F-STREAM: A Flexible Process for Deriving Architectures from Requirements Models

Jaelson Castro[1], João Pimentel[1,2], Márcia Lucena[3], Emanuel Santos[1], Diego Dermeval[1]

[1] Universidade Federal de Pernambuco – UFPE, Centro de Informática, Recife, Brazil
[2] Universitat Politècnica de Catalunya, Omega–122, CP: 08034, Barcelona, Spain
[3] Universidade Federal do Rio Grande do Norte – UFRN, DIMAp, Natal, Brazil
{jbc, jhcp, ebs, ddmcm}@cin.ufpe.br, marciaj@dimap.ufrn.br

Abstract. Some quality attributes are known to have an impact on the overall architecture of a system, requiring to be properly handled from the early stages of the software development. This led to the creation of different and unrelated approaches to handle specific attributes, such as security, performance, adaptability, etc. The challenge is to propose a flexible approach that could be configured to address multiple attributes of interest, promoting the reuse of best practices and reduction of development costs. We advocate the use of Software Product Line (SPL) principles to manage and customize variability in software processes targeted for the generation of architectural models from requirements models. Hence, in this paper we propose F-STREAM, a flexible and systematic process to derive architecture models from requirements. We define a common core process, its variation and extension points. The definition of this process was performed based on a survey of the existing approaches. As example, we instantiate a process for adaptive systems.

Keywords: Model-driven architectures; architecture derivation; non-functional requirements and architectures.

1 Introduction

It is well known that some kinds of systems present quality attributes, also called non-functional requirements (NFRs), that have an impact on the architecture of the system as a whole. These requirements must be elicited, analyzed and properly handled in the early requirements phase. Otherwise, it would compromise the software architectural design quality. Moreover, some NFRs demand specific approaches and mechanisms to enable their achievement. For instance, it is unlikely that an approach to develop mobile systems (portability) is also suitable to develop multi-server scalable systems (scalability).

The STREAM process [11] [13] allows a model-based systematic derivation of architectures—in Acme [8]—from requirements models—in *i** [19]. However, it does not properly address non-functional requirements (NFR). Instead of trying to define an entirely new process, we envision the integration of the original STREAM with already existing approaches for handling specific NFRs. With this purpose, in this

paper we propose a Flexible Strategy for Transition between REquirements models and Architectural Models (F-STREAM). To provide such flexibility, we are going to use the concepts of variability management—i.e., define the common core (commonalities) and the variations (variabilities) of the process. Therefore, we are going to define a base process that can be extended through integration with already existing approaches that are tailored to handle specific NFRs—in contrast with other approaches that handle generic NFRs [2] [3] [4] [6]. The integration of F-STREAM with a specific approach is called an F-STREAM instance.

In order to identify the commonalities and variabilities for the F-STREAM process, we performed a survey on different goal-based approaches that address these specific NFRs. Table 1 gives a brief description of some of the analyzed approaches. Usually the approaches require the use of an extended goal model notation, to include information that is not present on the original goal model. Some of the approaches also provide reasoning algorithms, specific components or a reference architecture. These are the key characteristics that will be considered on our approach.

Table 1. Some of the surveyed *i**-based approaches that target specific NFRs.

Approach	Description
For security [18]	Extends goal models by defining context annotations, preconditions and effects; Use anti-goal models; Provides a diagnostic component.
For adaptability [7]	Extends goal models with context annotations; Provides a self-configuring component.
For data warehouses [9]	Extends goal models by defining facts, attributes, dimensions and measures.
For software product lines [1711]	Extends goal models to express cardinality; Provides heuristics to elicit variability information.

The remainder of this paper is organized as follows. Section 2 describes the common core of the process, whilst Section 3 presents how the process might vary to accommodate the specific approaches, in terms of variation points and extension points. As a case study, we instantiated the F-STREAM process by integrating it with an approach that tackle the adaptability NFR (Section 4). The final remarks and future works are presented in Section 5.

2 The F-STREAM Common Core Process

The common core of the F-STREAM process is the subset of the original STREAM process that is generic enough to be used with different complementary approaches, requiring at most minimal modifications. This common core is able to generate architectural models from requirements models, with an incremental and models-transformation based approach.

