2012 IEEE 36th International Conference on Computer Software and Applications

STREAM-ADD – Supporting the Documentation of Architectural Design Decisions in an Architecture Derivation Process

Diego Dermeval[1], João Pimentel[1], Carla Silva[1], Jaelson Castro[1], Emanuel Santos[1], Gabriela Guedes[1]
[1] *Centro de Informática, Universidade Federal de Pernambuco – UFPE*
{ddmcm,jhcp,ctlls,jbc,ebs,ggs}@cin.ufpe.br

Márcia Lucena[2], Anthony Finkelstein[3]
[2] *Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte – UFRN*
marciaj@dimap.ufrn.br
[3] *Department of Computer Science, University College London – UCL*
a.finkelstein@ucl.ac.uk

***Abstract* – Requirements Engineering and Architectural Design are activities of the software development process that are strongly related and intertwined. Thus, providing effective methods of integration between requirements and architecture is an important Software Engineering challenge. In this context, the STREAM process presents a model-driven approach to generate early software architecture models from requirements models. Despite being a systematic derivation approach, STREAM does not support the documentation of architectural decisions and their corresponding rationale. Recent studies in the software architecture community have stressed the need to treat architectural design decisions and their rationale as first class citizens in software architecture specification. In this paper we define an extension of this process, named STREAM-ADD (Strategy for Transition between Requirements and Architectural Models with Architectural Decisions Documentation). This extended process aims to systematize the documentation of architectural decisions by the time they are made and to support the refinement of the architecture according to such decisions. In order to illustrate our approach, it was applied for creating the architecture specification of a route-planning system.**

Keywords – Requirements Engineering; Software Architecture; Architectural Decisions; Software Architecture Documentation; Architectural Knowledge

I. INTRODUCTION

Requirements Engineering (RE) and Architectural Design are initial activities of the software development process strongly related and overlapped [1]. In this context, some efforts have been done to understand the integration between these activities [2, 3, 4]. More specifically, some works present systematic methods to design software architecture from goal-oriented RE approaches [5, 6, 7, 8]. In particular, the STREAM (Strategy for Transition between REquirements and Architectural Models) process [8] presents a model-driven approach for generating initial architectures - in Acme [9] - from *i** requirements models [10]. It consists of the following steps: (i) *Requirements Refactoring*, (ii) *Generate Architectural Model* and (iii) *Refine Architecture*. Horizontal and vertical model-transformation rules were proposed to aid the steps (i) and (ii), respectively. Lastly, in step (iii) the architecture is refined by using architectural styles.

The architecture obtained from the STREAM process is represented in Acme using components and connectors. However, this representation is not sufficient. According to [11], software architecture must be sufficiently abstract to be quickly understood by new staff, concrete enough to serve as a blueprint for the development team and should contain sufficient information to serve as a basis for system analysis.

Moreover, recent studies have emphasized the need to treat architectural design decisions and their rationale as first class citizens in the software architecture specification [12, 13, 14]. Thus, it is necessary to include activities for documenting, capturing and managing architectural decisions in the architectural design process. In fact, performing these activities implies in an extra effort that can be compensated by some benefits obtained later in the software development process [11]. For example, traceability between requirements models and architectural models is produced during the software life cycle [14]. Traceability along with (architectural decision-making) rationale documentation enables estimating more precisely the impact of changes in requirements or architecture and decreasing costs of software maintenance [15]. Besides, documenting the rationale associated to the architectural decision-making process aids the communication between the stakeholders and serves as an auxiliary memory for the architect [14].

Facing the potential benefits of documenting architectural design decision, we propose the STREAM-ADD (Strategy for Transition between Requirements and Architectural Models with Architectural Decisions Documentation) process, an extension of the STREAM process to guide the documentation of architectural decisions and the refinement of the architectural model according to the decisions taken.

The remainder of this paper is organized as follows. Section II briefly describes our running example, the BTW route-planning system, along with the main concepts of *i** language. Section III gives an overview of our approach. Section IV describes the activities of the STREAM-ADD process, applying them to the running

0730-3157/12 $26.00 © 2012 IEEE
DOI 10.1109/COMPSAC.2012.81

example. Section V discusses related works. Finally yet importantly, Section VI summarizes our work, presents our conclusions and points out future works.

II. RUNNING EXAMPLE

This section briefly describes the BTW (By The Way) system, which is going to be used to illustrate our approach. The BTW system [16] consists on a route-planning system that helps users define a specific route through advices given by another user. This information might be filtered to provide only relevant information about the place that a user intends to visit.

Fig. 1 presents an excerpt of the requirements model of the BTW system, represented with the *i** notation [10]. Using *i**, we can describe both the system and its environment in terms of intentional dependencies among actors. In a dependency, an actor, called a *depender,* requires a *dependum* that can be provided by an actor, called *dependee*. The *dependum* may be a goal, a softgoal, a task or a resource. Goals represent the intentions, needs or objectives of an actor. Softgoals are objectives of subjective nature – they are generally used to express non-functional requirements. The tasks represent a way to perform some action to obtain the satisfaction of a goal or a softgoal. The resources denote data, information or a physical resource that an actor may provide or receive.

