On the Use of Metamodeling for Relating Requirements and Architectural Design Decisions

Diego Dermeval[1,2], Jaelson Castro[1],
Carla Silva[1], João Pimentel[1]
[1]CIn, Universidade Federal de Pernambuco (UFPE).
Recife, Pernambuco – Brazil
{ddmcm, jbc, ctlls, jhcp}@cin.ufpe.br

Ig Ibert Bittencourt[2], Patrick Brito[2],
Endhe Elias[2], Thyago Tenório[2], Alan Pedro[2]
[2]IC, Universidade Federal de Alagoas (UFAL).
Maceió, Alagoas – Brazil
{ddmcm, ig.ibert, patrick, endhe.elias, ttmo, alanpedro}@ic.ufal.br

ABSTRACT
Requirements models can be used to describe what is expected from a software system. On the other hand, architectural models can describe the structure of a system in terms of its components and connectors. However, these models do not capture the rationale of the decisions made during architectural design. This knowledge is important throughout the maintenance and evolution of the system, as it allows a better understanding of the system as well as the impact of changes on it. In this paper, we consider existing proposals for architectural decisions documentation to define a template for recording the rationale of architectural design decisions. This template is based on a metamodel, which borrows concepts from the NFR Framework to express such rationale. Documenting decisions enables the evaluation of architectural design alternatives when requirements evolve or when new alternatives are devised. Moreover, the metamodel provides a relationship between requirements and architectural design fragments, facilitating the maintenance of traceability between the problem and the solution. We illustrate and discuss the use of this metamodel in the context of Acme architectural models and *i** requirements models.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications

D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Decision tables*

D.2.11 [**Software Engineering**]: Software Architecture

General Terms
Documentation, Design

Keywords
Requirements Engineering; Software Architecture; Architectural Design Decisions;

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'13, March 18-22, 2013, Coimbra, Portugal.
Copyright 2013 ACM 978-1-4503-1656-9/13/03…$10.00.

1. INTRODUCTION
Requirements and architectural models are artifacts generated in connection with two strongly related and intertwined activities of a software development process, respectively, Requirements Engineering (RE) and Architectural Design (AD). Hence, it is critical to establish how these models are interconnected [1]. Indeed, some recent works, such as the STREAM (Strategy for Transition between REquirements and Architectural Models) process [2], present model-driven approaches for generating early architectural models – in Acme [7] – from *i** requirements models [15].

However, specifying software architecture only in terms of architectural models (e.g., Acme models) is not enough. In order to allow a more effective integration between RE and AD activities, the software architecture community highlights the need to treat Architectural Design Decisions (ADD) and their rationale as first class citizens in the software architecture design specification [10] [14].

Explicit mechanisms to link the decision to both the requirements and architectural models are required. Establishing these relationships [6] are essential to answer questions such as "how (well) does the architecture support the satisfaction of this requirement?", "why was this component created?", "what were the architectural design alternatives considered regarding this model fragment?"

In this paper, we present a metamodel that can be used as basis to build an ADD documentation template which can be used, for example, to record the rationale of the decisions taken, the requirements related to a specific decision as well as the alternatives that were considered during the decision-making process.

Our metamodel covers twelve documentation elements (as proposed in [13]) and includes a contribution analysis model (based on the NFR Framework [3]) to analyze how the architectural alternatives contribute to the satisfaction of the system's non-functional requirements. It also relates the requirements to the architectural design fragments responsible to address them.

The expected benefits of using an ADD documentation template are threefold: (i) traceability between requirements models and architectural models is produced during the software lifecycle; (ii) more precise estimation of the impact of requirements and architecture changes; and (iii) better communication between the

stakeholders. Therefore, a reduction in the maintenance and evolution costs is expected.

The remainder of this paper is organized as follows. Section 2 describes the ADD metamodel and its relation with *i** and Acme metamodels. Section 3 presents the ADD documentation template based on the metamodel. Section 4 discusses related works. Finally yet importantly, Section 5 summarizes our work, presents our conclusions and points out future works.