For expressing the requirements models and architectural models we use, respectively, *i* (iStar)* [19] and Acme [8], since the original STREAM process also use these languages. Goal modeling is a an widespread approach in the academy to express requirements, such as in the Tropos method [14]. *i** defines goal-based models to describe both the system and its environment in terms of intentional dependencies among strategic actors [12] (*who*). There are two different diagrams, or

views, of an *i** model: the Strategic Dependency (SD) view presents only the actors and the dependency links amongst them, whilst the Strategic Rationale (SR) view shows the internal details of each actor. Within a SR diagram is defined *why* each dependency exists and *how* they are going to be satisfied.

There is a variety of Architectural Description Languages (ADLs), each one with its set of tools and techniques. Acme ADL was proposed with the primary goal of providing an interchange format for tools and environments for architectural development. Therefore, it can be easily translated into an ADL of choice.

Based on the survey on specific goal-based approaches, we defined a core set of activities that may be carried out with any of the approaches. Fig. 1 presents the process diagram of this core set, which is the F-STREAM process common core. In the next sub-sections these activities will be further detailed.

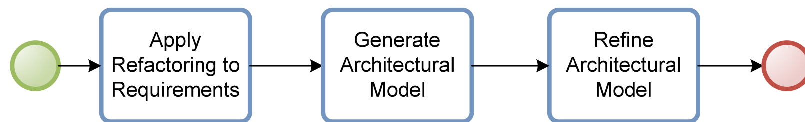


Fig. 1. Common core of the F-STREAM process

2.1 Apply Refactoring to Requirements

The aim of this activity is to modify the organization of the *i** diagram, splitting the responsibilities of the software actor into smaller actors. This allows the delegation of different issues of a problem, initially concentrated into a single actor, to new actors so that it is possible to deal with each of them separately. The decomposition of the main software actor into smaller actors has the objective of modularizing *i** models by delegating responsibilities of the software actor to other (new) software actors that are dedicated to a particular concern. The decomposition criterion is based on the separation and modularization of elements or concerns that are not strongly related to the application domain. Usual examples of this kind of domain independent elements are persistency, security, statistics, etc.

In order to assist the requirements engineer to identify the elements that can be extracted from the software actor, we use the following heuristics. H1: Search for internal elements in the software actor that are independent of the application domain. H2: Check whether these elements can be moved from the software actor to another software actor without compromising the behavior and the understandability of the internal details of the actor. H3: Verify whether these elements can be reused in different domains.

After the identification of the movable elements, they will be transferred to other actors, through horizontal transformation rules defined in previous work [13].

2.2 Generate Architectural Model

In this step, transformation rules will be used to translate the *i** requirements model onto an early architecture model in Acme. Since these transformations have different

source and target languages, they are exogenous, or translation transformations. They are also vertical transformations, since the source and target models have different level of abstractions.

In summary, these transformations define the mapping from *i** actors to Acme components, and from *i** dependencies to Acme connectors and ports. A component in software architecture is a unit of computation or a data store having a set of interaction points (ports) to interact with external world. An actor in *i** is an active entity that carries out actions to achieve goals by exercising its knowhow. Thus, an actor representing the software establishes a correspondence with modules or components [10]. In addition, an actor may have as many interactions points as needed. Hence, an actor in *i** can be represented in terms of a component in Acme.

Thus, the first vertical transformation rule is a straightforward one, that maps *i** actors onto Acme components. Further details of this component will be added later during the mapping of *i** dependencies. In *i**, a dependency describes an agreement between two actors playing the roles of depender and dependee, respectively [5]. In Acme, connectors mediate the communication and coordination activities among components. Thus, we can represent a dependency as an Acme connector. The complete transformation rules for mapping the *i** model to an Acme architecture are described in [11].

2.3 Refine Architectural Model

Having produced an early architectural design solution, we can now refine it. This activity relies on some commonly used architectural patterns, such as Model View Control (MVC), Layers and Client-Server. The components of early architectural model will be manually refined by the architect based on his/her expertise by applying these patterns. These patterns are analyzed to identify the similarity with the early architectural model. The refinement process follows three steps.

The first step is to analyze the components of the early architectural model and compare them with the elements of the pattern observing the similarities of their roles and responsibilities. The most similar architectural pattern can be used to structure the early architectural model. For instance, if the roles of the architectural model components are organized hierarchically they can be associated with the Layers pattern, then the components of a layer will communicate just with the components of the layer next to them. Thus, a new version of architectural model is generated. Since the components of architectural model have been related to components of an architectural pattern, also their connectors need be associated. Therefore, the second step is to analyze the connectors of the generated architectural model and compare them with the connectors of architectural. Applying the architectural patterns during the refinement can incorporate the qualities associated with the pattern to the refined model. However, if some component of the pattern is missing in the architectural model it needs to be included. The third step is to introduce new components to adjust the architectural model to the pattern, if any is missing. Since the architectural design can be iterative, components can be added anytime. Moreover, refining the architectural models with patterns to address system qualities (i.e., NFRs) is a common practice, existing several tactics to this end documented in the literature.

3 Variation and Extension Points of the F-STREAM Process

In this section we are going to present the Flexible STREAM process, which consists of the common core presented in Section 2 enriched with variability information. In a business process, a variation point is the place on which a variation occurs, and each possible alternative for a variation point is a variant [15]. In order to describe the variation points in the process without defining which are the variants themselves we are going to use the notation proposed by Schnieders and Puhlmann [16], which defines a set of stereotypes and association links for expressing variability in Business Process Modeling Notation (BPMN) diagrams.

The top of Fig. 2 shows the F-STREAM process. Its gray rectangle shows an instance of the process, which will be explained in Section 4. The variation points—*VarPoint* stereotype—are the activities that are already present on the F-STREAM common core. These activities are generic, but they still may be customized in order to better suit the approach being integrated. The extension points—*Null* stereotype—represent points of the process on which new activities may be inserted, in order to complement the process.

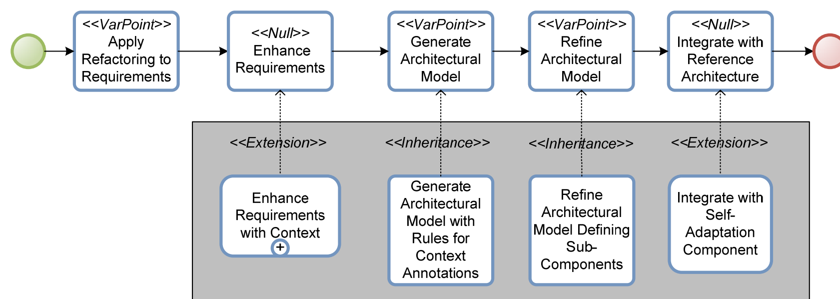


Fig. 2. The F-STREAM process with variability information. The gray rectangle shows modifications and extensions creating a process instance integrated with one approach to handle adaptability.

The process starts with the analysis of the requirements models to detect improvement possibilities. Then, the requirements are enhanced with more information followed by the generation of an initial architectural model. This early architectural model can be further refined and later integrated with a reference model.

The *Apply Refactoring to Requirements* activity consists of refactoring the goal models, based on a set of heuristics and transformation rules. This step is intended to improve the overall quality of the goal model and to turn its structure closer to the expected of an architectural model. Therefore, a modification of this activity would involve changing the heuristics to be used, changing the transformation rules or including new sub-activities.

However, some approaches require the model to be extended, for example with temporal, contextual annotations [7] or crosscutting concerns [1]. Furthermore, they may even require complementary models, such as data-entity models or contextual models. The activities to enrich the original goal models or to define new models may be inserted through the *Enhance Requirements* extension point. Some further

requirements elicitation activity may be required in order to provide the information for these enhanced models.

The next activity is *Generate Architectural Model*. This activity consists of deriving an architectural model from the goal model using vertical transformation rules. The set of transformation rules may be modified to address the peculiarities of the approach being integrated in the process. This is the case when an extended version of goal models is used or when other kinds of models are used, requiring the creation of new rules in order to provide a more complete mapping. This is also the case when an architectural description language other than Acme is required. For the latter case, there are two possible approaches: modifying the current set of transformation rules to derive an architecture on the new target language, or defining new transformation rules for performing the mapping from the Acme language to the new target language.

