ArchiTech: Tool Support for NFR-guided Architectural Decision-Making

David Ameller, Oriol Collell, Xavier Franch
Software Engineering for Information Systems research group (GESSI)
Universitat Politècnica de Catalunya (UPC)
Barcelona (Catalunya, Spain)
{*dameller, ocollell, franch*}*@essi.upc.edu*

***Abstract*—Researchers from from requirements engineering and software architecture had had emphasized the importance of Non-Functional Requirements and their influence in the architectural design process. To improve this process we have designed a tool, ArchiTech, which aims to support architects during the design process by suggesting alternative architectural decisions that can improve some types of non-functional requirements in a particular project, and facilitate the reuse of architectural knowledge shared between projects of the same architectural domain (e.g., web-based applications).**

***Keywords*-Non-functional requirement; architectural decision; computer-aided support system;**

I. Introduction

Non-Functional Requirements (NFRs) are one of the main targets of research in the Requirements Engineering community [1]. A trending topic along this line is the analysis of relationships between NFRs and software architectures (e.g., [2]). From the perspective of the software architecture community, many researchers have stated that architectural decisions are the core of software architecture [3], and how this decisions can be reused among projects by having a common knowledge base, normally referred as Architectural Knowledge (AK) [4]. Therefore, it seems natural to explore the links among NFRs, architectural decisions and AK and to look for tool support for their coordinated management.

In this paper we present ArchiTech, a tool to guide architects in the architectural decision-making process. ArchiTech integrates two subsystems: an Architectural Knowledge (AK) manager, ArchiTech-CRUD, and an architectural decision-making-assistant, ArchiTech-DM (see Figure 1).

II. ArchiTech Description

ArchiTech tool is one piece of a bigger envisioned framework to deal with NFRs in Model-Driven Development [5].

ArchiTech starts from a set of quality requirements and constraints derived from a SRS by the architect. As result of the decision-making process (described in Section II-B), ArchiTech will provide a set of architectural decisions and an overall evaluation of the expected quality as consequence of applying the resulting decisions. The results obtained from ArchiTech-DM require the management and maintainability of Architectural Knowledge, these facilities are provided

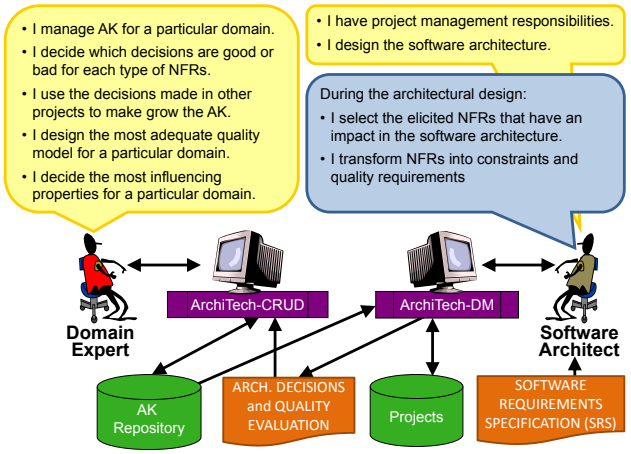

Figure 1. ArchiTech overview

by Architech-CRUD in form of interrelated CRUD (Create-Read-Update-Delete) operations (described in Section II-A) applied to the concepts defined in an AK ontology [6].

A. ArchiTech-CRUD

This subsystem provides a graphical user interface for the domain expert to operate with the AK. The CRUD operations are specialized for four different types of knowledge:

- *Architectural element.* The domain expert has to define the elements (e.g., architectural styles -SOA, layered, etc.-, components -services, packages, etc.-, technologies -DBMS, RESTful vs. W3C, etc.-) that are to be used to structure any software architecture. We use four architectural views (logical, deployment, development, and platform) to classify these elements.
- *Properties.* Each architectural element may have values for one or more properties that are defined by the domain expert (e.g., the property License may be used to classify and reason about OSS technologies).
- *Types of NFRs.* We give freedom to the domain expert to define (and reuse in multiple projects) the most appropriate quality model for her interests (e.g., the S-Cube quality model to design SOA systems [7]).

- *Architectural decisions.* The domain expert has to define the decisions that are more habitual in a particular architectural domain (e.g., web-based system, service-based system, etc.), and which types of NFRs are affected by each alternative. Decisions can be higher-level (e.g., which architectural style to apply) or lower-level (e.g., which DBMS to choose).

