

Automatic Models Transformation for the STREAM process

Monique Soares¹, João Pimentel¹, Carla Silva¹, Gabriela Guedes¹, Cleice Talitha¹,
Fernanda Alencar², Jaelson Castro¹, Jéssyka Vilela¹

¹Universidade Federal de Pernambuco – UFPE, Centro de Informática, Recife,
Brasil

{mcs4, jhp, ctlls, ggs, ctsn, jbc, jffv}@cin.ufpe.br

²Universidade Federal de Pernambuco – UFPE, Departamento de Eletrônica e
Sistemas, Recife, Brasil fernandaalenc@gmail.com

Abstract. The STREAM (Strategy for Transition Between Requirements and Architectural Models) process allows systematic generation of initial architectural design models from oriented goals requirements models through an informal definition of model transformation rules. It was observed that the first two activities in this process are time-consuming and error-prone because they involve model transformation rules that are not automated. This article advocates the automation of such transformation rules, with the intention of improving the productivity of the process and the quality of the models produced.

Keyword: Requirements Engineering, Software Architecture, Transformation Rules, Automation.

1 Introduction

The STREAM process (Strategy for Transition Between Requirements and Architectural Models) [2] is a systematic approach to integrate the activities of requirements engineering and architectural design, based on model transformations, to generate architectural models from requirements models.

To transform the requirements models in architecture models the STREAM uses i* (iStar) [3] as source language and Acme [4] as target language. This approach has four activities, namely: Prepare Requirements Models, Generate Architectural Solutions, Select Architectural Solution and Refine Architecture. The first two activities need attention and take a lot of time to be done. These activities define horizontal and vertical transformation rules (HTRs and VTRs), respectively, which are amenable to automation are implemented. Once automated, the first two STREAM activities would not require as much time and attention to be performed.

Currently, the transformation rules are made manually, demanding attention from the analyst. Our proposal is to use a transformation language to describe these rules and execute them with a supporting tool.

The i* language allows goal-oriented modeling [3], it is able to represent the organization and characteristics of the system being developed. Stakeholders and

systems are represented as actors. To achieve its goals, the actors depend on each other. The i* language is composed of two models: a SD (Strategic Dependency) model, that represents dependency relationships between actors, and a SR (Strategic Rationale) model, that details how actors achieve their goals and dependencies.

In a dependency, a *dependor* actor depends on another actor (*dependee*) to achieve some objective (*dependum*). A *dependum* can be a goal, a softgoal, resource or task.

Acme is an architectural description language [4] designed to describe the view of the components and connectors of the system architecture. It has six main types of entities for representing architecture: Components, Connectors, Systems, Ports, Roles and Representations.

Components represent the primary computational elements and storage of data from one system. Connectors characterize interactions between the components. Systems denote configurations of components and connectors. Each port identifies a point of interaction between the component and its environment. Roles define the connectors' interfaces. Representation describes support hierarchical architectures. Among these six types, the most basic elements of architectural description are components, connectors and systems.

2 Research Objectives

This paper aims to answer the following research question:

Is it possible to automate the transformation rules defined in the first two STREAM activities? And, if possible, how to automate these rules?

The main objective of this work is to automate the transformation rules defined in STREAM. In order to accomplish this, we have established the following strategy:

- Define the transformation rules in a transformation language;
- Make the vertical and horizontal transformation rules compatible with iStarTool [6];
- Make the vertical transformation rules consistent with AcmeStudio.

3 Scientific Contributions

The automation of the horizontal and vertical transformation rules proposed by STREAM requires a language that allows mapping elements and processing them. To do this, we have chosen the QVTO (Query / View / Transformation Operational) [5] language, which is a transformation language that has integration with Eclipse and whose community gives constant maintenance and support.

To facilitate the development of artifacts used as input in the first activity of STREAM, we have chosen iStarTool tool [6], which allows i* modeling. It is developed under a model-driven technology, the GMF (Graphical Modeling Framework) [7] Eclipse. The iStarTool uses XMI files to save its models, based on its own metamodel. The XMI file generated by iStarTool is compatible with the QVTO plugin in Eclipse.

The input artifacts of the first STREAM activity are created in iStarTool and the transformation of these artifacts is executed by the QVTO Eclipse plugin. This first activity is concerned with improving the modularity of the expanded system actor and it is subdivided into three activities: analysis of internal elements, application of horizontal transformation rules and evaluation of i* models.

In order to develop these activities it is necessary to use:

- Heuristics to guide the decomposition of the system actor;
- A set of rules for transforming i* models;
- Metrics to measure the modularization degree of initial and final i* models.

There are four HTRs (Horizontal Transformation Rules). **Table 1** presents application examples of each horizontal rule, showing the original model and the resulting model after applying the rule, in order to representate how each rule works.

