

Model-driven development of embedded systems on OSGi platforms

Julio Cano, Natividad Martínez Madrid, Ralf Seepold
Universidad Carlos III de Madrid
{julioangel.cano, natividad.martinez, ralf.seepold}@uc3m.es

Fernando López Aguilar
Telefónica I+D
fla@tid.es

Abstract

Large and complex systems design is still being a challenge even bigger when developing embedded, distributed or real-time systems. OSGi is a platform created to reduce some of the software design problems, increasing reusability modularity, etc. This paper describes a methodology based in MDA that aims at real-time embedded systems, The approach is based on a target platform using OSGi and thus reducing applications development time and complexity.

Keywords

Model-Driven Development, Metamodel, Modelling Language, Model Transformation, MDA, Transformation Language, UML, Web Engineering, XSLT, XMI, XML, OSGi.

1. Introduction

MDA[1] (Model-Driven Architecture) is nowadays a well known software design approach that emphasizes the separation of PIM (Platform Independent Models) and PSM (Platform Specific Models), following the MDD (Model-Driven Development) proposal of the OMG (Object Management Group).

The MARTES[2] (Model-based Approach to Real-Time Embedded Systems development) project tries to create a new approach based on MDA having as target the domain of real-time embedded systems.

Different software platforms need to be included and thus the use of these software platforms can make the application more independent of the final hardware platform.

In this paper it is described how the MARTES project meta-models can be extended to create an OSGi profile so that the MARTES process can be applied to generate code for this platform.

The next subsection presents the OSGi platform and related approaches before the MARTES meta-models are described. Afterwards the modelling meta-models are described.

1.1. OSGi

OSGi[3] (formerly known as Open Service Gateway initiative) is a platform developed in Java where multiple components of an application can be deployed, activated and updated independently. It also allows remote download and management of components or services.

Among other domains OSGi is, for example, used for home service networks providing an easily configurable platform for service provisioning.

Home networks also include different kind of platforms like PDAs, mobile phones, gateways, set-top-boxes, etc. A methodology based in MDA is encouraged here, where a platform independent model can be used for the application design so that the same components can be allocated to different target platforms without changing the application design.

At the Polytechnic University of Valencia (PUV), a methodology based in MDA is proposed to develop pervasive systems ([4][5][6]). A language called Perv-ML (Pervasive Modelling Language) is used for the PIM level containing pervasive systems semantics.

OSGi is used as the main target platform, so an OSGi meta-model is created for the PSM level and code generation.

This OSGi meta-model is also used as a basis for this work. The meta-model, containing OSGi concepts, is integrated into the Java meta-model [5] so that Java code can be easily generated for every OSGi element. The meta-model has been modified and extended to include some concepts coming from the PIM level used here.

1.2. MARTES Project

The MARTES project is part of the European ITEA programme.

The main goal is the definition, construction, experimentation, validation and deployment of a new methodology based in models and a new interoperable toolset for Real-Time Embedded Systems development, and the application of these concepts to create a development and validation platform for the domain of data stream dominant applications on embedded heterogeneous platforms architectures.

UML 2.0 is used as modelling language and MDA as the main methodology. Java and SystemC 2.0 are used as the MARTES languages.

The approach of the methodology is based in the Y-chart model, mapping a model of application onto a model of platform and simulating or generating code of the resulting allocated code (Figure 1).

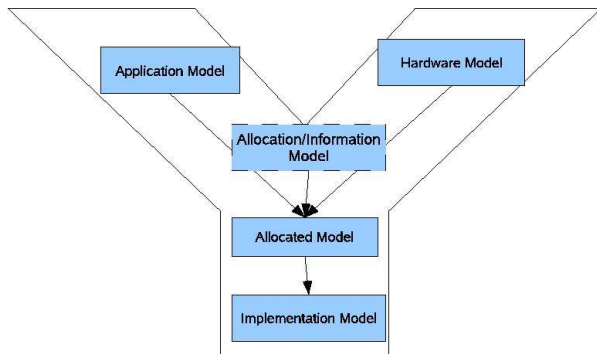


Figure 1: Y-Chart MARTES Process Model

The Application meta-model contains different application views, like functional modelling (structural and behavioural), execution modelling and workload modelling. This model is independent of the target implementation platform.