2. ARCHITECTURAL DESIGN DECISIONS METAMODEL

The elements and relations which are present in the ADD metamodel are inspired by the analysis of existing architectural design decisions models conducted by Shahin et al [13]. However, according to other authors (e.g. [9][10][14]) other elements and relations should be considered in the ADD documentation, such as the *Design Fragment* element. Furthermore, it was also necessary to identify binding points between elements used for documentation purposes and the ones used for capturing requirements and architectural information. As an example, in this paper we rely on the *i** [15] and Acme [7] languages to describe requirements and architectural models.

In the sequel, we describe the rationale involved in the definition of the ADD metamodel as well as we explain each of its elements and relations.

2.1 The ADD Documentation Elements

Shahin et al [13] identified 12 major elements addressed by several ADD documentation models in the literature. These elements are: *Decision*, *Constraint*, *Solution*, *Rationale*, *Problem*, *Group*, *Status*, *Dependency*, *Artifact*, *Consequence*, *Stakeholder* and *Phase/Iteration*.

All these elements were considered in the metamodel defined in our work, but, in some cases, we preferred to change their names to make them more appropriate to our context. For example, we changed the name of three elements: *Constraint*, *Solution* and *Problem*. The *Constraint* element was changed to *NFR* because, according to [13], specific kinds of non-functional requirements (NFRs) can be seen as constraints (this will be further explained later). The *Solution* element is equivalent to the concept of architectural alternative so that we preferred to change its name to *Alternative*. The *Problem* element is now called *Functional*, since a problem refers to a functional system requirement to be satisfied [13].

Figure 1 presents the metamodel, including the elements of the ADD metamodel (highlighted in yellow) and the *i** and Acme elements highlighted in red and blue, respectively. Firstly, we will explain the elements and relationships present in the ADD metamodel (highlighted in yellow).

An architectural alternative (***Alternative*** metaclass) must satisfy a set of requirements (***Requirement*** metaclass). However, a system requirement can be one of two types: functional requirement (***Functional*** metaclass) and non-functional requirement (***NFR*** metaclass). Functional requirements can be achieved with a clear level of fulfillment. By contrast, non-functional requirements are of qualitative nature, i.e. they are fulfilled at some partial degree of satisfaction [15]. NFRs are classified in three types [3]: Process NFRs, Product NFRs and External NFRs. The first one refers to product release NFRs, implementation constraints or standards that need to be followed. The second type refers to system quality attributes – for instance, performance, usability, security and so on. The last one refers to legal, economic or interoperability issues.

In Figure 1, we represent the three types of NFRs through the same metaclass. For this reason, the ***NFR*** metaclass has an enumerator attribute named *NFRType* that encompasses the three possible NFR types. Besides that, it has a boolean attribute to indicate the NFR priority.

As we have explained before, a NFR has different degrees of satisfaction. With this in mind, the architectural alternatives (***Alternative*** metaclass) contribute to some degree of satisfaction to non-functional requirements (***NFR*** metaclass). Hence, based on the NFR Framework [3], we propose to represent the contributions from the architectural alternatives to satisfy the NFRs through the ***Contribution*** metaclass. In order to specify the contribution degree to satisfy each NFR, the ***Contribution*** metaclass has an enumerator attribute – named *ContributionType* – indicating five possible contribution degrees: *Make*, *Break*, *Unknown*, *Help* or *Hurt*.

In addition to that, we can see in Figure 1 that any requirement (***Requirement*** metaclass) must be proposed by one or more Stakeholders (***Stakeholder*** metaclass). In its turn, stakeholders can be specialized to system users (***SystemActor***) and the organizational stakeholders (***OrganizationActor***), which can be for example, managers, requirements engineers, software architects and so on. We will see in the next section that a system actor is equivalent to an actor in *i**.

