An Automated Model Based Approach to Mobile UI Specification and Development

António Nestor Ribeiro(✉) and Costa Rogério Araújo

Departamento de Informática, Universidade Do Minho and HASLab / INESC TEC, Braga, Portugal
anr@di.uminho.pt, rogerio.ar.costa@gmail.com

Abstract. One of the problems of current software development lies on the existence of solutions to address properly the code portability for the increasing number of platforms. To build abstract models is one efficient and correct way to achieve this. The Model-Driven Software Engineering (MDSE) is a development methodology where models are the key for all project lifecycle, from requisites gathering, through modelling and to the development stage, as well as on testing. Pervasive computing demands the use of several technical specifications, such as wireless connections, advanced electronics, and the Internet, as well as it stresses the need to adjust the user interface layer to each one of the platforms. Using a model-driven approach it is possible to reuse software solutions between different targets, since models are not affected by the device diversity and its evolution.

This paper reports on a tool, which is highly parameterizable and driven to support Model-2-Model and Model-2-Code transformations. Also, instead of using a predefined technology, the tool was built to be scalable and extensible for many different targets and also by addressing the user interface layer generation.

Keywords: Model-Driven Software Engineering · Model transformation · Cross-platform generation · Pervasive software development

1 Introduction

The current trends about software development for mobile platforms, namely mobile apps development, are mainly focused on the portability for the rising number of devices to which user interface layers can be developed. This addresses the need to sustain this development by building abstract models as a mean to have an efficient and scalable way to achieve our purposes.

As its well known, model driving software engineering supplies a development methodology where models are the key for the entire project lifecycle, from requisites gathering, through modelling and development stage, as well as on testing. Using a model-driven approach it is possible to reuse software solutions between different targets, since models should not be affected by the device diversity and its evolution.

M. Kurosu (Ed.): HCI 2016, Part I, LNCS 9731, pp. 523–534, 2016.
DOI: 10.1007/978-3-319-39510-4_48

As said previously, actual technologies are developing up at great speed in a diversity of areas, such as hardware and software (middleware and user interface layer). Hardware has been evolving to standardized form factors, more powerful and cheaper, and software has become more complete, with increased functionalities at the user interface level.

However, this development led to the proliferation of platforms and technologies where constantly there is new base software with new features, which increasingly impose new restrictions to software portability. For example, each time a new Android smartphone is released, there is the risk of old released software become uncovered with problems such as fragmentation or "multiple screens". This is particularly true when dealing with the user interface layer source code.

The amount of complexity brought to the software side is only possible to be reasonably solved because of the notorious improvements around the development methodologies, which enables us to deliver software with lower production costs, longer lifecycle, and higher interoperability. Using models as basis of software development allows the overcome of the current platform proliferation and it also provides portability for new platforms that may appear in near future.

Model-Driven Architecture (MDA) [7], proposed in 2001 by the Object Management Group (OMG), encompass a set of standards for model-based software development. It is intended to support ever-changing business environments, minimising the software development time and project costs. MDA enables separating the system functionality from implementation details, keeping consistent glue between both elements.

Software development based on MDA starts with high-level models obtained in the specification phase. Gradually and automatically, the models should be transformed into more specific (low-level) models until source code is reached. The transition between models can be achieved by a set of well-defined rules (the models glue). Then, using a tool, it is possible to achieve automatic code generation from abstract (high-level) software models.

Tools supported by models make the software development more straightforward, because it enlaces the software portability, and the developer can choose the abstraction layer and programming language to be used. It is important to stress that data, behaviour and user interfaces can be modelled at adequate abstraction levels and then rely on transformation rules to generate the corresponding source code. Specific efforts on the development of each one of these layers usually implies that the models were not properly designed.

This paper uncovers the first results of a model-based tool, MDA SMARTAPP, which is driven to support highly parameterizable MDA transformation processes. The tool is to be used in the development of the application's business and user interface layers meant to be accessed by mobile apps (in a first approach Android specific) or hybrid web browser desktop applications.