The Execution Platform meta-model comprises hardware and software component descriptions including connections needed for application purpose. Similarly to the Application Modelling, hierarchical and repetitive structures may be used to explore the topology and parallelism.

The Allocation meta-model associates Application Components Platform Components used in the Execution Platform model, having as a result the Allocated model.

Finally, the implementation oriented meta-model represents the transformation of the components in the Allocated Model having as a target a specific model: simulation, software and hardware code generation, etc. The Allocated Model is transformed into a Platform Specific Model. Generic components in the Allocation model are replaced by implementation level elements that use the target platform API.

MARTES defines no specificImplementation meta-model. A different Implementation meta-model is needed for every target domain.

The work presented in this paper applies the MARTES methodology to software execution platforms, like OSGi, extending both the PIM, PSM and implementation models, and defines the necessary transformations among models. The models are described in section 4 and the transformations are described in section 5. Finally, the conclusions are stated.

2. Meta-Models

This section describes how MARTES meta-models have been extended and adapted mainly to the OSGi

domain. Application Component, Platform and Allocation meta-models had to be extended to include the needed semantics. Application meta-model was not modified as it is completely platform independent. The Implementation meta-model was completely generated based on the UPV meta-model.

2.1. Application Component Meta-Model

Figure 2 shows how application components have been modelled for the OSGi profile. Other platforms have been included too, like J2EE and J2ME although they are not described in this paper. This is done to show how an application component can be platform independent.

Application components have services associated. These services will be provided to the rest of the application. The implementation of the services will depend on the target platform and the way the other components will communicate to access these services.

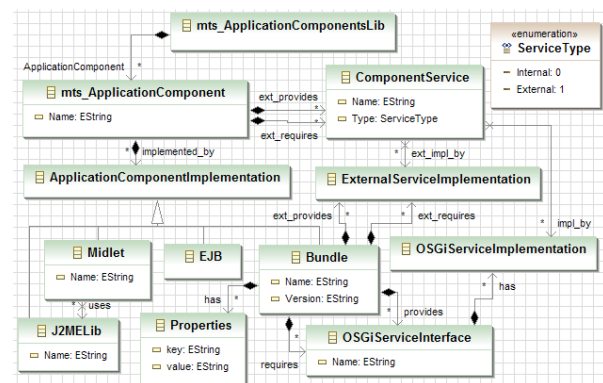


Figure 2: Application Component Meta-model

Application components library will not only contain the component's specification but the different available implementations. In case of working with OSGi as target software platform the implementation component will be a Bundle. A Bundle is an OSGi software component including all the Java classes needed to implement and provide services, grouped in a jar file.

The Bundle contains the associated properties and services. OSGi service interfaces and implementations are differentiated depending on whether these services are provided or only required by the bundle. Services not provided to the OSGi platform but to the rest of the system are considered to be *external* services and are implemented in a different way.

An Application components library can contain components implementations, but implementations can be added later in the development process. If the application component has an implementation when it is assigned to a platform component then that implementation is selected, otherwise the corresponding implementation elements can be generated for the implementation model.

2.2. Execution Platform Meta-Model

The MARTES Platform meta-model components adaptation is shown in Figure 3. Components used here are mainly high level representations of hardware platform components, like PC, PDA, mobile phones, etc.

The layer of software Execution Platform has been extended to include other platforms like OSGi, J2ME and J2EE, so that every application component can be allocated to the desired target platform in the Allocated Model.

Two of the main OSGi platform implementations have been represented. These are Oscar [7] and Equinox [8]. The OSGi platform could have been derived from J2SE, indicating that the target platform will be specifically coded in Java, making a difference between possible OSGi platforms programmed in different languages (although no OSGi platforms have been programmed in a different language than Java).

