

A Model Driven Architecture for REA based systems

Signe Ellegaard Borch, Jacob Winther Jespersen,
Jesper Linvald, Kasper Østerbye*

IT University of Copenhagen, Denmark

* Corresponding Author (kasper@it-c.dk)

Abstract. An important aspect in the OMG model driven architecture is the separation of the platform independent model (PIM) and the platform specific model (PSM) of a system. This paper presents a system generator for a specific kind of accounting systems where the PIM is specified in a domain specific XML specification, and where the PSM is automatically generated by using platform dependent templates. One contribution of this paper is to describe how we have been working with the concepts of MDA in a domain specific context. The accounting systems we generate are based on the REA model of accounting proposed by William McCarthy. Another contribution of the paper is to show that one can generate parts of an accounting system based on REA specifications.

1 Introduction

The REA accounting model was first presented by William McCarthy [1], and later elaborated in numerous papers, foremost in the work of McCarthy and Geerts [2], and it has been a main inspiration for the ebXML standardization effort [5]. An important aspect of the REA model is its attempt to capture three important aspects of business modeling in one framework, namely accounting, inventory, and supply chain management. In this paper, however, our main emphasis is using REA as a meta-model. Our models based on the REA meta-model are expressed in XML, and plays the role of platform independent models. These models are translated into EJB code for a specific J2EE server using code-templates.

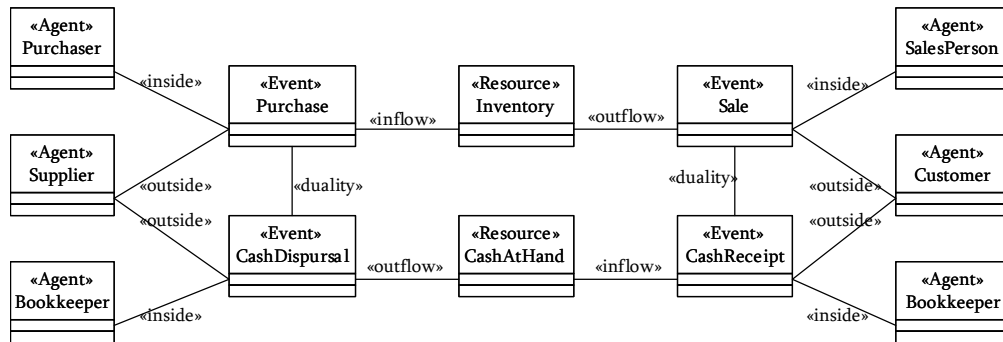
The choice of XML technology for specifying the platform independent model, rather than UML is deliberate. XML is widely supported by a range of transformation and verification tools, which unlike most UML transformation technology are freely available for the Java platform, or through open source projects.

The paper is structured as follows: First, we will briefly present the REA meta-model, and give a concrete example. Second, we will present the architecture of our implementation of the model driven architecture. Finally, we will conclude with some experiences and perspectives.

2 The REA meta-model

The REA model of accounting is a meta-model for building accounting systems, which is centered on registration of the basic economic events that occur within an organization. In its first incarnation, presented by William McCarthy in [1], the model consists of three basic meta-types, Resources, Events, and Agents. Each Event (such as a sale, payment, purchase, etc.), is related to a Resource (Inventory, Cash...) and two Agents, an internal Agent responsible for the Event, and an external Agent with whom the Event takes place. In [2], this basic model is extended to handle contractual issues as well.

A simple cycle shop that purchases bicycles and sells them again can be modeled as:



Besides the three REA entity meta-types, there is a number of association meta-types. Events are associated using a duality relation. This captures a simple exchange situation: When the customer receives a bicycle through a sales event, he must also be part in a cash receipt event (as the shop does not give away bicycles). For each event that is associated with a resource through an outflow association (meaning that the resource is decreased), its dual event must be attached to a resource through an inflow association.

In the above model, no cardinalities, attributes, or methods are described. Due to the fact that we are working in a domain specific context these aspects can be deduced from the model on the basis of the underlying ontology: In the simple model, each entity type has an attribute ID, Resources have an additional description, Agents have a name, and Events have a date, a quantity, and a value.

Resources have a static method to find out how much is in stock, which is the total quantity over its inflow events, minus the total quantity over outflow events. For a given instance of a Resource, e.g. men's bike, black, Centurion, 7 gear basic, one can find the number in stock by looking only at inflow and outflow events that relate to that specific inventory item.

Similarly, one can find out how much money a given customer owes by calculating the value of all outflow events minus the value of all inflow events associated with a given customer.

Cardinalities are as follows: One Agent can participate in many Events. One Event has one internal and one external Agent. A Resource can participate in many Events; a given Event is only about one Resource¹. The duality association is many to many, unless otherwise specified. Notice that these methods, attributes and cardinalities are given by the REA meta-model, and need not be specified in any concrete model.

3 Design and Implementation

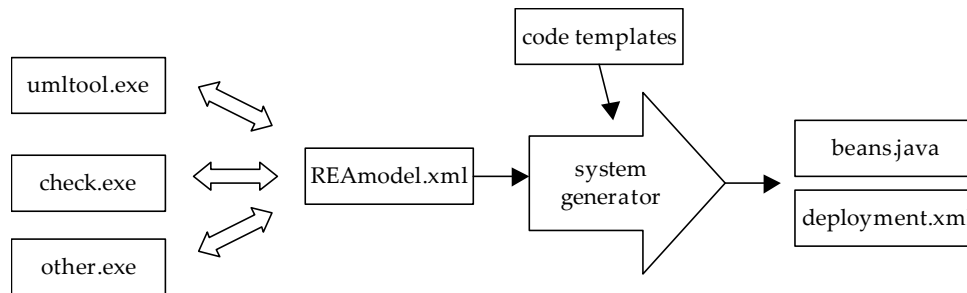
In the following we will describe an implementation of MDA on the basis of the REA meta-model.

3.1 Overall design

The overall design is illustrated in the figure below. The main component is the system generator, which based on the REA model and code templates generates the files needed to build the system.

¹ Obviously, a given sale can consist of several things, e.g. two bikes, one helmet, 4 lamps, and a bell. This multiplicity is specified at the contractual level, not at the operational level we present here.

The REAmodel.xml file describes the platform independent model in a simple XML format with tags representing the meta-model entities Resource, Event and Agent.



The system generator assumes the REA-model to be well formed, an assumption that is honored by the checking tool to the left. Leaving the checking tool out of the system generator has the advantage that the system generator becomes simpler, furthermore, the consistency of the REA meta-model has not yet been fully specified, and it is therefore a good idea to leave it as a separate tool. Furthermore, it is possible for a different team to work on the checker independently of the team working on the system generator.

We have chosen to use a simple customized XML Schema rather than building upon say XMI [9], which is clearly strong enough to express the models. The primary reason is that we wanted to be able to start designing without worrying about the inner workings of XMI. However, we are building an extension to RationalRose that will produce our REA-model from a UML class diagram, where each of the meta-types of REA is specified as UML stereotypes. We see UML as *one* important way to specify REA models, though we envision others. Thus, it is useful for us to keep a XML representation as the input to the system generator.

The system generator is the heart of the tool. It reads the REA-model, builds an internal intermediate model, and uses this intermediate model together with the templates to generate the java and XML files necessary for the system to run on the J2EE platform.

3.2 Model transformation

This section describes the process of transforming a REA model into a J2EE application through multiple automated transformations in more details.

The first transformation is carried out on the basis of the XML REA specification, which is parsed and mapped to a Java model. This model corresponds one-to-one with the XML REA specification. The purpose of this step is to provide a Java based specification of the PIM to give as input to the Velocity template engine [3].

The reason why we have chosen Velocity as our template engine instead of using other technologies such as XSLT is that it is Java-centric and therefore fits better into the Java environment we are working in. From the templates, we generate Bean classes with XDoclet [2] tags. This is the point where the mapping from the platform independent model to the platform specific model takes place.

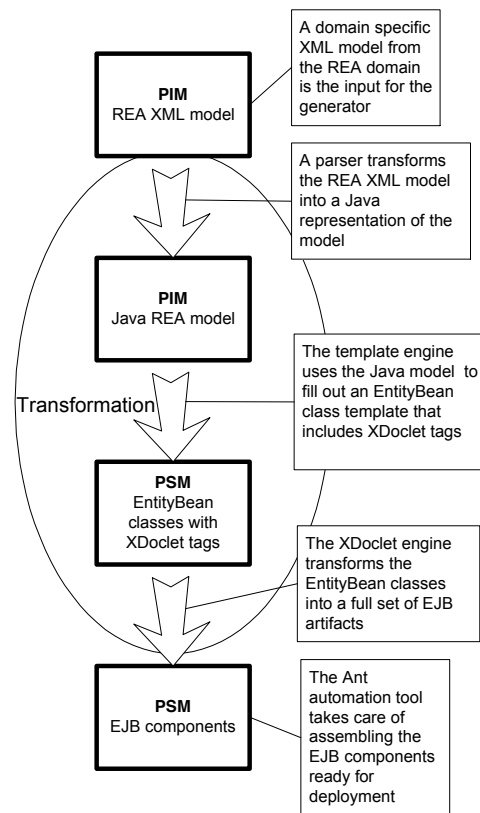
XDoclet generates the necessary interfaces and deployment descriptors to make up complete EJBs on the basis of the tagging done in the previous step. The reason for using XDoclet as an intermediary transformational step is to simplify code generation. As an alternative, we could have used multiple templates each corresponding to an EJB specific artifact (i.e. one template for the deployment descriptor, one for the LocalHome interface etc). The XDoclet approach has the advantage that we only have to maintain a single template for the components, which makes the template easier to debug.

The generated source files are finally packaged for deployment on a specific application server using the automation tool Ant [10].

Interpreting our approach to the terminology of the MDA guide [7], we first notice that neither the PIM nor the PSM are MOF based. The PSM has a reified meta-model in the form of an XML Schema, whereas there is no reified meta-model for the XDoclet output. We use a *direct transformation* approach c.f. [7, 3.7], which, despite of the non-reified PSM meta-model, is an example of a *meta-model transformation* [7, 3.10.2].

The fact that we are working with a domain specific PIM with rich semantics gives us the possibility of building in knowledge about the domain into the system generator.

This means that we do not have to annotate the PIM with additional information concerning business rules – for example on how to compute inventory stock or accounts receivable. Instead, we can rely on the fact that our transformation tool will recognize declarative statements in the PIM about which business rules to apply and translate them into the PSM. The MDA guide mentions *automatic translation* in [7, 4.1.4], which is what we do. However, our semantics are not defined using an action language, but is part of the REA meta-model itself.



4 Conclusions

This paper has described how we utilize a Model Driven Architecture in a domain specific environment. From a strictly declarative REA application specification (the PIM) our tool is able to create a Java Enterprise Application (the PSM) ready for deployment to a J2EE server. The user of the tool is not required to do manual coding.

As is apparent, we have focused on transformation of meta-models, i.e. we have devised a particular PIM-to-PSM mapping based on a typological view of the REA entities. This approach seems to work well in our case due to the common characteristics of our PIM elements; a commonality that is a result of the ontological foundation (REA) they share. The rich semantics in this foundation is what allows us to turn seemingly sparse specifications into meaningful applications in a fully automated transformation process.

The latter observation is in line with OMG's proclamation that semantically rich input models is an important condition to make full automation in the transformation a feasible approach [8], page 15. Another important condition mentioned is the independence of legacy, which also holds in our case.

The capabilities of the applications that our tool generates are currently limited to basic handling of the application data objects (the Resources, Events and Agents) and to carrying out simple business processing, e.g. the stock and accounts receivable calculations. Notably, our REA specifications do not currently include declarations concerning business rules, workflow, or information visualization

(e.g. report generation), nor can they be marked up to control platform specific issues, e.g. to fulfill QoS requirements.

However, going forward it seems promising to extend the capabilities in these areas by applying the MDA pattern repeatedly to the transformation process. For example, when workflows are modeled generically in REA-based systems, such models can be merged with a PIM (constituting a PIM-to-PIM mapping) before going to the PSM.

In [6] the notion of Platform Description Model is presented. The MDA guide [7], only defines the term Platform Model. Our interpretation of Platform Model is that it is not a formal model, but is expressed at a conceptual level, whereas Bézin and Ploquin's notion of PDM is a technical artifact. In our system templates play the role of PDM, and we believe that the system generator should be able to generate code for various platforms if equipped with a different set of templates, though that remains to be verified.

5 References

- [1] William E. McCarthy. *The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment*, The Accounting Review (July 1982) pp. 554-78.
- [2] William E. McCarthy, and Guido L. Geerts. *An Accounting Object Structure For Knowledge-Based Enterprise Model*, IEEE Intelligent Systems, July/August 1999 (pp. 89-94)
- [3] Velocity Template Engine. <http://jakarta.apache.org/velocity/>
- [4] XDoclet - <http://xdoclet.sourceforge.net/>
- [5] ebXML (Electronic Business using eXtensible Markup Language). <http://www.ebxml.org/>
- [6] Jean Bézin and Nicolas Ploquin, Combining the Power of Meta-Programming and Meta-Modeling in the OMG/MDA Framework, OMG's 2nd Workshop on UML for Enterprise Applications, San Francisco, USA, December 2001
- [7] MDA Guide Version 1.0. Edited by Joaquin Miller and Jishnu Mukerji. http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf
- [8] Model Driven Architecture (MDA). <http://www.omg.org/docs/ormsc/01-07-01.pdf>
- [9] XML Metadata Interchange (XMI). <http://www.omg.org/cgi-bin/doc?formal/2003-05-01>
- [10] The Apache Ant Project. <http://ant.apache.org/>