The *Refine Architectural Model* activity concerns evolving the architectural model by applying architectural patterns. This activity can be simplified when the approach being integrated to the process requires the usage of a specific architectural pattern. Furthermore, new activities may be inserted to provide a more detailed architecture as well as intermediary steps towards integration of the current architecture with the reference architecture.

Lastly, the *Integrate with Reference Architecture* is an extension point to insert the activities that will close the gap between the refined architecture and the reference architecture of the approach that is being integrated with the process, if any. Basically, it consists of defining how to link the existing components to the components of the reference architecture. Nonetheless, further activities may be defined to conclude the architecture generation.

These variation and extension points are summarized in Table 2.

Table 2. A summary of the variability information of the F-STREAM process

Type of Variability	**Activity**	**Variability description**
Variation Points	Apply Refactoring to Requirements	Add, change and remove refactoring heuristics; Add, change and remove horizontal transformation rules; Add new sub-activities.
	Generate Architectural Model	Add, change and remove vertical transformation rules; Add new sub-activities.
	Refine Architectural Model	Add, change and remove architectural patterns to be considered; Add new sub-activities.
Extension Points	Enhance Requirements	Add new activities to handle goal model extensions or other kinds of models.
	Integrate with Reference Architecture	Add activities to integrate the derived architecture with the reference architecture of the approach being used.

4 Example of F-STREAM Instantiation for Adaptive Systems

In this section we instantiate the F-STREAM process to include activities for handling the development of adaptive systems. This particular instance is the result of

integrating the F-STREAM process with an approach for developing adaptive systems, presented in [7], which uses an extended version of *i** models to represent context information. This information will be used by a self-configuring component, which performs all the runtime reasoning related to adaptation. Its reference architecture is based on the definition of sensors and actuators, which interface with the system environment.

This process is depicted in Fig. 2. The first activity, *Apply Refactoring to Requirements*, was maintained as-is. On the other hand, the third activity—*Generate Architectural Model*—was modified, which is expressed by the *Inheritance* association link from the *Generate Architectural Model with Rules for Context Annotations* activity. Similarly, the *Refine Architectural Model Defining Sub-Components* activity modifies the *Refine Architectural Model* activity. Also, on this adaptability instance of the F-STREAM process, the *Enhance Requirements with Contexts* sub-process and the *Integrate with Self-Adaptation Component* were inserted on the extension points, which is expressed by the *Inheritance* association link.

In the following sub-sections we describe each activity of this instantiated process. To exemplify the use of these activities, we are going to use an adaptive smart-home system. In the specific domain of smart homes the adaptivity is a transverse issue. Even when we do not explicitly model a softgoal called Adaptivity, most of softgoals in the model will require adaptivity in some degree (e.g., reliability, customization).

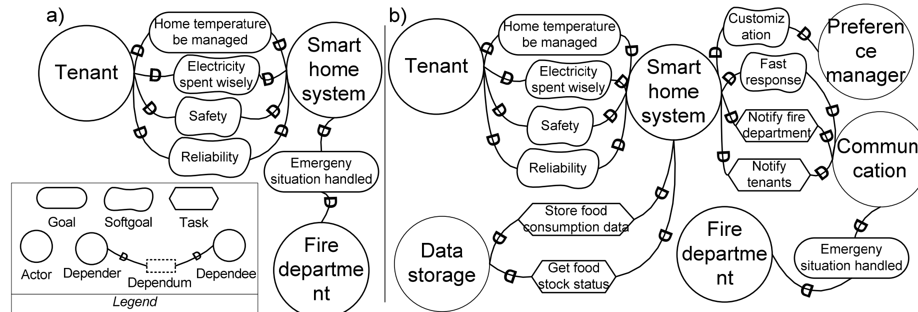


Fig. 3. a) Excerpt of the Strategic Dependencies model of the Smart home system b) Excerpt of the Strategic Dependencies model after refactoring, showing three new software actors.