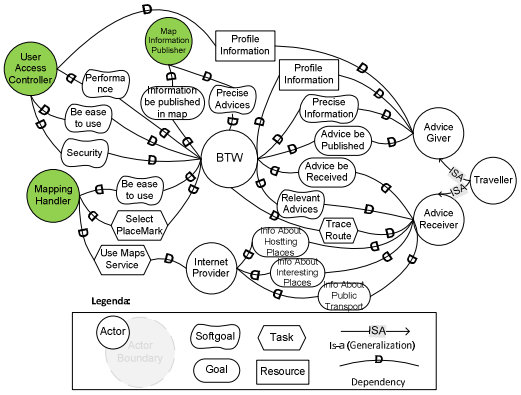


Figure 1. Modular *i** model of the BTW system

In Fig. 1 there is an actor which represents the software system to be developed (*BTW*), actors illustrative of human agents (*Travelers*, that can be *Advice Giver* and *Advice Receiver*), and an actor on behalf of an external system (*Internet Provider*). The software system actor (*BTW*) is also refined in the SR (Strategic Rationale) model by exploiting its internal details to describe how the dependencies are accomplished. For the sake of space, the SR model of BTW system is suppressed in this paper; however it can be seen in [8].

III. STREAM-ADD OVERVIEW

The goal of the original STREAM process is to generate architectural models, in Acme, from requirements models in *i**, by using model transformations [8]. That process has an activity focused on the refinement of architectural models by the application of architectural styles and architectural refinement patterns. However, this activity is not entirely systematized and does not allow the documentation of the rationale involved in the decision-making performed during the architectural design refinement.

To overcome this limitation, we have defined the STREAM-ADD process, which is an extension of the STREAM process richer support to making and documenting architectural decisions. The three activities of this new process are depicted in Fig. 2. The first two activities, *Requirements Refactoring* and *Generate Architectural Model*, were maintained as-is from the original STREAM [8]. The last activity (*Refine Architectural Model With Architectural Decisions*) has been extended to support the documentation of architectural decisions and to make the architectural refinement more systematic.

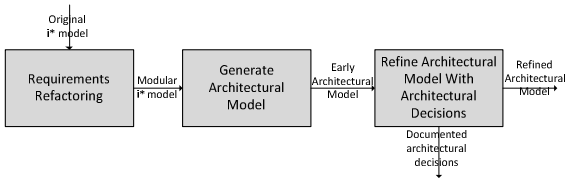


Figure 2. Overview of the STREAM-ADD process

The *Requirements Refactoring* activity is concerned with the modularization of the *i** model. It is a first step towards identifying the system's components. To achieve this, a set of model transformation rules is applied [17]. During the *Generate Architectural Model* activity the requirements model is mapped onto components and connectors of an early architectural model also, based on a set of model transformation rules.

Before presenting the last activity, we apply the two first activities to our running example, the BTW system. The *Requirements Refactoring* activity relies on using a decomposition criterion based on the separation and modularization of elements or concerns that are not strongly related to the application domain. Fig.1 illustrates the *i** model for the BTW system obtained as result of this activity. The highlighted elements in Fig. 1 represent new system actors, which are linked to the BTW itself.

During the *Generate Architectural Model* activity, transformation rules are used to translate the modular *i** model (Fig. 1) onto an early architecture model in Acme [8]. The main elements of Acme are components, connectors, ports, roles, and representations. Acme components represent computational units of a system. Connectors represent and mediate interactions between components. Ports correspond to external interfaces of components. Roles represent external interfaces of connectors. Thus, ports and roles are points of interaction,

respectively, between components and connectors. Representations allow a component, connector, port, or role to describe its design in detail by specifying a sub-architecture that refines the parent element.

The transformation rules provided in this activity define the mapping from *i** actors to Acme components, and from *i** dependencies to Acme connectors and ports. Applying this mapping to our running example (Fig. 1), seven components are generated: *BTW system* (the main actor); *User Access Controller*, *Map Information Publisher* and *Mapping Handler* (actors not related to the application domain); *Advice Giver*, *Advice Receiver* and *Internet Provider*. Fig. 3 shows the early architectural model mapped from the *i** model. More details about the systematic application of these activities can be found in [8].

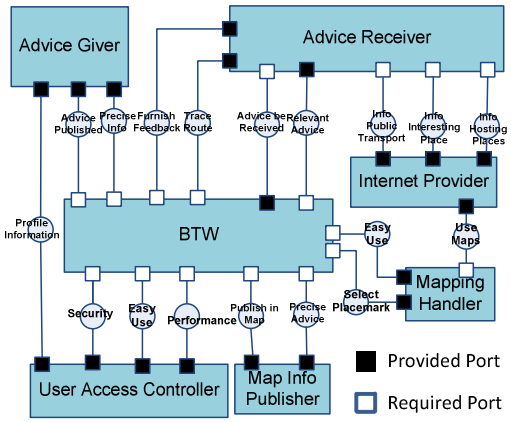


Figure 3. The result of mapping the BTW system model from *i** to Acme

In the following section we will focus on the *Refine Architecture With Architectural Decisions* sub-process, since it is the novel contribution of this present work.

IV. STREAM-ADD – ARCHITECTURAL MODEL REFINEMENT

The *Refine Architectural Model With Architectural Decisions* sub-process aims to refine the early architectural model, mapped from the *i** model, by making and documenting architectural decisions. To do so, we designed a template (based on [12][13][18]) that relates the requirements (*i**) and architecture (Acme) models used in the process. The defined template contains the following fields: *Requirements*, *Stakeholders*, *Non-Functional Requirements*, *Alternative* solutions, *Rationale*, *Decision*, *Design Fragment*, *Group*, *Status*, *Related artifacts*, *Phase/Iteration*, *Consequences*, and *Dependencies*. This template will be used to document architectural decisions made during this sub-process.

