OntMDE approach: Enhancing AIM with architectural ontology

NOURCHÈNE ELLEUCH, ADEL KHALFALLAH

Department of software engineering,
Higher Institute of Computer Science, El Manar,
2 Rue Abou Raihan El Bayrouni-2080 Ariana
TUNISIA

Abstract: Model-driven engineering (MDE) is a software development approach that mainly focus on models design and transformation. Models are basically built to represent different software system views and could be refined and evolved into lowered abstract level. This paper is a first investigation of the model-driven paradigm enactment using established architecture-centric concepts. His goal is to provide an OntMDE approach for model transformations in the context of MDE. The OntMDE attempts to combine both architecture-centric and model-driven paradigms. It is based on two ideas. Firstly, the architecture of model is independent from any platform implementation. Therefore, the architectural considerations that are independent of platforms’ considerations must be dealt with at Platform Independent Model (PIM) level. For that, PIM will be decomposed into two models: an Architecture Independent Model (AIM) and an Architecture Dependant Model (ADM). Secondly, domain ontology is used as marking technique to label the AIM and architectural ontology description is provided to generate an ADM.

Key-Words: MDE, ontology, architectural styles, model transformations.

1 Introduction

The change of paradigm in software engineering from object-oriented to model-oriented will have an important impact on the way in which software system will be built and maintained. Model Driven Engineering (MDE) [1][2][4][10][16] tackles the elusive problem of system development by promoting the usage of models as the primary artifact to be constructed and maintained. The system, developed following the MDE approach, is first described by a model at a very high level of abstraction, i.e., ignoring any kind of technology-related dependencies. Such model is represented by a Computation Independent Model (CIM). In order to make the model more platform-specific, a series of transformations may be performed. The model transformations can be classified in two categories [5]: Vertical model transformations are used to refine or to abstract a model during forward or reverse engineering. The higher level models are transformed into the lower level models, e.g. CIM to Platform Independent Model (PIM) or PIM to Platform Specific Model (PSM). The knowledge about platforms is encoded into platform description models (PDM). To explicitly define PDM makes this knowledge reusable. Horizontal model transformations however, don’t affect the abstraction level of a software model. They are mappings between models at the same level of abstraction and they are used to restructure or optimize a software model in order to improve its internal structure and/or quality. We think that another category must be defined: Oblique model transformations, which mix horizontal and vertical transformations. The oblique model transformations introduce a series of model’s optimization before code generation.

In this paper, we propose a new approach to transform PIM into PSM: OntMDE approach. It is an oblique model transformation and is based on two ideas: Firstly, the architecture of model is independent from any platform implementation. Therefore, the architectural considerations that are independent of platforms’ considerations must be dealt with at PIM level. We decompose the PIM into two levels: Architecture Independent Model (AIM) and Architecture Dependant Model (ADM). AIM is considered as a pure domain model and ADM combines the specifications in the AIM with the details that specify how that system uses a particular style of architecture. Secondly, to make transformation more and more automatic; we investigate the possibilities of applying techniques that are developed for knowledge-based systems modeling and configuration to mark AIM and to guide ADM’s generation. We define two ontologies: domain ontology is used as marking technique to label the AIM and architectural ontology description is provided to generate an ADM. Thus, this
paper emphasizes different aspects, which are, in our view, essential to develop reusable models:

- It offers a user-friendly vision to architects by providing re-usable architectural models by using architectural ontology;
- It hides the complexity of the final execution platform through abstraction models (CIM, AIM, ADM, and PSM);
- It promotes design and model re-uses to facilitate further developments e.g. the distinction between AIM and ADM can improve the development process by re-using AIM. In fact, several architectural styles applied in the same AIM, produce different ADMs.

The paper is organized as follows: section 3 introduces the MDE approach and the notations used to formalize architectural styles and their limitations. Section 4 describes our approach based on the architectural and domain ontology. Section 5 provides an overview over initial results. Finally, we will end up with a discussion and a presentation of the perspectives and areas for future improvement.

2 Model-driven and architecture-centric approaches

2.1 The MDE approach
The MDE knows growing interest as much as a research domain as an industry process for building software quickly and reliably. It is being considered as an important departure from traditional techniques in such areas as software engineering, system engineering and data engineering. The MDE, probably derived from the OMG Model Driven Architecture MDA initiative [17][21], provides a framework for software development and tackles the problem of system development by promoting the usage of models [23] as the primary artifact to be constructed and maintained.

The MDE’s aim is based on separation of concerns. It allows separation of platform dependent from platform independent aspects in software construction and maintenance in order to alleviate the complexity of platforms, improve the portability, increase the productivity, and express domain concepts effectively [7]. For that, it uses models to capture specific, business and platforms aspects of a system under construction or maintenance and uses model transformation approaches to enhance, refine and transform models.

