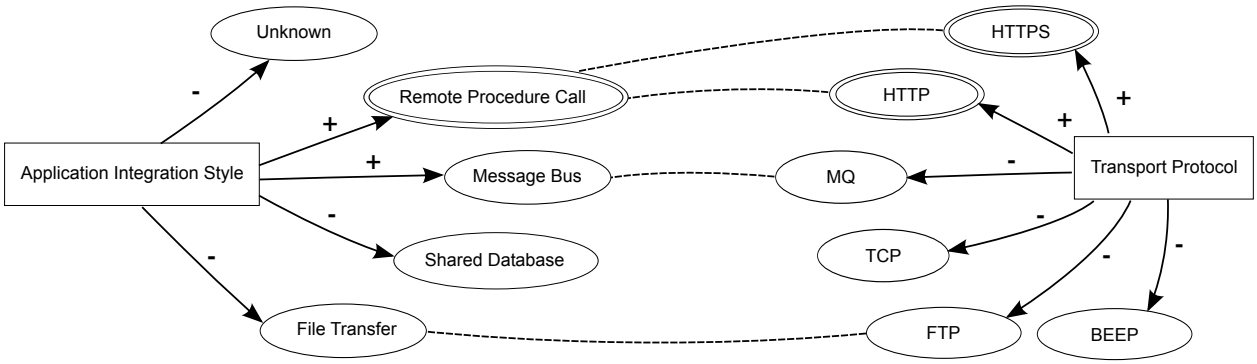

Figure 3: Fuzzy decision modeling with *Acceptable* (+) and *Unacceptable* (-) decisions.

needs to be complemented with an efficient way of communicating it to the architects so that its reuse can be promoted. Taking the very small example of the case study, which only contains 8 issues and 24 alternatives, it is already noticeable the difficulty of visualizing the design space artifacts and relations in a readable way [26].

The content of the design and project spaces could be visualized in two or three dimensions. Traversal through the design space nodes should be enabled by following tags and relations between artifacts. Metrics computed over the artifacts should be overlaid to enhance the visualization to ease intuitive localization of the design hot-spots and the visual assessment of selected artifacts properties. We expect interactive exploration of the project space to be a powerful tool in order to assist with collaborative design, to facilitate brainstorming workshops and to support learning about the knowledge captured in the design space.

5 Design Process

The software design process is a process of making decisions [18]. The more consciously and precisely decisions are taken, the higher quality of the design is. In the ideal situation, each software architect would have unlimited knowledge about the problem and a complete set of architecture alternatives to choose from. In practice, architects possess limited knowledge about the possible solutions and even more limited knowledge about the requirements. We introduce the concept of fuzzy decisions in order to support conscious decision making in conditions of limited certainty. We observe the need for verification in order to automatically check that decisions made within a limited view of the overall architecture do not introduce inconsistencies. Finally we present a sketch of our ideas on how to best support the process of software design and development within a decision-centric environment.

5.1 Fuzzy Decision Modeling

Making a design decision requires to pick a single architecture alternative. However, in some cases it may not be immediately possible to do so. A fuzzy approach to decision making provides the architect with support for pruning unwanted alternatives from the design space without immediately having to converge on a single choice.

For every issue, alternatives being considered can be separated from the ones likely not to be chosen. Allowing the existence of multiple *acceptable* alternatives opens a path leading from architectural synthesis to architectural evaluation [8], when multiple solutions are compared in order to optimize selected quality attributes.

By analyzing fuzzy decisions over a group of issues we plan to supply architects with information on the progress of the decision making process. This can be used to estimate the effort required to complete the design. Also, a range of possible solutions offering specific qualities can be compared [24].

