Weaving a Network of Architectural Knowledge

Elena Navarro
Computing Systems Department, University of Castilla-La Mancha
elena.navarro@uclm.es

Carlos E. Cuesta
Department of Computing Languages and Systems II, Rey Juan Carlos University
carlos.cuesta@urjc.es

Dewayne E. Perry
Department of Electrical and Computer Engineering, The University of Texas at Austin
perry@ece.utexas.edu

Abstract

Recent research in software architecture has reasserted an emphasis on keeping track of design decisions and their rationales during the development process, that is, on maintaining architectural knowledge (AK). This knowledge takes the form of explicit assets, interrelated in decision networks. We argue that the relationships structuring these networks contain valuable information, specially those describing negative semantics. For this reason, we have extended an architecture-centric, model-driven development process, ATRIUM, which already provides support for AK, with new AK relationships to define AK as a network of knowledge.

1. Introduction

The traditional approach for architectural description has been to provide the *final* architecture of the software system, i.e. the result of the design process. However, this means that much of the design information gets simply lost. Recent research suggests to avoid that by capturing the rationale and design intent, documenting every design decision in the process and relating it to the architecture.

The resulting structure, the *architectural rationale*, can be considered just documentation; but in recent research it has acquired quite an active role. It can be seen as a computational structure, composed of small assets of design knowledge, tracing back to some requirements and forward towards an implementation: this defines our *architectural knowledge* (AK). This AK is composed of architectural elements, requirements, and a number of design assets (DA). There are several ways to represent them; we talk about Design Decisions (DDs) and Design Rationales (DRs), which comprise a concrete decision in the process, and the reasoning behind it. When just the final architecture is described, all that is *unrepresented design knowledge*; but now architecture includes this information as part of the rationale.

From the AK perspective, architecture is therefore better defined as "a set of design decisions" [5]. But it is not a set, but a *network*. DDs are related to some others and both forward and backwards, defining an intertwined chain of decisions. When the rationale is conceived as a network of AK assets, much of the structure can be *subsumed* by the network itself. However, these relationships can either be *positive* or *negative*, indicating conflicts in the design and the resulting compromises. A decision *not taken*, or inhibited due to some other, is not visible (there are no *traces* of it) in the final architecture. This unrepresented knowledge cannot be deduced from the final design, and should be explicitly considered.

In this paper, we describe the AK relationships that we have defined up to date. They are analyzed highlighting their importance to comprehend the reasoning accomplished during software architecture (SA) specification. A core of concepts has been identified that has allowed us to extend an existing model-driven (MDD) methodology, ATRIUM [9]. It already has the support for DDs, but it previously lacked the support for relationships, which are added in the new version.

This paper is structured as follows. After this introduction, section 2 summarizes current approaches to AK relationships, discussing its benefits and extension. Section 3 describes how ATRIUM has been extended with AK relationships. Finally, section 4 describes our conclusions and lines of further research.

2. Background on AK Relationships

The most natural way to think about a design decision is to consider it separately from the rest of the system by a process of abstraction. However, such a view is necessarily incomplete and partial. Design decisions are connected, because they refer to and affect each other. In fact, there is a complex fabric of relationships surrounding them: even the simplest model of DDs in existing literature provides some structure. First, every decision can be *traced* back to the goal it achieves, or the requirement(s) it satisfies. Second, every decision is *implemented* by some design artifact, some architectural element. Apart from these two reifying relationships, any decision relates to other decisions in different ways.

Design Decisions and Rationales (DD&DRs) can be correctly considered as the basic assets which compose our design knowledge. However, to provide a coherent system-wide rationale these have to be complemented with the information about their mutual referrals and connections. Therefore, those Architectural Knowledge Relationships (AKRs) present themselves as another invaluable basic asset, which transform the *set* of design decisions into a *network* of design knowledge.

Some authors [5] describe this network of decisions using a single (dependency) relationship, or perhaps several assimilated ones. Even this simple layout is useful and much more than a simple "set" of decisions, as many details depend just on topology. But there are still many details left, such as *positive* and *negative* relationships, respectively exposing synergies and divergences within the design; these interactions should also be considered.

Many authors recognize the inherent complexity of managing and combining "knowledge assets", and thus they advocate an *ontological approach* [1][2][3] [6][7][8] to provide a solid basis for this reasoning. These ontologies are able to identify the basic properties of the knowledge contained in an asset. They can also help to define the basic relationships between assets, and to provide a more complete metamodel for them.