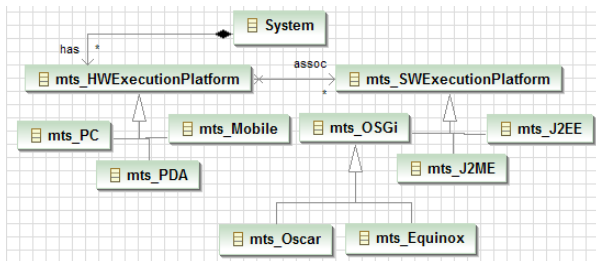


Figure 3: Execution Platform meta-model

2.3. Allocation Model

An example of Allocation Model is shown in Figure 4, indicating how rules would be applied and how application components can be assigned to platform components to the Allocated Model.

In this case it can be seen how a Bundle can be assigned only to an OSGi platform, while Midlets and J2ME libraries can only be allocated over the J2ME platform.

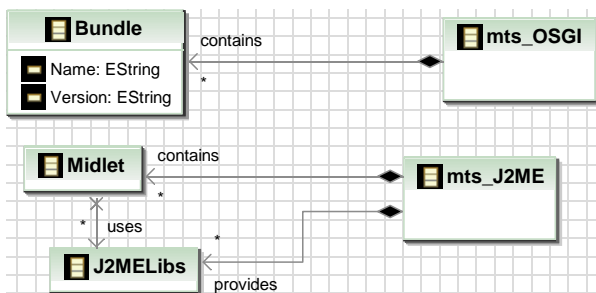


Figure 4: Allocation Model

Application components allocation over specific platform components will determine the kind of implementation of the application components. As a result of the allocation, the Allocated Model will be obtained. From this model the Implementation model

can be generated, where every application component can be transformed into a specific platform component.

2.4. Implementation Oriented Meta-Model

A complete profile must be created for the OSGi/Java platform. This OSGi meta-model includes most of the necessary OSGi concepts. So this meta-model was modified and extended to include new elements like services provided by a bundle but not published as OSGi services, as show in Figure 5. These services are named here as External Services.

This meta-model results as a refinement of part of the application component meta-model. New elements like OSGiManifest, CodeRepository are generated from the previous one.

The model allows to generate automatically a code skeleton for an OSGi bundle corresponding to every application component at PIM level. Once the component is associated to an OSGi platform it can be recognized as a bundle and the corresponding code is generated.

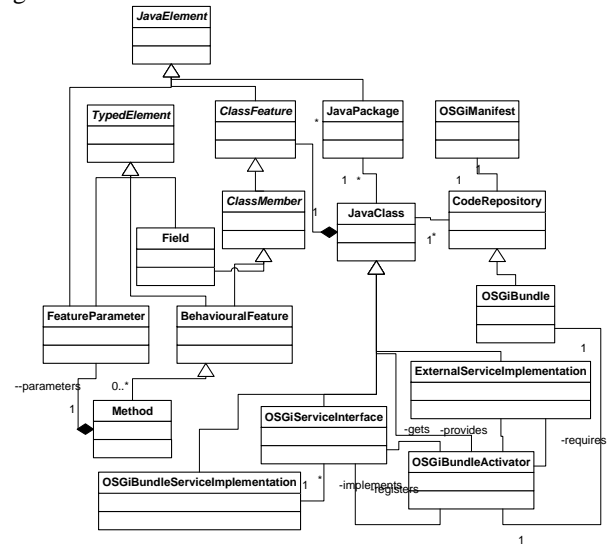


Figure 5: Implementation Oriented Metamodel for the OSGi profile

Since Java is the target platform this model is completely integrated in the Java meta-model.

Every bundle element is represented and associated to the Java element it is going to be implemented with (classes, methods, etc.).

This meta-model completes the OSGi profile for the MARTES process model. All process steps could be accomplished from PIM models to final code, with bundles packaged as “.jar” files.

It is possible to automate some of the transformations from PIM to PSM and from PSM to final code generation.

3. Transformations

This section describes how some of the main transformations between models can be achieved.

These transformations should be easily reversible, because each transformation represents a model

refinement containing all previous model information. The implementation model can be obtained again from the modified code.

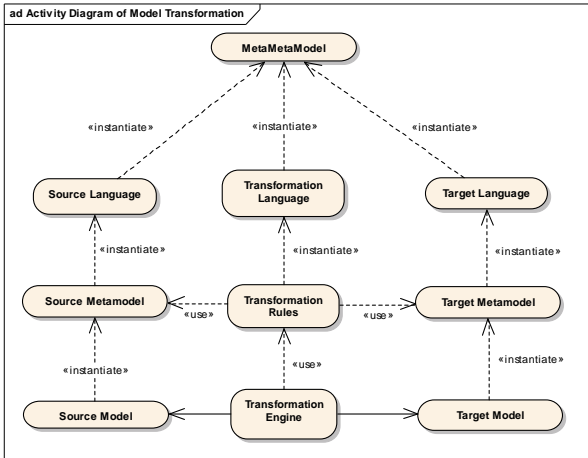


Figure 6: Model Transformation Pattern[12]

Model transformation represents the process of converting one or more models into an output model of the same system [10]. Mappings and relations are defined as specializations of a transformation. Contrarily to mappings, which are defined as a unidirectional transformation, relations are presented as bidirectional processes. Note that the result of a transformation is another model.

Model transformation languages are used to specify a model transformation. They are defined in a metamodel level. The relationship between source metamodel elements and target metamodel elements are established too, as we represented in the Figure 6 which is an evolution of pattern Bezivin [11].

As it can be seen in Figure 6, some information must be available to perform a model transformation. Basically, this comes from primitives of both meta-models (source and target). This type of information can be patterns, templates or UML profiles. We also need some information to guide the resulting transformation.

In summary, different transformation relations between source and target models are established, and, from them, transformations rules are derived.

3.1. Transformation Process

Several types of transformations in a MDD (Model Driven Development) process are considered, as it can be seen in Figure 7. The main characteristic of the MARTES model transformation process is that it is a semi-automatic, model-driven and transformation-based process.

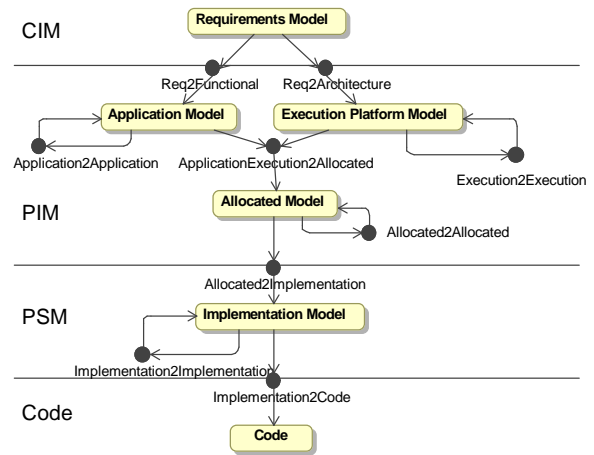


Figure 7: Transformation process

Figure 7 shows the transformations between the different models of the methodology. Some of them are described in the following subsections.

If we want to implement the Transformation Rules, we have to consider several options[13]:

- General programming language as Java, C++, PHP, and so on.
- Graph transformation language as AGG[14] and VIATRA[15]
- Languages for transformations such as ATL[16], QVT[17] or XSLT[20]

We decided to adopt the XSLT language to represent the Transformation Rules. This decision is adopted because the model representation can be exported into XMI files (XML format). Models in XMI format can be processed using XSLT, generating new models in XMI format. It can be represented more clearly if we modify Figure 6 to see the languages used in the transformation process (Figure 8). As it can be seen in this approximation, both models and Transformation Rules are implemented with XML formats.

To perform the transformation, we coded a Java program to read the XMI files (source model) and XSLT Transformation Rules and generate the new XMI files (target model). There are a lot of tools that we can use to design this parser (Xalan, Xerces, Saxon). We adopt the Saxon libraries to make this process[20].