In the scenario of the case study, the architect deciding over the *Application Integration Style* may choose both *Message Bus* and *Remote Procedure Call* as acceptable choices and discard the other two alternatives (Fig. 3). The final decision needs to be done considering additional constraints (for example the requirement implied by *Message Bus* to use the *MQ* transport protocol within the *SOAP-WS-** Web services paradigm). By pre-selecting the acceptable alternatives the decision making effort can be reduced as the set of alternatives to choose from is smaller. As shown in the example of Fig. 3, rejected alternatives can be pruned, leaving only the acceptable alternatives marked with double ellipses to choose from.

5.2 Verification

Good software quality does not come for free, because achieving quality requires quality control. The principal quality of the architectural decisions is their consistency. Our plan is to achieve consistency by means of both internal and external verification.

First we want to check if decisions are consistent locally, that is within the scope of a single issue. For example considering the issue of *Contract Design*, it would be illogical to decide for *Contract-First* and *Contract-Last* at the same time. At a wider scope, we also want to make sure that decisions made over one alternative in different contexts are not contradicting each other. For example choosing *Message Bus* for the *Application Integration Style* and not deciding

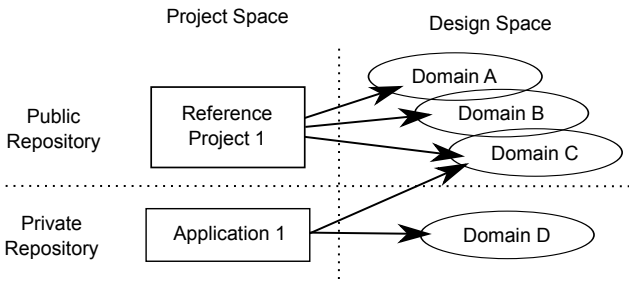

Figure 4: Relations between project and design spaces divided between private and public repositories.

for the *MQ Transport Protocol* is not consistent because the *Message Bus* alternative requires *MQ*.

Second, we want to verify the external consistency of a solution by following relations between artifacts (i.e., issues and alternatives) that belong to domains overlapping with the local domain of the project (Fig. 4).

Verification can be especially beneficial when the design space naturally evolves independently of the project space and the external consistency of the decisions needs to be rechecked. More in detail, accumulated knowledge in the design space grows over time. During the lifetime of a project it often happens that the state of the art evolves as new technologies emerge and new design patterns and paradigms are discovered. Verification can help to find out if some decisions made in the project are no longer suitable and consistent with updates in the design space.

By automating verification and by that reducing the effort required to verify the consistency of solutions, we expect to improve the quality of the design in the same way as best development practices such as continuous integration and testing improve the quality of an implementation.

5.3 Design Process Support

Navigating a new and large design space can be a daunting task for beginners and experts alike, if the specific body of architectural knowledge is unknown and needs to be learned from scratch. Where to start? When to stop? A tool should be able not only to visualize the relationships and implications between decisions in order to help architects build a mental map over the design space, but also to suggest important starting points (which decision should be considered first? which decision next?) to bootstrap the design process. As the architect makes progress going over the design space for a certain project, a tool should also be able to estimate how soon the design effort will be nearing completion.

We plan to investigate multiple methods of assisting the browsing over the design space. We think that depending on the nature of the project, specific top-down, cross-cutting, bottom-up approaches might be applicable. A basic top-down approach is to be implemented on the basis of topic group tree browsing. Second, cross-cutting methods can be applied by filtering the artifacts associated with chosen tags in order to quickly select issues of specific characteristics (e.g., all issues related to security). Third, random access assisted by full-text search can be very helpful to support a bottom-up, free hand approach to design.

For example, the case-study could be approached using a top-down design process. The architect should be advised to first decide over the higher-level *Conceptual* issues (such as *Application Integration Style*) and later with lower-level *Technological* issues (such as *Payload Format* or *Transport Protocol*). Still if there are *Technological* constraints which are specific for the application, they should be included into the project space first, so no conflicting decisions are made. For example, if the constraint of selecting *YAML* as a *Payload Format* exists, the architect should be advised that the single alternative of *Web Service Paradigm* that conforms with this constraint is *REST*.