The ***Decision*** metaclass is the key element of the ADD metamodel; it clearly represents the alternative selected in the decision-making process. In the metamodel proposed, we can note that a ***Decision*** is a specialization of an ***Alternative***, since a decision is the architectural alternative selected among several alternatives because it has the best contributions to satisfy the NFRs. The ***Decision*** metaclass also includes three attributes: *group*, *status* and *phaseOrIteration*. The *group* attribute represents the group associated to a decision, i.e., the decisions included in a group are related with each other by certain characteristics as, for example, all decisions related to a system graphical interface could indicate a graphical interface group. The *status* enumerator attribute identifies the decision status. It defines several kinds of status that a decision can assume [10], such as: *Idea*, *Tentative*, *Decided*, *Approved*, *Challenged*, *Rejected* and *Obsolesced*. The last attribute (*phaseOrIteration*) documents the process phase or iteration on which a decision is made (e.g., the software architecture phase).

Moreover, to capture the arguments that lead to the selection of a specific architectural alternative, the ***Decision*** metaclass is associated to the ***Rationale*** metaclass. The ***Rationale*** is composed of a set of architectural alternatives and a set of NFRs associated to each other through contribution links (instances of the ***Contribution*** metaclass). The rationale records that the decision made (i.e., the alternative selected) has the best contributions to satisfy the NFRs. It is worth noting that the purpose of recording the rationale is to provide an unambiguous way to specify the alternatives contributions to satisfy the NFRs; whilst the other ADD documentation models present in the literature, represent this information only using natural language.