However, most of the work in this area, even those inspired by an ontological approach, has focused on the description of the (internal) *structure* of DDs. For most of them, relationships play just a secondary role, if any.

For instance, Tang et al. [11] provide a structural model for their basic asset, *the architectural rationale*. This is defined as the composition of a quantitative and a qualitative rationale, with scenarios and possibly some aggregations, defining an *alternate* DD. The resulting structure is quite useful, particularly for documentation of a single decision. However, this model does not scale well: first, every decision is complex, including cases and alternates; and second, there is not support to relate a decision to any other, except for aggregation itself. Therefore, the resulting "global" rationale is defined as an unstructured discourse, which does not provide an organization for architectural knowledge.

The situation improves a lot when considering it as a *network* of design assets, as it captures much of the complexity itself. First, AKRs must assume ontological features, which can be independent from DDs themselves; and second, the *structure* of the network itself becomes significant. Moreover, the connections capture some essential facts, not only about the architecture itself, but also about the construction process. Therefore, a *network* of very simple DDs is able to capture perhaps even *more* information than a poorly structured set of complex DDs.

In existing literature, the most popular relationships are *constrains*, which expresses the self-evident positive implication, and *alternative*, probably the most cited among them, which expresses *variability*. It provides the support to express a choice and refer to otherwise unrepresented, *vaporized* design knowledge, and it is also very useful in the context of product lines [10].

Table 1. Analyzing current AKR approaches

Relationship	Synonyms	References
Constrains	Implies, Refines	[6][2][3][5][7]
Forbids	Excludes	[6][3][7]
Enables		[6][3][7]
Subsumes	(**)	[6][3][7]
Conflicts with	(*)	[6][3][7]
Overrides		[6][3][7]
Comprises	Made Of	[6][3][7][11]
Bound To		[6][3][7]
Alternative	Alternate DD	[6][2][3][7][11]
Related To		[6][3] [7]
Traces From/To	Addresses, Implements (*)	[6][2][3] [7] [11][12]
Not Complies		[6][3][7]
Depends on	(**)	[5][7]

Table 1 summarizes many of the existing approaches for AKRs described by current research, providing some terminological equivalence and some references covering their definition or use. Kruchten [6] provides the most complete reference about this topic. He not only provides the definition for most existing AKRs, but also the way they relate to each other. Other references, such as [1][3][7], basically use the same ontological framework. As mentioned, they are not mutually exclusive: *bound to*, for instance, is defined in terms of *constrains*; also *enables* and *comprises*, are described respectively as weaker and stronger versions of it.

Although the names of these relationships must be understood in terms of this particular context, some of them can be considered within a wider scope; these have been marked with a star (*) in Table 1.

This refers, in particular, to *conflicts with* and *traces from/to*. In the restricted sense, they obviously refer to marking conflicting decisions and keeping track in a chain of decisions. Both of them can also be conceived as derived relationships (see Table 2) in the context of DDs, but there is also a general meaning out of this scope, namely the traditional traceability relationship (*trace*), and schemas of conflict *between requirements*, combined to trace to provide derived decisions.

Other relationships have been marked with a double star (**). The issue here is *generalization*. Both *depends* and *subsumes* can be considered as generic versions of the rest. First, dependency can be considered the basic relationship, by definition, so that every other inherits from it. Then every AKR would also be a dependency [5]. On the other hand, *subsumption* is usually considered as the target relationship for ontologies, acting as the transitive closure for ontological relationships.

Considering all of the above, we can see that AKRs provide a particularly rich framework to capture design knowledge. As already said, this makes also possible to work with smaller DDs. Also, the conceptual closeness between some of them makes it possible to apply simple analysis techniques. The potential of exploiting all this information remains still unexplored, and there are very few works in this direction [12]. In summary, though not very frequent yet, AKRs provide a very useful framework to define, use and *reuse* architectural knowledge.

3. Extending ATRIUM with AKRs

ATRIUM [9] is a model-driven software development methodology, designed to automate the transition from a goal-based requirements model to the proto-architecture of a system, with explicit support for AK.

The first model in the ATRIUM process, the Goal Model, is in charge of manipulating most of the AK. The building blocks of this model are *Goal*, *requirement* and *operationalization* . *Goals* constitute expectations that the system should meet. *Requirements* are services that the system should provide, or constraints on these services. And *operationalizations* describe both the DDs and DRs made to satisfy the established requirements. A seamless transition from requirements to operationalizations is defined by the *contribution* relationship, which specifies how solutions contribute positively and/or negatively to meet the corresponding requirements.

