

Automatic Generation of Architectural Models From Goals Models

Monique Soares, João Pimentel, Jaelson Castro, Carla Silva, Cleice Talitha, Gabriela Guedes, Diego Dermeval

Centro de Informática
Universidade Federal de Pernambuco/UFPE
Recife, Brazil
{mcs4, jhcp, jbc, ctlls, ctns, ggs, ddmcm}@cin.ufpe.br

Abstract—The STREAM (Strategy for Transition Between Requirements and Architectural Models) process presents an approach that allows the generation of early architectural design described in Acme ADL from goal oriented requirements models expressed in i^* . The process includes activities that defines transformation rules to derive such architectural models. In order to minimize the effort to apply the process and decrease the possibility of making mistakes it is vital that some degree of automation is provided. This paper explains in detail the transformation rules proposed and their corresponding formalization in a model transformation language.

Requirements Engineering, Software Architecture, Transformation Rules, Automation.

I. INTRODUCTION

The STREAM (Strategy for Transition between Requirements Models and Architectural Models) is a systematic approach to integrate requirements engineering and architectural design activities, based on model transformations to generate architectural models from requirements models [3]. The source language is i^* (iStar) [11] and the target language is Acme [2].

Our proposal is in line with the current MDD (Model-Driven Development) paradigm, as we support the transformation of models of higher levels of abstraction to more concrete models. The MDD advantages are: greater productivity and, therefore, a lower development time; increased portability; increased interoperability; and lower maintenance costs, due to the improved consistency and maintainability of the code [7].

Currently, the transformation rules defined in the STREAM approach are informally described and are manually applied. Hence, their use is time consuming and effortful, as well as error prone. In order to overcome these shortcomings, we propose to use an imperative transformation language (QVTO [8]) to properly describe them and to provide tool support.

The aim of this paper is to automate the vertical transformation rules proposed in the STREAM. For this, it was necessary to: Define these rules in a proper transformation language; Make these rules compatible with the IStarTool and AcmeStudio tools; and illustrate the use of them rules.

The remainder of this paper is organized as follows. Section 2 describes the background required for a better understanding of this work. Section 3 presents the description and automation

of the vertical transformation rules in QVT. Section 4 presents an application example. Section 5 presents the related works and Section 6 discuss the results of this work and future research.

II. BACKGROUND

This section presents an overview on i^* , Acme and the STREAM process, which are used in our approach.

A. i^* Star

The i^* language is a goal-oriented modeling language able to represent features of both the organization and of the system to be acquired/developed by/for the organization. Stakeholders and systems are represented as actors, which are active entities able to perform tasks, reach goals and provide resources. In order to achieve their own goals, actors have dependencies with each other [11].

i^* is comprised of two models: a SD (Strategic Dependency) model describes dependency relationships among actors in the organization; a SR (Strategic Rationale) model explains how actors achieve their goals and dependencies.

In a dependency, a *dependor* actor relies on a *dependee* actor to achieve something (the *dependum*). A *dependum* can be a goal, which represents the intentional desire of an actor, to be fulfilled; a softgoal to be satisfied, which is a goal with the acceptance criterion not so clear; a resource to be provided; or a task to be performed. Figure 1 presents an excerpt of the i^* metamodel defined for the iStarTool tool [6].

B. Acme

Acme is an ADL (Architectural Description Language) designed to describe the components and connectors (C&C) view of the system architecture [2]. It relies on six main types of entities for architectural representation: *Components*, *Connectors*, *Systems*, *Ports*, *Roles*, and *Representations*. Figure 2 presents the Acme metamodel, based on the AcmeStudio tool [1].

Components represent the primary computational elements and data stores of a system. Connectors characterize interactions among components. Systems denote configurations of components and connectors. Each port identifies a point of interaction between the component and its environment. Roles define the connector's interfaces. Representation supports hierarchical descriptions of architectures. [2].