4.1 Apply Refactoring to Requirements

No modification was required in this activity. Therefore, it can be performed as described in the common core. Fig. 3-a shows an excerpt of the Strategic Dependencies model of the *Smart home system*. A *Tenant*, who is the user of this system, depends on the *Smart home system* to have the house temperature managed, to have electricity spent wisely and to be safe. She also requires the system to be reliable. In order to fulfill these dependencies, the system also depends on other actors. For instance, it needs the *Fire department* to handle emergencies. After refactoring the Smart home system actor (see Section 2.1), three new actors were created: *Preference Manager*, *Communication* and *Data storage*. This is shown in

Fig. 3-b. This refactoring is based on the content of the *Smart home system* actor, which is not presented here for the sake of space.

4.2 Enhance Requirements

In this extension point we inserted a sub-process concerned with context sensors and actuators, presented in Fig. 4. A context sensor is "any system providing up-to-date information about the context where the system is running", whilst a context actuator is "any actuator in the environment which can receive commands from the system to act on the environment context" [7]—i.e., a context sensor monitors the environment and a context actuator performs a change on the environment.

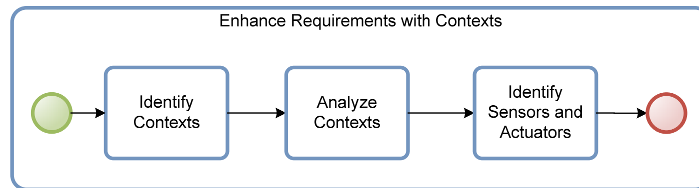
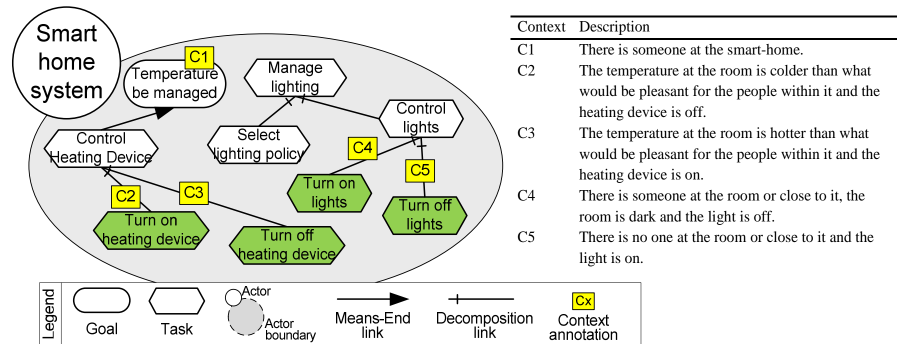


Fig. 4. Enhance Requirements with Contexts sub-process

The *Identify Contexts* activity defines the context information that has an impact on the system's behavior. This information is included in the goal model as context annotations. Fig. 5 shows an excerpt of the goal model of the Smart home system with context annotations. *Temperature be managed* is a goal of the system, but it is only required when the context C1 holds—i.e., when there is someone at home. To achieve this goal, the task *Control Heating Device* can be performed. This task is decomposed in *Turn on heating device*—when C2 holds—and *Turn off heating device*—when C3 holds. The system also has to perform the task *Manage lighting*, which is further decomposed.



Context	Description
C1	There is someone at the smart-home.
C2	The temperature at the room is colder than what would be pleasant for the people within it and the heating device is off.
C3	The temperature at the room is hotter than what would be pleasant for the people within it and the heating device is on.
C4	There is someone at the room or close to it, the room is dark and the light is off.
C5	There is no one at the room or close to it and the light is on.

Fig. 5. Excerpt of the Smart home system goal model with context annotations

The definition of these contexts is crucial for the proper specification of an adaptive behavior. In the *Analyze Contexts* activity these contexts are analyzed to

provide the actual data entities that need to be monitored in order to define the context. This information will be used in the *Identify Sensors and Actuators* activity to discover the context sensors that the system will need. During this activity the context actuators will also be identified, based on the tasks of the goal model. Both the sensor (monitor) and actuator for the Smart-home system are presented on the *i** excerpt of Fig. 6.

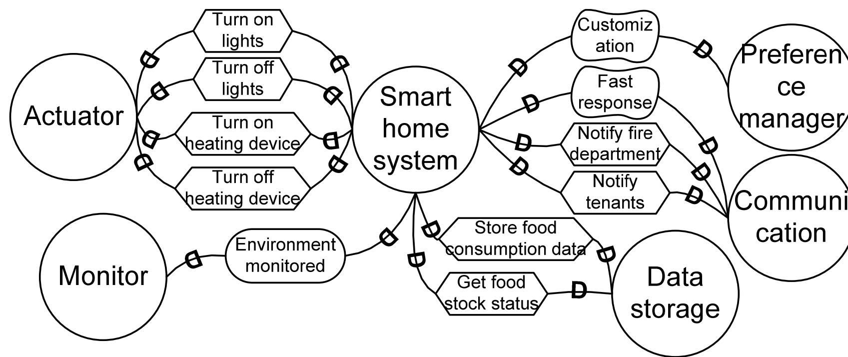


Fig. 6. Excerpt of the Strategic Dependencies model of the Smart home system after the *Identify Sensors and Actuators* activity. Non-software actors are omitted.

4.3 Generate Architectural Model

This variation is needed to add new transformation rules in order to consider the context annotations. Therefore, we defined a new activity that modifies the original activity by adding new rules: the *Generate Architectural Model with Rules for Context Annotations* activity. Fig. 7-a shows the resulting early architecture diagram of the Smart-home system—i.e., the mapping from the *i** model to an architecture in Acme. In summary, the actors are mapped to components and its dependencies are mapped to connectors. The context annotations are mapped as properties of the connectors, which are not explicit in the architecture diagram but are defined with the Acme textual notation.

4.4 Refine Architectural Model

Instead of applying the architectural patterns, at this point it is possible to define some sub-components of the components related to adaptability, using the information included in the goal models during the *Enhance Requirements with Contexts* activity. Therefore, we created a new activity named *Refine Architectural Model Defining Sub-Components*. This activity modifies the original *Refine Architectural Model* activity by including steps to define sub-components of the *Monitor* and *Actuator* components. This is achieved by analyzing the extra information added in the goal model during the *Enhance Requirements with Contexts* sub-process (Section 4.2). In

Fig. 7-b we show the resulting sub-components of the *Monitor* and the *Actuator* components.

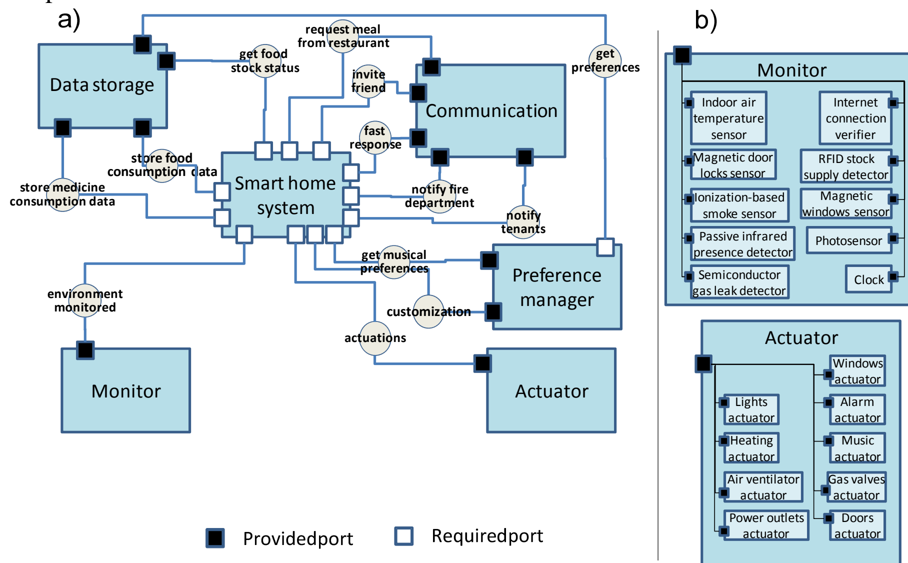


Fig. 7. a) Early architecture of the Smart-home system, after the *Generate Architectural Model with Rules for Context Annotations* activity. b) Sub-components of the *Monitor* and *Actuator* components, defined during the *Refine Architectural Model Defining Sub-Components* activity.