Moreover, to define the activities of this sub-process, we analyzed a classification scheme of architectural decisions presented in [13]. Our sub-process supports *Existence Decisions* related to structural aspects and *Executive Decisions* related to technology aspects. Hereafter, we briefly describe each of these architectural decisions types used in the STREAM-ADD process.

Existence Decisions state that some element/artifact will positively be included in the architecture. This concerns both structural and behavioral decisions. Structural decisions lead to the creation of subsystems, layers, partitions and components in some view of the architecture. Behavioral decisions are related to how the elements interact together to provide functionality or to satisfy some non-functional requirement. Executive Decisions do not relate directly to the design elements or their qualities, but are essentially driven by the business environment, and affect the development process, the people, the organization, and to a large extent the choices of technologies and tools, e.g., the programming language to be used.

Fig. 4 presents the activities that constitute the *Refine Architectural Model With Architectural Decisions* sub-process. It is worth noting that we do not intend to guide architectural decision-making in this process, but rather to guide the documentation of these decisions, at the moment they are made, using a specific template. Fig. 5 will show the refinement of the BTW early architectural model after making and documenting one structural and one technology decision.

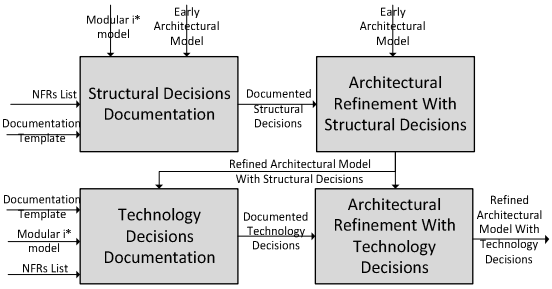


Figure 4. The *Refine Architectural Model With Architectural Decisions* sub-process

A. Structural Decisions Documentation

We consider as structural the following types of decisions: (i) architectural style application; (ii) refinement pattern application to specific non-functional requirements [5]; and (iii) component decomposition. The inputs of this activity are the modular *i** model, the early architectural model, the documentation template and a NFRs list. This activity will be illustrated with an architectural style decision in the BTW system.

In the following we describe a set of steps that must be performed to fill the documentation template for each architectural decision made for a system. For the sake of space, here we are going to focus only on the architectural style decision.

1) Identify Requirements and Stakeholders Addressed by the Decision

In this step, the requirements and stakeholders related to a decision are identified and documented in the template – the former in the *Requirements* and *NFRs*

fields, the latter in the *Stakeholders* fields. The *Requirements* field captures intentional elements of *i** models, i.e., goals, tasks, resources and softgoals that influence the current decision-making. The *Stakeholders* field captures the stakeholders interested in these requirements. They may be actors in *i** model that have dependency links with the identified requirements or actors from the organization that is developing the system, e.g., project manager, client, quality assurance staff, and so on.

In addition, considering the three types of NFRs – Product NFRs, External NFRs and Process NFRs [19] – the first type refers to NFRs that are essentially quality attributes of a product (e.g. performance). Thus, in this paper we consider that Product NFRs and softgoals are semantically equivalent. The second and third types influence architectural alternatives to consider during the decision-making process (e.g. use Free/Libre and Open Source Software technologies). However, these two NFR types are not usually represented in *i** models. Hence, the External and Process NFRs are recorded in *NFR* field of the documentation template, as shown in the next step.

Table I – BTW system NFRs list

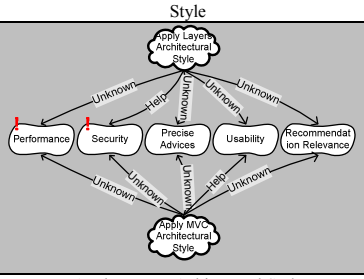
Softgoals	Other NFRs
Usability; Performance, Security; Recommendation Relevance; Precise Advices.	Minimize Costs; Minimize Development Time; Maximize Mashup Engineering.

Table I illustrates the list of NFRs for our running example. This list is an input of the *Structural Decisions Documentation* activity. Softgoals were captured from the *i** model (Fig. 1) and NFRs were identified from other artifacts of the BTW system, especially the project plan [16]. Since we are considering an architectural style decision, and Product NFRs (softgoals) affect the software architecture globally [7], the *Requirements* field of the template (Table II) is filled with all softgoals presented in Table I. The *Stakeholders* field of the template is filled with a non-software actor present in Fig. 1 that has some dependency with these softgoals – in this case, the *Traveller* actor. The "NFRs" field is empty because it was noticed that the system' NFRs do not affect the architectural style decision.

2) Identify Architectural Alternatives

This step is concerned with considering possible architectural alternatives to attend the captured requirements. These alternative solutions will be recorded in the *Alternatives* field of the documentation template. For the case of identifying a suitable set of alternatives for deciding which architectural style to apply, we suggest consulting architectural style catalogues (such as [20]). By analyzing such catalogues, we identified two possible styles that could be applied to the early architectural model of the BTW system (Fig. 3): Model-View-Controller and Layers. So, the field "*Alternatives*" of the template documenting this decision is filled with these architectural alternatives (Table II).

