Assisting software architects in architectural decision-making using Quark

David Ameller and Xavier Franch
Universitat Politècnica de Catalunya,
Barcelona, Spain,
{*dameller, franch*}*@essi.upc.edu*

Abstract

Non-Functional Requirements (NFRs) and constraints are among the principal drivers of architectural decision-making. NFRs are improved or damaged by architectural decisions (ADs), while constraints directly include or exclude parts of the architecture (e.g., logical components or technologies). We may determine the impact of an AD, or which parts of the architecture are affected by a constraint, but at the end it is hard to know if we are respecting the NFRs and the imposed constraints with all the ADs made. In the usual approach, architects use their own experience to produce software architectures that comply with the NFRs and imposed constraints, but at the end, especially for crucial decisions, the architect has to deal with complex trade-offs between NFRs and juggle with possible incompatibilities raised by the imposed constraints. In this paper we present Quark, a method to assist software architects in architectural decision-making, and the conceptualization of the relationship between NFRs and ADs defined in Arteon, an ontology to represent and manage architectural knowledge. Finally, we provide an overview of the Quark and Arteon implementation, the ArchiTech tool.

Keywords: Software quality, non-functional requirements, architectural decisions, architectural knowledge, software architecture design method, decision-making.

1 Introduction

In the last decade, software architecture has become one of the most active research areas in software engineering. As a significant trend in this community, many researchers have stated that architectural decisions (ADs) are the core of software architecture [1, 2, 3]. Under this view, software architecture has evolved from a structural representation to a decision-centered viewpoint [4].

In this paper, we present Quark (Quality in Architectural Knowledge), a method to assist software architects in architectural decision-making. Quark builds upon Architectural Knowledge (AK) [5, 6] using an ontology (Arteon [7]) to manage and reuse knowledge about ADs, their rationale and their link to Non-Functional Requirements (NFRs) and constraints. We also present in this paper the part of Arteon related to ADs (this part was not presented in [7]). The concepts defined in Arteon are tightly related to the Quark method, in particular the conceptualization of the relationship between NFRs and ADs. Finally, we give an insight into the Quark and Arteon implementation, the ArchiTech tool. This tool was already presented in [8], we only include a short description for completeness.

The highlights of Quark, Arteon, and ArchiTech are: first, they have been designed using empirical basis, we asked to architects what they want and need from an architectural design method; second, they use a decision-centered perspective, which is aligned with the trend in architectural research; and third, they use Non-Functional Requirements (NFRs) to drive the decision-making.

The rest of this paper is divided into the following sections: related work in Section 2. The Quark method in Section 3. The Arteon ontology in Section 4, in particular the module for decision-making knowledge. The overview of the ArchiTech tool in Section 5. An example of use of Quark, Arteon, and ArchiTech in Section 6. And finally, conclusions, limitations, and future work in Section 7.

2 Related Work

Related to AK, there are several topics that are of especial relevance for this work:

- *Software Architectural Design Methods.* SADMs are methods that help the architect to derive the software architecture from the software requirements [9]. In this paper is presented Quark, a SADM which was inspired in some of the works presented in this section.
- *Architectural Knowledge ontologies.* Ontologies are the principal way of representing knowledge, and AK is no exception. These ontologies are presented is some cases, as models or metamodels, but always as a way to organize the elements and concepts that are relevant to AK. In this paper is presented a part of an ontology for AK, Arteon, which was inspired in some of the works presented in this section.
- *Architectural Knowledge tools.* AK tools help to manage and assist software architects in the design and reasoning tasks. In this paper we show an overview of ArchiTech, a tool to manage AK and assist software architects in architectural decision making.

Works related to these topics are presented in detail in the following sections.

2.1 Software architectural design methods

In this section are analyzed some of the Software Architecture Design Methods (SADMs) available in the literature, and more concretely is important for the contents of this paper how these methods deal with NFRs. One of the principal producers of this type of methods is the Software Engineering Institute (SEI). SEI has created several design and analysis methods: SAAM [10], ATAM [11], CBAM, QAWs, QUASAR, ADD [12], ARID. Documentation for all of them can be found in SEI website[1]. The most relevant ones, in relation to the contents of this paper, are ADD and ATAM.

- Attribute-Driven Design Method (ADD, F. Bachmann and L. Bass) [12]. A second version of this method was published in 2007 (available in the SEI website). ADD is a method to design the software architecture of a system based on quality goals for the system. The method is extensible for any quality attributes but has been particularly elaborated for the attributes of performance, modifiability, security, reliability, availability and usability. The method considers three architectural views: module view, component and connector view, and deployment view. The method consist in decomposing the system recursively into subsystems and then into components.
- Architecture Tradeoff Analysis Method (ATAM, R. Kazman et al.) [11] is a methodology that evolved from Software Architecture Analysis Method (SAAM, 1996). It is a method to understand the tradeoffs of the architectures of software-intensive systems. This method analyzes the architecture for several quality attributes (e.g., security, performance, etc.). The method guides the design decisions that have an impact on quality attributes. It is a spiral method, consisting in four phases: requirements elicitation including constraints, architectural views, modeling and analysis, and identification of tradeoffs.

SEI methods, namely ADD for design and ATAM for analysis, are heavyweight methods that require large-scale projects to achieve a balance between what the method offers and the effort that supposes for the architects to use it. This balance is hard to achieve when projects are low- or medium-scale. In our case, we based our method in suggestions from architects working in low- or medium-scale projects (see Section 3), therefore we believe that the method can be successfully applied to this kind of projects.

We did several searches in academic databases (e.g., Google Scholar, ISI Web of Science, etc.), complemented with other methods that we were already aware of to build a list of SADMs:

- Quality Atribute-oriented Software ARchitecture (QASAR, J. Bosch) [13]. This method consists of three steps: first, the functional requirements are implemented in components, then the architecture is analyzed to decide whether the NFRs are fulfilled or not, and in the third step the architecture is adapted to be in conformance with the NFRs. As Quark, QASAR relies on NFRs, but in QASAR first there is a design based only on functional requirements, and then it is refined using the NFRs. Instead, Quark uses NFRs from the very beginning as the main driver of the decision-making.
- Quality-driven Architecture Design and Analysis (QADA, M. Matinlassi et al.) [14] is a set of methods: one method for selecting an appropriate architecture approach, one method for quality-driven architecture design, one method for evaluating the maturity and quality of architecture, and one technique for representing variation points in the family architecture. As Quark, QADA is quality-driven and it is built upon a knowledge base, but it is not centered in ADs.

[1] www.sei.cmu.edu/architecture

- Quality Achievement at the Architectural Level (AQUA, H. Choi et al.) [15]. A method that provides software architects means for achieving NFRs at the architectural level. AQUA involves two kinds of activities, which are architectural evaluation and transformation. AQUA uses a decision-centered approach as we do in Quark, the main differences between both methods are that AQUA does not have an ontological foundation and their method is not based on empirical studies. On the first hand, having an ontology to reason and mange knowledge is known as a good approach in many areas (e.g., artificial intelligence) but is true that currently it is not wide used in computer engineering research. On the second hand, having an empirical study on the main target community (software architects) helps to reduce the risk of having a solution disconnected from the real needs of this community.

- A. Bertolino et al. [16] presented an approach to automate the architecture design and implementation. The method starts from requirements in Natural Language (NL). The authors say that they want to integrate several existing tools to accomplish the task: QuARS (Quality Analyzer for Requirements Specifications, tool to obtain requirements from NL specifications), ModTest (a model-checking tool), and Cow Suite (a testing tool). The main difference with Quark is that Bertolino's method does not deal with architectural decisions. Also, it is not clear what is the interaction with the software architect in the method presented by Bertolino et al.

- T. Al-Naeem et al. [17] proposed a method centered on the decision making process, but not on generating the architecture. For the computation method, they rely on Multiple Attribute Decision Making (MADM), in particular Analytic Hierarchy Process (AHP), to score each alternative decision. Our method is based on artificial intelligence algorithms to score each alternative decision. We are not in a position to say which option is best, but they are clearly different.

- Tang et al. proposed the AREL method [18] to improve the traceability of ADs by linking them to the design rationale. They also propose a conceptual model to manage this rationale, and the concern for software quality during the architecture design, but they do not describe which is the reasoning method used (if any), in this situation it is hard to compare their approach with Quark.

- Montero and Navarro proposed ATRIUM method [19], Architecture Traced from RequIrements applying a Unified Methodology. ATRIUM is a methodology based in the MDD approach, its intention is to guide the architects in the definition of the architecture, the method considers both functional and non-functional requirements. Contrary to Quark, this method does not focus on decisions but in scenarios and requirements.

- L. Chung et al. [20] proposed a framework, Proteus, to develop software architectures considering NFRs in goal-oriented notation, using NFR Framework [21]. However, Chung's framework does not support to explicitly trade-off analysis between alternate design decisions.

There are many other SADMs that improve or specialize one of the previous ones, the following are the ones found related to the improvement of the handle of quality or non-functional requirements:

- S. Bode et al. [22] presented a method based on QASAR to design the system's security architecture. The authors state that they considered methods form software engineering and security engineering to deal with security requirements. This approach is specialized only in security, while Quark could be used for any type of requirements (it all depends of the knowledge base provided).

- S. Kim et al. [23] presented a method that is based on architectural tactics. Architectural tactics are explained in L. Bass et al. book "Software architecture in practice, second edition" [24], they are basically reusable pieces of the architecture. This method uses feature models to generate the architecture automatically. It is very similar to a product line for architectures. Product lines work for known and repetitive problems, but in Quark we leave the door open to customize the knowledge to any particular architectural area of interest (e.g., the architect may want to have many technological alternatives but does not care much about styles because s/he uses always the same).

- D. Perovich et al. [25] presented a method to design software architectures using Model-Driven Development (MDD) [26] considering quality aspects (based on ADD method). In this case they use a "megamodel" (a model composed of models) to represent the software architecture. The method uses feature models to construct the architecture. This approach has many similarities with ours, both are based on architectural decisions, and are iterative methods. But there are also some differences, Perovich's method generates an architectural models following the MDA approach [27], while Quark is more focused on the architectural decisions itself and the customization of the architectural knowledge.

Table 1: Comparison of SADMs

Ref.	Name	Kind of NFR	Computer-aided	Based on...
[11]	ATAM	Quality aspects	No	SAAM
[12]	ADD	Quality aspects	No	None
[13]	QASAR	Quality aspects	No	None
[14]	QADA	Quality aspects	Yes, as product lines	None
[15]	AQUA	Quality aspects	Somehow, transformations	None
[16]	No name	Requirements	Yes, no details	None
[17]	ArchDesigner	Quality aspects	Yes, limited to decisions	None
[18]	AREL	Concerns	No	None
[19]	ATRIUM	Any NFR	Yes, MDD method	None
[20]	Proteus	Any NFR	No	NFR Framework
[22]	No name	Security	No	QASAR
[23]	RBML	Any NFR	Yes, as product lines	ArchDesigner
[25]	No name	Any NFR	Yes, MDD method	ADD
[28]	QRF	Quality aspects	Yes, as product lines	QADA
[30]	No name	Quality aspects	Somehow, decision making	ADDv1
	Quark	*Quality aspects*	*Yes*	*Empirical evidence*

- E. Niemelä and A. Immonen [28] presented the QRF method, this method extends QADA by providing a systematic method for eliciting and defining NFRs, tracing and mapping these requirements to architectural models and for enabling quality evaluation. In Quark the elicitation of requirements is performed previously, in fact, the architect is expected to only introduce the requirements that are architecturally relevant. To this end the QRF method could help in the identification of requirements relevant for the architectural design.

The Table 1 summarizes the studied SADMs. There are many SADMs that use the ideas behind product lines to design architectures. It is interesting to see that almost all are capable to deal with quality aspects or NFRs in general, not limiting to a particular type as it happens in some MDD approaches. It seems to be more common to speak about quality aspects than NFRs in this area. SADMs that are based on other SADMs are more specific and are oriented to facilitate the automation of the method. There are many SADMs that consider NFRs and some of them are able to generate an architecture in a semi-automatic way.

Last but not least important, it is worth mentioning one interesting work published by Hofmeister et al. in 2007 [29]. Their intention was to model a general architectural design method based on empirical observation. The resulting model has three architectural activities:

Analysis: *"serves to define the problems the architecture must solve"*

Synthesis: *"proposes architecture solutions to a set of architectural significant requirements"*

Evaluation: *"ensures that the architectural design decisions made are the right ones"*

In Quark (described in Section 3), *architectural analysis* is covered with the specification activity, *architectural synthesis* is covered with the decision inference activity, and *architectural evaluation* is covered with the decision-making activity. We identified two important differences between Quark and the general approach of architectural design proposed by Hofmeister:

- Hofmeister's general approach does not give much details on how iterative methods should work, which is why we have an extra activity in our method.
- Hofmeister's general approach deals with complete architectural solutions, while Quark works at decisional level. The reason to design Quark in this way is because in our exploratory studies we have detected that architects will not trust a support system that generates full architectural solutions without their intervention.

2.2 Architectural Knowledge Management

Several works have been published for Architectural Knowledge Management (AKM), each with different nuances, but most of them making a special emphasis on the notion of ADs. One particularly relevant work in this direction is the ontology proposed by P. Kruchten et al. [5] to describe the types of ADs (it was

previously published in 2004 [3]). In this taxonomy ADs are classified into: existence decisions, property decisions, and the executive decisions. They are defined as:

Property decision: *"A property decision states an enduring, overarching trait or quality of the system. Property decisions can be design rules or guidelines (when expressed positively) or design constraints (when expressed negatively), as some trait that the system will not exhibit"*, e.g., *"all domain-related classes are defined in the Layer."*

Existence decision: *"An existence decision states that some element / artifact will positively show up, i.e., will exist in the systems' design or implementation"*, e.g., *"the logical view is organized in 3 layers."*

Executive decision: *"These are the decisions that do not relate directly to the design elements or their qualities, but are driven more by the business environment (financial), and affect the development process (methodological), the people (education and training), the organization, and to a large extend the choices of technologies and tools"*, e.g., *"system is developed using J2EE."*

In Arteon (described in Section 4), *existence decisions* are represented as the decision concept and its actions. The two other kinds of decisions are also represented in the ontology, but not in an evident way. *Property decisions* are represented in Arteon as the resulting decisions from conditions over the attributes, for example, all the ADs made because of the condition to have Open Source Software (OSS) license. *Executive decisions* are represented in Arteon as the resulting ADs imposed by restrictions that come from the software requirements, in particular the requirements unrelated to the software quality, for example, a software requirement says that the DBMS should be Oracle, because the architect's company has a deal with Oracle to only use its products.

The following works present conceptualizations for AKM. Being flexible, we may understand these conceptualizations as ontologies[2] to facilitate the comparison with Arteon:

- A. Jansen et al. [2] presented a metamodel that put ADs as the central concept. The metamodel is divided into three parts: composition model, architectural model, and design decision model. ADs are described by means of design fragments. Other relevant concepts that appear in this metamodel are: connector, interface and port.

- R. de Boer et al. [6] presented a model to represent the *core* of AK. This work defines ADs as alternatives. Other relevant concepts that appear in this model are: stakeholders, activities, and artifacts. The model is represented using a custom made modeling language.

- P. Avgeriou et al. [31] presented a conceptual model to represent an AD. This work defines decisions as options. In this work decisions are related to rationale, issues, and concerns. This model pretend to be an extension to the ISO/IEC/(IEEE) 42010 [32].

- R. Capilla et al. [33] presented a metamodel for architecting, managing and evolving architectural design decisions. This work divides the concepts into three groups: project model, architecture, and decision model. The project model includes concepts such as stakeholders, iterations, requirements, and views of the architecture. The part named as architecture have concepts such as variation points, patterns and styles. The decision model includes concepts such as constraints, dependencies, decision-making activity, and assumptions rationale.

- C. López et al. [34] presented an ontology that describes Soft-goal Interdependencies Graphs (SIGs) semantics concepts to represent NFR and design rationale knowledge. This ontology does not include architectural concepts, but the concepts related to interdependency, argumentation, and decomposition. The ontology is described using the OWL language.

- A. Akerman and J. Tyree [35] presented an ontology that focus on ADs. The ontology is divided into four parts: architecture assets, architecture decisions, stakeholder concerns, and architecture roadmap. The architecture assets concepts offer an accurate description of the structure of the architecture. Concerns are addressed by ADs, these, in turn, are implemented in roadmaps. The ontology is represented in UML.

[2]Sometimes the terms used to refer to these conceptualizations (model, metamodel, ontology, taxonomy, etc.) are used by convenience (e.g., the target audience) instead of their actual meaning. The differences between these terms are described in: `http://infogrid.org/trac/wiki/Reference/PidcockArticle`

Table 2: Comparison of AK conceptualizations

Ref.	ADs are...	ADs are related with...	Modular[1]	Other aspects covered
[2]	Design fragments	Composition techniques	Yes	Architecture structure
[6]	Alternatives	Concerns	No	Design process
[31]	Options	Rationale, issues, concerns	No	None
[33]	Patterns, styles	Constraints, dependencies	Yes	Project, architecture
[34]	N/A	N/A	Yes	NFRs
[35]	Alternatives	concerns, assumptions	Yes	Architecture structure
[36]	N/A	N/A	Yes	Architecture structure
[37]	N/A	N/A	No	Architectural styles
Arteon	*Selection of elements*	*Quality attributes*	*Yes*	*Architecture structure*

[1] The conceptualization is described in different modules or there is some kind of separation.

- ArchVoc [36] is an ontology for representing the architectural vocabulary. The terminology is classified in three main categories: architectural description (e.g., frameworks, views, and viewpoints), architectural design (patterns, styles, methodologies, etc.), and architectural requirements (non-functional requirements, and scenarios).
- Pahl et al. [37] presented an ontology that focused on components and connectors as a general way to describe architectural styles. This ontology uses a precise notation because the final objective is to provide a modeling language for architectural styles.

In Table 2 there is a summary of the works to represent AK mentioned in this section. The concept of alternative appear in three of the studied works [6, 31, 35], in these same three works also appear the concern concept related to ADs. These coincidences may be consequence of collaborations between the authors, it is also worth mentioning that they are very near in time. Most of the works present the concepts separated in different aspects, this is also a recommended practice when designing ontologies. Five works considered relevant to include concepts related to the structure of the architecture as part of the AK (in the case of [31] the intention is not to represent AK, only the part related to ADs).

None of the studied conceptualizations fulfills the underlying needs of a computer-aided support system to make architectural decisions: a computer oriented formalism and enough detail to design an architecture. In this paper we try to fulfill these needs, this is the reason why we designed this ontology. Arteon is inspired in many of the mentioned conceptualizations and complemented with the required detail and formalism to be used in a computer-aided support system context.

2.3 Architectural Knowledge tools

There are, already, many tools to manage AK. This may be the reason why, as far as we now, there are three papers published to compare tools related with AK:

- A. Tang et al. [18]: in this work is published a comparative of five AK tools, with especial emphasis in the name used for architectural concepts.
- K. Henttonen and M. Matinlassi [38]: a recompilation of OSS based AK tools.
- M. Shahin et al [39]: this work compares tools to manage architectural design decision and the ways used to model these decisions.

We selected 10 tools from these papers, the tools are: AEvol, Ontology-Driven Visualization (ODV), Archium, ADDSS, AREL, Knowledge Architect, PAKME, Web of Patterns, Stylebase for Eclipse, and Morpheus. We summarized the observations on these tools in Table 3, are we also identified some interesting facts related with this work for some of these tools:

- ODV [40]: this tool uses its an ontology names QoOnt (Ontology for the Reuse of Quality Criteria) and the ISO/IEC 9126 [41] to classify the quality attributes.
- AREL [42]: this tool takes in consideration quality concerns as one of the elements of the architecture design. This tool helps in the design of the architecture using UML models and views.
- PAKME [43]: in this tool NFRs can be specified as keywords of architectural patterns that then can be reused for other projects. This tool is limited to textual knowledge.

Table 3: Comparison of AK tools

Name/Ref.	SADM	AKM	Platform	MDD	Support of NFRs
AEvol [44]	No	No	Eclipse	No	Not mentioned
ODV [40]	No	No	Windows desktop	No	Yes
Archium [2]	No	Yes	Java/Compiler	No	Somehow
ADDSS [45]	No	Yes	Web	No	Somehow
AREL [42]	AREL	Yes	Enterprise Architect	No	Yes
Knowledge Architect [46]	No	Yes	Excel plug-in	No	Not mentioned
PAKME [43]	No	Yes	Web/Java	No	Yes
Web of Patterns [47]	No	Yes	Web/Eclipse	No	Not mentioned
Stylebase for Eclipse	QADA	Yes	Eclipse	Yes	Yes
Morpheus [19]	ATRIUM	No	Windows desktop	Yes	Yes
RBML-PI [23]	RBML	No	Windows desktop	Yes	Yes
ArchiTech	*Quark*	*Arteon*	*Eclipse plug-in*	*No*	*Yes*

- Stylebase for Eclipse[3]: this tool is a plug-in for Eclipse, that is capable to generate code for some architectural patterns, each pattern have a model associated (an image not a real model) and the principal NFRs that are improved (but in a very limited way).
- Morpheus [19]: this tool uses NFRs as constraints over functional requirements that then conditions the software architecture. It is presented as a MDD method that starts from requirements using goal oriented notations.

First of all it is worth to remark that most of these tools are discontinued or created just as a proof of concept. Also, one important fact is that all the tools that appear in this section are the result of an academic research (as far as we know, there is no software company offering similar products). If we look to the SADM and AKM columns, we can see that most of the tools have ways to manage the AK but only few have a well-defined method, this is not strange because most of them are oriented to document ADs but not to assist in the decision-making process. Finally, it is worth to mention that we did not find and explicit link between the AK conceptualizations mentioned in 2.2 and the tools mentioned in this section. The Table 3 summarizes the AK tools mentioned in this section.

3 The Quark Method

NFRs express desired qualities of the system to be developed such as system performance, availability, dependability, maintainability and portability. Over the years, a common claim made by software engineers is that it is not feasible to produce a software system that meets stakeholders' needs without taking NFRs into account. NFRs affect different activities and roles related to the software development process. One of the strongest links is with software architecture, especially architectural decision-making [48]. This observation is the main driver of Quark: facilitate and making more reliable architects' decisions with regard to the desired qualities. The design of Quark has been also driven from some observations gathered from empirical studies [49, 50]:

a) Software architects are the main source of NFRs. This is why the method is centered in the architect.

b) Software architects may be receptive to new design methods as far as they still keep the control on the final ADs. The method should suggest alternatives instead of making final ADs.

c) The amount of information provided by the architects should pay itself. Software architects are pragmatic, a balance between effort and benefit must be reached.

d) The produced ADs should be justified, because architects also have to justify them to other stakeholders.

In Quark, the software architect plays the central role, they specify the NFRs and constraints (a), and then they select among the inferred ADs, and decide when the process has to end (b). In the same direction, Quark is not intrusive. It notifies about possible incompatibilities and possible actions to solve them, but the method does not require resolving any incompatibility to continue with the design, it is up to the software

[3] `stylebase.tigris.org`

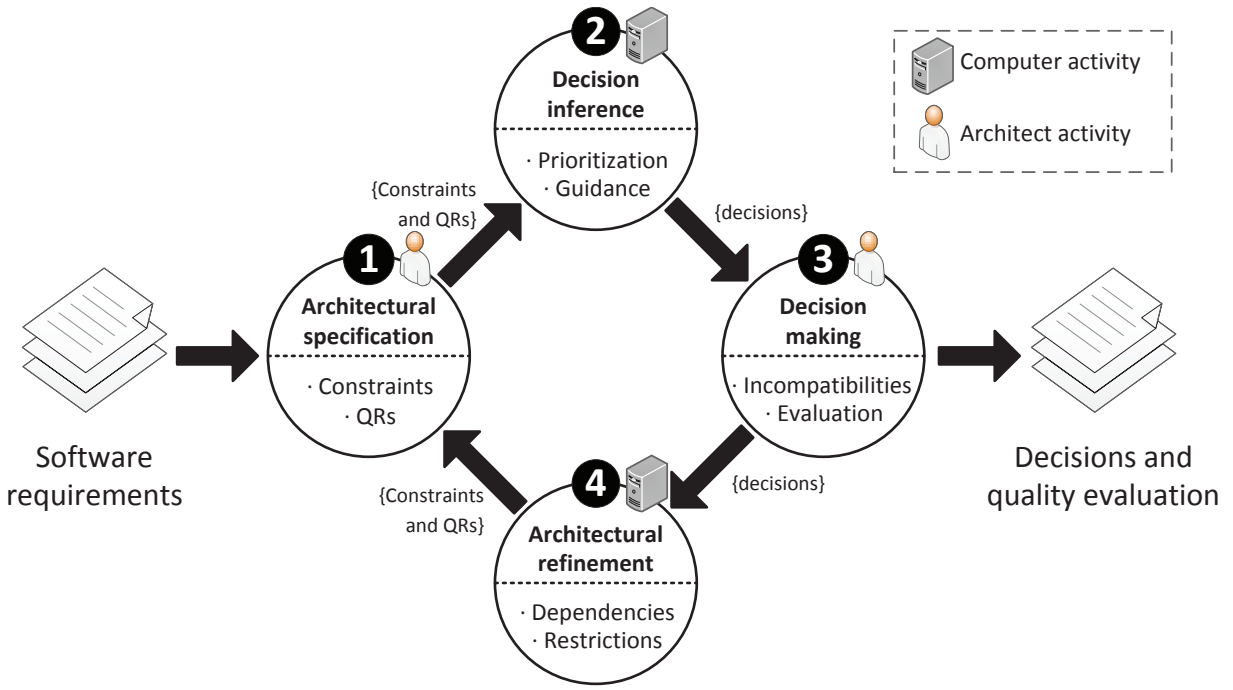

Figure 1: Quark overview.

architect (b). The use of Arteon helps to reuse ADs (c) and also allows to produce detailed information on how an AD was reached, and why it was motivated (d).

The Quark method delivers an iterative process divided into four activities (see Figure 1): first, specification of the NFRs and the imposed constraints related to the architecture; second, inference of ADs; third, decision-making; and fourth, architectural refinement (when necessary). Whenever the solution is refined, activities 1-3 are repeated. In the following subsections we give details on each activity.

3.1 Architectural Specification

In the first activity, the architect specifies the NFRs and constraints that are relevant for the architecture design. For example, a NFR could be *"performance shall be high"* (in other words, more a goal than a requirement) or something more concrete as *"loan processing response time shall not be higher than two seconds 95% of the times"*. Constraints are typically referring to technologies, e.g., *"the database management system (DBMS) must be MySQL 5"*, but may also refer to architectural principles, patterns or styles, as in *"the architectural style must be Service-Oriented Architecture (SOA)"*. These requirements and constraints may come from the project documentation or from the architect's experience (as we found out in our empirical studies [49, 50]).

Due to Quark's iterative nature, aligning with to the conclusions we obtained in our empirical studies (the architect wants to have full control of the process), the specification of these NFRs and constraints does not need to be complete. The architect has freedom to decide if s/he wants to start from a very short specification and then make the architecture grow in each refinement or if s/he wants to provide a more complete specification and see if the expected quality calculated by the method matches the expected NFRs, and then refine till the architecture complies with the requirements.

3.2 Decision Inference

In the second activity, the Quark method uses the AK available in the Arteon ontology to generate a list of ADs. Since the expected amount of ADs in a real case is large, they should be prioritized using some criteria (e.g., the ADs that satisfy more constraints and better comply with the stated NFRs are top priority).

ADs need to be informative. This means that, beyond their name, ADs must include information about: why the AD was offered?, what is the impact in the overall architecture quality?, and what other implications involve making the AD? (a more complete description could be, e.g., the template proposed in [1]). For example, for the AD of using *"data replication"* we could answer the above questions as follows: *"the AD of having data replication is offered because there is a NFR about having high performance"*, *"by making this AD, the overall performance will increase but will affect negatively to the maintenance, and can*

damage the accuracy", *"also, by selecting this AD, the used DBMS is required to be able to operate with data replication."*

3.3 Decision-Making

In the third activity, the architect decides which ADs wants to apply from the ones obtained in the previous activity. When the architect makes an AD, two things may happen. First, there could be incompatibilities with previous ADs (e.g., the architect decides to use *"data replication"*, but s/he already selected a DBMS that does not support data replication), and second, there could be one or more NFRs that are not supported by the ADs made (e.g., the ADs made indicate that maintainability will be damaged while there is a NFR that says that maintainability is very important for this project).

In both cases, the architect will be informed about which ADs are in conflict, but at the end s/he will decide if the set of ADs is satisfactory or not. In some cases there will be non-technical reasons (e.g., the method recommends to use PostgreSQL but the development team is more experienced with MySQL, and the overall quality between both DBMS is similar), for these cases we rely on the experience and knowledge of the architect to make the correct decision.

After the decision-making, the architect has the opportunity to conclude the process by accepting the current set of ADs and their impact in the NFRs. As mentioned in [1], we understand the software architecture as a set of ADs. Alternatively, the architect may choose to start a new iteration. To smooth the transition to a new iteration we have fourth activity, the *architectural refinement*.

3.4 Architectural Refinement

The Refinement activity is for detecting issues that may be resolved in the next iteration. As we commented before the Quark method is an iterative method, and this activity serves as a link between iterations. Our approach is to detect several types of issues that may need the attention of the architect. The three kinds of issues contemplated right now are: incompatibilities, dependencies, and suggestions for NFRs:

- *Incompatibilities* (mentioned in 3.3) are converted into new conditions over the attributes of the architectural elements (see Section 4.1.4). E.g., the architect decides to use *"data replication"*, and this decision implies that the DBMS architectural element must have the attribute *"supports replication"* activated, but the previously selected DBMS does not support replication. If the architect wants to resolve this incompatibility it will become a new constraint that will *ban* the previously selected DBMS. In consequence the *decision making activity* of the next iteration there will be new suggestions of DBMS that support replication (unless the knowledge base does not contain any satisfactory DBMS).

- *Dependencies* occur when some AD requires other parts in the architecture. E.g., when the architect decides to use SOA, several related ADs are needed: service implementation (SOAP, REST, ...), service granularity (service composition, single service, ...), etc. The architect will be asked for which of these dependencies of SOA are to be solved in the next iteration. If the architect selects, for example, the service implementation, there will be a new constraint that will *need* service implementation technology, and during the *decision making activity* of the next iteration there will be generated the ADs related to the technologies that can be used to implement services.

- *Suggestions for NFRs* may be inferred if a particular type of NFR is of special relevance due to the selected ADs. E.g., if Quark detects that there is a great majority of ADs with a have positive impact on security, and security is not part of the current NFRs, there will be suggested to include a new NFR about security. It is worth to remark that this also helps making NFRs explicit, which may help the architect to better understand the relevant concerns of the system to be.

There are other aspects that can be notified to the architect during this activity, for example, as mentioned in the last example of the *incompatibilities*, imagine that the current AK does not contain any component that satisfy the current constraints and NFRs. In this situation the architect may want modify the constraints or NFRs in order to have more alternative decisions generated, but it also could mean that the *domain expert* should add new architectural elements to the knowledge base.

In the next iteration, starting in the Specification activity, the architect will have a list of issues to be resolved (i.e., a *todo list*), and s/he will decide which of the issues are to be converted into new constraints or NFRs. Of course, it is not mandatory for the architect to follow the suggestions, our intention here is to highlight things that may require her/him attention.

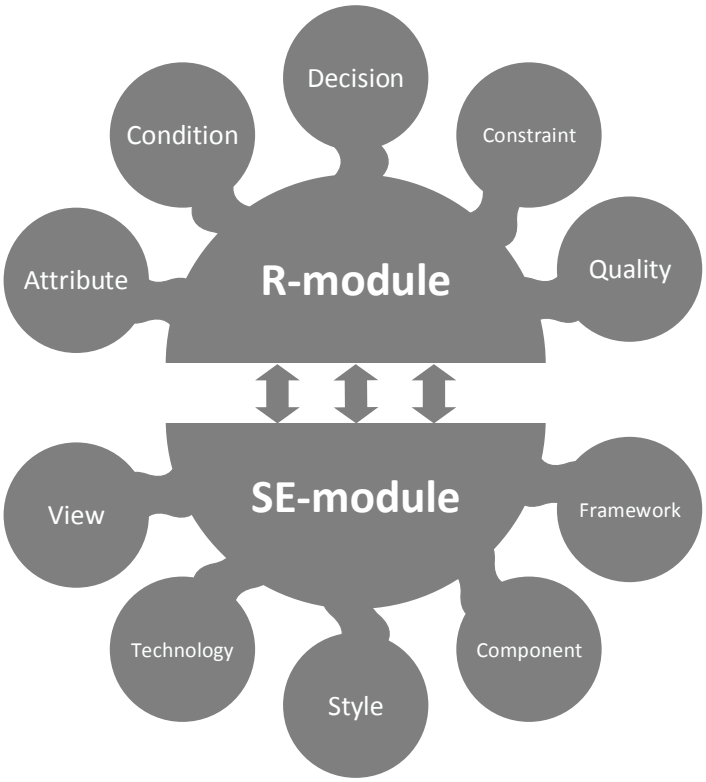

Figure 2: Arteon modules.

4 The Arteon Ontology

Central to Quark is the management and reuse of the AK that supports the architectural decision-making. Among other alternatives, we have chosen to use ontologies to represent the AK as done also by [35]. Ontologies have been successfully used for knowledge representation in other domains (e.g., software engineering, artificial intelligence, semantic web, biomedicine, etc.) and they offer other advantages such as reasoning and learning techniques ready to be applied (e.g., we could add new ADs using case-based reasoning techniques).

Arteon (Architectural and Technological Ontology) is currently divided into two modules (see Figure 2): the *R-module*, reasoning and decision-making knowledge (the module presented in this paper) and the *SE-module*, structural elements, views and frameworks knowledge (presented in [7]). Although interconnected, the two modules are loosely coupled and highly cohesive enough to be reused separately. In particular, the relationship between the R-module and the SE-module is done through a specialization of the Decisional Element concept (see Section 4.1.1) into a full classification of these elements (the classification of this elements is presented in [7]).

We have designed Arteon to manage and reuse AK. A first overview of Arteon appeared in [7]. In that paper, the focus was on the part of the ontology related to the structural elements (SE-module). Our focus here is on the part related to ADs (R-Module). Arteon was designed following the principles stated by Gruber [51], Noy and Hafner [52] Guarino [53] and Evernmann [54]:

Clarity. *"An ontology should effectively communicate the intended meaning of defined terms. Definitions should be objective. [...] All definitions should be documented with natural language"* [51]. *"Be clear about the domain. Any formal theory is a theory about a domain. Such a domain must be clarified in advance"* [53]. We had many iterations in the design of the Arteon ontology, and we have resolved many ambiguities of the terms selected to represent the concepts that appear in the ontology.

Coherence. *"An ontology should be coherent: that is, it should sanction inferences that are consistent with the definitions. At the least, the defining axioms should be logically consistent"* [51]. The coherence and consistency of the ontology has been checked during its design by instantiating the concepts that appear in the ontology with toy examples of AK. This practice produced a faster evolution of the ontology design.

Extendibility. *"An ontology should be designed to anticipate the uses of the shared vocabulary. [...] One should be able to define new terms for special uses based on the existing vocabulary, in a way that does not require the revision of the existing definitions"* [51]. Whenever possible, the definitions of Arteon's concepts are built upon the other concepts that appear in the ontology.

Reusability. *"In order to enable reuse as much as possible, ontologies should be small modules with high internal coherence and limited amount of interaction between the modules"* [52]. As mentioned before, Arteon is composed by two modules connected only by inheritance relationship.

Minimal encoding bias *"The conceptualization should be specified at the knowledge level without depending on a particular symbol-level encoding. An encoding bias results when a representation choices are made purely for the convenience of notation or implementation"* [51]. Arteon has been diagrammed using UML class diagrams to present and describe Arteon' concepts. UML has not been a limitation to express any concept or relationship. We also found in the literature many authors that use UML to diagram the ontologies (e.g., [55, 56]) and there is also the possibility to convert the UML representation of the ontology into OWL [57].