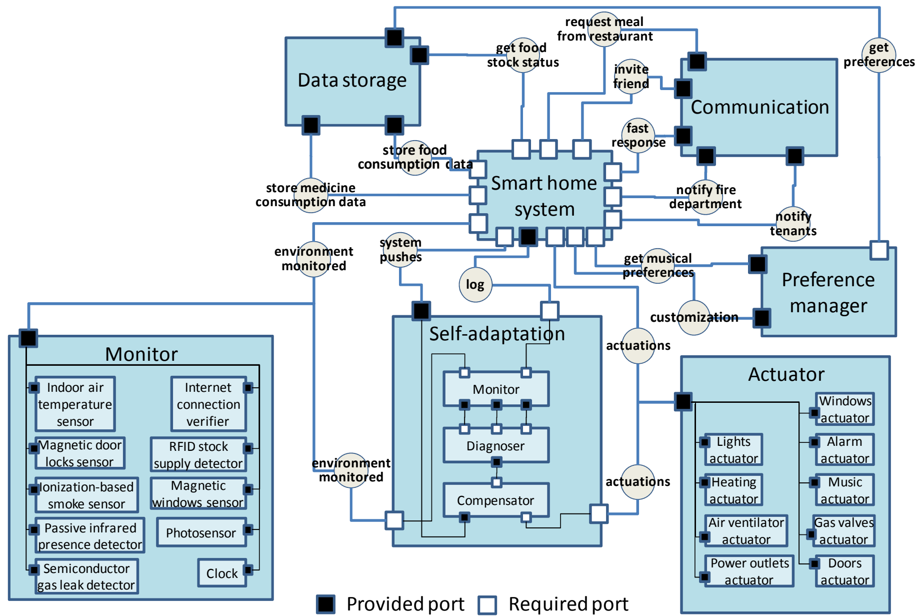


Fig. 8. Resulting architecture of the Smart home system, after the *Integrate with Self-Adaptation Component* activity.

4.5 Integrate with Reference Architecture

Since the components of the architecture modeled so far need to be linked to the component defined in the reference architecture [7], in this extension point we defined the *Integrate with Self-Adaptation Component* activity. This component performs a Monitor-Diagnose-Compensate (MDC) reasoning cycle, to check if the goals of the system are being achieved and, if not, what adaptations are required to achieve them. This is performed based on the context-annotated goal model and on the input of the context sensors. By encapsulating this reasoning, this component prevents the need of hard-coding the adaptation handling.

The *Self-adaptation* component will be linked to the main component of the system (in this example, the *Smart home system* component), to the *Monitor* component and to the *Actuator* component. The *Self-adaptation* component will receive a history of the system's execution from the main component (*log* connector) and the environmental data from the *Monitor* component (*environmentMonitored* connector). This data will be checked against the goal model of the system, and the required adaptations will be identified. Some of the adaptations will be required to be performed through the *Actuator* component (*actuations* connector), and others will be suggested to the main component (*system pushes* connector). The resulting architecture of the Smart-home system is presented in Fig. 8.

5 Conclusion and Future Work

In this paper we defined F-STREAM, a flexible, systematic and model-based process to derive architecture models from requirements. We faced the challenge of proposing an approach that could be configured to address multiple quality attributes of interest. Inspired by Software Product Line (SPL) principles we defined a set of common core, variation and extension points.

Our goal is to be able to deploy our generic approach to handle specific non-functional requirements, such as adaptability, security, reusability, etc, through integration with other existing approaches. Thus, all the support for NFR would come from these approaches, including NFR refinement and traceability. As a proof of concept, we described how F-STREAM could be applied to develop an adaptive Smart Home system.

As future work, we need to further validate our work with more case studies. We also intend to define a family of instances of the F-STREAM process, addressing some of works presented in Table 1. Additionally, we need to conduct further research to analyze how the different approaches may be weaved together to handle multiple and possibly conflicting NFR. There will be also a parallel effort to improve the STREAM process–for instance, by defining guidelines for its use.

Acknowledgments. This work has been partially supported by Erasmus Mundus External Cooperation Window - Lot 15 Brasil and the Brazilian institutions CAPES and CNPq.