Table II – Documentation Template for the *Apply Layers Architectural Style* Architectural Decision

Requirements	Usability, Performance, Security, Recommendation Relevance, Precise Advices
Stakeholders	Traveller
NFRs	--
Alternatives	Apply Layers Architectural Style; Apply MVC Architectural Style
Rationale	
Decision	Apply Layers Achitectural Style
Design Fragment	
Group	Architectural Style Application
Status	APPROVED
Related Artifacts	BTW *i** Model BTW Early Acme Model
Phase/Iteration	Architectural Design
Consequences	--
Dependencies	--

3) Perform Contribution Analysis of Alternatives

In this step, a contribution analysis is performed, based on [21]. The identified alternatives may contribute (negatively or positively) to the fulfillment of the softgoals and the NFRs documented in the template. Contributions are represented by a link whose source is an architectural alternative and whose target is a softgoal or a NFR. Each link has a label to express different kinds of contributions from the source to the target element: *help* (positive contributions), *hurt* (negative contributions) and *unknown* (neutral contributions).

The contribution analysis performed in this step for our running example was aided by the catalog presented in [20]. By consulting this catalog we have identified that, in general, the layer architectural style has neutral impact on *Performance* and *Usability*, and positive impact on *Security*. Additionally, we considered that the layer architectural style does not impact the *Precise Advices* and *Recommendation Relevance* softgoals. Therefore, this alternative has a neutral impact on these softgoals.

With respect to the MVC architectural style, the catalog indicates that this style has a positive impact on the *Usability* softgoal and a neutral contribution to the *Performance* and *Security* softgoals. The contributions to

Precise Advices and *Recommendation Relevance* sotfgoals are also neutral for this architectural alternative.

Last but not least, architectural alternatives can also impact on NFRs. Since there are no NFRs involved in this decision, there is no contribution from the alternatives to NFRs. The model capturing the contributions of the architectural style alternatives to the Softgoals/NFRs satisfaction are illustrated in the *Rationale* field of the documentation template (Table II). This model will be modified in the next step, by adding prioritization information.

4) Perform Trade-off Analysis of Alternatives

After performing the contribution analysis, the software architect must identify which are the priorities of each softgoal/NFR involved in the analysis. High priority elements are marked with exclamation marks [21]. Then, the architect must perform a trade-off analysis of the alternatives and choose the one that best fulfills the set of softgoals and NFRs as a whole. The trade-off analysis should focus on maximizing the satisfaction of softgoals and NFRs with higher priority. Some reasoning mechanism, such as the one presented in [21], can also be used in order to identify the most suitable architectural alternative.

Once the architectural alternative has been chosen, it should be inserted in the *Decision* field of the documentation template. Besides that, the model in the *Rationale* field of the template must be updated with the prioritization information.

In the BTW system, the softgoals with highest priority are *Performance* and *Security*. Analyzing the architectural alternatives contributions we can see that the *Apply Layers Architectural Style* alternative contributes neutrally to *Performance* softgoal and contributes positively to *Security* softgoal. On the other hand, the *Apply MVC Architectural Style* alternative contributes positively to *Usability* softgoal. Thus, as *Performance* and *Security* softgoals have higher priority than *Usability* softgoal, the selected alternative is *Apply Layers Architectural Style* (documented in the *Decision* field of Table II).

5) Specify Architectural Decision Design Fragment

Once the architectural alternative is already selected and documented in the template, the architect must specify a design fragment associated with the architectural decision. A design fragment is composed of architectural elements in Acme, to be incorporated to the early architectural model during the *Architectural Refinement With Structural Decisions* activity.

A design fragment has different characteristics, depending on the type of the structural architectural decision. For the case of architectural style application, it usually has a global impact in software architecture [7]. Thus, a design fragment produced by an architectural style application decision may modify the architecture as a whole, or a large part of it. In doing so, an architectural style design fragment should be composed of a high level architectural configuration representing the structure of the selected architectural style.

To specify the design fragment associated with the architectural style of BTW, we used the guidelines proposed by [8] to define a three-layer design fragment, whose layers are: *Interface*, *Business* and *Services*. This fragment illustrates how the layers are interconnected (Table II). Each layer is represented as an Acme representation to enable the insertion of other architectural elements in it.

It is important to note that the fragment specified in this step is going to be incorporated to the early architectural model in the *Architectural Refinement with Structural Decisions* activity (Section B).

6) Fill Additional Information

The last step for the documentation of an architectural decision is to fill the additional information in the documentation template. It includes: (i) *Group* – information about the type of architectural decision; (ii) *Status* – the status of the architectural decision (rejected, approved, and so on [13]); (iii) *Related Artifacts* – documents the artifacts related to the documented decision; (iv) *Phase/Iteration* – captures the phase or iteration in which the architectural decision was made; (v) *Consequences* – all consequences that arise when an architectural decision is made must be recorded in this field. For instance, the decision may result in other architectural decisions, or require to create or modify requirements, to create new constraints in the environment, and so on [12]; (vi) *Dependencies* – the dependencies between new architectural decisions with decisions already made are recorded in this field. The identification of decisions dependencies can be aided by the work presented in [13].