In order to provide management facilities, the AK must be persistent and easy to share among projects. To this end we provide an embedded database, and we have also added an option to export the stored AK to an XML file.

B. ArchiTech-DM

This subsystem uses our method, called Quark, to guide software architects in NFR-driven decision-making. Quark starts from the SRS (from which we focus in NFR at this stage), and ends with a set of architectural decisions and the overall evaluation of the software quality. The Quark method delivers an iterative process divided into four activities:

1) *Architectural Specification.* The architect specifies the QRs and constraints (using the SRS as basis) that will drive the architecture decision making. For example, a QR could be "performance should be high" (in other words, more a goal than a requirement), and constraints could be "the database management system (DBMS) must be MySQL 5". As happens in this example, QRs may be at a high level of abstraction.
2) *Decision Inference.* The ArchiTech tool uses the AK available to generate a prioritized list of decisions (e.g., the decisions that satisfy more constraints and better comply with the stated QRs are top priority) using a local search algorithm.
3) *Decision Making.* The architect decides what decisions are to be applied from the ones generated in the previous activity. When the architect makes a decision, some issues may rise (e.g., we may be selecting the "data replication" decision, but we could have already selected a DBMS not supporting this feature).
4) *Architectural Refinement.* The ArchiTech tool identifies possible issues and suggest actions that to resolve them (e.g., following the previous example, the tool may suggest to use a DBMS with data replication).

After the fourth activity, we may end the process by accepting the resulting set of architectural decisions or use the suggested actions provided by the tool and start a new iteration. Once a new iteration starts, the architect is free to change any of the previously defined requirements (e.g., s/he may want to soften some of them to get more alternatives).

III. Conclusion

In this paper we have described the principal parts of ArchiTech and summarized the specific use cases of the tool.

A detailed description of the system may be found at (www.upc.edu/gessi/architech/index.html), where the current version is available for download. Also, the details of the design of ArchiTech (architectural description, technologies used, etc.) are available in [8]. Finally, a screen-shot of the tool is shown in Figure 2.

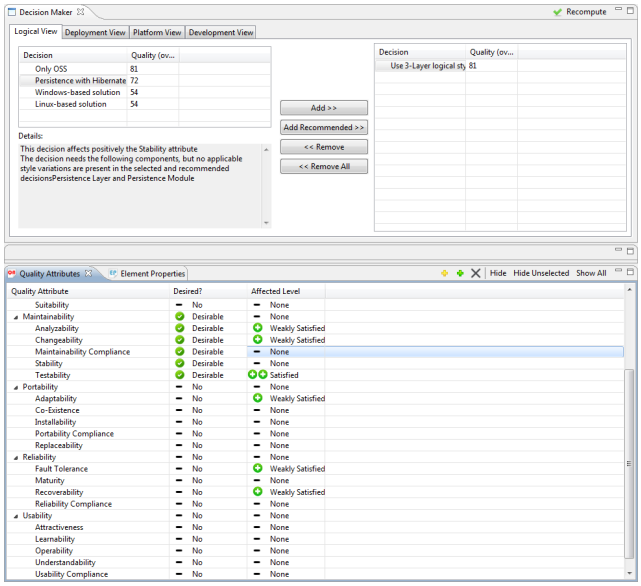

Figure 2. ArchiTech screen-shot

Acknowledgment

This work has been partially supported by the Spanish MICINN project TIN2010-19130-C02-02.

References

[1] M. Glinz, "On Non-Functional Requirements," in *RE*, 2007.

[2] J. A. Miller, R. Ferrari, and N. H. Madhavji, "An exploratory study of architectural effects on requirements decisions," *JSS*, vol. 83, no. 12, pp. 2441–2455, 2010.

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

[4] P. Kruchten, P. Lago, and H. van Vliet, "Building Up and Reasoning About Architectural Knowledge," in *QoSA*, 2006.

[5] D. Ameller, X. Franch, and J. Cabot, "Dealing with Non-Functional Requirements in Model-Driven Development," in *RE*, 2010.

[6] D. Ameller and X. Franch, "Ontology-based Architectural Knowledge representation: structural elements module," in *IWSSA, CAiSE*, 2011.

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

[8] D. Ameller, O. Collell, and X. Franch, "Reconciling the 3-layer architectural style with a plug-in-based architecture: the Eclipse case," in *TOPI, ICSE*, 2011.

ArchiTech: Tool Support for NFR-guided Architectural Decision-Making (Demo Outline)

I. Introduction

This document describes how the ArchiTech tool will be presented during the conference. The tool is addressed to software architects and requirements engineers collaborating in the design of the architecture of a given software system in a way that maximizes the fulfillment of its non-functional requirements (NFRs). During the demo, the presenter will take the role of the software architect and will show how the tool can assist him on the architecture design activity by following the Quark method (see figure 1). Some attendee will be contacted in advance to play the role of requirements engineers. As explained in the extended abstract, ArchiTech also includes a CRUD subsystem (ArchiTech-CRUD) to manage the Architectural Knowledge (AK) to be used by the second subsystem, ArchiTech-DM, in the decisional process to suggest architectural decisions.The demo will focus on the decisional subsystem which is far more interesting. ArchiTech-CRUD will be just briefly introduced although, in case a participant shows interest, we will show it in more detail.

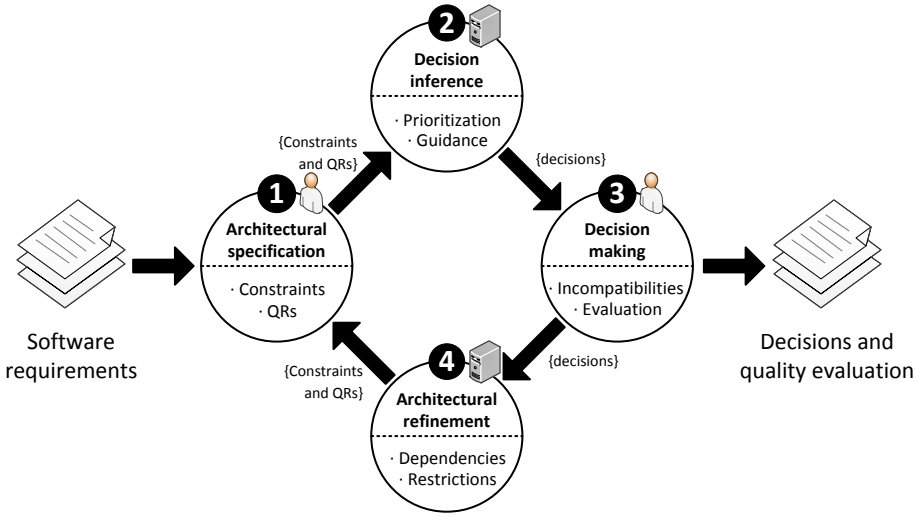

Figure 1. Graphical representation of the Quark method

The method starts from the set of requirements contained in the initial software requirements specification (SRS) defined by the Requirements Engineer, which can include both Quality Requirements (QRs) and constraints. We will first show how the Software Architect, most probably with the support of the Requirements Engineer, can reformulate these requirements inside the tool and then, how these requirements guide the rest of the process until the architect, helped by the tool, decides on a set of decisions to make. During the development of the tool, we have strongly considered the real needs of software architects coming from several empirical studies (one of them reported in the main conference itself). We will show how these considerations are reflected in the tool, such as the balance between degree of freedom and assistance provided by the tool.

The demo will use predefined data (AK, NFRs, ...) already installed in the tool, but we will try during the demo to engage participants by proposing them to add new AK they may have from their own projects, or new NFRs they may consider relevant for their decision-making. Given that a complete architectural design example may take too much time, we will focus on one decision, the DBMS selection (it can be demonstrated in between 2 and 5 minutes, depending on the degree of interaction).

The rest of the document shows how we will orientate the demo for each of the steps in the Quark method, and how we will relate each step with the functionality provided in the tool. We will also exemplify the execution of each step of the method with the aforementioned example.

II. Architectural Specification

The first step in the method is to select the project NFR types that have an impact on the software architecture, and then formalize them in the tool, so that they can be considered in the rest of the steps to infer which decisions to take in order to maximize its fulfillment.

Lets suppose that the Requirements Engineer has produced an SRS document containing among others the three requirements below:

(R1) The software system shall keep information about clients and providers
(R2) The software system shall be developed using OSS whenever possible
(R3) The software system shall be highly reliable

ArchiTech includes two concepts in order to represent these requirements:

- **Quality Attributes**: Represent the quality requirements (QRs) to attain, such as Security, Reliability, etc. Each architectural decision will directly impact over one or more Quality Attributes by helping or damaging them (we have used the levels of impact described in [1]) . One good example of a full set of Quality Attributes can be found in the ISO 9126-1 quality standard. In the provided example, the "Reliability" attribute would be present.
- **Element Properties**: Represent characteristics of architectural elements, such as architectural styles, components or technologies. This concept corresponds to constraints in the Quark process. In the provided example, there would be the "provides persistence" and "license" properties that correspond respectively to R1 [1] and R2.

We will explain how these requirements can be elicited inside ArchiTech using both the "Quality Attributes view" and the "Element Properties view".

On one hand, in the "Quality Attributes view", the architect declares (from the SRS) which goals are *Necessary* for the project, which ones are *Desirable* and which ones are *not relevant* for the current project (see figure 2). Following the example, we would state that "Reliability" is *Necessary*. Moreover, all sub-attributes of "Reliability" are marked also as *Necessary* by default to indicate that all aspects of "Reliabiliy" are important. The architect, however, has freedom to unselect all those attributes that are not relevant for the given project.

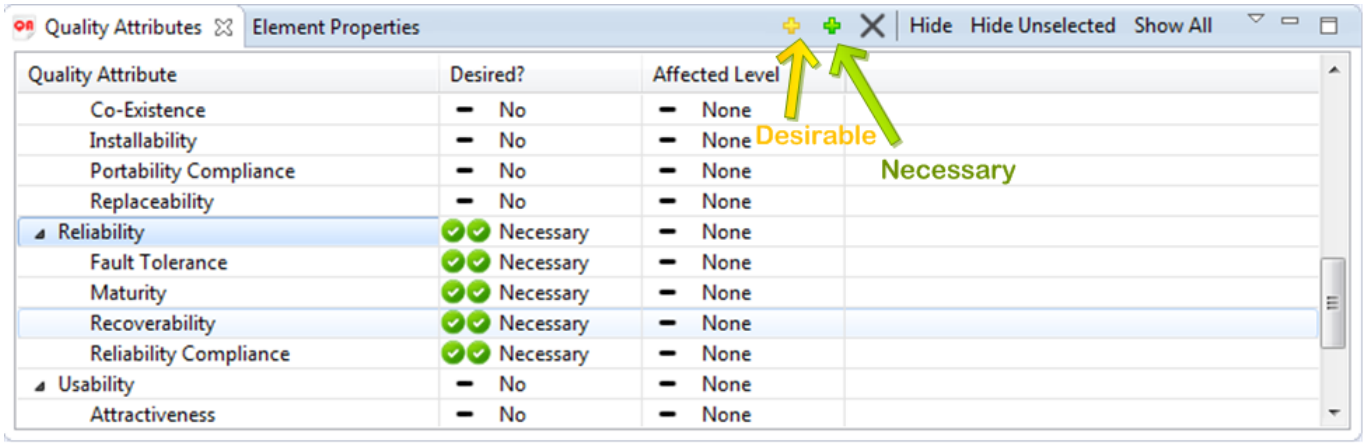

Figure 2. Quality Attributes View

On the other hand, in the "Element Properties view", the architect selects which properties are relevant for the project and which are not, and also sets conditions over the properties according to the requirements. Following the example, the "provides persistence" property would be set to "true", and the "license" to "equals OSS" (see figure 3).

Additionally, the architect has the option to set another type of constraints over architectural elements, in the form of *Use* an element or *Ban* it. For example, we could set a constraint saying that the "Oracle Database" technology is banned (see figure 4).

When showing this step of the process, we will offer the possibility to also explain how knowledge about requirements (both Quality Attributes and Element Properties) can be changed, given that in a real scenario the knowledge base may not be the same for each company, project or user. To show this situation the presenter will take the role of the Domain Expert.

[1] R1 is, in fact, a functional requirement, however it has some implications over NFRs. With respect to ArchiTech, only these implications are relevant.

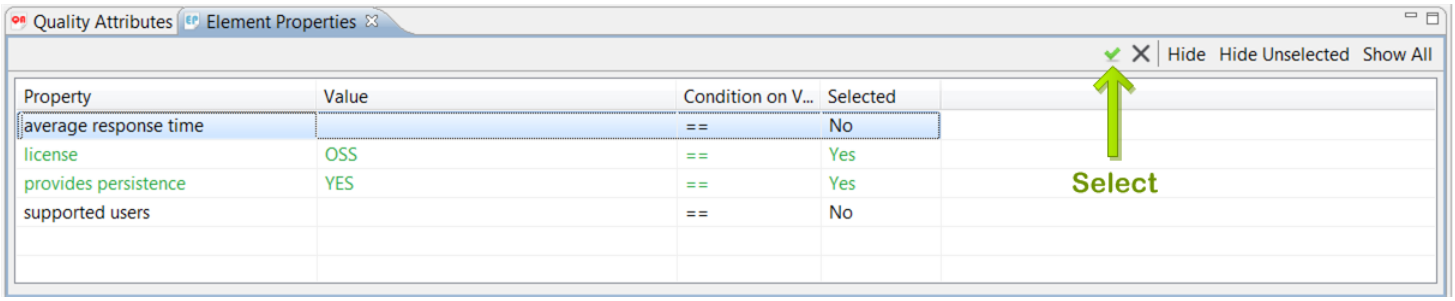

Figure 3. Element Properties View

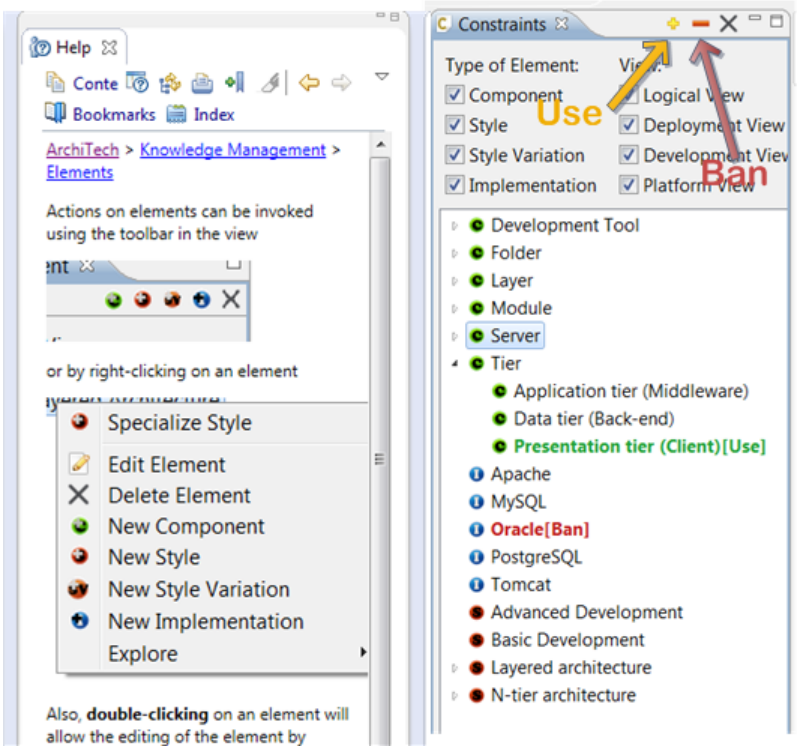

Figure 4. Constrains can be set over elements. The figure also shows one of the help documents offered by the tool

III. Decision Inference

Once we have shown how to elicit the requirements, we will explain how the next step is performed, which corresponds to the calculation of a prioritized list of all decisions. This prioritization is done according to the requirements stated and, also, the detected incompatibilities between decisions and violations of user constraints.

The tool uses the "Decision Maker view" for this purpose (see figure 5). To help the architect in the decisional process, the tool will show the *quality* of each decision, according to how well it fits the requirements, incompatibilities and constraints imposed. Also, it will mark in green the set of decision which the tool considers as the *recommended set*, in the sense that selecting all the green decisions will generate a good solution with the minimum of incompatibilities and constraint violations and the maximum of NFRs fulfilled.

Additionally, as a result of another observation from our empirical studies with software architects, we decided to include a description with each decision that gives insight about how it has been evaluated, so that the architect has more information at hand to make informed decisions. For example we could describe "data replication" as follows: "this decision improves the quality attribute High Performance, which has been selected".