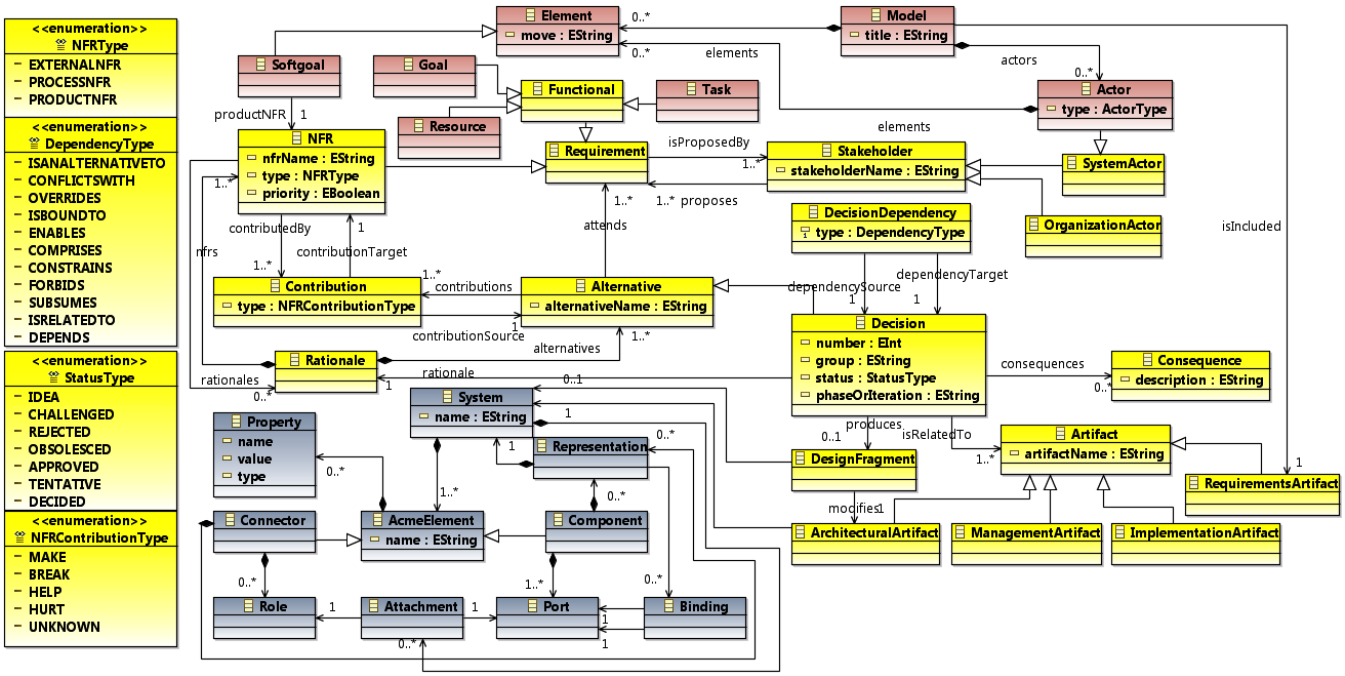


Figure 1. Unified metamodel that relates ADD, *i and Acme elements. Some *i** elements are omitted for clarity.**

Figure 1 also presents the ***DecisionDependency*** metaclass to enable the documentation of the dependencies between different architectural decisions made in the software architecture design. For example, the "Use JavaServerFaces" decision depends on the "Use Java" decision, i.e., the "Use Java" decision *constrains* the "Use JavaServerFaces" decision. According to [10], there are different types of decision dependencies, such as: *Constrains*, *Forbids*, *Enables*, *Subsumes*, *ConflictsWith*, *Overrides*, *Comprises*, *IsAnAlternativeFor*, *IsBoundTo*, *IsRelatedTo* and *Depends*. Thus, the ***DecisionDependency*** metaclass has an enumerator attribute (named *DependencyType*) which represents all these dependency types.

An architectural design decision is also related to software artifacts, so that in Figure 1 we represent this relationship through an association between the ***Decision*** and the ***Artifact*** metaclasses. Software artifacts may have different specializations such as: requirements specifications (***RequirementsArtifact***), architectural artifacts (***ArchitecturalArtifact***), implementation artifacts (***ImplementationArtifact***) or management documents (***ManagementArtifact***).

The ***Consequence*** metaclass in Figure 1 is also mentioned in [13] and corresponds to the concept of *implications* described in [14]. This element depicts all the consequences related to a decision made. For instance, a decision-making can introduce the need to make other decisions, create new requirements or new constraints in the environment, modify existing requirements and so on.

An architectural decision can also be related to a design fragment [9]. The ***DesignFragment*** metaclass represents this concept in the ADD metamodel (Figure 1). A design fragment consists of a set of architectural structure entities which are associated to an architectural decision. In this work, this element is directly related to Acme language constructs that will be further described in Section 2.3. Besides – as we can see in Figure 1 – a design fragment modifies the architectural artifact which contains the architectural model.

2.2 Relating the ADD Metamodel to the *i** Metamodel

After identifying the core elements that compose the ADD metamodel, we examine how they could be related to the language used to describe requirements, in our case *i** [15], which is a popular goal oriented modeling language..

Among the various available *i** metamodels, we considered in this work the one used in a specific *i** modeling tool [11]. However, some modifications were necessary. For example, the intentional elements types (goal, softgoal, task and resource) which originally were specified as an enumerator attribute are now metaclasses. In fact, the need to link specific intentional elements types to specific elements of the ADD metamodel led to the exclusion of the enumerator type and the inclusion of metaclasses to represent the intentional elements. Thus, as can be seen in Figure 1 (see red elements) the intentional element types are represented by the ***Goal***, ***Softgoal***, ***Task*** and ***Resource*** metaclasses. Note that we have defined the *i** elements representing functional requirements as metaclasses (***Goal***, ***Resource*** and ***Task***) specializing the ***Functional*** metaclass.

Moreover, to capture the contributions from the architectural alternatives to the softgoals present in the *i** model, the ***Softgoal*** metaclass is related to the ***NFR*** metaclass of the ADD metamodel – as we can see in Figure 1. Nevertheless, process and external NFRs are not usually modeled in *i** models; they are generally documented as external artifacts (e.g., project plan document). Thus, softgoals elements represent only product NFRs. Therefore, the relation between the NFR and Softgoal metaclasses needs to be constrained by an OCL rule to specify that the **Softgoal** metaclass is uniquely related to the *PRODUCTNFR* type of the ***NFR*** metaclass.

In *i** models, system stakeholders are represented by actors which are related to intentional elements. By contrast, in the ADD metamodel described in the previous section, the ***Stakeholder*** metaclass has two specializations: ***SystemActor*** and ***OrganizationActor***. Note that the ***SystemActor*** metaclass is equivalent to an actor in *i** models. As a result, we specified an inheritance relationship from the ***Actor*** metaclass – present in the *i** metamodel – to the ***SystemActor*** metaclass.

Last but not least, a requirements artifact must contain in its specification, among other information, an *i** model which specifies the system requirements. This way, as it can be seen in Figure 1, the ***RequirementsArtifact*** metaclass has an association relationship with the ***Model*** metaclass.

2.3 Relating the ADD Metamodel to the Acme Metamodel

After the relationships between the architectural design decision elements and the *i** elements are identified, we now need to relate the ADD elements to the architectural model elements of the chosen Architectural Description Language. We opt for Acme [7] because it is a generic ADL that can be used as a common interchange format for architecture design tools and/or as a foundation for developing new architectural design and analysis tools. The blue elements of Figure 1 illustrate the Acme metamodel defined in this work.

In the Acme metamodel, the ***System*** metaclass represents the model which contains all Acme elements (**Acme*Element*** metaclass) that comprise a software architectural structure. An Acme element can be one of two types: ***Connector*** or ***Component***. Any Acme element may have properties (***Property*** metaclass) [7]. Furthermore, *Ports* (**Port** metaclass) and *Roles* (**Role** metaclass) are points of interaction, respectively, between *Components* and *Connectors* – they are bound together through attachments (***Attachment*** metaclass) inside an acme ***System***. Besides, representations (***Representation*** metaclass) allow a component or connector to describe its design in detail by specifying a sub-architecture (***System***) that refines the parent element. The elements within a representation are linked to (external) ports through bindings (***Binding*** metaclass).

After defining the Acme metamodel, we specified its relationship with the architectural design decision documentation metamodel elements. This way, we identified two link points between them. The first relationship links an ADD fragment (***DesignFragment*** metaclass) to an Acme system (***System*** metaclass). It happens because a design fragment can only be an architecture specified according to the Acme metamodel. The second one relates an architectural artifact (***ArchitecturalArtifact*** metaclass) to an Acme system (***System*** metaclass), since an architectural artifact represents the architecture document which contains an Acme model.

The metamodel is specified in a semi-formal way (using the ECORE notation - Eclipse Modeling Framework), which facilitates the development of a tool to support the architectural decisions documentation activities and to trace from requirements to architectural design and vice-versa.

It is worth noting that the ADD metamodel borrows concepts from the NFR Framework [3] in order to express the rationale for a decision. This representation can assist the decision-making process; since it enables to reason about the architectural alternative whose contributions best satisfy the non-functional requirements. The rationale is also of utmost importance when requirements change (or evolve) or when new alternatives need to be considered.

In the next section, we present an ADD documentation template defined according to the elements of the proposed metamodel.

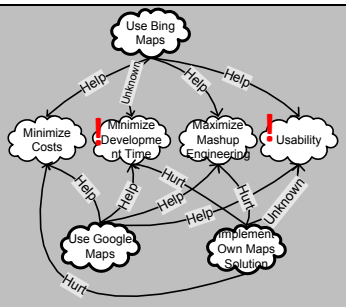
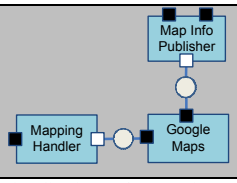
3. ADD Documentation Template

In the sequel, we rely on an example available in the literature (see [2]) to explain the template proposed to record architectural design decisions. BTW is a route-planning system that helps users to define a specific route through advices given by another user.

Table 1 illustrates a documentation template which was based on the metamodel elements described in the previous section. It will be used to record the decision related to the alternative technologies used for visualization of maps and interaction in the BTW system. The *i** and Acme models of the BTW system are suppressed in this paper; but they can be found in [2]. However, the *i** and Acme elements involved in the architectural design decision are recorded in the template and will be explained in the sequel.

The functional requirement addressed by the decision present in the Table 1 is the publication of the information in a map (obtained from the *Information Be Published in Map* goal in the *i** model), which is recorded in the *Functional Requirements* field of the template.

Table 1. ADD Documentation Template

Functional Req.	Information be Published in Map
NFRs	Usability, Minimize Costs, Minimize Development Time, Maximize Mashup Engineering
Stakeholders	Traveller
Alternatives	Use Google Maps; Use Bing Maps; Implement Own Maps Solution
Rationale	
Decision	Use Google Maps
Design Fragment	
Group	Maps Visualization and Interaction Services
Status	APPROVED
Related Artifacts	BTW *i** Model; BTW Acme model
Phase/Iteration	Architectural Design
Consequences	Developers must learn how to use Google Maps API.
Dependencies	--

Regarding the associated non-functional requirements (*NFRs* field of Table 1), the choice of a specific technology can affect the *Usability* product NFR – which is also a softgoal element of the BTW *i** model. Besides, there are several process and external NFRs affected by this decision, such as: *Minimize Costs*, *Minimize Development Time* and *Maximize Mashup Engineering*.

Concerning the *Stakeholders* field, in the BTW *i** model, the *Traveller* actor has a dependency relationship with *Usability* softgoal and, therefore, it is inserted in this field of the ADD template (Table 1).

Three possible architectural alternatives are considered to satisfy the maps visualization and interaction requirement of the BTW system. They were recorded in the template's *Alternatives* field: *Use Google Maps*, *Use Bing Maps* and *Implement Own Maps Solution.*

The contribution analysis from the alternatives to the satisfaction of the NFRs is recorded in the *Rationale* field of the documentation template (see Table 1). Performing the contribution analysis, the *Use Google Maps* alternative contributes positively to all NFRs. The *Use Bing Maps* alternative has a neutral (*Unknown*) contribution to the *Minimize Development Time* and positive contributions to the other NFRs. Last but not least, the *Implement Own Maps Solution* has a neutral contribution to the *Usability* softgoal and negative contributions to the other NFRs.

Once the contribution analysis from the architectural alternatives to the NFRs is concluded, the NFRs are prioritized – in this example, by assigning exclamation marks to them. As it can be seen in the *Rationale* field of Table 1, the *Usability* and *Minimize Development Time* NFRs have the highest priority.

Thus, some analysis can be performed (for example using some of the current available reasoning techniques [8]) to define the best alternative for the given preferences. In the scenario presented, the *Use Google Maps* is the most suitable alternative and, therefore, it is documented in the *Decision* field (see Table 1).

As a consequence, a design fragment for the *Use Google Maps* decision is produced and presented in the *Design Fragment* field of the documentation template. This fragment is composed of an architectural configuration that shows how the *Mapping Handler* and *Map Info Publisher* components of the BTW Acme model [2] use the services of the *Google Maps* component (see Table 1).

Finally, the additional information regarding the decision made is going to be filled in the documentation template. Thus, the *Group* field informs the requirements group addressed by this architectural decision: *Maps Visualization and Interaction Services*. The *Status* field is filled with the *APPROVED* attribute, indicating that the decision was approved. The *Related Artifacts* field records the project artifacts involved in this decision, i.e., the *BTW i* model* and *BTW* Acme *model*. The *Phase/Iteration* field is filled with *Architectural Design*. Regarding the *Consequences* field, the decision made implies that software developers must learn how to use Google Maps services. Finally, it was not identified any dependencies between this decision and others, so that the *Dependencies* field is empty.