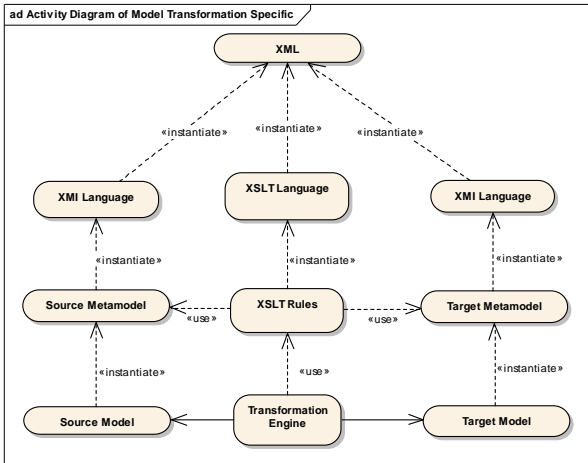


Figure 8: Model Transformation Language

3.2. Application – Platform – Allocation to Allocated model transformation

Three models are needed in this transformation. Application model and Platform model are combined into a new Allocated model while Allocation model is used to test the final Allocated model coherence.

We need to know the target platform for each application component in this transformation, when this information is known, application components can be replaced by their respective implementations.

The way in which services are implemented, is selected depending on its attributes or the way in which the components are connected in the Allocated Model. This determines if a service is provided to OSGi platform or to other components through a different interface like a network one. The result of this transformation will be other UML 2 model, which represents the Allocated Model, where Application Components are associated to Platform Components.

The procedure to realize this task consists of a manual mapping followed by automatic or semi-automatic verification. Automatic procedure may not be easily implemented here since models do not provide enough information to determine the desired allocation. Hence, calculating the transformed model may have exponential complexity, but checking the result as given by a human designer has only polynomial complexity.

3.3. Allocated model to Implementation model

Once the previous transformation is done, the second transformation to source code can be accomplished.

The Allocated model contains the platform and services information, the code structure can be generated, even some code can be filled, like initialization of values, registering services, etc.

The result of this transformation, as we could see in the section 3.1, will be an output UML 2 model, represented by an XMI file. This file represents an implementation model, which expresses details of the specific implementation environment. To achieve this

result, we need some type of information, how it can be seen in Figure 9.

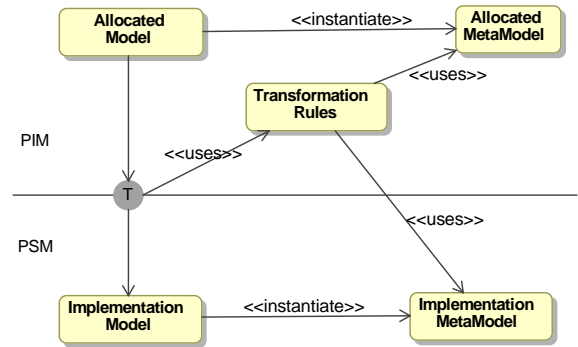


Figure 9: Allocated Model to Implementation Model transformation

The diagram shows all the data needed during this process.

- Allocated Model, used to describe the source model used in the transformation process.
- Implementation Model which is the target model in the process.
- Allocated meta-model, used to specify the components that we can use in the Allocated Model and also in the transformation rules.
- Implementation meta-model, used to specify the components that we can use in the Implementation Model and also in the transformation rules.
- Transformation rules. Based on the source and target meta-model, different transformation relations are established. Our approach is packed all of these transformation rules and describes them using the XSLT language.

They have properties that express all the application semantics and the navigable associations between different classes, for example working on a Java/OSGi scenario, each application description component, defined at the Allocated level, will become an Implementation component that includes all the main peculiarities related to the target technology (bundle attributes, OSGiBundleActivator, OSGiServiceInterface, etc.).

This new model contains all information needed to achieve later a complete transformation into OSGi services.

4. Conclusions and future work

The goal of this work was to extend the MARTES project profile to add new target platforms. The work has been centred in the OSGi platform, so although other platforms have been included in the application component meta-model only the OSGi implementation oriented meta-model has been described here.

The only meta-model that has not been modified is the application meta-model as it must stay completely

platform independent. The rest of meta-models have been adapted to create a new OSGi profile. With this new profile the OSGi platforms like Oscar and Equinox can be selected as targets, generating code for them without modifying the application model at the PIM level.

Future work will further develop the design and implementation of transformations between models. Also, the methodology will be extended in order to include other software execution platforms.