The remaining document is structured as follows: in Sect. 2 it is exposed some related work; in Sect. 3 it is presented how Model-2-Model transformations are achieved; Sect. 4 is related to MDA SMARTAPP model editor; in Sect. 5 it is presented the tool architecture; Sect. 6 is related to the case study; and Sect. 7 presents the conclusions.

2 Related Work

To build a mature model-based tool, such as MDA SMART, it is important to overcome two major different points of view: what is expected from a model-based tool, and what could be done to support efficiently models transformation.

In a model-based tools space, there are some highly evolved tools, being the OutSystems Platform[1], or the IBM Rational Software Architect[2] two successful examples of such tools. Usually, these tools provide development environments with simple and high quality rendered interfaces, and a lot of features for drag-and-drop modeling. As a result, users becomes more concerned about the envisaged solution, instead of the implementation details. However, highly evolved tools have a restricted structure, and the user has sometimes some difficulty to custom and expand beyond their "sandbox". And the advantage of model portability is many times fully dependent on the tool ecosystem and not properly interoperable.

Several MDA implementations have already been proposed in the past [5,6]. In [2], for example, it is done a study on the applicability of MDA in the development of large-scale software. As a result, the study proved that MDA based approaches increases the quality and quantity of the deliverables and reduces the overall cost once it allows people to interact at a more abstract point of view. It is also important to note that using MDA models provides for some durability and resistance because they are not affected by the proliferation of available middlewares.

In [1] the development of a Fujaba [8] plugin to support Business Process Modeling (BPM) tasks is presented. The main goal is to port BPM models for UML activity diagrams and vice versa through Fujaba mechanisms "MoRTEn" (ModelRound-Trip Engineering) and "MoTE" (Model Transformation Engine). To support the transformations it was implemented a mechanism of Triple Graph Grammars (TGG) [12] in order to achieve bi-directionality and incremental model processing.

In [13] a prototype for the semi-automatic construction of Web Information Systems (WIS) was built. The objective is to achieve the tool architecture through other existent tools and some Model-Driven Development (MDD) components.

The most successful initiatives of MDA supported tools are the ones which use Domain-Specific Language (DSL) approaches to define model transformations. Here the tools are divided in several domains such as mobile devices, web services and applications, and standard desktop solutions.

Another work worth of mention is the one presented by Vaupel et al. [14]. It presents a modelling language and an infrastructure for the model driven development of Android apps. It also uses Ecore meta-models and it provides model transformation and source code generation using the Eclipse plugins. It defines a meta-model for the business layer, one for the user interface and another for

[1] http://www.outsystems.com/.
[2] http://www.ibm.com/developerworks/rational/products/rsa/.

specifying the application's behaviour. It uses simplified meta-models, in order to cope with complexity, for the transformation stages. One major difference from our approach is the fact that it only supports the transformation for Android applications not covering both the Web and hybrid clients.

3 Model to Model Transformation Engine

There are tools that manage web, mobile and desktop development at the same conceptual level. Even inside each one of these categories not all the existing tools support, or can be extended to the plethora of possible technological targets. In order to achieve this compatibility degree is the main objective of MDA SMARTAPP, a tool that allows the using of models and provides a way to support transformations for different target device families.

The kernel of MDA SMARTAPP is based on a model to model (Model-2-Model) transformation mechanism, the M2M Engine. The main purpose of M2M Engine is to iterate over all models of a MDA standard architecture until the models reach low-level abstraction layers. This is particularly useful when addressing the user interface controls and widgets, knowing that at model level the developer needs that technological particularities will not change the models, allowing to keep the discussion at a reasonably high and abstract level.

The DSL approach has been repeatedly used in model-based tools. There are well known cases where using a DSL become a success, such as is the case of ATLAS Transformation Language (ATL) from ATLAS Model Management Architecture (AMMA) platform.

ATL, proposed by the Group ATLAS INRIA & LINA, was aimed to implement Meta-Object Facility (MOF)/Query-View-Transformationg (QVT) [10,11] request standard from OMG. It's a hybrid language since it allows rules construction on both imperative an declarative paradigms. In a declarative way, simple mappings are implemented in a straightforward way. The imperative way to use the language is mostly used for higher complexity definitions.

The ATL virtual machine is properly equipped with a well-developed Object Constraint Language (OCL) [9] architecture. This feature provides flexibility in models manipulation (and respective meta-models) allowing it to cope with more

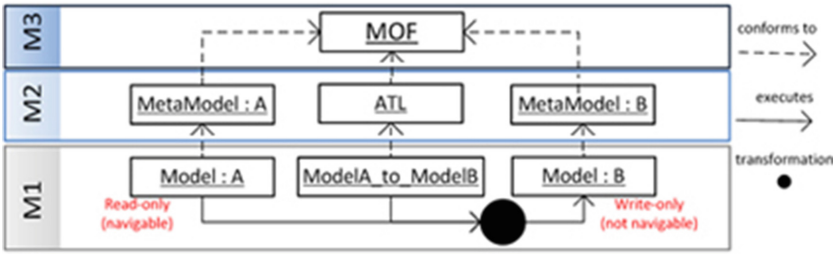

Fig. 1. EMF ATL - Operational context

complex models. Moreover, models can present problems in the transformation process, and these could be difficult to resolve if there is not a significant support from the OCL side.

As presented on Fig. 1, ATL operational context follows a MOF [10] compliant architecture. In this context, the input model (A) is translated to the output model (B) through a well defined set of ATL rules (ModelA_to_ModelB). The input model (A), the output model (B), and the set of ATL rules (ModelA_toModelB) conforms to the M2 (level) meta-models, MetaModel:A, MetaModel:B, and ATL, respectively. All three M2 meta-models are bridged by the (M3) MOF meta-meta-model.

MDA SMARTAPP takes advantage of this MOF compliant architecture to be extensible and scalable. For a new Model-2-Model configuration there is the need to provide the input and the output meta-model (written in the Ecore format), and the ATL set of rules. With only these three elements it is possible to achieve software portability for any device configuration.

4 M(odel) Editor

In order to give the end user a friendly environment to edit the models we developed a small scale graphical editor. The graphical editor component was built using the JGraph[3] library. This library presents good usability patterns, with a rich look and feel, it is well documented, and it has become used with success in a series of successful case studies [1]. However, it should be noted that our aim is not to replace other tools that can be used for model edition and manipulation, but to provide for prototyping purposes the means to easily create a model. We believe that most developers will use their preferred tool for model creation and through the existing formats for model interchange the models can exchanged with other applications.

In addition to the most well known functionalities, JGraph also provides a mechanism to implement the model validation. It is possible to reuse this mechanism to build "a priori" a model checker, and therefore by using this functionality, MDA SMARTAPP can validate the user actions and their conformity to the UML's meta-model. For example, it doesn't allow the user to specify an implementation of an UML class with respect to other class, as it should have been done to an interface definition.

In [2] is reported how hard and unmanageable is to restart a sequence of model transformations because of delayed detected errors. That is even more evident when dealing with very large and complex models, with a magnitude of several thousand objects (business and interface objects) as discussed in [3].

5 Tool Architecture

MDA SMARTAPP is intended to support the bottom layers from the MDA architecture: Platform Independent Model (PIM), Platform Specific

[3] http://www.jgraph.com/.

Model (PSM) and source code. Therefore, this tool provides one component dedicated for PIM models manipulation; one component for the PIM to PSM transformations; and one component for source code generation taking the PSM models as input. All three were designed to be abstract components, and can be extended by specific configurations.

The first component, the M(odel) Editor, is responsible for capturing the visual information (objects and locations) that describes the memory model representation. Similar to a CASE tool, this includes model manipulation according to the respective meta-model context. Also, it allows for a design environment with good usability patterns and without the need to the user to develop any source code.

The tool core component, the M2M Engine, is accomplished with an ATL configuration. This component is responsible for managing models definitions and to execute the instantiated Model-2-Model transformations.

The third component, the M2C Engine, covers the last step of a MDA architecture, and by using a template approach the PSM models are translated into source code.

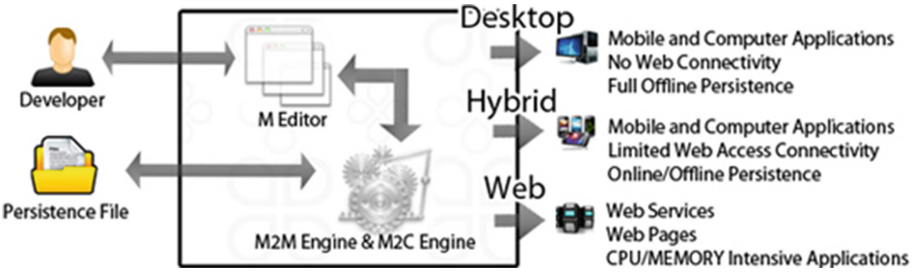

Fig. 2. MDA SMARTAPP - Tool logical architecture

MDA SMARTAPP supports UML2 for the PIM layer and Java and Android for the PSM and source code layer. Also, there are considered three main output targets (Fig. 2): Web applications, Hybrid clients with a server side and a client side components, and Desktop applications. Although the definitions of web and desktop applications are self-explanatory, it is important to define what we understand by hybrid applications. Hybrid applications are applications built specifically for native platforms (namely Android, iOS, or others) that exchange information with the server side using standard web protocols (eg. Web Services). At this stage we use Java as the platform for desktop applications and server side components and Android code will be generated to run in the mobile devices.

6 Case Study

As a proof of concept our case study is a simple Field Force Automation (FFA) application. The application objective is to retrieve lists of technical information

shaped for different use cases. The biggest challenge in this domain lies on the definition of a usable Graphical User Interface (GUI) for the mobile devices, specially the smaller ones, that force the developer to think very carefully about the usability and the user experience. There's another significant challenge that arises from the fact that the source code portability is important especially when dealing with such constraints with the target hardware and base software.

This case study was solved with one unique abstract model, that later was derived for desktop (Java) and mobile (Android) applications.

First it was necessary to develop some primary MDA SMARTAPP features: this includes an UML2 domain editor (the M Editor component), UML to Java model transformations (the M2M Engine), and finally the source code generation for Java and Android targets (the M2C Engine).

For the UML2 domain editor it was developed a graphical view (V) of UML2 model using the JGraph library. This view is supported by a bespoke controller (C) and the UML2 meta-model[4] application program interface (M) available in the Eclipse platform.

To support the Model-2-Model transformation, it was considered a simplification of Java meta-model, in order to reduce the number of entities and relationships. Some ATL rules were specified, and strengthened with OCL definitions. In this particular case OCL allowed us, for example, to ensure that the UML2 packages are well unfolded to Java packages (Fig. 3), or the name of any Java element respects the reserved words, although other more complex restrictions could have been specified.

The source code generation of the user interfaces, from both desktop and mobile clients, was derived from the UsiXML [4] models of the interface layer.

```
helper context UML!Namespace def: getExtendedName() : String =
    if self.namespace.oclIsUndefined() then ''
    else if self.namespace.oclIsKindOf(UML!Model) then ''
    else self.namespace.getExtendedName() + '.'
    endif endif + self.name;
```

Fig. 3. UML2 to Java ATL rule - UML2 to Java package unfolding

Once we had a robust Java meta-model, two sets of Velocity templates were developed for Java and Android technologies. Since it is possible to build multiple template fragments and choose at runtime what best fits on the target device, it is possible to overcome the slight differences from similar targets with one unique PSM meta-model.