HTR1 moves a previously selected sub-graph, according to the heuristics. The analyst may choose to move this sub-graph for a new actor or an existent actor. HTR1 was not automated because it depends on the choice of the analyst, so it is necessary to apply this rule manually. The analyst uses the heuristics and performs the HTR1 rule.

After applying HTR1, the resulting model may not be correct according to the i* language syntax. Therefore, we must check the preconditions of the others rules, HTR2, HTR3 and HTR4.

The HTR2 moves a means-end relationship across the actor boundary. HTR2 considers the situation where the sub-graph to be moved has the root element as a "means" of a means-end relationship.

The HTR3 moves a contribution relationship across the actor boundary. The HTR3 considers the situation in which the sub-graph to be moved has a contribution relationship with other elements that cannot be moved.

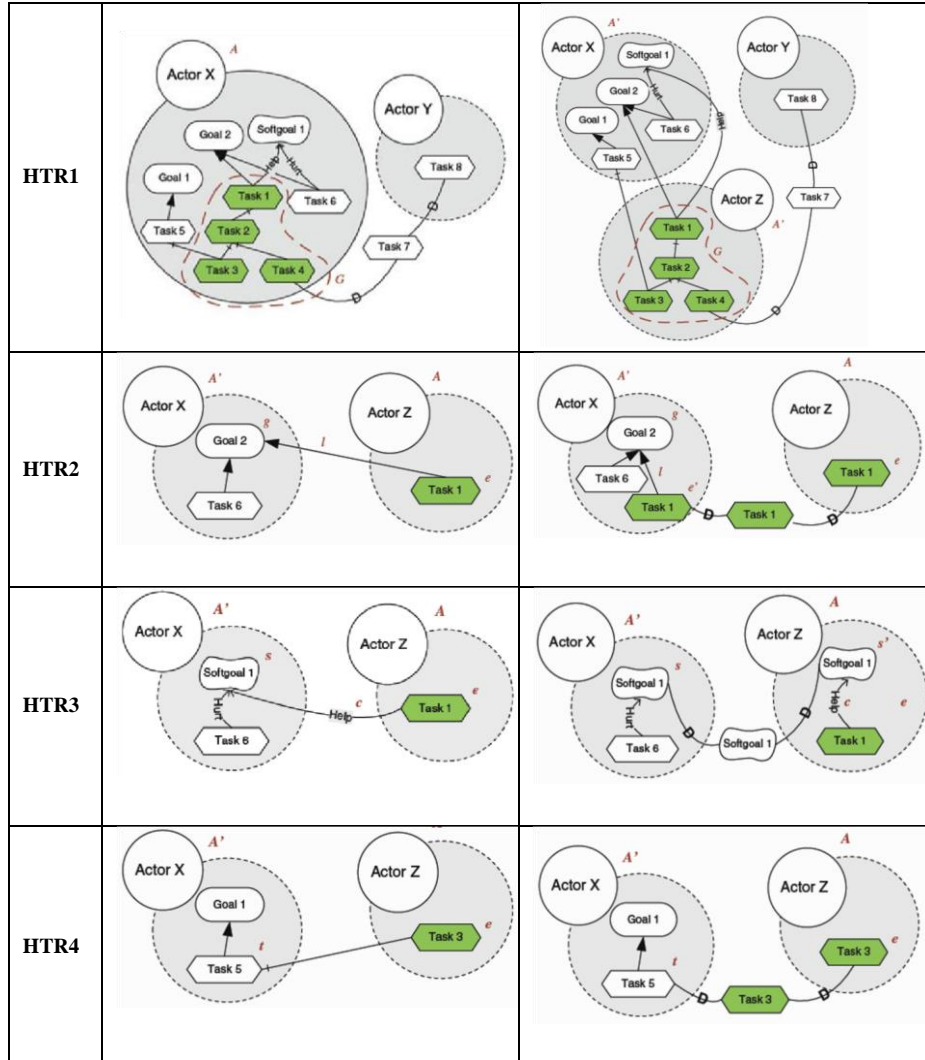
The HTR4 moves a task decomposition relationship across the actor boundary. HTR4 considers the situation where the sub-graph to be moved has a task decomposition relationship with other elements that cannot be moved.

The transformation rules are intended to delegate internal elements of software actor to other actors. This delegation must ensure that new actors and the original actor has a dependency relationship. Thus, the original model and the final model are supposed to be semantically equivalent.

Upon completion of this activity, the actors representing the software are easier to understand and maintain, since there are less internal elements. The original requirements model is decomposed into a more modularized one.

Table 1. Application example of HTRs adapted from [8]

Rule	Original Model	Resulting model after applying the rule
------	----------------	---




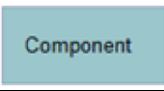


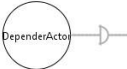
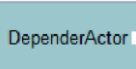
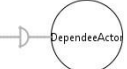

The second activity of STREAM process (Derive Architectural Solutions) is concerned with the mapping of i^* actors and dependencies into Acme elements through the application of Vertical Transformation Rules(VTRs).

As the vertical transformation rules do not consider the internal elements of the actors, first we create a SD model from the modularized i^* model, produced on the first activity.

The first rule (VTR1) maps i^* actors into Acme components, while VTR2 transforms i^* dependencies in Acme connectors. The VTR3 maps *depender* actors as required port of Acme connector. Finally, the VTR4 maps *dependee* actors as provided port of Acme connector.

Table 2 shows the VTRs.

Table 2. Vertical Transformation Rules

	Fonte (i*)	Alvo (Acme)
VTR1		
VTR2		
VTR3		
VTR4		

For the construction of i* artifacts we used iStarTool and the rules were described in QVT. It was necessary to make the vertical rules consistent with iStarTool and AcmeStudio tools, to make it possible, we used the i* metamodel present iStarTool and the Acme metamodel designed according to the AcmeStudio syntax to describe and run the rules.

To apply automated rules you need to perform these steps [9]:

1. Create the i* model of the system using iStarTool;
2. Use three heuristics for selecting the subset of candidate elements for refactoring (modularization);
3. Apply HTR1 manually, i.e., move the subset of candidate elements to another actors, in order to modularize the i* model;
4. Use the SR model obtained with HTR1 as input to the application of other horizontal transformation rules, which are already automated;
5. Transform the modularized i* SR model, resulting from the application of horizontal transformation rules, into an SD model;
6. Use the SD model as input to the application of vertical transformation rules, which results in an initial architectural model in Acme.

Of the four horizontal transformation rules, three were automated (HTR2, HTR3 and HTR4). The vertical transformation rules were divided into four rules for better understanding and all of them were automated.

4 Conclusions

The main objective of this work was to automate the horizontal and vertical transformation rules proposed by the activities 1) Prepare Requirements Models and 2) Generate Architectural Solutions of the STREAM process.

We used the iStarTool tool to create the input artifact for performing the horizontal rules. This artifact is the i* SR model obtained with manual execution of HTR1 applied according to the choice of the analyst.

The transformation rules proposed by STREAM were described in QVT, a specification language for processing models, for the automation and execution of the

transformation rules. The Eclipse environment was used to make possible the implementation of the rules in QVT language.

We applied the automated rules in three software projects to exemplify their use. More details are presented in [1].

5 Ongoing and Future Works

Currently, the output of our process is a XMI file with an architectural model in Acme. However, the AcmeStudio tool is able to read only files described using the Acme textual language. Therefore, it is not possible to display the architectural model graphically. So, our plan is to provide another set of transformation rules for generating the textual file also, thus facilitating the graphical visualization in AcmeStudio tool. It is also necessary to include the automated rules on iStarTool, to facilitate the execution of the rules in the same tool, the modeling and the transformation will occur in the iStarTool. And last, execute the rules on various application examples with the intention of evaluating the real benefits and limitations of approach.

References

1. 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 Informatics: UFPE, Brazil, 2012, available in: http://www.cin.ufpe.br/~mcs4/dissertation_mcs4.pdf.
2. Lucena, M. STREAM: A Systematic Process to Derive Architectural Models from Requirements Models. Thesis (PhD). Center of Informatics: UFPE, Brazil, 2010
3. Yu, E. Modelling Strategic Relationships for Process Reengineering. Thesis (PhD). University of Toronto: Department of Computer Science, 1995.
4. Acme. Acme - The Acme Architectural Description Language and Design Environment., 2011. Available in: <http://www.cs.cmu.edu/~acme/index.html>. Accessed: March 2013.
5. 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: March 2013.
6. Malta, A., Soares, M., Santos, E., Paes, J., Alencar, F., Castro, J. (2011). iStarTool: Modeling requirements using the i* framework. IStar 11.
7. ECLIPSE GMF. GMF - Graphical Modelling Framework. Available in: <http://www.eclipse.org/modeling/gmf/>. Accessed: March 2013.
8. Pimentel, J., Lucena, M., Castro, J., Silva, C., Santos, E., Alencar, F. (2011). Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach. Requirements Engineering Journal.
9. Soares, M. C., Pimentel, J., Castro, J., Silva, C., Souza, C. T., Guedes, G. S., Dermeval, D. (2012). Automatic Generation of Architectural Models From Goals Models. SEKE 2012: 444-447.