Applying this step to the BTW system, the *Group* field is filled with the type of the decision made, that is *Architectural Style Application*. The *Status* field is filled with the attribute *APPROVED* to indicate that this decision has been accepted. With respect to the *Related Artifacts* field, it is filled with the names of the requirements model and the early architectural model of BTW system. The *Phase/Iteration* field is filled with the "Architectural Design" phase. Finally, it was not identified consequences and dependencies for this decision, thus the *Consequences* and *Dependencies* fields of the documentation template are empty. The complete structural architectural documentation template of the *Apply Layers Architectural Style* decision is presented in Table II.

B. Architectural Refinement With Structural Decisions

In this activity, all structural architectural decisions made on the previous activity are used to refine the early architectural model derived from the *i** model. This activity receives as input a set of structural architectural

decisions documented and the early architectural model of the system. The architectural refinement occurs by applying the structural decisions design fragments to the early architectural model.

There is no predefined order to refine the architectural model using the structural decisions design fragments documented in the decisions made. Nevertheless, we propose two general guidelines that might help the selection of an appropriate sequence for the architectural refinement:

Guideline 1. The architectural style decisions should have the highest priority in the architectural refinement sequence. This guideline is motivated by the idea that, in general, architectural styles affect the system architecture in a global way [7].

Guideline 2. Architectural decision design fragments whose architectural configuration is more complex should have higher priority. The complexity of the fragments may be measured according to the number of Acme components, representations and connectors.

After establishing a refinement sequence for the architectural decisions, for each structural decision, the architect must analyze its design fragment, identify the architectural model elements affected by the fragment and perform the refinement. It is important to note that the refinement of architectural models is incremental, so that a decision should be applied to refine the architectural model derived from the refinement according to the previous decision.

In our running example, the early architectural model of the BTW system (Fig. 3) is refined with the *Apply Layers Architectural style* (Table II). The architecture represented in Fig. 5 is the BTW architectural model refined with this architectural decision. For the moment ignore the blue dashed (*Google Maps*) component as it is the result of an activity to be explained below. This refinement was made based on a set of guidelines proposed in [8]. Hence, each component was mapped as follows: (i) *Advice Giver* and *Advice Receiver* components were mapped to the "Interface" layer; (ii) *Internet Provider* and *Map Info Publisher* components were included in the "Services" layer, and; (iii) *BTW*, *User Access Controller*, *Mapping Handler* and *Internet Business* components were mapped to the "Business" layer. Therefore, in order to respect the strict definition of the Layers pattern, *Internet Business* component is introduced in the middle layer to provide internet services to the top layer.

C. Technology Decisions Documentation

Similarly to structural decision, technology decisions need to be documented. Technology decisions should not contain specific details about implementation but decisions that affect the architecture globally or specify how particular structural aspects must be implemented [11]. This is the main reason why the technology architectural decision-making usually occurs after the structural architectural decision-making. In addition, technology architectural decisions also limit the technologies used when implementing the system [13]. Technology decisions may be related to programming language, specific frameworks or APIs (Application Programing Interfaces), component reuse, database management system and so on.

The inputs to this activity are: the modular *i** model (Fig. 1), the architectural model refined with structural decisions in the previous activity (Fig. 5), the system NFRs list (Table I) and the decisions documentation template.

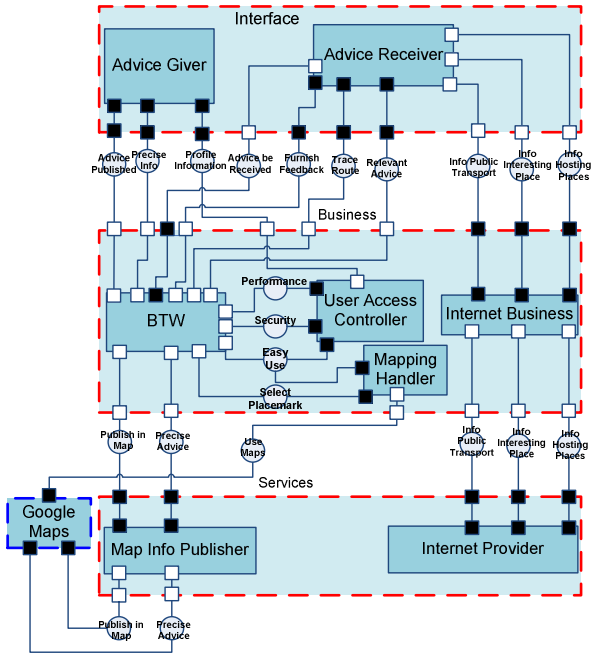


Figure 5. BTW final architectural model

In order to fill the documentation template with technology decisions, we rely on the same steps presented in the *Structural Architectural Decisions* activity, but with some variations. Given the similarities, we will focus on the application of these steps in a decision related to the selection of a technology for maps visualization and interaction to be used in the BTW system.

1) Identify Requirements and Stakeholders Addressed by the Decision

In order to illustrate this activity, we are going to consider the decision made in our running example to determine which maps visualization and interaction technologies available are more appropriate to be used in the BTW system.

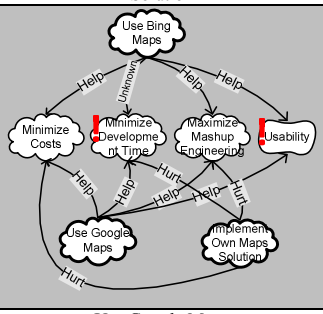
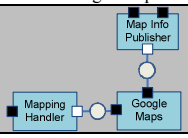
The *Information Be Published in Map* goal, present in the *i** model, is affected by this technology decision, as well as the *Usability* softgoal. In this sense, the *Requirements* field of this decision documentation template (Table III) is filled with these requirements.