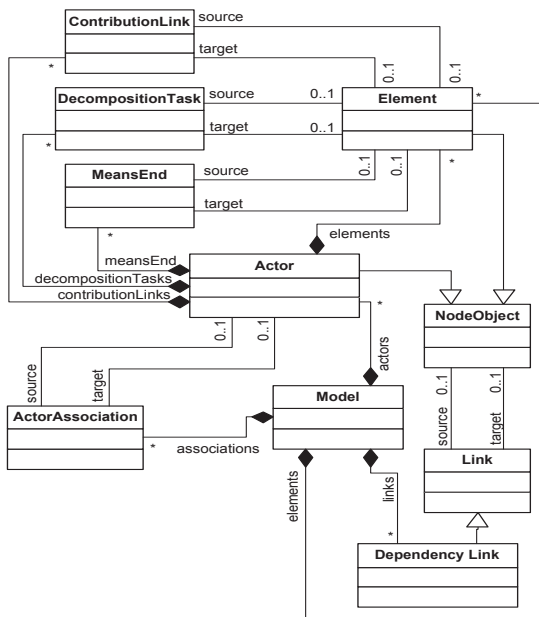


Figure 1. Excerpt of the i^* metamodel

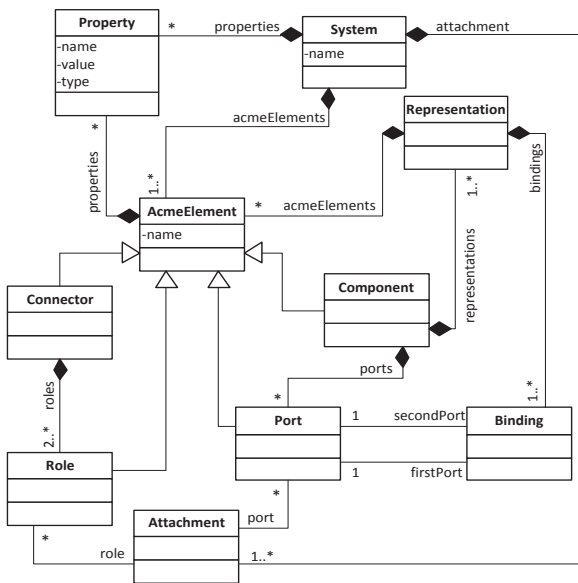


Figure 2. Acme metamodel

C. STREAM

The STREAM process includes the following activities: 1) Prepare requirements models, 2) Generate architectural solutions, 3) Select architectural solution, and 4) Refine architecture. In activities 1) and 2), horizontal and vertical rules are proposed, respectively. Horizontal rules are applied to the i^* requirements models to increase its modularity and prepare them for early architectural transformation. There are four horizontal transformation rules (HTRs) [9]. Vertical rules are used to derive architectural models in Acme from modularized i^* models. Non-Functional Requirements are used to select initial candidate architecture in the 3) activity. Certain architectural

patterns can be applied to allow appropriate refinements of the chosen candidate architectural solution in 4) activity [5].

In the Vertical Transformation Rules (VTRs), the i^* actors and dependencies are mapped to Acme elements. Thus, an i^* actor is mapped to an Acme component. An i^* dependency becomes an Acme connector. The *depender* and *dependee* actors in a dependency can be mapped to the roles of a connector. In particular, we can distinguish between required ports (mapped from *depender* actors) and provided ports (mapped from *dependee* actors). Thus, a connector allows communication between these ports. A component provides services to another component using provided ports and it requires services from another component using required ports.

The four types of dependencies (goal, softgoal, task and resource) will define specific design decisions in connectors, ports and roles that are captured as Acme *Attachments*. An object dependency is mapped to a Boolean property. A task dependency is mapped directly to a port provided. The resource dependency is mapped to a return type of a property provided. A softgoal dependency is mapped to a property with enumerated type

These transformation rules were defined in a semi formal way in [13] and now they need to be precisely specified using a suitable model transformation language, such as Query/View/Transformation Operational – QVTO [8]. In doing so, we can validate them, as well as provide support to (partially) automate the process, hence enabling the STREAM process to become a full-fledged MDD approach.

III. AUTOMATION OF THE VERTICAL TRANSFORMATION RULES

In the STREAM process, the user begins by using i^* to model the problem at hand. Some heuristics can guide the selection of sub-set(s) of candidate elements to be refactored. Once they are selected, the HTRs can be applied to improve the modularization of the i^* model [5].

Since the vertical transformations do not consider the internal elements of the actors, we first create an intermediary SD model from the modular i^* SR model. We proceed to apply the VTRs (see Table I). The first rule (VTR1) maps i^* actors to Acme components, while VTR2 transforms i^* dependencies to Acme connectors. The VTR3 converts the *depender* actor onto a required port of the Acme connector. The VTR4 translates the *dependee* actor onto a provided port of the Acme connector.

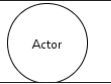




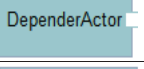
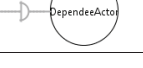
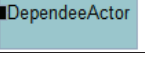
We relied on the Eclipse based tool for i^* modeling, the *iStarTool* [6], to create the i^* model. This model is the input for the first STREAM activity. This tool generates a XMI of the modularized i^* SD model, which can be read by the QVTO Eclipse plugin [4], to serve as input for the VTRs execution.

The VTRs described in QVTO are based on the metamodels presented in Section 2, they are referenced during the execution of the transformation. The models created with the VTRs execution are represented as XMI files.

In our work, we were able to automate 3 horizontals (HTR2-HTR4) and 4 verticals transformation rules [14]. However, due to space limitation, in this paper, we only discuss how

we dealt with the verticals rules. Table 1 illustrates the elements present in the source model and their corresponding elements present in the target model.

TABLE I. VERTICAL TRANSFORMATION RULES

| | Source (i^*) | Target (Acme) |
|------|---|---|
| VTR1 |  |  |
| VTR2 |  |  |
| VTR3 |  |  |
| VTR4 |  |  |

To map i^* actors to Acme components, we need to obtain the number of actors present in the modularized i^* SD model artifact. So, we create the same amount of Acme components, giving to this components, the same names of the i^* actors. The XMI file obtained as output of this transformation will contain the components represented by tags (Figure 6).

```
while(actorsAmount > 0) {
  result.acmeElements += object Component{
    name := self.actors.name->at(actorsAmount);
  }
  actorsAmount := actorsAmount - 1;
}
```

Figure 3. Excerpt of the QVTO code for VTR1

In the VTR2 each i^* dependency creates two XMI tags, one capturing the *dependee* to the *dependum* connection and another one captures the *dependum* to the *dependee*.

In order to map these dependencies in Acme connectors it is necessary to recover the two dependencies tags, observing that have the same *dependum*. It is necessary not consider the actor which plays the role of *dependee* in some dependency and *dependee* in another. Once this is performed, there are only *dependums* left. For each *dependum*, one Acme connector is created. The connector created receives the name of the *dependum* of the dependency link. Two roles are created within the connector, one named *dependeeRole* and another named *dependeeRole*. The XMI output file will contain the connectors represented by tags (see Figure 4).

```
<acmeElements xsi:type="Acme:Connector" name="Conn">
  <roles name="dependeeRole"/>
  <roles name="dependeeRole"/>
</acmeElements>
```

Figure 4. Connector in XMI

The VTR3 converts a *dependee* actor into a required port present in the component obtained from that actor (see Figure 5). First, we create one Acme port for each actor *dependee*. Each port created has a name and a property. The port name is given at random, just to control them. The property must have a name and a value, so to the property name is assigned "Required" as we are creating a required port and the value is *true*.

The XMI output file will contain "ports" tags within the *acmeElement* tag of component type. Moreover, since they are required ports, there will be one property with an attribute named "Required" whose value is set to "true".

Last but not least, the VTR4 maps all *dependee* actors to provided ports in the corresponding components obtained by those actors. For this, we list all *dependee* actors in the model. Every port generated has a name and a property. The port name is given at random. The port property name is "Provided", the port type is set to "boolean" and the port value is set to "true".

```
while(dependencyAmount > 0) {
  if(self.actors.name->includes(self.links.source-
  >at(dependencyAmount).name) and
  self.actors.name-
  >at(actorsAmount).=(self.links.source-
  >at(dependencyAmount).name)) then {
    ports += object Port{
      name := "port"+countPort.toString();
      properties := object Property {
        name := "Required";
        value := "true"
      };
    };
  } endif;
  dependencyAmount := dependencyAmount - 1;
  countPort := countPort + 1;
};
```

Figure 5. Excerpt of the QVTO code for VTR3

```
<acmeElements xsi:type="Acme:Component" name="Comp">
  <ports name="port17">
    <properties name="Provided" value="true"
    type="boolean"/>
  </ports>
</acmeElements>
```

Figure 6. Provided port, component and properties in XMI

After creation of the basics Acme elements, it is necessary to create the *Attachment* object, element responsible for associating the connectors to the required and provided ports present in the components. Therefore, an attachment is created for each port of a component. Each *Attachment* has a component as an attribute, a port, a connector and a role.

Next section presents an example to illustrate our approach.

IV. RUNNING EXAMPLE

MyCourses is a scheduling system that provides, as optimal as possible, a plan for scheduling courses. It allows universities to perform tasks, such as checking and listing available lecture rooms, teachers, students enrolled in any course. It was one of the project proposals for the ICSE 2011 Student Contest on Software Engineering [10].

The modularized i^* SD model for the MyCourses system (Figure 7) was used as input model for the execution of the VTRs. These rules have the objective to transform a modularized i^* SD model into an Acme initial architectural model.

After the automated application of the VTRs, a XMI model representing the output model and compatible with the Acme metamodel, will be generated. Figure 8 shows the graphical representation of that XMI model for the MyCourses system.

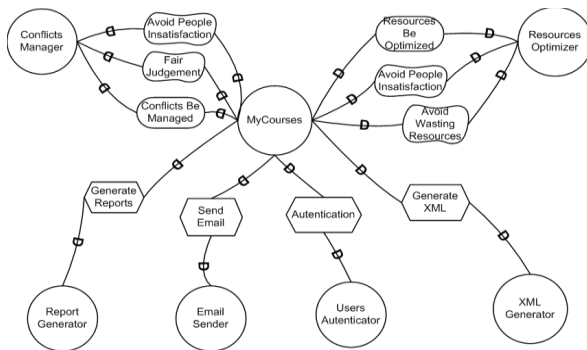


Figure 7. Modularized *i** SD model MyCourses

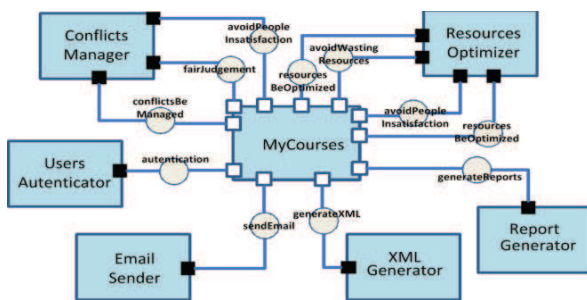


Figure 8. Acme Model from MyCourses

V. RELATED WORK

Our work is unique in supporting the STREAM approach.

MaRiSA-MDD [15] presents a strategy based on models that integrate aspect-oriented requirements, architecture and detailed design, using AOV-graph, AspectualACME and aSideML languages, respectively. It defines representative models and a number of transformations between the models of each language. The transformation language used was ATL.

Silva et al [16] specify, through a model-driven approach, the transformations necessary for architectural models described in UML, from architectural organizational models described in *i**. The transformation language used was ATL.

VI. CONSIDERATIONS AND FUTURE WORKS

In this paper, we advocated the use of model transformation to generate architectural models from requirements models. We reviewed the STREAM process, which defines and applies (manually) a set of model transformation rules to obtain Acme architectural models from *i** requirements models.

In order to decrease time and effort required to perform the STREAM process and minimize the errors introduced by the manual execution of the transformation rules, we proposed to use the QVTO language to automatize the execution of these rules. Our focus was on the automation of the VTRs, responsible to generate an initial Acme architectural model. Metamodels of the *i** and Acme languages were provided. The input models of the VTRs are compatible with the iStarTool and the output models are compatible with Acme metamodel, supported by the AcmeStudio tool.

Currently, the output of our process is an XMI file with the initial Acme architectural model. But the AcmeStudio tool reading files described in Acme textual language. As a consequence, the current architectural model cannot be graphically displayed. Hence, our plan is to provide new transformation rules to generate the textual representations. Case studies are still required to validate our approach.

REFERENCES

- [1] ACME. Acme. Acme - The AcmeStudio, 2009. Available in: <http://www.cs.cmu.edu/~acme/AcmeStudio/>. Accessed in: May 2012.
- [2] GARLAN, D., MONROE, R., Wile, D. Acme: An Architecture Description Interchange Language. In: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research (CASCON'97). Toronto, Canada.
- [3] 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: 8th IWSSA - OTM Workshops 2009. Lecture Notes in Computer Science, 2009, Volume 5872/2009, 370-380.
- [4] ECLIPSE M2M. Model To Model (M2M). Available in: <http://eclipse.org/m2m/>. Accessed in: May 2012
- [5] CASTRO, J.; LUCENA, M.; SILVA, C.; ALENCAR, F.; SANTOS, E.; PIMENTEL, J. C. Changing Attitudes Towards the Generation of Architectural Models. Journal of Systems and Software, 2012.
- [6] MALTA, A.; SOARES, M.; SANTOS, E.; PAES, J.; ALENCAR, F.; CASTRO, J. iStarTool: Modeling requirements using the *i** framework. IStar 11, August 2011.
- [7] OMG. Object Management Group. MDA Productivity Study, June 2003. Available in: <http://www.omg.org/mda/mda_files/MDA_Comparison-TMC_final.pdf/>. Accessed in: May 2012
- [8] OMG. QVT 1.1. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, January 2011. Available in: <http://www.omg.org/spec/QVT/1.1/>. Accessed in: May 2012
- [9] LUCENA, M.; SILVA, C.; SANTOS, E.; ALENCAR, F.; CASTRO, J. Applying Transformation Rules to Improve *i** Models. SEKE 2009: 43-48.
- [10] SCORE 2011. The Student Contest on Software Engineering - SCORE 2011, 2011. Available in: <http://score-contest.org/2011/>. Accessed in: May 2012.
- [11] YU, E.; GIORGINI, P.; MAIDEN, N.; MYLOPOULOS, J. Social Modeling for Requirements Engineering. Cambridge, MA: MIT Press. 2011. ISBN: 978-0-262-24055-0.
- [12] MENS, T.; CZARNECKI, K.; VAN GORP, P. A Taxonomy of Model Transformations. In: Proceedings of the Language Engineering for Model-Driven Software Development. Dagstuhl, Germany 2005.
- [13] PIMENTEL, J.; LUCENA, M.; CASTRO, J.; SILVA, C.; SANTOS, E.; ALENCAR, F. Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach. Requirements Engineering Journal, 2011.
- [14] SOARES, M. C. Automatization of the Transformation Rules on the STREAM process (In Portuguese: Automatização das Regras de Transformação do Processo STREAM). Dissertation (M.Sc.). Center of Informatic: UFPE, Brazil, 2012.
- [15] MEDEIROS, A. MARISA-MDD: An Approach to Transformations between Oriented Aspects Models: from requirements to Detailed Project (In Portuguese: MARISA-MDD: Uma Abordagem para Transformações entre Modelos Orientados a Aspectos: dos Requisitos ao Projeto Detalhado). Dissertation (M.S.c). Center for Science and Earth: UFRN, Brazil, 2008.
- [16] SILVA, C.; DIAS, P.; ARAÚJO, J.; MOREIRA, ANA. From Organizational Architectures in *i** Agent-based: A model-driven approach (De Arquiteturas Organizacionais em *i** a Arquiteturas Baseadas em Agentes: Uma abordagem orientada a modelos). WER'11.