1 Introduction

Increasingly, software needs to dynamically adapt its behavior at runtime in response to changing conditions in the supporting computing, communication infrastructure, and in the surrounding physical environment [6]. Self-adaptive software systems are able to reconfigure at run-time to adapt to changes. The implementation of ad-hoc solutions to cover all possible system configurations and reconfigurations is not feasible. Dynamic Software Product Lines (DSPLs) provide a systematic basis for the engineering of self-adaptive systems [4]. A key characteristic in DSPLs is the intensive use of variability at run-time in order to adapt the system configuration caused by an environment change. Following this approach, a self-adaptive system can be seen as a family of feasible system configurations with a mechanism to move from one configuration to another. The development of self-adaptive systems involves great complexity and becomes a tedious task. We propose Moskitt4SPL (M4SPL) an open-source tool to ease the development of self-adaptive systems. In this tool, we combine model-driven and DSPLs to better cope with the complexities during the construction of self-adaptive systems. M4SPL can be used for modeling systems which make use of variability at run-time in order to adapt the system configuration caused by an environment change. M4SPL provides edition capabilities for Feature Models, Configuration Models and Resolution Models which are part of a self-adaptive system specification. Furthermore, M4SPL incorporates a series of refinements to automatically ensure interesting behavior issues in adaptation specifications. Dealing with those issues before execution is essential for reliable self-adaptive systems that fulfill many of the user’s needs. M4SPL can be used standalone as an Eclipse plug-in or integrated in the MDE MOSEKitt environment.

2 M4SPL Tool: Overview

Moskitt4SPL (M4SPL) is a free open-source tool which gives support for modeling self-adaptive systems. M4SPL is based on Eclipse plug-ins: Eclipse Modeling Framework (EMF) [1], Graphical Modeling Framework (GMF) [2] and Atlas Transformation Language (ATL) [5]. The major novel feature of this tool is the application of model-driven and product-line engineering for designing self-adaptive systems. M4SPL provides different editors to ease the specification of self-adaptive systems:
– **Feature Model Editor.** Feature Models are a widely used notation to capture the variability of a system in terms of features. Feature Models describe, in an intentional way, the possible configurations in which the system can evolve. M4SPL includes a graphical Feature Model Editor. The Feature Model Editor is based on a previous tool called Mokitt Feature Modeler [3]. Figure 1 (left) illustrates a screenshot of the Feature Model Editor.

– **Feature Model Configuration Editor.** From the defined Feature Model, designers can create Configuration Models. A Configuration Model represents a feasible configuration of the Feature Model. A configuration is defined by the set of states of each feature in the Feature Model. The possible feature states are: active, inactive or discarded. A graphical editor is provided to describe possible configurations of a Feature Model. Figure 1 (right) illustrates the environment of the editor. Features are represented with different colors depending on their state: green features represent active features, red features represent inactive features and orange features represent discarded features.

![Fig. 1. Feature Model Editor (left). Configuration Model Editor (right)](image)

– **Resolution Model Editor.** Resolutions define reconfigurations among the different system configurations in a declarative manner. Each Resolution is associated with a context condition and represents the sequence of actions in terms of feature activation/deactivation produced in the system when a context change occurs and the condition is fulfilled. An editor is provided to support the definition of the Resolution Model using the EMF capabilities. Figure 2 (left) shows the environment of the editor.

One of the main reasons for following a MDE development is that it is focused on automation. Models can be transformed automatically into new models or code by means of model transformation techniques. This enables automation in system development. In order to automate the design method, we have implemented a series of model-to-model transformations using ATL:

– **Adaptation Space Generation.** Using the tools presented above, designers can define the initial self-adaptive system specification by means of: a Feature Model,
an initial Configuration Model and a Resolution Model. This system specification can be automatically transformed into a State Machine Model that represents the implicit Adaptation Space of the system. The state machine representation eases the analysis of the system behavior. The Adaptation Space contains all the feasible configurations that the system can reach through execution, and the reconfigurations among them. The configurations are graphically represented as states and the reconfigurations are graphically represented as transitions. Each state has associated a Configuration Model, and each transition has associated a Resolution of the Resolution Model. Figure 2 (right) shows a screenshot of the Adaptation Space generated by M4SPL from a specification. Once the State Machine Model has been obtained, several refinements can be applied successively to guarantee properties about the adaptive behavior at run-time. Such refinements are made by means of rules implemented in the ATL. All the transformations take as input two models: the State Machine Model and the Resolution Model; and generate a Refined State Machine Model and a Refined Resolution Model.