Continuing with the example, the tool will suggest the following decisions:

1) The decision to use MySQL 5. This decision will be marked as "high quality" because it is OSS. MySQL will be

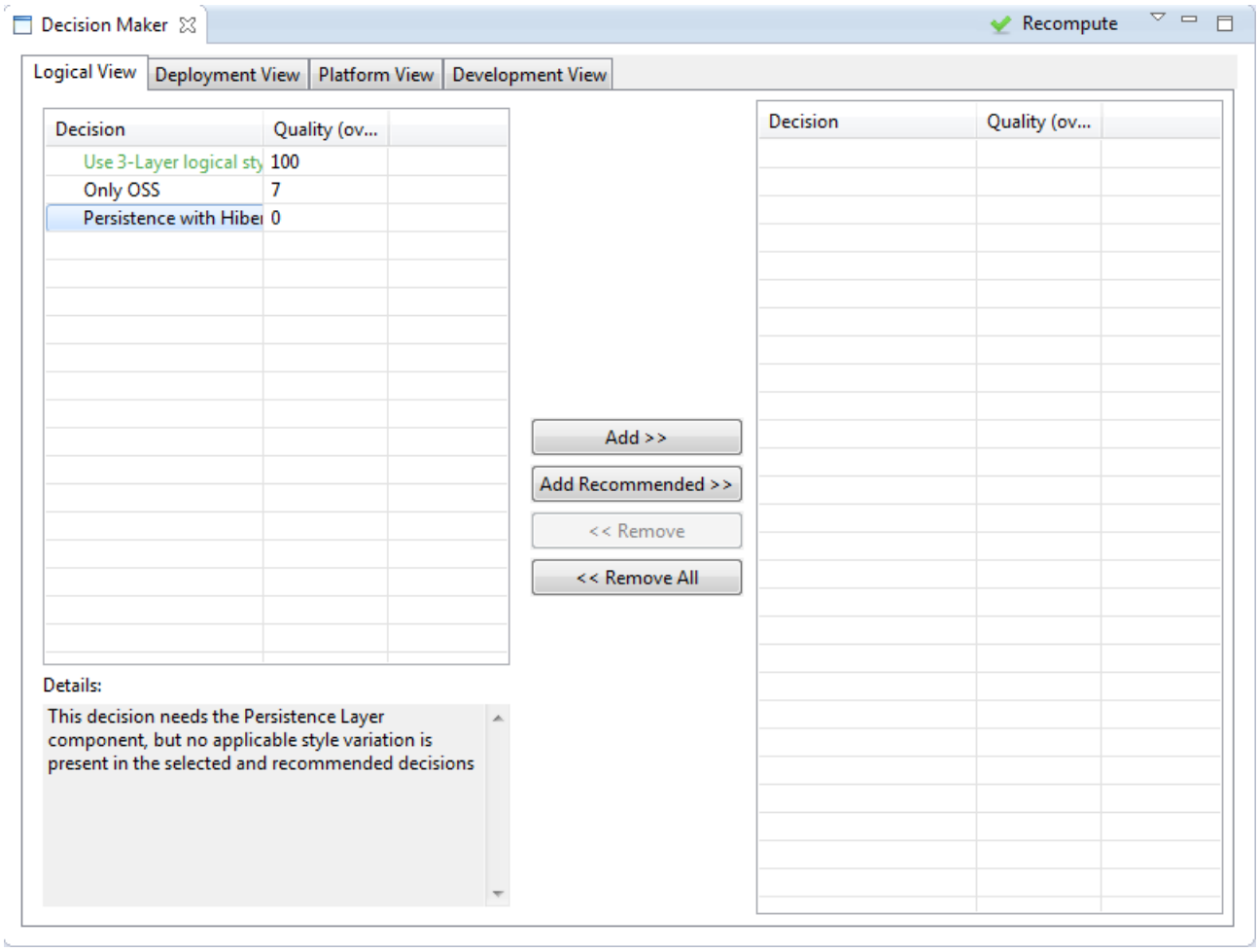

Figure 5. Decisions Maker View. Architectural decisions in the recommended set are shown in green. The details indicate that this decision corresponds to using a component, but the recommended set does not include the style variation to which the component applies.

preferred because it supports more OSS technologies. Also, in the description it will be explained that using MySQL has neutral impact on reliability because ACID compliance depends on the configuration.

2) The decision to use PostgreSQL 8.3. This one will also be marked as high quality because it is OSS. There are few OSS technologies with support for PostgreSQL. Quality will also be increased because using PostgreSQL improves reliability given that it is ACID compliant.
3) The decision to use SQL Server 2005. This last one will be given less quality because it is not OSS and there are few OSS technologies with support for SQL Server. SQL Server will require Windows operating system. However, this technology is ACID compliant and offers backup facilities, which helps reliability.