Acknowledgments

This research has been partially supported by the project ITEA MARTES “Model-based Approach to Real-Time Embedded Systems development” (ITEA 04006) and financed by Ministerio de Industria, Turismo y Comercio (FIT-340000-2006-166), Spain.

We would like to thank the MARTES team of Telefónica I+D which contributed with his ideas and work to this paper, in particular to Samuel Jiménez Martín, Alejandro Guarido Rincón and Ángel Jesús Ramos Moreno.

References

- [1] OMG Object Management Group. January 2007. URL: <http://www.omg.org/>
- [2] Model-driven Approach to Real-Time Embedded Systems development (MARTES), 2007. URL: <http://www.martes-itea.org>
- [3] OSGi Alliance, WWW, 2007. URL: <http://www.osgi.org>
- [4] J. Muñoz, V. Pelechano, J. Fons, "Model Driven Development of Pervasive Systems", en *Proceedings of International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES)*, June 2004, pp. 3-14, ISBN: 952-12-1359-0.
- [5] J. Muñoz, V. Pelechano, E. Serral, "Providing Platforms for Developing Pervasive Systems with MDA. An OSGi Metamodel", in *Actas de las Jornadas de Ingeniería de Software y Base de Datos (JISBD)*, September 2005. pp. 19 - 26, ISBN: 84-9732-434-X.
- [6] J. Muñoz, V. Pelechano, "MDA vs Factorías de Software", en *Actas de Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM)*, September 2005. [CEUR Workshop Proceedings \(vol-157\)](#), ISSN 1613-0073
- [7] Oscar, WWW, 2007. URL: <http://forge.objectweb.org/projects/oscar/>
- [8] Equinox, WWW, 2007. URL: <http://www.eclipse.org/equinox/>
- [9] D. Varró, A. Pataricza, “Generics and Meta-transformations for Model Transformation Engineering”, T. Baar et al. (Eds.): UML 2004, LNCS 3273, pp. 290-304, 2004.
- [10] Object Management Group (OMG). MDA Guide Version 1.0.1. <http://www.omg.org/docs/omg/03-06-01.pdf>.
- [11] Baumeister, H., Knapp, A., Koch, N. And Zhang, G. Modelling Adaptive with Aspects. In Proc. 5th Int. Conf. On Web Engineering (ICWE 2005), LNCS 3579, Springer, July 2005.
- [12] Taenzer, G., AGG: A Graph Transformation Environment for System Modelling and Validation. Proc. Tool Exhibition at “Formal Methods 2003”, Pisa, Italy, September 2003.
- [13] Koch, N. and FAST GmbH. Transformation Techniques in the Model-Driven Development Process of UWE. Int. Conf. On Web Engineering (ICWE 2006) Workshops, Palo Alto, California, July 2006.
- [14] Taenzer, G., AGG: A Graph Transformation Environment for System Modelling and Validation. Proc. Tool Exhibition at “Formal Methods 2003”, Pisa, Italy, September 2003.
- [15] Varró, D. and Pataricza A. Generics and Metatransformation for Model Transformation Engineering. In Proc. 7th Int. Conf. On Web Engineering (ICWE 2005), Sydney, Australia, LNCS 3579, Springer, July 2005
- [16] Jouault, F. and Kurtev, I. Transforming Models with ATL. In Proc. of the Model Transformations in Practice Workshop at MoDELS 2005, Jamaica, 2005.
- [17] Object Management Group (OMG). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Final Adopted Specification, ptc/05-11-01. <http://www.omg.org/docs/ptc/05-11-01.pdf>, November 2005.
- [18] Czarnecki K. and Helsen S., Classification of Model Transformation Approaches. OOSPLA'03 Workshops on Generative Techniques in the Context of Model-Driven Architecture, Anaheim, USA, 2003.
- [19] Object Management Group (OMG). Unified Modelling Language (UML): Superstructure, version 2.0. Specification, <http://www.omg.org/cgi-bin/doc?formal/05-07-04>
- [20] Saxon XSLT and XQuery processor, WWW, 2007, URL: <http://sourceforge.net/projects/saxon>