Refinements. M4SPL enables to analyze the Adaptation Space and to automatically refine the model specifications to ensure behavioral issues. M4SPL supports five refinements:

- **Determinism**: M4SPL implements a refinement to ensure that the system is determinist. The refinement guarantees that from a given state, when a determined context condition is fulfilled, the system can only reconfigure to one destination state. This refinement modifies the Resolution Model in order to avoid simultaneous reconfigurations in the system.

- **Reversibility**: M4SPL implements two model refinements to achieve a specification (1) with rollback capabilities or (2) without rollback capabilities. The purpose of the first refinement is to ensure that, for all configurations contained in the Adaptation Space, exist a reconfiguration that leads directly to the previous configuration. The refinement generates new resolutions which define compensation actions to reverse all the reconfigurations. The purpose of the second refinement is to ensure that there is no reconfiguration that leads di-
directly to the previous configuration. This refinement modifies the Resolution Model in order to avoid reversible reconfigurations

- **Remove Redundancy**: M4SPL provides a refinement to guarantees that the specification does not contain duplicated behavior. This refinement eliminates redundant reconfigurations in a self-adaptive specification. Two (or more) reconfigurations are redundant if they evolve the system from the same source configuration to the same target configuration. The refinement, first finds redundant resolutions. Then, it selects the simplest resolution and removes the others. We consider the simplest resolution as the one that involves the minimum number of change actions. The elimination of redundancy can actually improve execution time and understandability of the system behavior.

- **Remove Loops**: M4SPL implements a model refinement to remove loops in the system specification. The refinement modifies the Resolution Model in order to avoid reconfigurations that evolve from a source configuration to the same configuration.

M4SPL also provides some metrics about the specification (i.e., number of active features, number of configurations, number of reconfigurations etc). M4SPL is available online at: [http://www.pros.upv.es/m4spl/](http://www.pros.upv.es/m4spl/).

**Acknowledgments.** This work has been supported by Conselleria d’Educació of Generalitat Valenciana under the program VALi+d and by Ministerio de Ciencia e Innovación (MICINN) under the project EVERYWARE TIN2010-18011.

**References**

Aplicando los principios del DSDM al desarrollo de transformaciones de modelos en ETL

Álvaro Jiménez, Verónica A. Bollati, Juan M. Vara, Esperanza Marcos

Grupo de Investigación Kybele,
Universidad Rey Juan Carlos, Madrid (España).
{alvaro.jimenez,veronica.bollati
juanmanuel.vara,esperanza.marcos}@urjc.es

Resumen Las transformaciones de modelos son uno de los principales artefactos en el Desarrollo de Software Dirigido por Modelos. Sin embargo, a pesar de ser otro artefacto software más, existen pocas aproximaciones que apliquen los principios del DSDM a su desarrollo. En este trabajo presentamos una aproximación para el desarrollo de transformaciones de modelos dirigido por modelos para el lenguaje Epsilon Transformation Language (ETL). Para ello, presentamos un metamodelo para el lenguaje ETL, una transformación que permite obtener un modelo ETL a partir de un modelo de la transformación de alto nivel y la generación del código ETL que implementa la transformación.

Palabras Clave: Desarrollo Dirigido por Modelos, Transformaciones de Modelos, Modelos de Transformación, ETL, ATL.

1. Introducción

En el contexto del Desarrollo de Software Dirigido por Modelos (DSDM, [27], [30]), las transformaciones de modelos actúan principalmente como elementos de enlace entre los diferentes pasos del proceso de desarrollo, refinando los modelos de alto nivel en modelos de menor abstracción, hasta que estos puedan ser ejecutados/interpretados o convertidos en código ejecutable. Además, las transformaciones de modelos pueden emplearse para llevar a cabo otras tareas como la sincronización o la migración de modelos [24],[31]. Por ello, independientemente del objetivo con el que hayan sido construidas, las transformaciones de modelos juegan un papel clave en cualquier propuesta relacionada con la Ingeniería Dirigida por Modelos (Model-Driven Engineering, MDE) [26].

Como prueba de ello, durante los últimos años ha surgido un gran número de lenguajes y herramientas que dan soporte al desarrollo de transformaciones [6], [9], [29]. Estos lenguajes y herramientas difieren en múltiples aspectos, como la aproximación que adoptan (declarativa, imperativa, híbrida, basada en gráficos, etc.) y esta diversidad trae consigo una complejidad adicional en el desarrollo de transformaciones como la de la selección del lenguaje, el tiempo de aprendizaje, la migración, etc.