IV. Decision Making

Once the prioritized list has been generated we will explain how the architect can select which decisions to take, helped by the tool and its recommendations. The tool will inform about the consequences of taking a decision, but the architect will always have the last decision about the adequacy of each decision (there may be other reasons, beyond requirements fulfillment, that have higher priority). At this moment, we will be willing to discuss with participants about the degree of freedom offered and the reasons that may drive an architect to not always select the "best" decisions.

In the proposed example, the architect could decide to use MySQL 5 (the decision with higher priority) as the implementing technology for the DBMS component. But as said before, the architect may prefer to use PostgreSQL, even it is not the top decision, e. g. because she used it in her last project and ended very satisfied. The important point is that the architect is able to make informed decisions, and, eventually, new decisions that were unknown to her/him are taken into consideration.

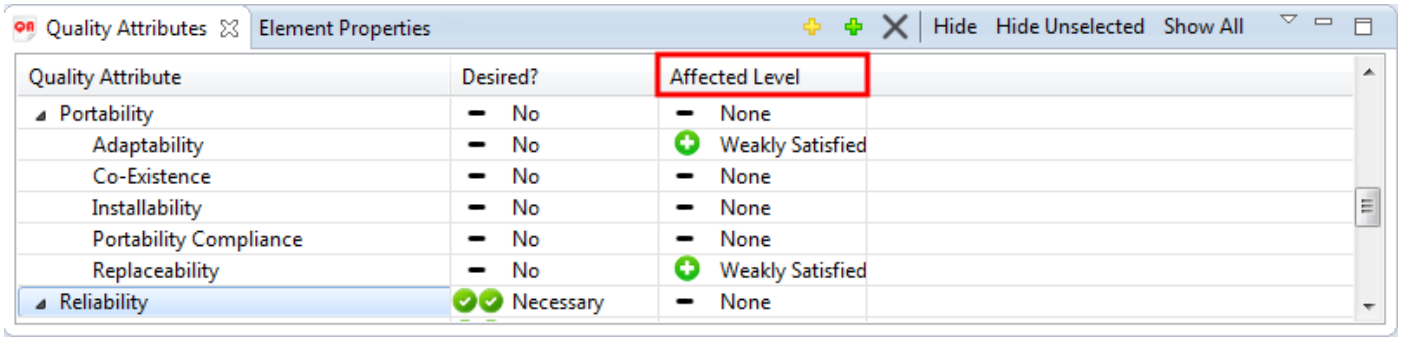

Figure 6. Quality Attributes monitor. The "Affected Level" column shows the level of fulfillment of quality goals when selecting the MySQL Decision.

The tool also includes a "Quality Attributes monitor" (see figure 6) that shows how requirements are fulfilled by the set of selected decisions, so that the architect can have an accurate idea about the impact of making a decision.

Once some decisions have been selected, the architect has to decide whether he wants to finish the process right now, obtaining the set of selected decisions as a result, or if he wants to perform another iteration in order to refine the list of decisions. In case the architect wants to refine the list, he proceeds to the next step.

V. Architectural Refinement

After selecting an initial set of decisions to make, the architect can execute an action to recalculate the *quality* of decisions and the *recommended set*. This will update the list, giving the architect a clear idea about the situation after some decisions have been made. Descriptions over decisions will also be updated to reflect the new status.

Before recalculating, the architect can decide to change requirements if he feels that the decisions he is making are related to some other requirement that he wants to make explicit, or if he wants to change constraints.The architect can execute as many iterations as he feesl necessary until he is satisfied with the set of selected decisions.

Finalizing with the example, the architect could continue with new iterations, where the decision to use MySQL will impact, for example, in the selection of other technologies that are compatible with MySQL.

VI. The Domain Expert role

We have decided to focus on the role of the architect to present the tool, because we feel that this is the most appealing usage and in which participants will have more interest. However, we are also planning on doing some role-playing together with participants in order to illustrate the other users of the tool, namely the Domain Experts, which are the responsible for managing the Architectural Knowledge. As said previously in the document, we will be willing to show how the CRUD module works to any participant interested in it.

VII. More information

A detailed description of the system may be found at its site (www.upc.edu/gessi/architech/index.html), where the current version is available for download. We have prepared a short video about the tool, it can be viewed in the same website. Also, any requests about the tool should be directed to `dameller@essi.upc.edu`

References

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