A Model-Based Transformation Approach for Embedded Systems Development*

Aram Hovsepyan, Didier Delanote, Stefan Van Baelen, Yolande Berbers, Wouter Joosen

Katholieke Universiteit Leuven, Departement Computerwetenschappen, Celestijnenlaan 200A, B-3001 Leuven, Belgium
{Aram.Hovsepyan, Didier.Delanote, Stefan.VanBaelen, Yolande.Berbers, Wouter.Joosen}@cs.kuleuven.be

Abstract. We have devised an approach, called Generic Upsilon Transformations, which supports and promotes reusable model transformations. As a proof-of-concept we have developed an ATL-based tool which realizes the concepts proposed in our framework. Even though the main focus of GUT is reusable design patterns, our approach is generic enough to support model transformations for embedded systems. We have successfully applied our approach on an embedded case study.

1 Model Transformations: State-of-the-art

The development of embedded systems is very difficult. The Y-chart approach is known as a suitable MDD flow to cope with the complexities. It advocates the separate development of software and hardware architecture models. The two models are afterwards merged using an allocation model which specifies the mapping of software to hardware components. As a result an allocated model is obtained which can be further refined to a platform specific model and source code. MDD supports this process by providing a technology for automatic model transformations.

In current MDD practices transformations are usually implemented as black-boxes. A typical black-box consists of a number of patterns which should transform a given source model according to those patterns. The black-box also includes a mapping/matching mechanism which directs the transformation engine how to apply this pattern on a set of matched elements. A source model is annotated with UML stereotypes and tagged values which are used by the transformation engine to identify a match and map it to a corresponding pattern (fig. 1).

This a black-box approach has a number of disadvantages. Transformations cannot be easily combined because the combination of 2 black-boxes may produce an unpredictable target model. Transformation patterns are tangled and woven in transformation implementation. As a result the relationship between the source and target models is hidden. Transformations are often written in a complex language which allows only transformation experts to modify them. Most of the current transformation languages support limited or no parameterization. Input models are often "polluted" by stereotypes and tagging information which is only used by the transformation engine.

2 Generic Upsilon Transformations (GUTs)

In our approach we try to solve these issues by introducing a conceptual framework for representing model transformations as graphical models themselves. We can see each transformation as the Greek letter upsilon, hence the name of the approach. The left and right branches of the upsilon contain two source models (fig. 2). Along with the two source models we also provide a *Mapping Model* that instructs the model transformation engine how elements from the *Source Model 1* map on elements from *Source Model 2*. *Source Model 1* and *2* are standard UML2 class models. The

* The described work is part of the EUREKA-ITEA MARTES project, and is partly funded by the Flemish government institution IWT (Institute for the Promotion of Innovation by Science and Technology in Flanders).

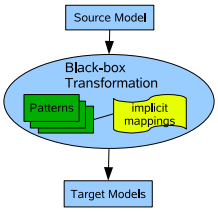

Figure 1. Traditional transformations

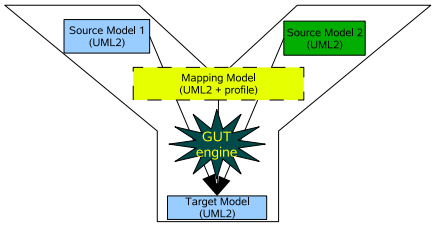

Figure 2. Generic Upsilon Transformation

Mapping Model is a UML class model using a dedicated GUT profile that allows to express different mapping strategies. The *Mapping Model* cannot exist on its own, but only contains links to the two source models. In order to realize different mapping strategies, the mapping links are stereotyped and may use tagged values for additional mapping parameterization.

The GUT approach can be used in different MDD flows. It is suitable for embedded system development using the Y-chart approach where approach can be used in two scenarios.

The first scenario realizes the merging of the software and hardware architecture models represented as *Source Model 1* and *Source Model 2* respectively. The *Mapping Model* specifies the allocation of software entities to hardware components.

In the second scenario GUT can be used to realize refinement transformations on any given model (software, hardware architecture, allocated system, etc.). We believe that refinement transformations often follow a certain pattern, e.g., software design, architectural, hardware refinement, etc. Hence we represent the model as *Source Model 1* and the pattern as *Source Model 2*. The *Mapping Model* specifies how the pattern model is applied on the source model by defining all pattern parameters and their bindings.

Currently we have applied our approach on a producer-consumer embedded system case-study that was defined by CoFluent. We have started from system requirements and modeled software and hardware aspects of the system separately. Then we have applied refinement transformations to each of the models. Afterwards we have merged the two models to obtain an allocated system model. Finally we have applied refinement transformations to the system model and generated code for the software part.

3 GUT Tool

The GUT tool is implemented using the Eclipse technology. All source models are provided via the EMF framework using the Ecore implementation of UML. The transformation logic is written in several ATL modules. These modules contain rule sections which handle each mapping strategy in our approach. Each element is modified according to the mapping it is involved in. Elements that are not referenced in the *Mapping Model* are copied to the *Target Model*. The GUT tool is extensible since new mapping strategies can easily be added to the GUT profile and implemented in an ATL module.

4 Future Work

As a future work we are looking to build libraries of patterns and a methodology to easily incorporate patterns from these libraries in model transformations.

Our approach is very related to the Y-chart framework often used in embedded system modeling. We are investigating to what degree GUT can support embedded system development using the Y-chart framework. We will validate our approach on a broader embedded system case-study.