One of the main advantages of AK management is the capability to explore the reasoning in the software architecture by exploiting the network of AK. In order to provide ATRIUM with this facility, several relationships were defined in its metamodel, to allow the analyst to describe the AK as a network. As shown in [9], these relationships were first defined on *operationalizations*, as they are in charge of describing both the DDs and the DRs. The analysis performed in section 2 was considered in their definition, establishing the following relationships:

- *constrains* is a binary and unidirectional relationship with *positive* semantics. Let's consider operationalizations A and B , describing different design decisions. Having a *constraint* relationship from A to B means that B's design decision cannot be made unless A's design decision is also made.
- *inhibits* is a binary and unidirectional relationship used to specify *negative* semantics. Let's consider operationalizations A and B, describing different design decisions. Having an *inhibition* relationship from A to B means that if A's design decision is made, it hinders B's design decision to be made.
- *excludes* is a binary and unidirectional relationship with *stronger negative* semantics than inhibits. Let's consider operationalizations A and B, describing different design decisions. Having an *exclusion* relationship from A to B means that if A's design decision is made, it prevents B's design decision to be made.

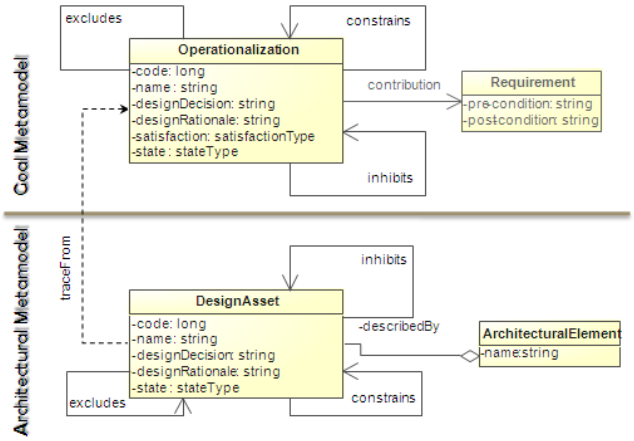

Fig. 1 Describing AK relationships in ATRIUM

Table 2 shows a brief resume of how the selection of these specific relationships provides us with the necessary expressiveness to cover most of the existing approaches in the area. It compares these three relationships to the set of AKRs surveyed in section 2. When Table 2 indicates "*equivalent*", there is a full coincidence with some existing definition in the bibliography. When it indicates "*opposite*", the reversed relationship would have been equivalent; therefore that is a trivial conversion. When it indicates "*derived*", it means that the current relationship can be non-atomically built using some of the selected, using some conditions to strengthen or weaken their definition.

Both *subsumes* and *traces* relationships deserve a special mention. For the former we simply note that subsumption is usually defined as the transitive closure of a positive relationship – hence *constrains*. For the latter, the reasoning is similar, it can be extended to consider also requirements and architectural elements; and in this case it must be combined with the *trace* relationship.

Of course, the nature of most of the proposed AKRs is essentially ontological; consequently, we are not claiming that our three relationships capture every semantic feature in them. But both their basic meaning and, even more importantly, the *positive* or *negative* nature of the connections they define, can indeed be expressed in terms of these, as summarized in Table 2.

As presented in [9], one of the advantages of ATRIUM is that it facilitates the generation of the DDs along with the proto-architecture, so that each architectural element is related to the set of DDs that motivated its specification and the DRs that justify those decisions. Fig. 1 shows (part of) the Architectural metamodel. It can be observed that every Architectural Element is related to a set of Design Assets that describe both the DDs and the DRs.

Table 2. Evaluating proposed AKRs in ATRIUM

Relationship	*Constrains*	*Inhibits*	*Excludes*
Constrains	Equivalent		
Forbids			Equivalent
Enables		Opposite	
Subsumes	Closure		
Conflicts with		Derived	
Overrides		Derived	
Comprises	Derived		
Bound To	Derived		
Alternative		Derived	
Related To	Derived		
Traces From/To	Plus *trace*		
Not Complies	Derived	Opposite	
Depends on	Derived		

As can be observed, the DesignAssets can be related by means of *constrains*, *excludes* and *inhibits* relationships in a similar way to the operationalizations in the Goal Metamodel.