The benefits of documenting architectural design decisions using our template become clearer during the system maintenance or evolution. For example, after implementing the BTW system using the chosen technology, it may be noticed that the system performance is not adequate. However, analyzing the architectural decision documentation, it can be seen that the performance quality attribute was not taken into account during the decision-making process (see the *NFRs* field). Hence, some new analysis may be required. Gratefully, the information recorded in the *Alternatives* and *Rationale* fields may help the architect to remember which architectural alternatives were originally considered and how they were related to some NFRs. In face to the new information available, the performance quality attribute needs to be added to the rationale and a new contribution analysis must be performed. This new analysis can lead to the selection of another architectural alternative (e.g., *Use Bing Maps*). Moreover, in the case of an architectural change, the *Design Fragment* field allows the architect to evaluate how the new decision impacts the system architecture.

Documenting a set of architectural design decisions can also be helpful in the self-adaptive systems domain. The information recorded in the documentation template – mainly in the *Rationale* field – can be used to reason about a suitable architectural reconfiguration for accommodating environmental changes both in real and development time.

4. RELATED WORK

The documentation of architectural design decisions has been addressed by several works in the literature. For example, Shahin et al [13] presents a survey on ADD documentation models. It defines four major elements – decision, constraint, solution, rationale – and eight secondary elements – problem, group, status, dependency, artifact, consequence, stakeholder, phase/iteration. In this context, with a few terminology modifications, our metamodel covers all those twelve elements. Besides, based on [9], we also included the design fragment element in the set of entities supported by the metamodel. As a result, in comparison to the nine models presented by that survey, our metamodel encompasses a more comprehensive set of ADD documentation elements which provide more information that can aid the evolution or maintenance of a system.

Furthermore, our approach uses the NFR contribution analysis model, which not only describes and records the rationale, but also may help in the decision-making process. Hence, we are able to model the contributions from architectural alternatives to a set of given NFRs in a far more precise way than the other documentation strategies that rely on natural language to capture these information. Thus, we can benefit from goal model reasoning mechanisms [8], to select the most suitable architectural alternative for a given set of NFRs. In particular, for systems that frequently change (e.g., self-adaptive systems), using a contribution analysis and, therefore, reasoning mechanisms, are key features to enable architectural reconfiguration from an ADD documentation.

Other works have also tried to identify traceability links between requirements and architectural models. For instance, the Goal Centric Traceability (GCT) approach [5] uses the Softgoal Interdependency Graphs (SIGs) – from the NFR Framework [3] – to monitor and to trace the impact of model changes in the software lifecycle. However, compared to our proposal, the GCT approach does not relate requirements elements to specific

architectural fragments and only documents a rather limited set elements involved in the decision-making process, such as: alternatives, rationale and NFRs.

Moreover, both works presented in [12] and [4] propose a tool to capture traceability between software models. However, these approaches do not take into account the ADD documentation, which can establish a trace link from requirements to architectural fragments (and vice-versa) and can also aid the decision-making process during the maintenance or evolution of software systems.

Last but not least, although both the metamodel and the template, presented in this work, were tied to specific requirements and architectural languages, the ADD metamodel is neutral and was specified in generic way. This way, the ADD metamodel can be used by other architectural design approaches by using metamodels of other requirements and architectural languages and relating their elements to the ADD metamodel elements. These relationships can be identified without much effort, since the binding points between the requirements and architectural languages with the ADD metamodel are already identified. For requirements language, it is need to link its elements with the *Functional*, *NFR*, *SystemActor* and *RequirementsArtifact* ADD elements. While, for the architectural language, it is only necessary to link its elements with the *DesignFragment* and *ArchitecturalArtifact* elements.

5. CONCLUSIONS

This paper presented a unified metamodel for architectural design decisions documentation as well as an architectural design decisions documentation.