Minimal ontological commitment. *"An ontology should require the minimal ontological commitment sufficient to support the intended knowledge sharing activities. An ontology should make as few claims as possible about the world being modeled"* [51]. Most of the concepts that appear in Arteon are adopted from the software architecture literature. They are defined carefully, and whenever possible we simply adhere to the most widely-accepted definition.

Identity. *"Identity criterion (and especially Lowe's principle, no individual can instantiate both of two sorts if they have different criteria of identity associated with them) can play a crucial role in clarifying ontological distinctions"* [53]. Since Arteon's generalizations are all disjoint we cannot incur in an identity issue.

Basic taxonomy. *"All ontologies are centered on a taxonomy, Such a taxonomy is the main backbone of the ontology, which can be fleshed with the addition of attributes and other relations among nodes. Isolate a basic taxonomic structure. Form a tree of mutually disjoint classes"* [53]. In Arteon this backbone taxonomy are the decisional elements, that are specialized in the SE-module.

Cognitive quality. *"An ontology, as a formal description of a domain, must conform to the way in which the domain is perceived and understood by a human observer"* [54]. We have tried to be as near as possible to the understanding of architects, to this end we used the experience earned from the empirical studies on software architects [49, 50].

The common objective seen in most of the works mentioned in Section 2.2 is materializing AK to share and reuse the knowledge in different software projects and/or communities of architects. In our work, a part from the these objectives, we propose to use AK to guide and facilitate the architects' decision-making. Which, eventually, may bring more reliability to this process by surfacing new alternatives that were not initially considered by the architect. To apply the reasoning techniques necessary to walk this step towards decision-making, we need to be able to formalize the AK. In the next subsection we provide an example of formalization to show the feasibility of our approach.

4.1 The R-Module

In Figure 3 we show the principal concepts of the decision-making knowledge module (R-module) and their relationships. Following we provide a description of each concept of this part of Arteon.

4.1.1 Decisional Element

A Decisional Element is an elemental part of an architecture the architect can decide upon, i.e., the object of decisions. This concept is specialized in the SE-module, so it is left unrefined in the R-Module. The different types of Decisional Element proposed in the SE-module are: architectural styles (e.g., 3-layers), style variations (e.g., 3-layers with data replication), components (e.g., persistence layer), and technology (e.g., Hibernate). Of course, this is not the unique possible specialization of this concept. Being a modular ontology makes it is easy to design and use a different specialization hierarchy for the Decisional Element, which is aligned with the extensibility ontology design principle.

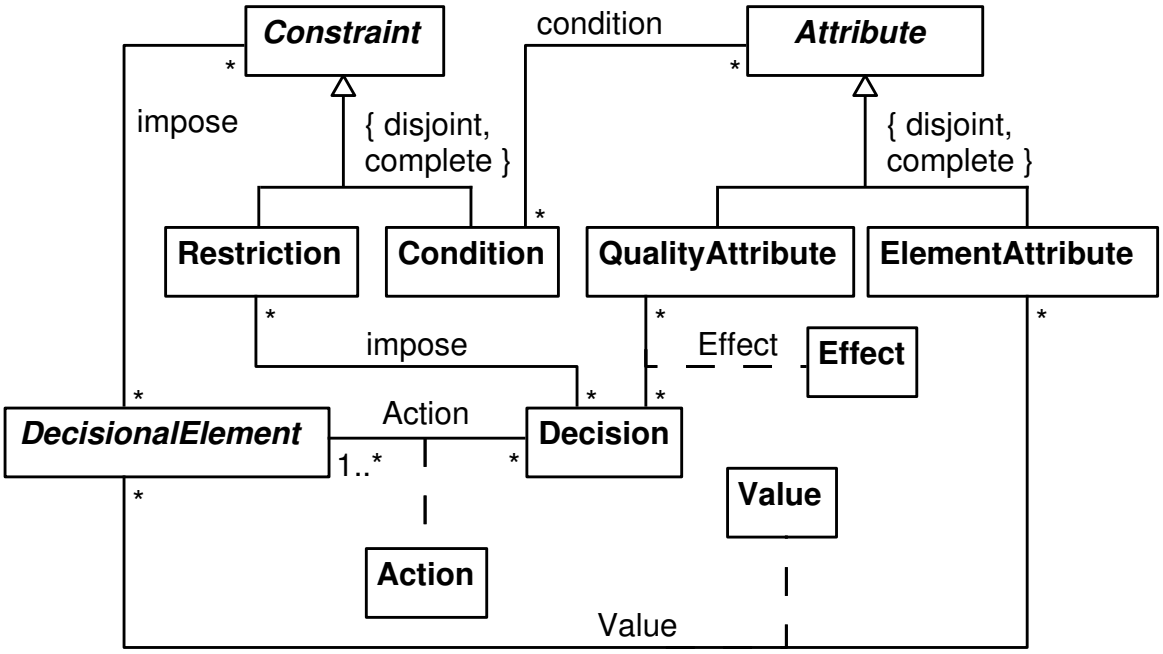

Figure 3: Arteon's R-module.

4.1.2 Decision

According to RUP [58], software architecture is the *"selection of the structural elements and their interfaces by which a system is composed, behavior as specified in collaborations among those elements, composition of these structural and behavioral elements into larger subsystem, architectural style that guides this organization"*. This definition is about making ADs, structural and behavioral, both classified as existence decisions by Krutchen et al. [5].

In Arteon, the decision concept is very similar to the existence decision concept. Decisions are actions over Decisional Elements where the action determines the effect of the decision. Due to the extensibility design principle, we have not closed the ontology to a predefined set of actions, but possible actions could be, for example, the ones proposed in [5]: *use*, the Decisional Element will be in the architecture, and *ban*, the Decisional Element will not be in the architecture.

4.1.3 Constraint

Constraints can be imposed by software requirements or by Decisional Elements (the concept of requirement belongs to the Req-module of Arteon). Constraints coming from requirements are normally described in natural language (e.g., *"the system shall be developed in C++"*), sometimes using templates (e.g., Volere [59]) or a specialized language (e.g., temporal logic, the NFR Framework [21], etc.). Constraints coming from Decisional Elements are formalized as part of the AK (e.g., when the architect uses a technology that is only available for a particular platform, s/he is restricting the architecture to this platform).

Independently from the origin, we distinguish two kinds of constraints:

Restriction A constraint that directly imposes one or more ADs. For example, *"SQL Server"* DBMS needs *"Windows"* operating system.

Condition A constraint that specifies the valid values for attributes (see 4.1.4). For example, if we only want to use OSS software, the condition limit the *"License"* attribute to OSS licenses.

Instead of using constraints to reduce the valid alternative decisions, we could use the constraints to prioritize ADs (e.g., in the previous example OSS technologies would have higher priority than other technologies). In this way we are not hiding alternatives to the architect, but we facilitate her/his work by highlighting the most susceptible alternatives.

As mentioned at the beginning of this section, in order to be able to reason with these constraints they must be formalized as evaluable expressions. Again, the ontology does not commit to any particular proposal, but we provide an example expressed as a Context Free Grammar (CFG) [60] (see Figure 4). We

```
RestrictionSet → Restriction (LogicOp Restriction)*
Restriction → Action [DecisionalElement]
Action → <use> | <ban>
ConditionSet → Condition (LogicOp Condition)*
Condition → ComparativeCond | ConjuntiveCond
ComparativeCond → [Attribute] CompOp [Value]
ConjunctiveCond → [Attribute] ConjOp [Value]+
LogicOp → <and> | <or>
CompOp → <greater_than> | <lower_than> | <equal_to>
ConjOp → <includes> | <excludes>
```

Figure 4: CFG to formalize constraints.

included two extra notations in the CFG to facilitate the reading: *[concept]* means one valid instance of the concept and *<symbol>* means a terminal symbol. We also did not include semantic rules because they are the common ones of typified language (e.g., *"the data type of the value should be the same of the data type of the attribute"*). It is worth to mention that an increase of the expressiveness of the language imply an increase of the complexity of the reasoning system, it is important to find a good balance between both.

4.1.4 Attribute

An Attribute is an *"inherent property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means"* [61]. In Arteon we differentiate two kinds of attributes:

Element Attribute An attribute of a Decisional Element. E.g., the values of the *"license"* attribute are the names of the licenses. Only Decisional Elements for which the license is relevant will have a value (e.g., technologies).

Quality Attribute An attribute that characterizes some aspect of the software quality. For example, ISO/IEC 25000 [61] defines a hierarchy of characteristics: functionality, reliability, usability, efficiency, maintainability, and portability.

In this case, we also followed the extensibility principle by leaving the attributes customizable. Initially, we thought to propose a set of attributes, the most generic and independent of domain, but when we tried, we found out that domain-specific quality models may be more adequate in each situation (e.g., the S-Cube quality model [62] is specific for SOA) and that the element attributes are uncountable, and even worse, the same information can be modeled with different attributes (e.g., for the license, we may have a boolean attribute, true when is a OSS license and false otherwise, or as before have an attribute with a list of licenses). We opted to let the domain expert decide which attributes are convenient in each case, but we acknowledge that more research is needed in order to make this knowledge reusable from one project to another.