Regarding the *Stakeholders* field, as the *Traveller* actor has a dependency relationship with *Usability* softgoal, it is inserted in this field of the documentation template. By analyzing Table I, we have identified that all NFRs are affected by the considered architectural alternatives. This way, the NFRs field of Table III is filled with *Minimize Costs*, *Minimize Development Time* and *Maximize Mashup Engineering*.

2) Identify Architectural Alternatives

In order to illustrate this step to a technology decision, we considered possible alternatives to the maps visualization and interaction technology for the BTW system. In this sense, the available technologies to handle the visualization and interaction of maps that were identified are: *Use Google Maps*, *Use Bing Maps* and *Implement Own Maps Solution.* These alternatives were included in the *Alternatives* field of the documentation template presented in Table III).

Table III – Documentation Template for the *Use Google Maps* Decision

Requirements	Information be Published in Map; Usability
Stakeholders	Travellers
NFRs	Minimize Costs, Minimize Development Time, Maximize Mashup Engineering
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 Modular *i** Model; BTW Acme Model Refined with Structural Decisions
Phase/Iteration	Architectural Design
Consequences	BTW software developers should learn how to use Google Maps API.
Dependencies	--

3) Perform Contribution Analysis of Alternatives

At this point, we analyze the contributions from the alternatives to the satisfaction of the softgoals and NFRs, identified in the previous steps. It was identified that the *Use Google Maps* alternative contributes positively to the *Minimize Costs* NFR because the former is a free solution. This alternative also contributes in a positive way to the *Minimize Development Time* NFR, because it is already implemented and has a good documentation, and also contributes positively to the *Maximize Mashup Engineering* NFR, since it is a service available online. Moreover, *Google Maps* provides an intuitive and easy graphical user interface. Hence it contributes positively to the *Usability* softgoal.

Regarding the *Use Bing Maps* alternative, it contributes positively to the *Minimize Costs* NFR, because it is also a free solution, but it has neutral impact on the *Minimize Development Time* NFR, because its documentation is not satisfactory. Moreover, this alternative contributes in a positive way to the *Maximize Mashup Engineering* NFR, because it is available as an online service as well. At last, the *Use Bing Maps* alternative provides an intuitive and friendly graphical user interface, so that it contributes positively to the *Usability* softgoal. Finally, the *Implement Own Maps Solution* contributes negatively to the *Minimize Costs* NFR, since it is necessary to spend time and people to the development of this solution. It also contributes negatively to the *Minimize Development Time* NFR, because it needs to be implemented from scratch. This alternative also contributes negatively to the *Maximize Mashup Engineering* NFR, because this alternative does not use online services. Moreover, it has an unknown impact on the *Usability* softgoal, since its usability can only be evaluated after its development starts. Please note that in the *Rationale* field of the Table III NFRs and alternatives are both graphically represented as clouds; however, architectural alternatives are represented as clouds with a thicker border.

4) Perform Trade-off Analysis of Alternatives

This step is concerned with choosing the most suitable alternative regarding the fulfillment of softgoals and NFRs. For doing this, it is required to define the priorities of softgoals and NFRs. In our running example, the *Usability* softgoal and the *Minimize Development Time* NFR have the highest priority. Observing the model present in the *Rationale* field of Table III, we conclude that the *Implement Own Maps Solution* alternative is dismissed because it does not contribute at all to the NFRs/Softgoals. We also can see that the *Use Google Maps* alternative contributes positively to the *Minimize Development Time* NFR, whereas the *Use Bing Maps* alternative contributes neutrally to the same NFR. Regarding the *Usability* softgoal, both remaining alternatives have positive impact on its fulfillment. Thus, we conclude that *Use Google Maps* is the most suitable alternative and it is included in the *Decision* field of the template presented in Table III, whereas the model used to perform this analysis is updated in the *Rationale* field.

5) Specify Architectural Decision Design Fragment

In this step, the design fragment associated with the selected architectural alternative is specified. In general, an architect needs to analyze if an architectural decision produces a design fragment that can affect/modify the structure of the software architecture. However,

depending on the technology decision type, a design fragment can be produced in different ways, or even not be produced at all. Therefore, we do not propose in this step general guidelines to aid the specification of design fragments. In this case, the architect is in charge to specify the design fragment according to each architectural decision made.

The design fragment produced for the *Use Google Maps* decision is presented in the *Design Fragment* field of the documentation template (Table III). This fragment is composed of an architectural configuration that shows how the *Mapping Handler* and *Map Info Publisher* components of the BTW system (previously responsible for addressing the requirements affected by this decision) use the services of the *Google Maps* component.

6) Fill Additional Information

Finally, in this step, the additional information regarding the technology decision made is filled in the documentation template. Thus, the *Group* field is filled with the requirements group addressed by this architectural decision: *Maps Visualization and Interaction Services*. The *Status* field is filled with the *APPROVED* attribute. The *Related Artifacts* field is filled with the project artifacts involved in this decision: *BTW modular i* model* and *BTW* Acme *model refined with structural decisions*. The *Phase/Iteration* field is filled with *Architectural Design*. Regarding the *Consequences* field, the decision taken implies that software developers have to 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.