For our FFA application the simplified domain model is presented in Fig. 4 and it depicts the core business entities: the worker, the service and the client.

[4] http://www.eclipse.org/modeling/mdt/?project=uml2.

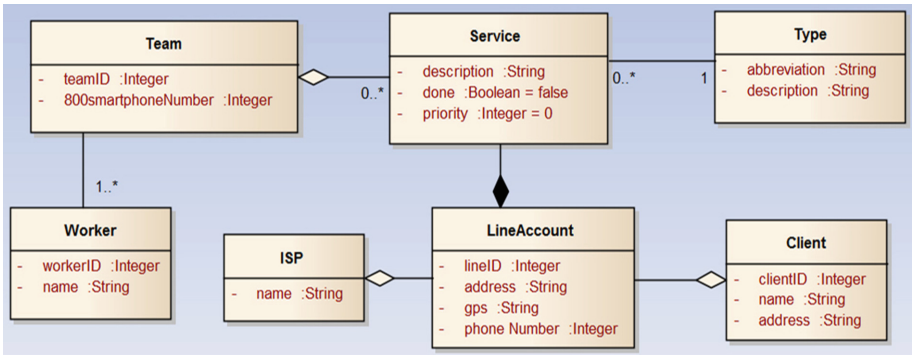

Fig. 4. Simplified domain model of a FFA application.

The domain model is transformed into a PIM model derived from the transformations needed to ensure the necessary compliance to the Java meta-model. Figure 5 shows MDA SMARTAPP platform independent model for the FFA application.

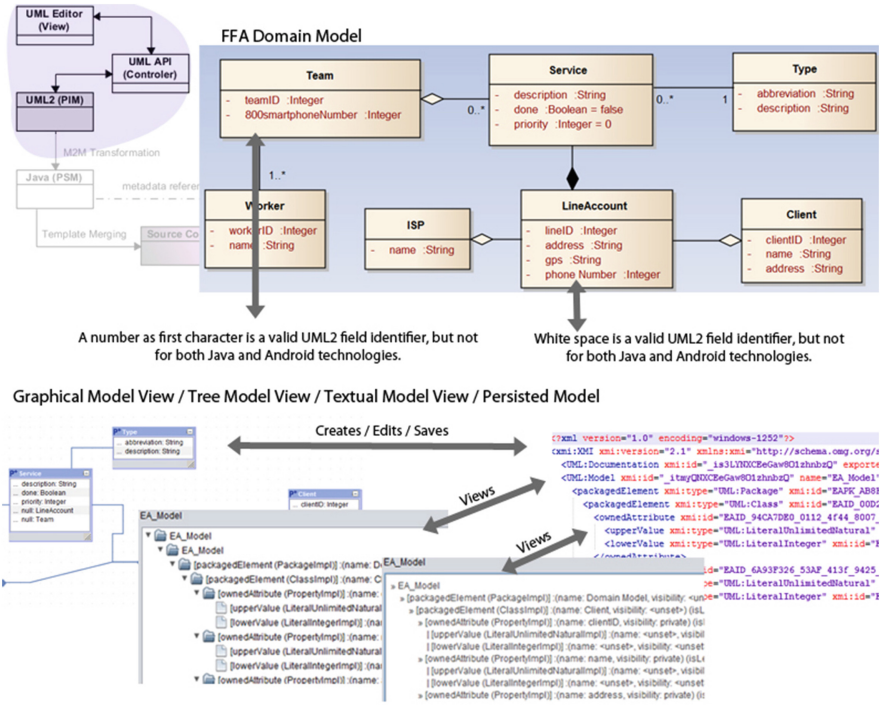

Fig. 5. PIM model construction.

Fig. 6. PSM model construction.

The platform specific model, the PSM, will use the same target language (Java), so it is not necessary to change the existing UML model. Figure 6 shows the PSM model for the FFA application.

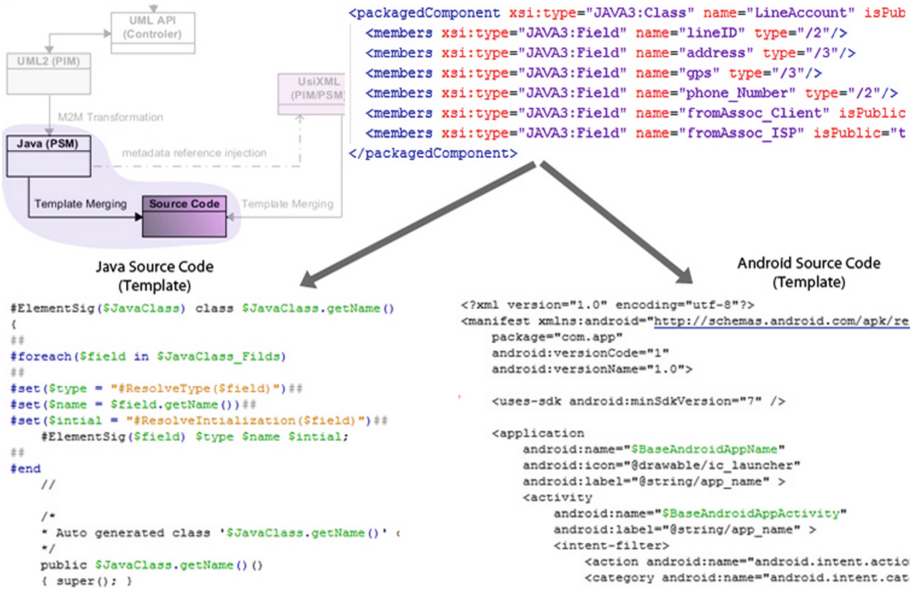

Fig. 7. Source code generation for Java and Android.

Figure 7 illustrates the source code generation for both the Java and Android platforms.

Using the same template's strategy, and starting from a UsiXML model, the source code for the user interface layer is also generated. Figure 8 shows the usage of templates to generate this layer.

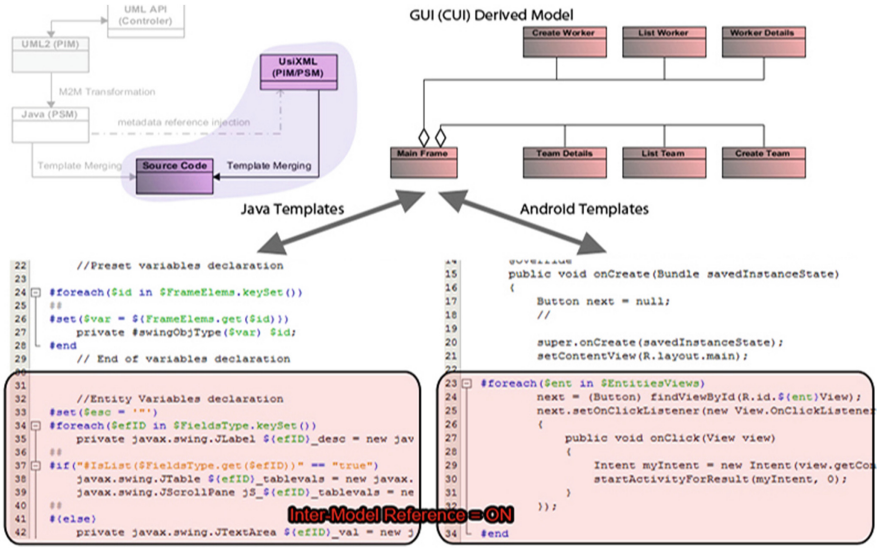

Fig. 8. User interface source code generation for Java and Android.

7 Conclusions

In this paper a model-based tool for hybrid systems development was presented. Through a DSL configuration the MDA SMARTAPP tool can translate abstract models in implementations artefacts for web, hybrid and desktop targets.

This paper described the first results of a model-based tool, MDA SMARTAPP, meant to support highly parameterizable MDA transformation processes. The tool is to be used in the development of business layer and user interface layers of applications that can be reached using mobile apps (in a first approach it is Android specific) or hybrid web browser desktop applications. Specifically, it supports PIM (Platform Independent Model) manipulation, PIM to PSM (Platform Specific Model) transformations, and automatic source code generation for both web and mobile clients. MDA SMARTAPP does the setup of a robust, extensible, and scalable model-based tool architecture where its skeleton is independent from any platform domain, having its main core based on model transformations.

The use of models, and the possibility of having them to parameterize the tool, ensures durability for any software and promote independency on changes of the base software of mobile devices. We presented a case study that covered this process as well as makes it possible to strive new application domains, since the tool can work with new target platforms, such as iOS or other custom fit solutions.

Also, this approach highlighted that with OCL it is possible to create robust and simple (not simpler) transformation processes, with business rules included, allowing us to shape better and target specific models, reducing the need to rearrange the generated lower-lever models.

The use of templates for source code generation allow us to easily reshape models in order to cover all the implementations variations from an original PSM specification. The work reported focused on a first set of components developed for MDA SMARTAPP and proves that highly parameterizable and complex user interface apps for mobile platforms can be specified using well known models and the transformations from models to source code can effectively deliver a ready to deploy product.

Acknowledgments. This work is financed by the ERDF - European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013.

References

1. Altan, G.S.: On the Usability of Triple Graph Grammars for the Transformation of Business Process Models - An Evaluation based on FUJABA. Master's thesis, TU Wien, Austria (2008)
2. de Almeida, P.: MDA - Improving Software Development Productivity in Large-Scale Enterprise Applications. Master's thesis, University of Fribourg, Switzerland (2008)
3. Egyed, A.: Fixing inconsistencies in uml design models. In: Proceedings of the 29th international conference on Software Engineering, ICSE 2007, pp. 292–301. IEEE Computer Society, Washington, DC (2007)
4. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: USIXML: a language supporting multi-path development of user interfaces. In: Feige, U., Roth, J. (eds.) DSV-IS 2004 and EHCI 2004. LNCS, vol. 3425, pp. 200–220. Springer, Heidelberg (2005)
5. Ma, K., Yang, B.: A hybrid model transformation approach based on j2ee platform. In: 2010 Second International Workshop on Education Technology and Computer Science (ETCS), vol. 3, pp. 161–164, March 2010
6. Meads, A., Warren, I.: Odintools-model-driven development of intelligent mobile services. In: 2011 IEEE International Conference on Services Computing (SCC), pp. 448–455, July 2011

7. Miller, J., Mukerji, J.: Mda guide version 1.0.1. Technical report, Object Management Group (OMG) (2003)
8. Nickel, U., Niere, J., Zundorf, A.: The fujaba environment. In: Proceedings of the 2000 International Conference on Software Engineering, pp. 742–745 (2000)
9. Object Management Group. Object Constraint Language, v2.0. Technical report, May 2006. http://www.omg.org/cgi-bin/doc?formal/2006-05-01
10. OMG: Meta Object Facility (MOF) Core Specification Version 2.0 (2006)
11. Partners, Q.V.T.: Revised submission for MOF 2.0 Query / Views / Transformations RFP. Technical report, OMG (2003)
12. Schrr, A.: Specification of graph translators with triple graph grammars. In: Mayr, Ernst W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903. Springer, Heidelberg (1995)
13. Vara, J.M.: M2DAT: a Technical Solution for Model-Driven Development of Web Information Systems. Ph.D. thesis, ETSII, University Rey Juan Carlos, Madrid, Spain, November 2009
14. Vaupel, S., Taentzer, G., Harries, J.P., Stroh, R., Gerlach, R., Guckert, M.: Model-driven development of mobile applications allowing role-driven variants. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E. (eds.) MODELS 2014. LNCS, vol. 8767, pp. 1–17. Springer, Heidelberg (2014)