5 The ArchiTech Tool

The ArchiTech tool [8] (see the video at `www.upc.edu/gessi/architech/` for a running example) is the proof of concept of the Quark method and the Arteon ontology. This tool is composed by two subsystems: ArchiTech-CRUD, and ArchiTech-DM:

ArchiTech-CRUD This subsystem provides facilities for the management of AK as it is defined in the Arteon ontology. It also provides an embedded database, and options to export the AK. The CRUD operations (i.e., create, read, update and delete) are available for the following concepts:

- *Architectural element.* A full description of what is an *architectural element* is available in [7] (e.g., architectural styles -SOA, layered, etc.-, components -services, packages, etc.-, technologies -DBMS, RESTful vs. W3C, etc.-)
- *Architectural Decisions* Defined in Section 4.1.2 (e.g., which architectural style to apply or which DBMS to choose)
- *Attributes and Quality models.* Defined in Section 4.1.4 (e.g., the property *"License"* may be used to classify and reason about OSS technologies, and the S-Cube quality model to design SOA systems [62])

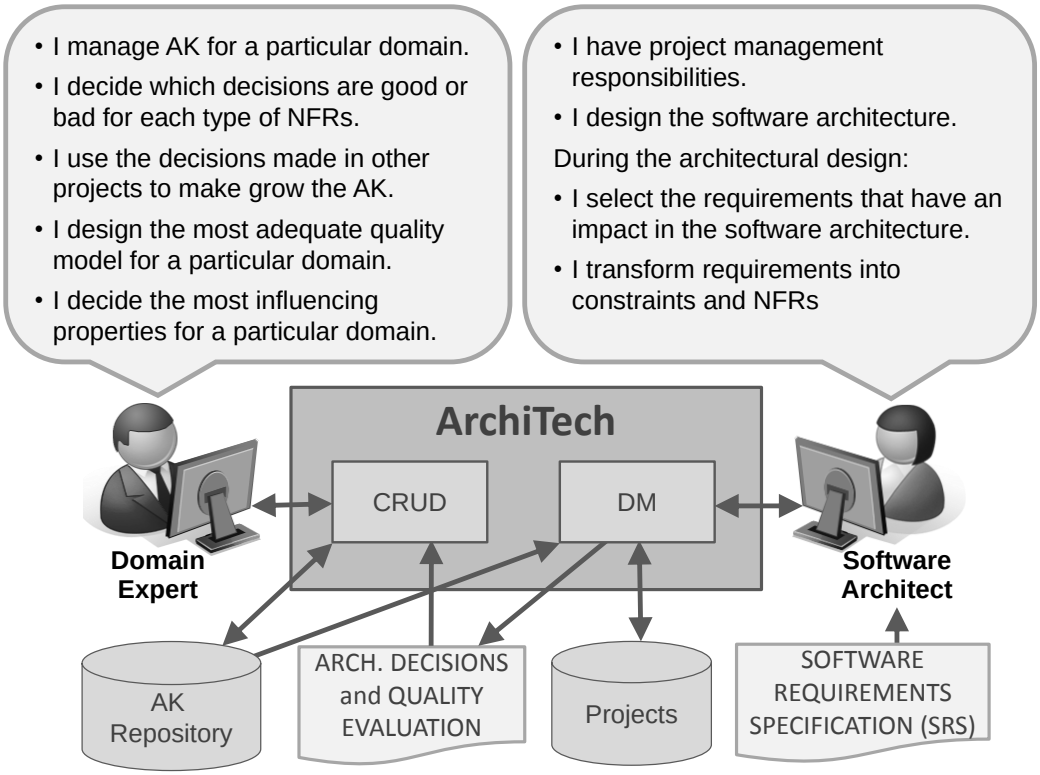

Figure 5: ArchiTech overview.

ArchiTech-DM This subsystem assists software architects in architectural decision-making as described in the Quark method as described in Section 3. Nevertheless, there are few limitations in the implementation:

- *Architectural Specification activity* supports the goal kind of NFRs (e.g., *"security must be high"*) but not the more concrete kind (e.g., *"loan processing response time shall not be higher than two seconds 95% of the times"*)
- *Architectural Specification activity* but restrictions are fully supported.
- *Decision Inference activity* uses the simulated annealing algorithm [63] as the ADs inference mechanism. There are many algorithms and variants that can be used for this task, and the results may slightly differ depending on the one used.

One extra feature in this subsystem is that the architect can monitor the overall status of the NFRs while making ADs. This feature gives to the architect a clear notion of what is happening at any moment.

Figure 5 shows an overview of this tool, and the tasks and responsibilities of each subsystem's user. Since *ArchiTech-CRUD* is not directly related to Quark we will focus on *ArchiTech-DM* in the example of Section 6.

6 Example

A complete example of an architectural decision-making process of a whole software architecture would be too long and very difficult to explain in a comprehensible manner. Therefore, in this example we will focus on one architectural decision, the selection of the DBMS. This example is mostly about the selection of one technology, but the same idea can apply to, for example, the selection of an architectural pattern.

Following the Quark method, first the architect will identify the software requirements that are relevant to the architecture. For this example, we can imagine that the software architect has identified the following requirements as relevant:

(R1) *"The software system shall keep the information about clients and providers"*

(R2) *"The software system shall be developed using OSS whenever possible"*

(R3) *"The software system shall have backup systems for reliability"*

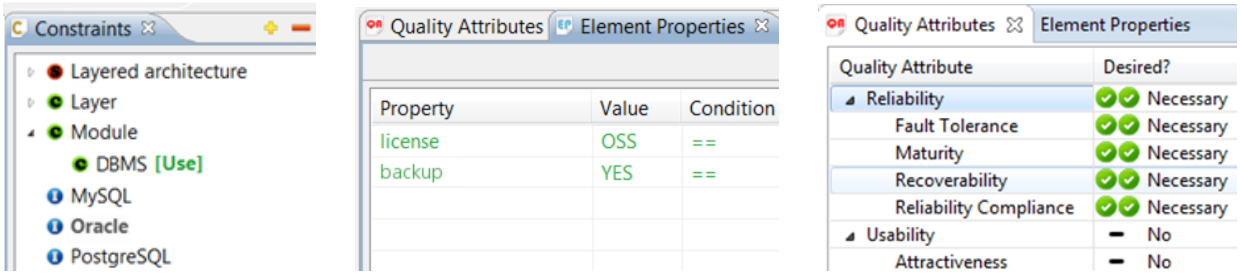

Figure 6: Specification of restrictions, conditions, and NFRs in ArchiTech.

6.1 Specification Activity

Once software requirements are identified, the software architect should translate them into constraints formalized with the language described in Figure 4. From R1, the architect may deduce that the project is an information system, so a DBMS will be required. R2 sets a constraint on the technologies used to be OSS. R3 sets constraints for backup facilities, and also mentions that reliability is a desired type of NFRs. The formalization would be:

(C1) From R1 the restriction: *use* "DBMS"

(C2) From R2 the condition: "License" *equal* "OSS"

(C3) From R3 the condition: "Backup facility" *equal* "yes"

(C4) From R3 the NFR: "Reliability" *greater than* "average"

Note that R2 could have been also codified as: "License" *includes* {"GPL", "LGPL", "BSD", etc.}. It depends on how the *domain expert* codifies the AK (see Figure 5). In some cases it would be interesting to have the concrete license of each technology, and in other cases it would be enough to have them classified between OSS and non-OSS.

In Figure 6 we can see how restrictions, conditions and the soft goal kind of NFRs are specified in the ArchiTech tool. On the left we have the restriction to *use* a "DBMS" which is a "module" to be used in the "Persistence Layer" because in the tool we have provided the AK for a the typical "3-Layer" architectural style. In the center of the Figure 6 we have the two conditions, and on the right we have the NFR for "Reliability". Note that we are using the ISO/IEC 9126 [41] in the example. This quality model is hierarchical and "Reliability" is a top quality attribute, and this is why when we selected it all the sub-attributes were also selected.

6.2 Decision Inference Activity

Now that we have formalized the constraints and NFRs, the next activity in Quark is the inference of ADs. Depending on the AK codified in the knowledge base and the prioritization criteria used, the method will prioritized list of ADs.

For this example, the AK is based on the information published in the Postgres Online Journal [64] and the prioritization criteria is to give higher priority to ADs that satisfy more constraints and improve the selected types of NFRs. This prioritization criteria is the same used by ArchiTech, and, right now, this cannot be customized in the tool.

The resulting list of justified[4] ADs is:

1. The AD of using MySQL 5 is offered because it is OSS. There is no information available about backup facilities in MySQL. MySQL is preferred because it supports more OSS technologies. Using MySQL has neutral impact in reliability because ACID compliance depends on the configuration.

2. The AD of using PostgreSQL 8.3 is offered because it is OSS. There is no information available about back-up facilities in PostgreSQL. There are few OSS technologies with support for PostgreSQL. Using PostgreSQL improves reliability because it is ACID compliant.

3. The AD of using SQL Server 2005 is offered because it satisfies the backup facility condition. SQL Server is not OSS. There are few OSS technologies with support for SQL Server. SQL Server will require a Windows operating system. Using SQL server improves reliability because it is ACID compliant.