It is worth noting that the main difference is the use of operationalizations in the goal model and Design Assets in the architectural model. The former are in charge of specifying all the design decisions and design rationales that were analysed during the specification of the system, that is, they describe the reasoning carried out to evaluate which were the best alternatives for the system. The latter describe the reasoning behind the current specification of the system, that is, why the system has its current specification. Therefore, they help to maintain AK from different perspectives.

4. Conclusions and further research

Our work in this paper has shown the influence of relationships in AK. What once was a set of small-sized design decisions is now a complex decision network. In fact, once that relationships have been included, the network of AK can get as complex as the architecture itself; comparatively, even more, as the final architecture is just a part of the rationale.

Now architecting, instead of simply constructing the architecture, and losing valuable context knowledge in every decision, becomes *the process of writing the architectural rationale*. Two complex structures are obtained: the architecture and the rationale; but being closely intertwined, they are just *one* structure. The act of building the first is also the act of writing the second.

The combination of the basic support with a model-driven approach gives enormous benefits, as already shown in [9]. As traceability relationships are also provided by MORPHEUS, the environment supporting ATRIUM, they are easily composed to each other, and their consequences can be fully exploited – including the scope of a requirement, the implementation of a decision, or finer strategies for network analysis.

Further work for the future includes the definition of special decisions, as well as automatic support to exploit the decision network, using standard techniques for network analysis, which will led to the identification of special nodes and critical decisions. We also plan to provide more sophisticated methods to visualize the information contained in the structure, and to complete this knowledge with the definition, and even local implementation of several adequate metrics.

Acknowledgments. This work has been funded in part by the National R&D&I Program, META/MOMENT Project, TIN2006-15175-C05-01, and in part by NSF CISE SRS Grant CCF-0820251.

References

[1] A. Akerman, J. Tyree, "Using Ontology to Support Development of Software Architectures," *IBM Systems Journal*, 45(4), 2006, pp. 813-826.

[2] A. Erfanian, F.S. Aliee, "An Ontology-Driven Software Architecture Evaluation Method", Proc. Workshop SHARK, ACM Computing, 2008, pp. 79-86.

[3] R. Farenhorst, R.C de Boer, "Core Concepts of an Ontology of Architectural Design Decisions," Technical Report IR-IMSE-002, Dept. Computer Science, Vrije Universiteit Amsterdam, 2006.

[4] S.A. Hendrickson, S. Subramanian, A. van der Hoek, "Multi-Tiered Design Rationale for Change Set Based Product Line Architectures", Proc. 3rd Work. SHARK, ACM Computing, ACM New York, 2008, pp. 41-44.

[5] A. Jansen, J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," Proc 5th Working IEEE/IFIP WICSA, IEEE CS, 2005, pp. 109-120.

[6] P. Kruchten, "An Ontology of Architectural Design Decisions," in J. Bosch (ed.), Proc. 2nd Workshop of Soft. Variability Man., Groningen, 2004, pp. 54-61.

[7] P. Kruchten, P. Lago, H. van Vliet, "Building Up and Reasoning About Architectural Knowledge," Proc. 2nd QoSA, LNCS 4214, Springer Verlag, 2006, pp. 43-58.

[8] T. Lenin Babu, M. Seetha Ramaiah, T.V. Prabhakar, D. Rambabu, "ArchVoc – Towards an Ontology for Software Architecture", Proc. 2nd W. SHARK, IEEE CS, 2007, pp. 5.

[9] E. Navarro, C. E. Cuesta, "Automating the Trace of Architectural Design Decisions and Rationales Using a MDD Approach", Proc. ECSA 2008, LNCS 5292, Springer Verlag, 2008, pp. 114-130.

[10] M. Sinnema, S. Deelstra, "Classifying Variability Modeling Techniques," *Journal on Information and Software Technology*, 49(7), 2007, pp. 717-739.

[11] A. Tang, J. Han, "Architecture Rationalization: A Methodology for Architecture Verifiability, Traceability and Completeness," Proc. 12th IEEE Intl. Conf. ECBS, IEEE CS, 2005, pp. 135-144.

[12] A. Tang, Y. Jin, J. Han, A. Nicholson, "Predicting Change Impact in Architecture Design with Bayesian Belief Networks," Proc. 5th Work. IEEE/IFIP WICSA, IEEE CS, 2005, pp. 67-76.