Some of the model driven engineering approaches [1][2][3] provide languages and tools to transform models by means of transformation rules in order to describe and design complex systems at a high level of abstraction. The main objective of such methods is to hide the complexity and constraints induced by the target execution platform, during the design phase. Thus, an architect can mainly focus on functional requirements of his application rather than on non-functional ones. Moreover, this approach doesn’t separate between architecture and platform aspects what can complicate the model transformation process and decrease the models’ re-use. Manset and all [25] propose a new approach called gMDE based on the concept of model-driven software engineering applied to the Grid application domain, which addresses the challenge of designing, optimizing and adapting Grid-abstract architectures in order to automatically generate a complete set of Grid services to be deployed on a physical grid of resources. The gMDE approach addresses functional and non-functional aspects by proposing several models that represent constraint, platform independent, concrete and physical views (see [25] for more details).

According to the OMG’s vision, when mapping system models to concrete platforms (PIM to PSM), it is often necessary to include model to model transformations to fill the gap between the abstract description and its concrete representation. For that, models being decoupled from architectural concepts, system descriptions remain relevant and re-usable. Providing two kinds of artifact’s output for model transformations (model and/or code) makes the approach more modular and also facilitates the final source code generation. In order to support this approach, we enhance the model-driven engineering with architecture concepts. Thus, we study the languages used to model architecture knowledge, in order to choose a well-formalism.

2.2 The architectural modeling languages
A number of common best practices, for instance architectural patterns and styles, can be used to model software architecture. The architectural styles contain condensed skeletons of the architectural knowledge gained by experienced software designers, and provide a way to reuse that knowledge [11]. Several common architectural styles are described in [8][24] i.e. Client-server, blackboard, etc., and several notations can be used to model styles. Störrle defines in [13] three classes of architectural modeling languages. The first class is grouping architectural description languages (ADLs) [6][12], which feature formal semantics but lack notational support. The second class is composed of industry-oriented software design languages, for example UML or Specification and Description Language (SDL). In one hand, these languages are offering powerful tools and development processes but in the other hand they lack of precise definitions of architectural design. The third class is a set of UML
extensions.

The ADLs allow for specification of abstract components and their interactions. They also offer formal semantics to model software architecture and architectural styles. The ADL’s problem is recurrent in formal method. In fact, using formal methods requires a strong theoretical user background. Moreover, the model developed with ADL has textual or ad-hoc graphical notations and lack of a legibility. These elements are considered as barriers to the ADLs spread among developers and practitioners. UML too have many deficiencies; it lacks of adequate concepts, notations and tools for describing software at the architecture level (see [9][14][19][22] for more details). Many researchers [14][18] working to integrate ADLs and UML. They made several propositions to map ADL concepts into UML ones, but none of them provided a real embedded UML meta-model. These propositions are limited to translate the main ADLs’ concepts into class, component and package marks and they lack of a well-defined semantics.

A good formalism should have four features to describe architectural styles. (1) A comprehensive and adequate set of concepts for all relevant aspects of architecture; these concepts must be equipped with (2) a convenient syntax (preferably a visual notation); (3) a precise semantics (preferably a formal one); and (4) a process support.

3 Our OntMDE approach

By convention and in order to separate clearly the concept described here from that of the OMG MDA, we call our approach an Ontology-based model-driven engineering approach (OntMDE) and use an architecture-specific terminology. Following our OntMDE paradigm, we address the challenge of designing, optimizing and adapting software architectures, in order to automatically generate a PSM. For that, we build a set of models and we investigate their orchestration among the OntMDE design process. Moreover, we describe how ontology can be used to mark model and to define architectural styles.

3.1. The OntMDE Key Models

Semi-automated or automated model transformations become a crucial aspect in the MDE context. The OntMDE is an oblique model transformation and is based on two ideas: firstly, by proposing several models, our approach separates concerns, mainly between architecture dependent and architecture independent parts of software systems, and addresses different aspects of architectural styles. Secondly, our approach needs to use ontologies to guide and to automate the transformation. We strongly believe that modeling a system according to a specific architecture is completely platform-independent. Moreover, the architectural considerations that are independent of platform’s considerations must be dealt with at PIM level. For that, we decompose the PIM level into two models: the architectural independent model (AIM) and the architectural dependent model (ADM). Both of them are system’s model that doesn’t have any technology-specific implementation information. AIM is considered as a pure domain model that exhibits a specified degree of architecture independence so as to be suitable for use with a number of different architectures. ADM is represented a refined AIM in which technology independent architectural consideration are introduced. This increases the reusability of the AIM and its adaptability to changes. In fact, when a change occurs either at a high level of abstraction (e.g., a change in system requirements) or at a low level of abstraction (e.g., moving to another architecture), its impact will be localized. The input to the OntMDE is the AIM and the mapping. The result is the PSM and the record of transformation. To prepare AIM-to-PSM transformation, we must to mark AIM and to define architecture styles. We use ontology to mark AIM and to partially automate a transformation of AIM into ADM (see Fig 1).