[4]We mentioned in Section 3 the importance for software architects to have justified ADs

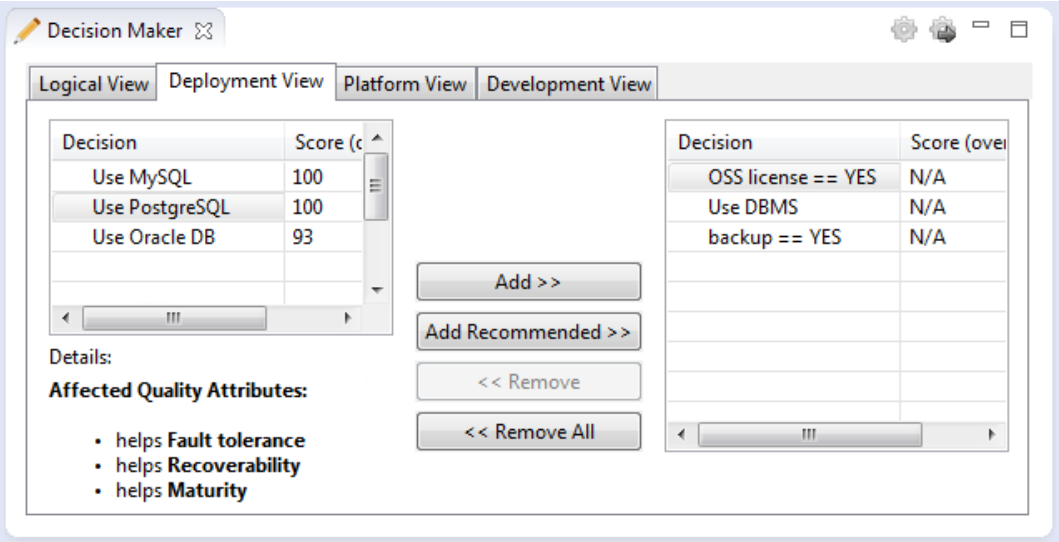

Figure 7: Decision-Making in ArchiTech.

6.3 Decision-Making Activity

In the Decision-Making activity, the architect, for example, will decide to use MySQL 5 (the AD with higher priority) as the implementing technology for the DBMS component. But as said before in this paper, the architect may prefer to use PostgreSQL, even it is not the highest-ranked AD. The important point is that the architect is able to make informed ADs, and, eventually, new ADs that were unknown to her/him are taken into consideration.

In the Figure 7 we can see how this decision making process would look in the ArchiTech tool. In this figure we can observe several aspects of the tool:

- Decisions are ranked from 0 to 100 depending on how they affect the Quality attributes that are marked as relevant. For example, *Oracle* may have a better effect on performance than *PostgreSQL*, but since performance is not one of the relevant quality attributes in the example it is not considered in the computation. Conditions are also considered in this computation, this is why *Oracle*, which is not OSS, is ranked lower than the two other options.

- We can see that ADs are organized into the four architectural views: Logical, Deployment, Platform, and Development. From the Quark and Arteon perspective, the architectural framework is totally customizable but in ArchiTech it is limited to these four views to reduce the knowledge fragmentation.

- We can see that the description of the architectural decision selected in the tool (i.e., *Use PostgreSQL*), is not as expressive as the ones in the Section 6.2. However, we can only see the list of affected quality attributes, and if the decision is conflicted with some of the conditions (e.g., if we select *Use Oracle DB* the tool would show a warning for the violation of the condition on the *"License"* attribute).

- On the right, we can see the list of architectural decisions. The ArchiTech tool includes in this list the specified conditions and restrictions in Section 6.1. In this way the architect can see all the decisions made, not only the *existence decisions*, but also the *executive and property decisions*. Note that these ADs are not ranked, because they were imposed by the architect.

At this point, the architect would only need to select the desired AD from the list on the left and press the button "Add", which would move the decision to the right list.

6.4 Architectural Refinement Activity

After the Decision-Making activity the architectural design will continue with new iterations, where the AD of using MySQL will impact, e.g., in the selection of other technologies that are compatible with MySQL. This information will appear during the Refinement activity as dependencies and incompatibilities.

Currently, the ArchiTech tool, will recalculate the ranking of decisions in the refinement activity. If some of the ADs made had a dependency or incompatibility the affected decisions would be ranked higher or lower than before, but it does not show any dialog to add or remove architectural elements due to the detected dependencies or incompatibilities.

7 Conclusions, limitations, and future work

One of the most known Kruchten's statements is *"the life of a software architect is a long (and sometimes painful) succession of suboptimal decisions made partly in dark"* [65]. The lack of knowledge is one of the reasons to produce suboptimal decisions. For example, the architect may not know all the effects of using some technology or architectural pattern: it may need of other components to work correctly (e.g., some of them may be incompatible with other ADs), it may have unexpected effects in the overall evaluation of some NFRs (e.g., lowers the resource utilization efficiency). Also, the lack of knowledge may cause a worse situation when some alternative is not considered because it is unknown to the architect. To improve this situation we presented Quark, a method to assist software architects in architectural decision-making.

About the limitations of the presented work, one of the major problems we have to deal with is the amount of knowledge required. Our position is that the best way to acquire and maintain such amount of knowledge is making architects active participants of its acquisition and maintenance. A possible way to achieve this participation is using networks of knowledge, which have been successful in other areas (e.g., Stack Overflow for software developers). Other techniques that have been considered to acquire and maintain this knowledge are knowledge reuse and knowledge learning, but both have drawbacks, for example, reusing knowledge you may find out that a solution that provides high security in a information system may not be secure enough for a critical system, and in order to use learning techniques first is necessary to have a big source of knowledge.

With regard to the future work we envisioned the transformation of the resulting set of ADs form the Quark method into models for the architectural views, and then into the actual software architecture implementation using a MDD approach (our first ideas were presented in [66]). It is worth to remark that this is not contradictory with our empirical observations about architects not willing to have automatic tools that produce the full architecture because the ADs will continue being produced by interacting with the architect, who had full control on which ADs are made. Currently neither Quark, Arteon, or ArchiTech have been applied in practice, but we are planning to use experiments and a real case study to improve the validation of the presented contributions.

Acknowledgments

This work has been partially supported by the Spanish MICINN project TIN2010-19130-C02-01. Thanks to Oriol Collell for the development of the ArchiTech tool.

References

[1] J. Tyree and A. Akerman, "Architecture decisions: demystifying architecture," *IEEE Software*, vol. 22, pp. 19–27, 2005.

[2] 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)*, Pittsburgh, PA, USA, 2005, pp. 109–120.

[3] P. Krutchen, "An Ontology of Architectural Design Decisions in Software Intensive Systems," in *Proceedings of the 2nd Groningen Workshop Software Variability*, Groningen, The Netherlands, 2004, pp. 1–8.

[4] P. Kruchten, R. Capilla, and J. C. Duenas, "The Decision View's Role in Software Architecture Practice," *IEEE Software*, vol. 26, pp. 36–42, 2009.

[5] P. Kruchten, P. Lago, and H. van Vliet, "Building Up and Reasoning About Architectural Knowledge," in *Proceedings of the 2nd international conference on Quality of Software Architectures (QoSA)*, Västerås, Sweden, 2006, pp. 43–58.

[6] R. de Boer, R. Farenhorst, P. Lago, H. van Vliet, V. Clerc, and A. Jansen, "Architectural Knowledge: Getting to the Core," in *Proceedings of the 3rd International Conference on Quality of Software Architectures (QoSA)*, Medford, MA, USA, 2007, pp. 197–214.

[7] D. Ameller and X. Franch, "Ontology-based Architectural Knowledge representation: structural elements module," in *Proceedings of the Conference on Advanced Information Systems Engineering Workshops (CAISE)*, London, UK, 2011, pp. 296–301.

[8] D. Ameller, O. Collell, and X. Franch, "ArchiTech: Tool Support for NFR-Guided Architectural Decision-Making," in *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE)*, Chicago, IL, USA, 2012, pp. 315–316.

[9] D. Falessi, G. Cantone, and P. Kruchten, "Do Architecture Design Methods Meet Architects' Needs?" in *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Mumbai, India, 2007, p. 5.

[10] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-Based Analysis of Software Architecture," *IEEE Software*, vol. 13, pp. 47–55, 1996.

[11] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *Proceedings of the 4th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, Monterey, CA, USA, 1998, pp. 68–78.

[12] F. Bachmann and L. Bass, "Introduction to the Attribute Driven Design method," in *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, Toronto, Ontario, Canada, 2001, pp. 745–746.

[13] J. Bosch, *Design and use of software architectures: adopting and evolving a product-line approach.* ACM Press/Addison-Wesley Publishing Co, 2000.

[14] M. Matinlassi, E. Niemelä, and L. Dobrica, "Quality-driven architecture design and analysis method. A revolutionary initiation approach to a product line architecture," in *VTT Technical Research Centre of Finland*, 2002. [Online]. Available: http://www.vtt.fi/inf/pdf/publications/2002/P456.pdf

[15] H. Choi, K. Yeom, Y. Choi, and M. Moon, "An approach to quality achievement at the architectural level: AQUA," in *Proceedings of the 8th IFIP WG 6.1 International Conference (FMOODS)*, Bologna, Italy, 2006, pp. 20–32.

[16] A. Bertolino, A. Bucchiarone, S. Gnesi, and H. Muccini, "An architecture-centric approach for producing quality systems," in *Proceedings of the 1st International Conference on the Quality of Software Architectures (QoSA) and Second International Workshop on Software Quality (SOQUA)*, vol. 3712, Erfurt, Germany, 2005, pp. 21–37.

[17] T. Al-naeem, I. Gorton, M. Ali-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 (ICSE)*, St. Louis, Missouri, USA, 2005, pp. 244–253.

[18] A. Tang, J. Han, and R. Vasa, "Software Architecture Design Reasoning: A Case for Improved Methodology Support," *IEEE Software*, vol. 26, pp. 43–49, 2009.

[19] F. Montero and E. Navarro, "ATRIUM: Software Architecture Driven by Requirements," in *Proceedings of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, Potsdam, Germany, 2009, pp. 230–239.

[20] L. Chung, K. Cooper, and A. Yi, "Developing adaptable software architectures using design patterns: an NFR approach," *Computer Standards & Interfaces*, vol. 25, pp. 253–260, 2003.

[21] L. Chung, B. Nixon, and E. Yu, *Non-functional requirements in software engineering.* Kluwer Academic, 2000.

[22] S. Bode, A. Fischer, W. Kuehnhauser, and M. Riebisch, "Software Architectural Design meets Security Engineering," in *Proceedings of the 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, San Francisco, CA, USA, 2009, pp. 109–118.

[23] S. Kim, D.-K. Kim, L. Lu, and S. Park, "Quality-driven architecture development using architectural tactics," *Journal of Systems and Software (JSS)*, vol. 82, pp. 1211–1231, 2009.

[24] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, Second Edition.* Addison-Wesley Longman Publishing Co., Inc., 2003.

[25] D. Perovich, C. Bastarrica, and C. Rojas, "Model-Driven approach to Software Architecture design," in *Proceedings of the 4th Workshop on Sharing and Reusing Architectural Knowledge (SHARK)*, Vancouver, BC, Canada, 2009, pp. 1–8.

[26] M. Brambilla, J. Cabot, and M. Wimmer, *Model-driven Software Engineering in Practice*, ser. Synthesis digital library of engineering and computer science. Morgan & Claypool, 2012.

[27] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained, The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[28] E. Niemela and A. Immonen, "Capturing quality requirements of product family architecture," *Information and Software Technology (IST)*, vol. 49, pp. 1107–1120, 2007.

[29] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A general model of software architecture design derived from five industrial approaches," *Journal of Systems and Software (JSS)*, vol. 80, no. 1, pp. 106–126, 2007.

[30] F. Bachmann, L. Bass, M. Klein, and C. Shelton, "Designing software architectures to achieve quality attribute requirements," *IEEE Software*, vol. 152, no. 4, pp. 153–165, 2005.

[31] P. Avgeriou, P. Kruchten, P. Lago, P. Grisham, and D. Perry, "Architectural knowledge and rationale: issues, trends, challenges," *ACM SIGSOFT Software Engineering Notes*, vol. 32, pp. 41–46, 2007.

[32] ISO/IEC/(IEEE), "42010 (IEEE Std) 1471-2000: Systems and Software engineering – Recomended practice for architectural description of software-intensive systems," 2007.

[33] R. Capilla, F. Nava, and J. C. Duenas, "Modeling and Documenting the Evolution of Architectural Design Decisions," in *Proceedings of the 2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI)*, Minneapolis, MN, USA, 2007, p. 9.

[34] C. Lopez, L. M. Cysneiros, and H. Astudillo, "NDR Ontology: Sharing and Reusing NFR and Design Rationale Knowledge," in *Proceedings of the 1st International Workshop on Managing Requirements Knowledge (MARK)*, Barcelona, Spain, 2008, pp. 1–10.

[35] A. Akerman and J. Tyree, "Using ontology to support development of software architectures," *IBM Systems Journal*, vol. 45, no. 4, pp. 813–826, 2006.

[36] L. Babu T., M. Seetha Ramaiah, T. V. Prabhakar, and D. Rambabu, "ArchVoc–Towards an Ontology for Software Architecture," in *Proceedings of the 2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI)*, Minneapolis, MN, USA, 2007, p. 5.

[37] C. Pahl, S. Giesecke, and W. Hasselbring, "Ontology-based modelling of architectural styles," *Information and Software Technology (IST)*, vol. 51, pp. 1739–1749, 2009.

[38] K. Henttonen and M. Matinlassi, "Open source based tools for sharing and reuse of software architectural knowledge," in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA)*, Cambridge, UK, 2009, pp. 41–50.

[39] M. Shahin, P. Liang, and M.-R. Khayyambashi, "Architectural design decision: Existing models and tools," in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA)*, Cambridge, UK, 2009, pp. 293–296.

[40] R. de Boer, P. Lago, A. Telea, and H. van Vliet, "Ontology-driven visualization of architectural design decisions," in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA)*, Cambridge, UK, 2009, pp. 51–60.

[41] ISO/IEC 9126, "Product quality – Part 1: Quality model," ISO, 2001.

[42] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," *Journal of Systems and Software (JSS)*, vol. 80, pp. 918–934, 2007.

[43] M. A. Babar and I. Gorton, "A Tool for Managing Software Architecture Knowledge," in *Proceedings of the 2nd Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent (SHARK-ADI)*. Minneapolis, MN, USA: IEEE Computer Society, 2007, p. 11.

[44] D. Garlan and B. Schmerl, "AEvol: A tool for defining and planning architecture evolution," in *Proceedings of International Conference on Software Engineering (ICSE)*, 2009, pp. 591–594.

[45] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas, "A web-based tool for managing architectural design decisions," *ACM SIGSOFT Software Engineering Notes*, vol. 31, p. 4, 2006.

[46] A. Jansen, T. Vries, P. Avgeriou, and M. Veelen, "Sharing the Architectural Knowledge of Quantitative Analysis," in *Proceedings of the 4th International Conference on the Quality of Software-Architectures (QoSA)*, Karlsruhe, Germany, 2008, pp. 220–234.

[47] J. Dietrich and C. Elgar, "Towards a web of patterns," *Journal of Web Semantics*, vol. 5, pp. 108–116, 2007.

[48] L. Chung and J. C. S. do Prado Leite, *Conceptual Modeling: Foundations and Applications. Essays in Honor of John Mylopoulos.* Springer Berlin Heidelberg, 2009, ch. On Non-Functional Requirements in Software Engineering, pp. 363–379.

[49] D. Ameller, C. Ayala, J. Cabot, and X. Franch, "How do Software Architects Consider Non-functional Requirements: An Exploratory Study," in *20th IEEE International Requirements Engineering Conference (RE)*, Chicago, IL, USA, 2012, pp. 41–50.

[50] ——, "Non-Functional Requirements in Architectural Decision-Making," *IEEE Software*, vol. 30, no. 2, pp. 61–67, March-April 2013.

[51] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *International Journal of Human-Computer Studies*, vol. 43, pp. 907–928, 1995.

[52] N. F. Noy and C. D. Hafner, "The state of the art in ontology design: A survey and comparative review," *AI Magazine*, vol. 18, pp. 53–74, 1997.

[53] N. Guarino, "Some Ontological Principles for Designing Upper Level Lexical Resources," in *Proceedings of the 1st International Conference on Language Resources and Evaluation*, Granada, Spain, 1998, pp. 527–534.

[54] J. Evermann and J. Fang, "Evaluating ontologies: Towards a cognitive measure of quality," *Information Systems*, vol. 35, pp. 391–403, 2010.

[55] G. Guizzardi, G. Wagner, and H. Herre, "On the Foundations of UML as an Ontology Representation Language," in *Proceedings of the 14th International Conference on Engineering Knowledge in the Age of the Semantic Web (EKAW)*, Whittlebury Hall, UK, 2004, pp. 47–62.

[56] D. Berardi, D. Calvanese, and G. De Giacomo, "Reasoning on UML Class Diagrams," *Artificial Intelligence*, vol. 168, no. 1-2, pp. 70–118, 2005.

[57] D. Gasevic, D. Djuric, V. Devedzic, and V. Damjanovi, "Converting UML to OWL ontologies," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters (WWW Alt.)*, New York, NY, USA, 2004, pp. 488–489.

[58] P. Kroll and P. Kruchten, *The rational unified process made easy: a practitioner's guide to the RUP.* Addison-Wesley, 2003.

[59] J. Robertson and S. Robertson, "Volere: Requirements Specification Template," Atlantic Systems Guild, Tech. Rep., 2010.

[60] A. Nijholt, *Context-Free Grammars: Covers, Normal Forms, and Parsing.* Springer, 1980.

[61] ISO/IEC 25000, "Software product Quality Requirements and Evaluation (SQuaRE)," 2005.

[62] A. Gehlert and A. Metzger, "Quality reference model for sba (deliverable cd-jra-1.3.2)," S-Cube, Tech. Rep., 2009.

[63] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[64] L. Hsu and R. Obe, "Cross Compare of SQL Server, MySQL, and PostgreSQL," http://www.postgresonline.com/article_pfriendly/51.html, 2008.

[65] P. Kruchten, "The Software Architect," in *Proceedings of the 1st Working IFIP Conference on Software Architecture (WICSA)*, San Antonio, TX, USA, 1999, pp. 565–584.

[66] D. Ameller, X. Franch, and J. Cabot, "Dealing with Non-Functional Requirements in Model-Driven Development," in *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE)*, Sydney, NSW, Australia, 2010, pp. 189–198.