D. Architecture Refinement With Technology Decisions

After making and documenting all technology decisions, they are used to refine the architectural model that was previously refined with structural decisions. To perform this refinement, this activity receives as input a set of technology decisions documented in a template and the architectural model obtained from the *Architectural Refinement with Structural Decisions* activity. It is worth noting that only technology decisions that produce a design fragment can be used to refine the architectural model.

As commented before, there is no predefined sequence to apply the architectural decisions in the architectural model refinement. Hence, the architect is in charge of defining this sequence. However, the architect can rely on the Guideline 2 defined in Section B to determine the proper refinement sequence. After doing this, the design fragment of each documented decision has to be analyzed, to identify which parts of the architectural model are going to be affected by them. Then, the refinement can be performed.

As a result, the dashed blue component in Fig. 5 represents the *Google Maps* component that was inserted in the architectural model. The *Map Info Publisher* and *Mapping Handler* components delegate the responsibility of providing maps visualization and interaction services to the *Google Maps* component.

V. DISCUSSION

The original Strategy for Transition between REquirements models and Architectural Models (STREAM) [8] is a systematic process aimed to define architectures through a model-driven approach. It strongly relies on transformation rules to incrementally evolve a requirements model in *i** onto an architectural model. If necessary, this architecture is further detailed through architectural style or refinement patterns application. However, even though STREAM offers a systematic way for deriving architectural models that takes benefits of using goal-oriented models and models transformations, it does not support the documentation of architectural decisions and their rationale.

In order to address this limitation, we defined in this work the STREAM-ADD process. This extended process brings to the original STREAM the benefits of documenting architectural design decisions. Even though we did not perform a thorough evaluation of our process, the results reported by the software architecture literature suggests that documenting architectural decisions compensates by far the extra architectural design effort required to document architectural decisions in this process.

Moreover, our process also allows the early architectural model derived by the application of the two first STREAM activities to be better refined through architectural decisions. The architectural model refinements with structural and technology decisions activities of STREAM-ADD allows the specification of a more complete components-and- connectors architectural view than the original STREAM process.

It is important to note that the new STREAM-ADD process does not aim to systematize the actual decision-making. Instead, it provides a set of activities that aid the architect in documenting these architectural decisions. In other words, a software architect can make every architectural decision that she considers necessary but she needs to document the rationale that lead her to make the decision and how this decision affects requirements and architectural models. Nonetheless, as a positive side effect, the documentation steps provide some guidance to the software architecture regarding the decision-making, as it requires the documentation of some elements that could be otherwise overlooked.

In the interest of clarity, the proposed process was sequentially presented, like a waterfall model. However, in fact it was conceived as an iterative and incremental process. For instance, considering the two first activities of the Architectural Model Refinement sub-process in the context of an industrial project, it is more likely that some structural decisions will be documented, and then applied

to the model. Next, other new structural decisions will be documented, and also applied to the model, so on and so forth.

It is also worth noting that, as in the original STREAM approach, the outcome the process depends on the quality of the input artifacts (e.g., *i** models). Thus, if poor quality *i** models are used it is likely that the resulting architectural model derived will also be of poor quality.

In the following sub-section we discuss our approach in comparison with other approaches for architecture derivation from requirements models and for architectural decisions documentation.

A. Related Work

Different strategies, techniques and models can be used when deriving architectures from requirements models. The SIRA approach [6], for instance, uses *i** models as input, resulting an architectural model using organizational architectural styles. The work by Chung et al. [22] has some similarities with our process, but it lacks the documentation activities that are essential in our proposal. The *UML Components* process [23] proposes a set of activities in order to derive a UML component diagram from use cases models and from a business conceptual model. However, it derives a limited architecture (always in fours layers) and does not allow the structural and technological decision-making. Other approaches that use *i** modeling language as the starting point of software specification, such as PRIM [24], do not support a systematic transition from requirements specifications to architectural design description.

Silva et al. [25] proposes a set of mapping rules between an aspectual goal model and an aspectual version of Acme. However, it does not support any kind of architectural decision. The CBSP approach [26] creates intermediate models to facilitate the development of architectures from requirements. It lacks proper support for making and documenting technology decisions. Galster et al. [27] defines requirements for architecture derivation processes, based on a review of approaches presented in the literature. Our approach does not properly satisfy the following requirements: underlying formal approach; manage different architectural views; reuse of architectural knowledge; handle different modeling notations. In particular, we plan to tackle the issue related to architectural views in future work.

Architectural design decisions documentation plays a key role in our approach and has been the focus of several studies presented in the literature. Shahin et al. [18] describes a survey on architectural decisions documentation models, which vary on their degree of formality from textual templates to well-defined metamodels. It defines four major elements – decision, constraint, solution, rationale – and eight minor elements – problem, group, status, dependency, artifact, consequence, stakeholder, phase/iteration. All these twelve elements are included in our template. A novel contribution of our paper is the use of goal-based requirements model to drive documentation activities. Moreover, our approach relies on NFR-based models to define the decision rationale, which not only describe the rationale but also may help in the decision-making process. It is worth noting that architectural design decisions documentation is the foundation of architectural knowledge management area, see [3] for some works in this research line.

VI. CONCLUSIONS

This paper presented STREAM-ADD, a process that extends the original STREAM architectural derivation process in order to create a more complete architectural model by encompassing both the architectural models and the architectural decisions. The first and second activity were maintained as-is from the original STREAM process. In the third activity, the early architectural model generated by models transformations is refined with further architectural decisions.