References

1. Alencar, F., Castro, J., Moreira, A., Araujo, J., Silva, C., Ramos, R., Mylopoulos, J.: Integration of Aspects with i* Models. In: Agent-Oriented Information Systems IV, LNCS 4898, Springer-Verlag, 2008, pp. 183-201.
2. Alencar, F., Marn, B., Giachetti, G., Pastor, O., Castro, J., Pimentel, J.: From *i** Requirements Models to Conceptual Models of a Model Driven Development Process In: PoEM, 99-114, 2009.
3. Ameller, D., Franch, X., Cabot, J.: Dealing with Non-Functional Requirements in Model-Driven Development. In: RE 2010: 189-198.
4. Bastos, L., Castro, J.: From requirements to multi-agent architecture using organisational concepts, ACM SIGSOFT Software Engineering Notes, vol 30 pp. 1-7, 2005
5. Castro, J., Silva, C., Mylopoulos, J.: Modeling Organizational Architectural Styles in UML. In: CAISE 2003 - LNCS, v. 2681, pp. 111-126, 2003.
6. Chung, L., Gross, D., Yu, E. S. K.: Architectural Design to Meet Stakeholder Requirements. In: Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1), Kluwer, B.V., 1999, 545-564
7. Dalpiaz, F., Giorgini, P., Mylopoulos, J.: An architecture for requirements-driven self-reconfiguration. In: CAiSE 2009. LNCS, v. 5565, p. 246-260.
8. Garlan, D., Monroe, R., Wile, D.: Acme: An Architecture Description Interchange Language. In: Proc.CASCON'97, 1997. Toronto, Canada.
9. Giorgini, P., Rizzi, S., Garzetti, M.: Goal-oriented requirement analysis for data warehouse design. In: Proceedings of the 8th ACM international workshop on Data warehousing and OLAP (DOLAP '05). ACM, New York, NY, USA, pp. 47-56, 2005.
10. Grau, G., Franch, X.: On the adequacy of *i** models for representing and analyzing software architectures. In: Advances in conceptual modeling: foundations and applications - LNCS, v. 4802, pp. 296-305, 2007.
11. Lucena, M., Castro, J., Silva, C., Alencar, F., Santos, E., Pimentel, J.: A Model Transformation Approach to Derive Architectural Models from Goal-Oriented Requirements Models. In: Proc. the OMT Workshop IWSSA,LNCS Springer-Verlag Berlin Heidelberg, Vilamoura, Portugal, pp. 370-380, 2009.
12. Lucena, M., Santos, E., Silva, M., Silva, C., Alencar, F., Castro, J.: Towards a Unified Metamodel for *i**. In: Proc. of RCIS'08, pp. 237-246, 2008.
13. Lucena, M., Silva, C., Santos, E., Alencar, F., Castro, J.: Applying Transformation Rules to Improve i* Models, In: Proc. the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE 2009), Boston, USA, pp. 43-48, 2009.
14. Mylopoulos, J., Castro, J., Kolp, M.: Tropos: Toward agent-oriented information systems engineering. In Second International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS2000), 2000.
15. Santos, E., Pimentel, J., Castro, J., Sanchez, J., Pastor, O.: Configuring the Variability of Business Process Models Using Non-Functional Requirements. In: Enterprise, Business-Process and Information Systems Modeling - LNBIP, v. 50, part 2, pp. 274-286, 2010.
16. Schnieders, A., Puhlmann, F.: Variability Mechanisms in E-Business Process Families. In: Proc. of the 9th Int. Conference on Business Information Systems, BIS 2006, 2006
17. Silva, C., Borba, C., Castro, J.: A Goal Oriented Approach to Identify and Configure Feature Models for Software Product Lines. In: Proc. of the 14th Workshop on Requirements Engineering (WER 2011), Rio de Janeiro, Brazil, 2011.
18. Souza, V., Mylopoulos, J.: Monitoring and Diagnosing Malicious Attacks with Autonomic Software. In: ER 2009 LNCS, v. 5829, pp. 84-98, 2009.
19. Yu, E.: Modeling Strategic Relationships for Process Reengineering. PhD Thesis, Toronto University, 1995.