5.4 Development Process

The architecture is at the center of the software development process [37]. Where should the architectural decision making process be placed? We think that design is an essential part of software development process. The challenge is to find a process-agnostic method of coupling development and architectural decision making.

Waterfall and iterative processes benefit from stable (but slow) growth of the domain knowledge. Agile processes [9] profit from the rapid adjustment of the project to a continuously changing environment. We see a great potential in combining the best of both, that is making decisions right and making decisions fast.

To do so, we aim to provide architects with an immediate interpretation and feedback of decisions as soon as they are made. This helps to grasp their implications in the design space [36]. We believe that this functionality will let architects state many "what-if" questions and verify the applicability of many solutions. This will positively affect the quality of decisions taken. For example, an architect deciding the *Application Integration Style* to follow in a project should be advised that, for example, the decision to use a *Message Bus* is connected to other conceptual decisions such as the choice of *SOAP-WS** as *Web Service Paradigm* and that only the *Contract-First* or the *Contract-Last* alternatives will remain acceptable for the *Contract Design* issue. Likewise, the issue on *Resource Identification* will not need to be considered any longer.

An additional opportunity to improve the quality of design decisions is to make the most of the experience gathered over the course of a project. By analyzing the experience gathered by the various actors of the development process we want to provide a multidimensional view over past history of a decision. For example, architects pondering over the *Contract Design* issue may find the negative experience record associated by developers with the *Contract-First* method, a good reason to chose *Contract-Last* instead.

6 Knowledge Sharing

Gathering, transforming and analyzing knowledge is important, but how far one can go with one's own wisdom? We see big potential in delivering the knowledge to those who really need it [35] by means of appropriate support for sharing and dissemination of knowledge across large development organizations [19].

6.1 Collaboration

The issue of collaborative work over the project and design spaces offers two kinds of challenges. One is the exchange and sharing of the general knowledge of the design space, the other concers how to best support the collaborative decision making process.

The idea is to introduce the concept of knowledge exchange between the architectural knowledge repositories. Nonlinear repository management [1] could contribute the most appropriate technique to handle the unstructured and uncoordinated build-up of the content. The challenge is to find an efficient, semi-automatic mechanism to propagate changes and updates to the knowledge.

In industrial scenarios, the concept of intellectual property and ownership of the knowledge shared in a repository becomes important. We believe this could require to distinguish two types of such repositories, namely private and public. A clear separation of public and private knowledge will make it possible for architects to benefit from the public knowledge without disclosing proprietary information (Fig. 4).

Collaboration also requires tool interoperability so that the design space can be accessed from different tools and exchanged between multiple repositories. Architectural Knowledge thus requires a flexible data interchange format that will carry not only data, but also the metadata describing it. Following [4] we think that using the Resource Description Framework (RDF) [2] might offer interesting opportunities to explore how to serialize a design space for exchanging its architectural knowledge. RDF is not only a well known standard, but it also provides the mechanisms to include the data together with the corresponding metadata.

6.2 Knowledge Access Control

Building upon the concept of distinguishing between public and private content of a design space, we identify the challenge of providing fine-grained access control to individual artifacts. Applying role-based access control allows project managers to adjust the access rights to specific scopes of the design space according to the needs of the specific development team. This is a requirement of commercial development projects where design, management, and implementation often are done by independent entities and separation of competencies is desired.

For example, in a structured software development process architects designing an architecture should use, analyze, and produce an explicit description of the rationale to justify their decisions [12]. This detailed information (which could contain explicit references to business drivers and management constraints) may only be useful to other architects. It could be safely hidden from the rest of the development team to reduce their information overload, or should be removed from the final set of design decisions delivered to an outsourced development group.

6.3 Tool Integration

It still remains a challenge how to best integrate software development environments (such as Visual Studio or Eclipse) together with architectural knowledge repositories. Not only more work on the linkage between the architectural decisions and the architecture design models is needed [13], but also the propagation of architectural decisions down to the level of the code is an open problem.

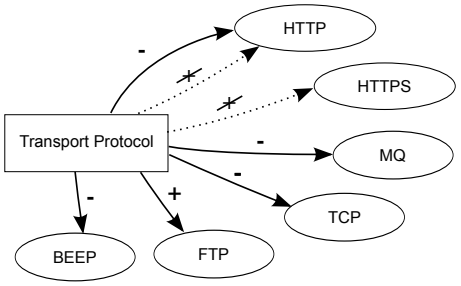

Figure 5: *Anti-Decision* on *Acceptable* (+) alternatives (HTTP and HTTPS) and new *Unacceptable* (-) decision (HTTP).

On the one hand, the goal of decision enforcement [39] requires an easy and intuitive access to the content of the project space with minimal effort required to switch context between development activities and the architectural level. The right kind of tool integration will thus provide engineers performing fine grained design and coding with amount of architectural knowledge (and possibly reference to rationale) that will improve their awareness of the nature of code that is to be implemented.

On the other hand, feedback and experience accumulated in the development process should flow back and enrich the design space knowledge base in order to improve the quality of future decisions. For example, the complexity of the code, the cost of the middleware infrastructure, and the efforts in terms of manpower required to implement support for the various *Transport Protocol*, *Security* or *Payload Format* alternatives could be tracked and stored in appropriate attributes of the design space.

7 Evolution

Decision making is hardly a linear process. Within a project, architects do not usually traverse the design space systematically and may need to go back and change previously made decisions. Also, the architectural knowledge captured in the design space may evolve over time independently of the projects depending upon it. At a fine-grained scale, we identify the requirement for tracking the history of individual decisions. At a coarse-grained scale, being able to compare different design solutions to detect and analyze changes will be an important feature of future architectural modeling tools [14].

7.1 Tracking Decision History

As decisions are made, the rationale for them should be referenced so that the arguments for taking a given decision is captured. However, decisions may need to be changed. For example, in response to changing requirements, or simply because the project follows agile methodologies and a refactoring is being applied, or because the design space has evolved with additional knowledge.

Architectural modeling tools should not just allow to change already made decisions, but also keep track of the history of

the decisions within every issue considered during a project's lifetime. This could be done with different approaches. For example, version control helps to track changes and compare different snapshots of the project space. Or, as we would like to pursue, introducing so-called *anti-decisions*, which would capture the rationale associated with the undoing of the decision. This would enable architects to distinguish between alternatives that were never explicitly considered from the ones that at some point during the project history were actually decided and later revoked (Fig. 5). When used in conjunction with metrics, this approach would allow to detect "wavering" architects, who cannot make up their mind and keep going back and forth between alternatives.

7.2 Comparing Solutions

Tracking decision changes is only one aspect of the project evolution. It has to be completed with mechanisms allowing architects to find out if the project is progressing with the expected speed in the desired direction.

We propose a versatile, comparative approach to tracking the evolution of design solutions. The challenge is to quantitatively measure the dynamics of the solution design process by comparing the properties of multiple snapshots taken at different times. Likewise, useful insight could be provided with a differential view over parallel branches originating from a common root. Comparing solutions across unrelated projects would also help to estimate the effort required for their integration and merge [5, 27].

We believe that a quantitative description of decision model dynamics will be very useful for project controlling and eventually will improve quality of the decisions taken. For example, considering the issue of *Transport Protocol* selection, which includes the *Unknown* alternative, by drawing attention to the fact that many new alternatives are currently emerging (such as *BEEP*, *WAKA*, or *SPDY*), we can inform the architect that more design and development work is expected around this issue. Conversely, observing that no change has recently occurred to set of known *Application Integration Style*s, we could characterize the issue as stable, and thus less likely to be reconsidered in the future.

8 Related Work

Within the architectural knowledge management community a number of contributions that identify and discuss open problems and outline possible research directions have been made. The seminal paper by [10] advocated a new perspective to the study of software architecture, in which architectural decision modeling plays a central role and software architecture is seen as composition of architectural design decisions. In [26], the authors put forward a research agenda for architectural knowledge modeling centered around visualization and providing task-specific support. In [28], the authors discuss the problem of building a knowledge community in the field of software architecture and stress the importance of cooperation between industry and academia. More recently, a survey of decision-centric design methods and techniques has been presented in [11]. The authors also identify a number of gaps related to the capturing of architecturally significant requirements, the reuse of decision rationale with the goal of improving the software design reasoning processes.

9 Conclusion

This paper collects a number of research challenges and opportunities related to architectural decision modeling for reuse and illustrates them with a concrete case-study scenario. Regarding the representation of complex architectural knowledge, we suggest to use tags for improved content classification. Also, both metrics and visualization can give a quantitative and qualitative overview over large architectural knowledge repositories. As architectural decisions play a central role in the software design process, we point out how fuzzy decision modeling and rapid verification could improve decision quality. We have also identified the need of collaborative decision modeling, access control, tool interoperability and integration in order to improve the propagation and sharing of architectural knowledge throughout distributed software development teams. Finally we have proposed new means of monitoring evolution of a design by detailed change tracking combined with a differential view over the history of a project.

Acknowledgements

This work is partially supported by the Swiss National Science Foundation with the CLAVOS project (Grant Nr. 125337).

10 References

[1] `http://eagain.net/articles/git-for-computer-scientists/`.

[2] `http://www.w3.org/RDF/`.

[3] T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi, and B. Benatallah. A quality-driven systematic approach for architecting distributed software applications. In *Proceedings of the 27th International Conference on Software Engineering*, pages 244–253, 2005.

[4] Aman-ul-haq and M. Ali-Babar. Tool support for automating architectural knowledge extraction. In *Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge, SHARK '09*, pages 49–56, 2009.

[5] M. Aoyama. Metrics and analysis of software architecture evolution with discontinuity. In *Proceedings of the International Workshop on Principles of Software Evolution, IWPSE '02*, pages 103–107, 2002.

[6] M. A. Babar. The application of knowledge-sharing workspace paradigm for software architecture processes. In *Proceedings of the 3rd Workshop on Sharing and Reusing Architectural Knowledge SHARK '08*, pages 45–48, 2008.

[7] M. A. Babar, T. Dingsøyr, P. Lago, and H. Vliet. *Software Architecture Knowledge Management - Theory and Practice*. Springer, 2009.

[8] M. A. Babar and I. Gorton. A tool for managing software architecture knowledge. In *Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, SHARK-ADI '07*, page 11, 2007.

[9] K. Beck. *Extreme Programming Explained: Embrace Change (2nd edition)*. Addison-Wesley Professional, 2004.

[10] J. Bosch. Software architecture: The next step. In *EWSA*, volume 3047 of *LNCS*, pages 194–199. Springer, 2004.

[11] W. Bu, A. Tang, and J. Han. An analysis of decision-centric architectural design approaches. In *Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge SHARK '09*, pages 33–40, 2009.

[12] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrìk. *Rationale-Based Software Engineering*. Springer, 2008.

[13] R. Capilla. Embedded design rationale in software architecture. In *Proc. of the Joint Working IEEE/IFIP Conference on Software Architecture and the European Conference on Software Architecture (WICSA/ECSA 2009)*.

[14] R. Capilla, F. Nava, and J. C. Duenas. Modeling and documenting the evolution of architectural design decisions. In *Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent SHARK-ADI '07*, 2007.

[15] R. Capilla, F. Nava, S. Pérez, and J. C. Duenas. A web-based tool for managing architectural design decisions. In *Proceedings of the First Workshop on SHAring and Reusing architectural Knowledge SHARK '06*, 2006.

[16] H. Choi, Y. Choi, and K. Yeom. An integrated approach to quality achievement with architectural design decisions. *Journal of Software*, 1(3):40–49, 2009.

[17] X. Cui, Y. Sun, and H. Mei. Towards automated solution synthesis and rationale capture in decision-centric architecture design. In *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture WICSA '08*, pages 221–230, 2008.

[18] P. Eeles and P. Cripps. *The Process of Software Architecting*. Pearson, 2009.

[19] R. Farenhorst and H. van Vliet. Understanding how to support architects in sharing knowledge. In *Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge, SHARK '09*, pages 17–24, 2009.

[20] N. Harrison, P. Avgeriou, and U. Zdun. Using patterns to capture architectural decisions. *IEEE Software*, 24(4):38–45, July-Aug. 2007.

[21] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture WICSA '05*, pages 109–120, 2005.

[22] A. Jansen, J. Bosch, and P. Avgeriou. Documenting after the fact: Recovering architectural design decisions. *Journal of Systems and Software*, 81(4):536–557, April 2008.

[23] A. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer. Tool support for architectural decisions. In *Working IEEE/IFIP Conference on Software Architecture, WICSA '07*, page 4, 2007.

[24] R. Kazman, M. Klein, and P. Clements. Atam: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, August 2000.

[25] P. Kruchten. A taxonomy of architectural design decisions. In *2nd Groningen Workshop on Software Variability Management*, 2004.

[26] P. Kruchten, P. Lago, and H. van Vliet. Building up and reasoning about architectural knowledge. In *Conference on Quality of Software Architectures*, volume 4214 of *Lecture Notes in Computer Science*, pages 43–58, 2006.

[27] J. M. Küster, C. Gerth, and G. Engels. Dependent and conflicting change operations of process models. In *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications, ECMDA-FA '09*, pages 158–173, 2009.

[28] P. Lago, P. Avgeriou, R. Capilla, and P. Kruchten. Wishes and boundaries for a software architecture knowledge community. In *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture WICSA 2008*, pages 271–274, 2008.

[29] L. Lee and P. Kruchten. Customizing the capture of software architectural design decisions. *Canadian Conference on Electrical and Computer Engineering*, pages 693–698, May 2008.

[30] M. Mahemoff. *Ajax Design Patterns*. O'Reilly Media, 2006.

[31] R. Marinescu. Detection strategies: metrics-based rules for detecting design flaws. In *Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 350–359, 2004.

[32] I. Ostacchini and M. Wermelinger. Managing assumptions during agile development. In *Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge SHARK '09*, pages 9–16, 2009.

[33] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. big web services: Making the right architectural decision. In *17th International World Wide Web Conference WWW2008*, pages 805–814, Beijing, China, April 2008.

[34] G. Robertson. From hierarchies to polyarchies: visualizing multiple relationships. In *Proc. of the working conference on Advanced Visual Interfaces AVI'00*, page 13, 2000.

[35] I. Rus and M. Lindvall. Guest editors' introduction: Knowledge management in software engineering. *IEEE Software*, 19:26–38, 2002.

[36] A. Tang, Y. Jin, J. Han, and A. Nicholson. Predicting change impact in architecture design with bayesian belief networks. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, pages 67–76, 2005.

[37] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture - Foundations, Theory and Practice*. Wiley, 2010.

[38] R. Wettel and M. Lanza. Visually localizing design problems with disharmony maps. In *Proceedings of the 4th ACM Symposium on Software Visualization SoftVis '08*, pages 155–164, 2008.

[39] O. Zimmermann. *An Architectural Decision Modeling Framework for Service-Oriented Architecture Design*. PhD thesis, Universität Stuttgart, 2009.

[40] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster. Managing architectural decision models with dependency relations, integrity constraints, and production rules. *Journal of Systems and Software*, 82(8):1249 – 1267, 2009.