The third activity was extended by defining a sub-process composed of four sub-activities: the first two are related to the structural architecture, whereas the last two are related to technology decisions-. In order to support the realization of these sub-activities, we presented a set of steps and general guidelines for each activity.

As future work, we expect to develop tool support for our approach. Such a tool would need to support all documentation and modeling activities of the process. We also intend to apply the STREAM-ADD process in the architecture specification of more complex systems, especially in an industrial context.

Our approach still needs to be extended in order to support the systematic specification of other architectural views specification in systematic way, including behavior characteristics of the architecture. Finally, we acknowledge that a thorough experimentation must be performed in order to evaluate and improve the STREAM-ADD process.

ACKNOWLEDGMENTS

This work has been supported by the Brazilian institutions: Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Coordenacão de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

REFERENCES

[1] B. Nuseibeh, "Weaving together requirements and architectures," *Computer*, vol. 34, no. 3, pp. 115-119, Mar. 2001.

[2] J. Castro and J. Kramer, "From software requirements to architectures," in *Proceedings of the 23rd International Conference on Software Engineering*, 2001, p. 764--765.

[3] P. Lago, P. Avgeriou, and P. Kruchten, "Sixth international workshop on SHAring and Reusing architectural Knowledge," *Proceedings of the 33rd International Conference on Software Engineering, ICSE*, 2011.

[4] P. Avgeriou, J. Grundy, J. G. Hall, P. Lago, and I. Mistrík, *Relating Software Requirements and Architectures*, vol. 152, no. 4. Springer, 2011, pp. 141-142.

[5] A. Van Lamsweerde, "From system goals to software architecture," *Formal Methods for Software Architectures*, pp. 25–43, 2003.

[6] L. Bastos and J. Castro, "From requirements to multi-agent architecture using organisational concepts," *SELMAS 05 Proceedings of the fourth international workshop on Software engineering for largescale multiagent systems*, vol. 30, no. 4. ACM Press, pp. 1-7, 2005.

[7] M. Lucena, "STREAM□: A systematic process to derive architectural Models from requirements models". Thesis, Universidade Federal de Pernambuco, 2010.

[8] J. Castro, M. Lucena, C. Silva, F. Alencar, E. Santos, and J. Pimentel, "Changing attitudes towards the generation of architectural models," *Journal of Systems and Software*, vol. 85, no. 3, pp. 463-479, Mar. 2012.

[9] D. Garlan, R. Monroe, and D. Wile, "Acme: an architecture description interchange language," in *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, 1997, p. 7.

[10] E. Yu, "Modelling strategic relationships for process reengineering". Thesis, University of Toronto, 1995.

[11] D. Garlan et al., *Documenting software architectures: views and beyond*, 2nd ed. Addison-Wesley Professional, 2010.

[12] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," *Software, IEEE*, vol. 22, no. 2, pp. 19–27, 2005.

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

[14] P. Kruchten, R. Capilla, and J. C. Dueñas, "The Decision View's Role in Software Architecture Practice," *IEEE Software*, vol. 26, no. 2, pp. 36-42, Mar. 2009.

[15] L. Bratthall, E. Johansson, and B. Regnell, "Is a design rationale vital when predicting change impact?-A controlled experiment on software architecture evolution," in *Product Focused Software Process Improvement*, 2000, vol. 1840, pp. 126-139.

[16] J. Pimentel, C. Borba, and L. Xavier, "BTW: if you go, my advice to you Project," 2009. [Online]. Available: https://jaqueira.cin.ufpe.br/jhcp/docs/. [Accessed: 18-Nov-2011].

[17] J. Pimentel, M. Lucena, J. Castro, C. Silva, E. Santos, F. Alencar, "Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach", in *Requirements Engineering Journal*, published online, 2011.

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

[19] I. Sommerville and G. Kotonya, *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, Inc., 1998.

[20] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*, vol. Vol. 1. Wiley, 1996, p. 476.

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

[22] Lawrence Chung, Sam Supakkul, Nary Subramanian, José Luis Garrido, Manuel Noguera, Maria V. Hurtado, María Luisa Rodríguez, and Kawtar Benghazi. *Goal-Oriented Software Architecting.* P. Avgeriou et al. (eds.), Relating Software Requirements and Architectures, DOI 10.1007/978-3-642-21001-3_7.

[23] J. Cheesman and J. Daniels, *UML components: a simple process for specifying component-based software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[24] Grau G, Franch X, Ávila S (2006) J-PRiM: A Java Tool for a Process Reengineering *i** Methodology. In: Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE□06). Minneapolis, USA, pp. 359–360. doi:10.1109/RE.2006.36

[25] Silva L, Batista T., Garcia A, Medeiros L, Minora L (2007) *On the symbiosis of aspect-oriented requirements and architectural descriptions.* Early Aspects: Current Challenges and Future Directions - LNCS 4765/2007:75-93. doi:10.1007/978-3-540-76811-1_5

[26] Paul Grünbacher, Alexander Egyed, Nenad Medvidovic. *Reconciling software requirements and architectures with intermediate models.* SoSyM 2003. DOI: 10.1007/s10270-003-0038-6

[27] M. Galster, A. Eberlein, and M. Moussavi, "Transition from Requirements to Architecture: A Review and Future Perspective," *Software Engineering Artificial Intelligence Networking and Parallel Distributed Computing 2006 SNPD 2006 Seventh ACIS International Conference on*, vol. 0, pp. 9-16, 2006.