
Slimane Hammoudi, ESEO, Ecole supérieure de l’Ouest Angers, France
Wajih Alouini, FST, Faculté des Sciences de Tunis, Tunisia
Denivaldo Lopes, Federal University of Maranhão, São Luis, Brazil
Marianne Huchard, Université de Montpellier II, France

ABSTRACT

Model Driven Engineering (MDE) has been proposed for supporting the development, maintenance and evolution of software systems. Model Driven Architecture (MDA), Software Factories, and the Eclipse Modeling Framework (EMF) are representative MDE approaches. These MDE approaches have concepts and techniques in common such as modeling, metamodels, and model transformation. Today, it is well recognized that model transformation is the most important of MDE approaches and is one of the most important operations in MDE. Despite the multitude of model transformation language proposals emerging from university and industry, these transformations are often created manually, which is a tedious and error-prone task and therefore an expensive process. In this paper, the authors propose an extended architecture that aims to semi-automate the process of transformation in the context of MDA. This architecture involves concepts and techniques of metamodel matching, inspired and borrowed from schema matching techniques that are an important issue in the database and ontology domain. This architecture is enforced by a methodology which details the different steps leading to a semi-automatic transformation process. Matching foundations and the authors’ approach for metamodel matching are discussed.

Keywords: Architecture and Methodology, Mapping, Matching, MDA, MDE, Transformation

INTRODUCTION

The main motivation behind Model Driven Engineering (MDE) is to transfer the focus of work from programming to modeling by treating models as first class entities and consequently the primary artifacts of development. Nowadays there are several approaches based on MDE principles, the most well known being Model Driven Architecture (MDA) (OMG, 2007) by the Object Management Group (OMG), Software factories by Microsoft (Dominguez, 2006).
or the Eclipse Modeling Framework (EMF) by IBM (Eclipse, 2004). In the literature, several issues around MDE have been studied and subjected to intensive research, e.g., modeling languages (Bezivin, 2004; Blanc, 2005), model transformation languages (Jouault, 2006; OMG, 2005), mapping between metamodels (Hammoudi, 2005b; Lopes, 2005a), and design methodologies (Almeida, 2006). Among these issues, model transformation languages occupy a central place and allow the definition of how a set of elements from a source model are analyzed and transformed into a set of elements in a target model. However, these transformations are created manually, often a tedious and error-prone task, and therefore an expensive process. These transformations consist of creating a set of rules involving, and at the same time merging, mapping and transformation techniques between two metamodels. A semi-automation of the transformation process leads to a real challenge allowing many advantages: It enhances significantly the development time of transformation and decreases the errors that may occur in a manual definition of transformations. In Hammoudi (2005a) and Lopes (2005b), the authors have initiated a first attempt towards this semi-automation. They introduced an approach separating mapping specifications from transformation definitions, and implemented this approach in a tool called Mapping Modeling Tool (MMT). In this first approach, a mapping specification was created manually to define the relationships between metamodels (i.e., equivalent metamodel elements), with the transformation definition being generated automatically from the mappings.

In this paper, we propose to push the semi-automation process one step further by using matching techniques (Kappel, 2007; Hong, 2007), to semi-automatically generate mappings between two metamodels. The produced mappings could then be adapted and validated by an expert for the automatic derivation of a transformation model, as a set of transformation rules. The contributions of this paper are three-fold. First, we propose a new architecture for the transformation process in the context of MDA. This new architecture extends the standard OMG model transformation architecture (OMG, 2005), and introduces mapping and matching as first class entities in the transformation process, represented by models and metamodels. Second, we develop a methodology which illustrates the different steps of the semi-automatic transformation process. These steps are distinguished between expert user activities for the generation of transformation rules and designer (or software engineer) activities for the derivation of a final platform specific model. Third, we implement a Plug-in under the Eclipse framework called Semi-Automatic Matching Tool for MDE (SAMT4MDE). This tool supports all the main steps of our approach: from matching between metamodels to the generation of transformation rules. An evaluation of our matching algorithm between metamodels is done through match quality measures proposed for schema matching in databases.

This paper is organized as follows: First, we introduces the MDE/MDA approach and presents its most common scenario of transformation in MDA. Next, an extended architecture for a semi-automatic transformation process and discusses the matching and mapping metamodels as two important components in this process are discussed. We then present the methodology which in the first part, starts with the presentation of the main users involved in an MDA project. The second part of this section details the steps of our methodology distinguishing two main activities: preparation and execution activities. Next, a review of the main steps of the transformation process according to two main criteria: how the steps are achieved and who is responsible for their achievement. Finally, we present our approach for metamodel matching through an algorithm and presents its implementation using a plug-in for Eclipse. The algorithm developed in this paper is evaluated using UML and C# metamodel matching on a test case. In the same section, we also provide a quantitative evaluation through match quality measures.