The unified metamodel specifies, in a semi-formal way (using the ECORE - Eclipse Modeling Framework notation), a more complete set of ADD documentation elements, as it covers the twelve elements described in a comparative study on architectural decisions documentation models [13] and also includes the design fragment element (defined in [9]). To the best of our knowledge, there is no ADD metamodel that includes all those elements.

The unified metamodel also relates requirements (in *i**) and architecture (in Acme) models, so that it can enable to estimate more precisely the impact of requirements change on the architecture. In fact, the metamodel specifies what architecture fragments are affected by the changing requirements.

A possible limitation to our approach may be the considerable effort required to document each architectural decision. However, the expected costs reduction in maintenance and evolution, broadly reported by the literature [1], suggests that the benefits of documenting ADD compensate its overhead.

To support the ADD documentation activities and to alleviate its extra effort, we intend to develop a tool based on the metamodel defined. We also expect to specify OCL rules to constrain, in a more effective way, the relationships between the metaclasses present in the unified metamodel. Finally, we plan to perform a thorough experimentation aiming to evaluate and improve our ADD documentation proposal and apply it in more complex scenarios.

6. ACKNOWLEDGEMENTS

This work has been supported by the Brazilian institutions: CNPq and CAPES; and by the ERC advanced grant 267856 "Lucretius: Foundations for Software Evolution".

7. REFERENCES

[1] Avgeriou, P., Grundy, J., Hall, J.G., Lago, P. and Mistrík, I. 2011. *Relating Software Requirements and Architectures*. Springer.

[2] Castro, J., Lucena, M., Silva, C., Alencar, F., Santos, E. and Pimentel, J. Changing Attitudes Towards the Generation of Architectural Models. *Journal of Systems and Software*. (2012).

[3] Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J. 1999. *Non-Functional Requirements in Software Engineering*. Springer.

[4] Cysneiros, G. and Zisman, A. Traceability and completeness checking for agent-oriented systems. *Proceedings of the 2008 ACM symposium on Applied computing – SAC'08*. (2008), 71.

[5] Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E. and Christina, S. Goal-centric traceability for managing non-functional requirements. *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* (2005), 362-371.

[6] Dermeval, D., Pimentel, J., Silva, C., Castro, J., Santos, E., Guedes, G., Lucena, M. and Finkelstein, A. STREAM-ADD: Supporting the Documentation of Architectural Design Decisions in an Architecture Derivation Process. *Proceedings of the 36th Annual IEEE International Computer Software and Applications Conference* (2012).

[7] Garlan, D., Monroe, R. and Wile, D. Acme: An Architecture Description Interchange Language. *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research* (1997).

[8] Horkoff, J. and Yu, E. Comparison and evaluation of goal-oriented satisfaction analysis techniques. *Requirements Engineering Journal*. (Jan. 2012).

[9] Jansen, A. and Bosch, J. Software architecture as a set of architectural design decisions. *WICSA 2005. 5th Working IEEE/IFIP Conference on* (2005), 109–120.

[10] Kruchten, P., Lago, P. and van Vliet, H. Building up and reasoning about architectural knowledge. *Quality of Software Architectures*. (2006), 43–58.

[11] Malta, Á., Soares, M., Santos, E., Paes, J., Alencar, F. and Castro, J. iStarTool: Modeling Requirements using the *i** Framework. *Proceedings of the Fifth International i* Workshop* (2011), 163-165.

[12] Sardinha, A., Yu, Y., Niu, N. and Rashid, A. EA-tracer. *Proceedings of the 27th Annual ACM Symposium on Applied Computing – SAC'12* (New York, New York, USA, 2012), 1035.

[13] Shahin, M., Liang, P. and Khayyambashi, M.R. Architectural design decision: Existing models and tools. *WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on* (2009), 293–296.

[14] Tyree, J. and Akerman, A. Architecture decisions: Demystifying architecture. *Software, IEEE*. 22, 2 (2005), 19–27.

[15] Yu, E. *Modelling strategic relationships for process reengineering*. Ph.D. Thesis, University of Toronto, Canada, 1995.