![OntMDE models and ontologies](image)

We define two kinds of ontology: domain ontology and architectural ontology. The domain ontology is used to define a set of marks in the goal to guide the transformation of AIM into ADM. The architectural ontology is used to describe the system architectural aspects.

3.2 The OntMDE development process

The OntMDE separates the specification of system architecture from the specification of the implementation of that architecture on a specific technology platform. The OntMDE approach envisions mappings from AIM to one or more ADM and ADM to one or more PSM. The system design with OntMDE approach is based on
three views: business, architecture and platform views. The business view or domain view focuses on the operation of a system while hiding the details necessary for a particular architecture. The business view shows that part of the complete specification that does not change from one architectural style to another. The architecture view focuses on the architectural ontology description. It provides, for use in an ADM, concepts representing the different kinds of elements to be used in specifying the use of the architectural style by an application. The platform view defines a platform model that provides a set of technical concepts, representing the different kinds of parts that make up a platform and the services provided by that platform. It also provides, for use in a PSM, concepts representing the different kinds of elements to be used in specifying the use of the platform by an application.

These views are linked to different tracks of Thales’s double 2 Tracks Unified Process (2TUP) [4]. The OntMDE’s roadmap is composed of five steps:
- Create a set of marks defined by domain ontology as part of knowledge base;
- Build a series of PIM’s refinements (horizontal model transformation). The set of marks are used to mark elements of the AIM in order to guide the AIM’s transformation. The output of this refinement is a marked AIM;
- Create an architectural ontology in order to describe architectural styles. The architectural ontology is be saved into knowledge based system;
- Transform the marked AIM into the ADM by using architectural sub-ontology (horizontal model transformation).
- Produce the PSM by using platform description model (PDM) and ADM (vertical model transformation).

Figure 2 introduces the orchestration of the previously presented models inside the OntMDE design process. In the depicted process, a distinction is made between two major levels, one is the architecture level of transformation and the other is the implementation level of transformation. Models and transformations can differ in nature and objectives. Thus models can be of two distinct types; either the model is manually created (CIM, AIM) or it is automatically obtained by transformation (marked AIM, ADM, PSM). Transformations can then be of two different types; either the transformation is a composition of one or more refinement actions (model to model transformation e.g. AIM-to-Marked AIM, AIM-to-ADM and ADM-to-PSM) or it is a translation mapping (model to source code transformation e.g. ADM-to-Executable PSM).

4 Initial Results
By introducing the new models, the software architect can specialize a software architecture progressively. Once the system architecture complies with the expressed requirements, the software architect can specify a platform for code generation.

The transformation of AIM into ADM can be partially automated by enriching AIM with marks. The marks prepare the transformation, and indicate the role of marked AIM elements in mapping between AIM and ADM. When an AIM element is marked, it indicates that a certain mapping has been done to transform that element into one or many other elements in the ADM. In OntMDE approach, these marks are defined by domain ontology [15][20], which models a specific domain. The CIM can be used to develop domain ontologies that should be saved in knowledge based system for further reuse for model transformations. The domain ontology provides a semantic for the marking concepts. This semantic is to be exploited for automating transformations and it is considered as a new knowledge, which would be used to mark the AIM. For example, Global Position System (GPS) domain ontology, within external operating context, would model the steps that a user should follow to successfully operate a GPS system. But, GPS domain ontology, within internal operating context, would model the internal functioning mechanism of the GPS system.

Formalism has to be chosen to describe the architectural styles for the sake of transforming a marked AIM into ADM. This choice must be in conformity with the four features listed in section 2.2. In fact, ontology...
could provide a useful representation of different aspects of software architecture. In this research, we suggest a new ontology called ‘architectural ontology’, which will target the creation of ADM. The architectural ontology is based primarily on description of the basic architectural design elements and the hierarchal relationships among them. The architectural design elements are composed by different components, connectors, ports, roles, properties and systems (see Fig 3.). The architectural ontology regroups all architectural styles. The transformation of marked AIM into AMD required the choice of an architectural style. For that, it is necessary to select an architectural sub-ontology.

Fig 3.: The architectural ontology concepts

5 Conclusion

The efficiency of the approach relies strongly on the correctness of these models; consequently great care is being taken to ensure this. As a proof of concept, the engineering framework must be developed (OntMDEnv) to enact the combination of architecture-centric and model-driven approaches introduced previously. Since this approach is based on the concepts of re-use, architecture independence and execution platform independence, our engineering framework must be no limited to the specific-domain.

To provide the OntMDEnv that will implement our OntMDE approach, we evaluated several model transformation languages and approaches. Future work will focus, firstly, on investigating how model transformations can use ontologies to automate the major transformation steps. A binding between metamodels and ontologies can be done. Secondly, on choosing which, a description language of ontologies can be used to build domain and architectural ontologies.

References:


