Migrating a Domain-Specific Modeling Infrastructure to MDA Technology

Duncan Doyle[1,2], Hans Geers[2], Bas Graaf[2], and Arie van Deursen[2,3]

[1] Fortis, The Netherlands
duncan.doyle@nl.fortis.com
[2] Delft University of Technology, The Netherlands
{h.j.a.m.geers,b.s.graaf}@tudelft.nl
[3] CWI, The Netherlands
arie.van.deursen@cwi.nl

Abstract For many companies, legacy applications developed using a model-driven approach based on (proprietary) domain-specific modeling languages (DSML's) form an important asset, as they implement key business functionality. Unfortunately, if the underlying infrastructure comprising code generators, libraries, and model repositories fails to meet new demands of, e.g., modern web applications, the original benefits of the model-driven DSML approach can turn into a significant drawback.

To remedy this, we explore how models specified with a proprietary DSML in use at a large financial services company can be migrated to models conform the MDA, in order to benefit from the range of MDA standards. We describe the legacy model-driven engineering infrastructure, propose a migration approach, discuss how we implemented each migration step using metamodeling and transformation activities, and offer an analysis of the lessons learned.

1 Introduction

The MDA promise of model-driven development is becoming a reality. In the MDA vision the software development process mainly consists of a series of model transformations applied to application models, resulting in a generated application. UML modeling tools have been available for a long time; now these tools start to offer more advanced features, such as support for UML profiles and code generation. As a result of the MOF QVT RFP [1] even techniques for model transformations, 'the missing link of MDA' [2], are becoming available.[1]

Many companies already adopted the fundamental ideas of MDA and, more in general, model-driven engineering (MDE). As such, many software 'production lines' and 'factories' are in use today. As standardized approaches to model-driven engineering are only becoming available recently, these are often based on 4th Generation Languages (4GLs) and proprietary technologies for, for example, modeling, model transformations, and code generation. These environments were often developed using a bottom-up approach. As a result, languages are weak-defined, and concerns not well separated

[1] http://www.planetmde.org/tools/

between models, transformation tools, and code generators, leading to various maintainability problems.

In this paper we discuss how to migrate such a legacy MDE approach towards the MDA. More in particular we focus on the migration of the legacy application models to UML or other MOF-compliant models. The discussion is illustrated by a model-driven development environment in use at Fortis [2]. We migrated a representative sample of the models from which this environment, called INVINCIBLE Software Factory [3] generates applications.

In Section 2 we discuss some of the relevant related work. The motivating case study is introduced in Section 3. Section 4 briefly summarizes the essence of DSML and MDA approaches to software development and outlines the necessary steps for a migration between the two. Then, the actual implementation of this approach for our specific case study is discussed in Section 5. In Section 6 we reexamine a number of issues, after which we conclude the paper in Section 7.

2 Related Work

Mansurov and Campara [3] argues that a first step in the migration towards the MDA is the introduction of modeling in the software development process. The paper proposes an approach to raise the maturity of software architectures to a level where software maintenance and evolution are driven by the architecture instead of by the code. For this, so-called Container Models are introduced. Although the approach enables a transition to the MDA, it only focusses on the extraction of these Container Models from existing *code*. In our case, however, we already start from a model-driven situation, and migrate from there to the MDA.

The OMG is currently working on a set of standards related to the modernization of existing systems, referred to as Architecture-Driven Modernization (ADM) [4]. These standards effectively define a set of metamodels that allow for modeling the information required for modernization efforts. In fact one of these standards is based on Mansurov and Campara's Container Models [3]. The OMG describes a number of scenarios in which ADM can be applied [5]. One of those scenarios involves the migration towards the MDA. For this scenario it is assumed that the starting point is a situation were software development is not model-driven, whereas in our case we are moving from one approach for MDE, based on proprietary DSML's, to another, based on MDA.

Kurtev et al. [6] coins the term technological space for a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. MDA is one of the example technological it mentions. In the context of this paper we can consider the software factory in use at Fortis a technological space as well. Although the paper argues that the different characteristics of technological spaces can motivate the move from one technological space to another, it does not discuss how to implement the required 'bridges'. In our migration approach a (particular) technological spaces bridge is an essential part.

[2] Fortis is an international financial services provider engaged in banking and insurance. With a market capitalization of EUR 34.7 billion (30/06/2006), Fortis ranks among the twenty largest financial institutions in Europe (`http://www.fortis.com`).

[3] We changed some of the names to protect Fortis' intellectual property.

Staikopoulos and Bordbar [7] acknowledges the importance of reusing artifacts from existing technological spaces and provide an approach based on refinement of metamodels for doing so. However, it only discusses the refinement of the involved *metamodels*, but not the transformation of the associated *models*.

A very generic framework for bridging between technological spaces is discussed in Wimmer and Kramler [8]. In this framework a compiler-compiler is used that, based on an attributed grammar mapping EBNF to MOF, generates a so-called grammar parser. This grammar-parser not only transforms EBNF grammars into MOF-based metamodels, but also generates a tool to transform programs associated with that EBNF grammar into models associated with the generated metamodel. This approach requires the availability of an EBNF grammar for the source of the transformation, consequently it only considers the grammarware technological space as a starting point.

Graaf et al. [9] uses the MDA itself as migration technology. In this work the source of the migration was obtained manually by normalizing models found in the design documentation. To this end, a set of domain-specific concerns was identified for which standardized idiom was defined. In the current work the involved models are much too large to manipulate manually.

The previously mentioned work assumes that metamodels and grammars are available. Unfortunately, obtaining these metamodels and grammars often is the essence of the problem, as was argued in face of the year 2000 problem in Lämmel and Verhoef [10]. It provides a convenient description of different cases with respect to obtaining a grammar (grammar stealing). In our case we had to recover the grammar from documentation, model instances, supporting tools, and developer knowledge, a situation not covered by their scheme.

Van Deursen and Klint [11] provides an assessment of the maintenance benefits of domain-specific languages. One of the conclusions is that as more domain-specific languages and code generators are being used, the maintenance burden will shift towards the adjustment of the code generators to ever changing platform requirements. In the present paper we study how to resolve these problems by shifting this maintenance burden to the MDA technological space.

3 Motivating Case: The INVINCIBLE Software Factory

3.1 Current Situation

In 1997 Fortis Insurance Netherlands introduced a new software development environment: the INVINCIBLE Software Factory. This environment is based on application modeling using proprietary domain-specific modeling languages (DSML's) and code generation. It is used to develop applications related to insurance and banking. The most important application INVINCIBLE currently supports is a mortgage tender system.

At the time, the introduction of a software factory based on proprietary DSML technology had several advantages. First, by abstracting the company's technical infrastructure into application frameworks and code generators, modelers only have to focus on the functionality of the information system, enabling domain experts, instead of software engineers, to model insurance products. This provided the ability to rapidly adapt to changes in the market and launch new products. As such time-to-market was reduced. Second, by modeling and generating the user interface, all applications developed using

INVINCIBLE have a uniform look-and-feel. Finally, communication between systems is dealt with by a communication framework, which implies that all applications developed using the Software Factory use the same communication mechanisms.

The INVINCIBLE Software Factory consists of four main platform components (Fig. 1):

- *Definition Manager* The definition manager is INVINCIBLE's integrated development environment. It is used to create application models in one of the supported DSML's.
- *Model Repository* A DB2 relational database in which application models are stored.
- *Generators* The generators of the Software Factory generate implementation code from the application models, including:
  - C++ client application code,
  - C++ server application code,
  - Oracle SQL to update or create server-side database tables, and
  - Staffware Workflow Manager scripts
- *Frameworks* The generated code and the deployment infrastructure is linked by frameworks. These frameworks provide a platform-independent interface to support the generated code, and a platform-specific implementation to support communication with the underlying infrastructure.

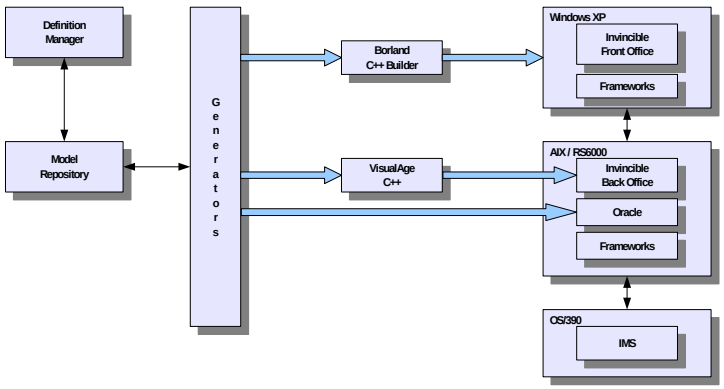

Figure 1. Overview of the INVINCIBLE infrastructure

Application development using INVINCIBLE is based on six modeling languages and an object-oriented domain-specific programming language called COOL (Common Object Oriented Language). Each of the six modeling languages is used to define one of the six models that (together with the code generators and frameworks) define an application. Each of the modeling languages allows to address different aspects of an application; the COOL language is used in these models to specify the business logic and constraints. The models include a model of the application domain (Core Product Model), a model of the user interface (Interface Model), and a model of the application's runtime structure (Process Model).

In our case study we limit ourselves to the Object Model defined in INVINCIBLE's Object Modeling Language. The Object Model is similar to a UML Class diagram as it contains modeling constructs such as 'Class', 'Attribute' and 'Association', along with more specific constructs such as, 'Domain' and 'Multiplicity Type'. The term 'Object' in the name Object Model Language is misleading, however, as this language is used to model design-time *classes* instead of run-time *objects*.

3.2 Current Problems

At the time of its introduction, the INVINCIBLE Software Factory was presented as Fortis' new strategic platform for insurance information systems. With the upcoming of the Internet, and with it, e-business and 24/7 online information systems, the strategic platform shifted to .NET and J2EE based technologies. This shift of platform can be explained by some drawbacks of the INVINCIBLE Software Factory, related to the inability of systems based on DSML's to easily accommodate changes to technological and user domains:

- The applications that are built in this factory are fat-client/server applications, which do not meet the web-based application standards to support online business-to-consumer transactions, for instance. Unfortunately INVINCIBLE is not flexible with respect to such requirements.
- Because INVINCIBLE is designed for one specific domain, applying it to other domains implies that it has to be modified. For example, the Core Product Model does not support collectivity as is known in a collective pension insurance. Expanding INVINCIBLE to support such constructs is expensive.
- The proprietary character of the DSML's implies a low availability of supporting tools and skilled developers. Expansion of the tool-set is expensive as everything has to be custom built. Furthermore, new developers first need to be trained. This implies inflexibility with regards to the expansion of the application development team.

At the time of writing the INVINCIBLE Software Factory approaches the end of its lifecycle. However, the complex applications that are built (and maintained) with this software factory are business critical and will have to be supported in the future. Because of this, developing these applications from scratch is not an acceptable solution.

3.3 Envisioned Solution

Fortis is investigating the possibilities to migrate INVINCIBLE and the supported applications to a new approach that overcomes the problems mentioned in the previous section. As such, the new solution has to support development platforms such as .NET and J2EE, is preferably based on standards, and is more flexible with respect to domain or technology changes. Finally, the benefits of model-driven engineering have to be maintained.

At the same time a number of Fortis developers are already familiar with UML. Together, this made MDA a candidate to investigate as a replacement. The MDA is based on standards, of which the UML is an important one, and supports a wide range of platforms. Furthermore, domain- and technological-specific concerns can be separated by encapsulating them in separate models and model transformations.

4 DSML's vs. MDA

4.1 DSML's

A way to bridge the gap between requirements and design models is to limit the domain of the requirements model and to raise the functional level of the design models. A domain specific (modeling) language is specialized to a specific user domain. The number of different entities and relations between entities are limited, as are the transitions entities can undergo. The functional level of the system models is raised by using frameworks tuned to the entities, relations and operations of the user domain.

The power of a domain-specific language is also its disadvantage. Because of its specialization in both the user as well as the technological domain, it is not able to easily adjust to changes within these domains [11]. In the user domain new entities, new relations and new operations might be introduced, in the technological domain new technologies.

4.2 MDA as an Alternative

The MDA aims at solving some of these problems by providing a set of standards for defining (modeling) languages and model transformations. MDA allows, with the help of UML profiles and with the MetaObject Facility (MOF), the definition of specialized languages appropriate for various abstraction levels and aspects of systems [12]. For this the MDA defines a 'modeling stack' that includes models specified in languages defined by metamodels (e.g., UML), that, in turn, are specified in a language defined by a metametamodel (MOF).

In the MDA vision the software development process consists of a series of gradual transformations from requirements to design. Starting with a requirements model, new models are constructed by adding technical artifacts and by instantiating parameters, types, multiplicities, and so on. Each model can, by the addition of technical artifacts, be considered as an implementation of the preceding model. The abstraction level of the technical artifacts will change from abstract to concrete, at the level of a 3GL or a framework. The key-enabler for these model transformations is the shared metametamodel, MOF.

Because the MDA is based on standards it is interesting for tool vendors and research groups to implement and study possible tool support. MOF implementation are such an example. The MOF is implemented by the Eclipse Modeling Framework (EMF) project [13] as an Eclipse plugin. Another implementation is made available by NetBeans as the Metadata Repository (MDR) [14]. These MOF implementations allow for the generation of API's based on MOF-based metamodels that make it possible to create, manipulate and persist associated models. As such, MOF implementations can, in turn, be used for the implementation of MDA model repositories and transformation tools. Examples of the latter are IBM alphaWorks' Model Transformation Framework (MTF) [15], and the ATLAS Transformation Language (ATL) [16].

4.3 Migration Steps

For a complete migration towards the MDA including migration of the supported legacy applications to new platforms, both the components of the MDE environment (application frameworks, code generators, and other tools) and the supported applications

(application models, and application data) have to be migrated. Code generators, for instance, might be replaced by model transformations that can be executed using generic model transformation tools. Here, we only discuss the migration of the existing application models. Migration towards the MDA implies that these models are migrated to MOF-based models.

As the target of the migration is within the MDA technological space, we decided to use MDA model transformation technologies for the *migration* itself. The use of a MDA model transformation language enables the definition of the transformations on a high level of abstraction (that of metamodels) such that they can be reused for all applications supported by INVINCIBLE. Furthermore, we circumvent the need to consider the concrete XMI syntax, for example.

The application of the MDA in a migration context implies the definition of source and target metamodels. In many cases UML is the target metamodel, but it can also be some custom defined (MOF-based) target metamodel. The source metamodel has to be a MOF-based metamodel of the involved legacy application model. This raises two problems: 1) how to derive this metamodel, and 2) how to instantiate it.

Deriving a MOF-based source metamodel is often not straightforward, for instance, when up-to-date specifications of the modeling languages are not available. This can be solved by considering other sources, such as instances of legacy models, development tools, structure of model repositories, and developer knowledge.

Once a MOF-based source metamodel is defined (recovered), conforming models have to be instantiated based on the legacy source models. This model population step clearly distinguishes the application of MDA in a *development* context from its application in a *migration* context. Assuming the legacy applications are not developed using the MDA, the starting point for a migration is outside the MDA technological space. Therefore, a bridge between the source technological space and the MDA technological space needs to be created before MDA model transformations can be applied. To implement this bridge the legacy application models need to be extracted from the repository. The result is subsequently used to populate a model that conforms to the MOF-based source metamodel.

An additional problem is the fact that legacy MDE systems often store models in a repository based on a relational database. For model storage the legacy MDE system implements a mapping between the structure implied by the application modeling language (e.g, as used in INVINCIBLE's Definition Manager) and the relational structure of the repository. To use the models as stored in the repository as input for the migration, a normalization step is required to reverse that mapping. As such, normalization recreates the original structure of application models. The stage at which to introduce this step, i.e., after a model is populated, or in the bridge, is a decision to be taken when defining the migration process. It is related to the type of MOF-based metamodel that is recovered and the type of normalization required. The metamodel could, for instance, follow the relational structure of a model repository. On the other hand it could also follow the intended structure of the legacy modeling language. In the former case the normalization step would be inserted as MDA model transformations after a conforming model is populated; in the latter case it can also be inserted in the bridge, preceding model

population and following model extraction. As such, this decision is context-dependent and can have a practical motivation.

5 Migration of INVINCIBLE towards the MDA

The previous section identified a number of migration steps: model transformation, model population, model extraction, and normalization. We started the implementation of these steps with the definition of a MOF-based source metamodel for the transformation step. To instantiate conforming models based on the models as stored in the repository, we then implemented the bridge consisting of the model extraction and population steps.

Because the Object Model Language is object oriented and the Model Repository is based on a relational database we have to define a normalization step. Furthermore, we have to decide where to insert this step in the migration process. This decision depends on the source metamodel and the type of normalization required. To be able to validate the metamodel and model transformation step in collaboration with INVINCIBLE developers, we decided that the source metamodel should resemble the intended conceptual model of the Object Model Language shared by those developers (instead of the relational structure of the Model Repository). Furthermore, we implemented the bridge using XML and for a large part normalization involved simple restructurings. Therefore, we decided to insert the normalization step in the bridge between model extraction and model population using Extensible Stylesheet Language Transformations (XSLT).

Finally, a target metamodel and corresponding model transformations to map source and target models need to be defined. As the Object Models are similar to UML Class Diagrams, the UML was selected as the target model.

We used the ingredients discussed above to implement the migration process depicted in Fig. 2. They are discussed in more detail below.

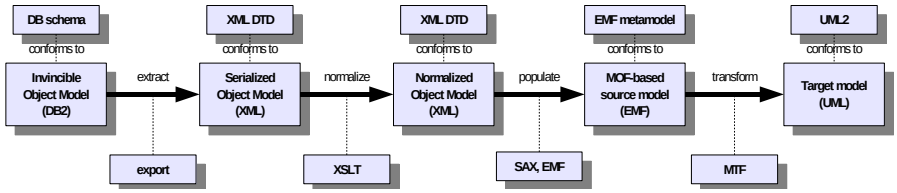

Figure 2. Steps in the migration process

5.1 Deriving a MOF-based Source Metamodel

In order to arrive at a MOF definition of the source metamodel corresponding to the INVINCIBLE Object Model, three sources of information were used: instances of the Object Model, a specification of the Object Model Language that was known to be obsolete (yet the best one available), and the INVINCIBLE Definition Manager. Note that, we could not use the database schema of the repository directly, because the current

state of the INVINCIBLE Software Factory is such that much of the structure of Object Models is actually introduced by the Definition Manager, and is not represented in the structure of the underlying database.

Object Model instances are stored in the INVINCIBLE Model Repository. From these instances (as represented by the Definition Manager) an initial version of the metamodel was constructed. Although it is not guaranteed that this metamodel is complete and correct, all the models in the repository conform to it. Considering the fact that DSML's are typically small, which also applies to INVINCIBLE Object Model Language, this is a practical and sufficient solution in many cases.

The obsolete Object Model Language specification was used to validate the initial metamodel. When an inconsistency between model and specification was discovered, the model instances and the language were reexamined in order to find the correct metamodel structure.

In turn, the metamodel that resulted from the validation and adaption, was validated by inspecting all possible model elements and relations that could be created with the INVINCIBLE Definition Manager. If it supported the creation of model instances that were inconsistent with the metamodel, that specific part of the metamodel and the language were reexamined. This proved to be necessary in a number of cases where the Definition Manager supported creation of model constructs, that appear in the original design of the modeling language, but that were not supported by the other components of the INVINCIBLE Software Factory, and thus were never used.

Finally we manually added generalizations to the resulting metamodel, for elements that share a common set of properties and are conceptually related to each other. This resulted into the (final) MOF-based source metamodel.

5.2 A Bridge to the MDA Technological Space

The first step in the bridge to the MDA technological space is the extraction of existing models from the legacy repository. The INVINCIBLE Definition Manager has functionality to export the models to the C++ code generators. This facility creates a directory for each table and a file for each row, the files are in plain text using key-value pairs. We decided to use this to implement a bridge based on XML.

We modified the Definition Manager's export module to export the application models as stored in the database to a single XML file (a database-XML bridge). Based on the database schema we constructed a DTD to which the extracted model-data could be validated. At this stage, the structure of the XML file represented the relational structure of the model repository. Subsequently, we normalized the XML file using several XSLT transformations to recover the OO structure of the Object Model. This is discussed in Section 5.3.

In the next step, we set up a MOF repository. We adopted the Eclipse Modeling Framework EMF, as it is free, is based on Eclipse (the current standard development environment for J2EE applications within Fortis), and is supported by various transformation frameworks (e.g., MTF [15] and ATL [16]). The drawback of using EMF is that it is not an exact implementation of MOF, which implies that we are using a non-industry-standard technology. There is, however, the possibility to transform EMF models into MOF models [17,18].

EMF provides functionality to generate a Java implementation of a metamodel, that includes an API for the manipulation of conforming models. As such, using the Java API generated for the Object Model metamodel, it is possible to instantiate Object Models. The metamodel implementation instantiates the model in memory and serializes it to an XML file using XMI when it is saved.

Note that circumventing the need to setup a MOF repository by directly transforming the legacy XML into an XMI representation is not a viable option; it would be a tedious and error-prone task to generate a document in a complicated XMI syntax [12].

Finally, we populate the EMF repository using the extracted (and normalized) XML models. As the structure of the XML file corresponds to the structure of the source metamodel, it is straightforward to use a SAX parser to parse the XML model and instantiate the appropriate model elements using the generated Java API in the SAX-handler. In an external property file the mapping between XML nodes and metamodel constructs is specified. For instantiating the model, the Object Model metamodel API is called using reflection; this technique uses the names of the XML nodes to generate the names of the classes and methods to be called using the generated Java API.

Because the populator makes use of reflection and a property file to define the mapping between XML nodes and metamodel constructs, the populator can be reused to populate other metamodels, as long as the input data conforms to certain defined prerequisites (e.g., the names of the XML nodes have to be the same as the names of the metamodel constructs).

5.3 Normalization: Recovering Object-Oriented Structure

The INVINCIBLE application models are object oriented, i.e. they contain classes, relations (associations, aggregations, and generalizations). However, they are stored in a model repository based on a relational database. Furthermore, the XML representation of the application models generated by the (modified) export module follows the relational structure of the repository. As a result we require a normalization step to populate the source model for the transformation step.

Several mappings from OO structures to relational structures have been described in literature [19]. These typically map a class model to a database schema in which tables correspond to classes. In a running application, objects can then be persisted as data in the corresponding tables.

In our case, the application models (that include elements such as Classes *and* Objects) are table data, while the database schema corresponds to the metamodel. This implies we have to deal with OO constructs on both the level of the database schema and the table data. Conceptually this makes no difference, as how to deal with the OO constructs on the data level is implied by the database schema [12]. However, to avoid confusion, in practice it is important to separate the two levels explicitly.

INVINCIBLE uses a horizontal mapping. It maps each metamodel element to a separate table. Properties of such elements (attribute list, method list, generalized by, etc) are mapped to table columns. In a horizontal mapping properties of generalizations are simply copied in the tables of their specializations as separate columns. Multi-valued properties of meta-elements are mapped to a separate table for the properties and an association table to connect the meta-elements table to the property table via primary

and foreign keys. Relations in the metamodel are also properties mapped to primary and foreign keys. Whether a relation is an association or a composite aggregation cannot be derived from the database structure.

The MOF-based source metamodel we defined is object-oriented. To populate a conforming source model using models in the repository, we must first extract. Subsequently, to normalize the extracted model, we reconstruct the OO structure of the source model; we have to reconstruct model elements and relations (aggregation, association, generalization).

The normalization is carried out in the bridge. We based the bridge on XML; the normalization is implemented using XSLT. The tree structure of the XML documents allows for simple representation of composite aggregations in the metamodel. The source metamodel indicates which relations are actually composite aggregations. In the (XML) model data recovered from the database, relations are represented by reference attributes (i.e., the primary/foreign key mechanism). During normalization these are resolved and, in the case of composite aggregations, represented using the XML tree structure. After normalization a model elements' attributes are also represented using XML's tree structure, i.e. as sub-elements of the XML elements that represent the involved model element. Other relations are also represented in the recovered model by reference attributes. These are used during the model population to instantiate association relations.

We implemented the normalization using XSLT in two stages. The first stage recreates the composite aggregations, associations and attribute structure. We compared the normalized XML file with the EMF Object Model metamodel. We identified violations of the metamodel in the XML representation of the model instance. In the second stage we used this information to bring the XML representation closer to the metamodel structure, including naming conventions and abstract class definitions.

During the normalization we encountered problems with respect to external objects and performance. External objects are objects that are referred within a model but which are not defined in it. This is for instance the case for objects defined in the INVINCIBLE Framework that are referred to from within the application models. In order to resolve these references, the INVINCIBLE Framework was exported from the repository and added as input to the normalization step.

The performance of XSLT was another problem. This was mainly caused by the fact that every element in the recovered XML file, i.e. every representation of an instance of a metamodel construct, is a direct child of the root element. This implies that when the XML file needed to be searched to create a composite aggregation, every child element of the root element was included in the search.

5.4 Transforming the MOF-based Source Model

The last migration step is the actual transformation of the MOF-based source model conforming to the EMF Object Model metamodel to a target UML Class Model. For this, we use MTF [15].

MTF uses a special language to define model transformations in terms of transformation rules. Each rule specifies how source metamodel elements are related to target metamodel elements. We followed a straightforward strategy for defining the rules; we

simply determined for each source metamodel element what UML model elements can be used to represent that element in a UML Class Diagram.

5.5 Case Study Results

We applied our approach to the Object Model of the most important application developed using INVINCIBLE: a mortgage tender system. This application has approximately 350 end-users. Our generic model extractor was capable of extracting all six models. This resulted in a model containing tens of thousands of model elements, serialized in 55 MB of model data. The Object Model was one of the larger sub models resulting in 10 MB of XML data. As said normalization of this extracted data was expensive, the XSLT transformation took 4 to 5 hours. Subsequent steps, model population and execution of MTF model transformation, executed much faster. Each took less than a minute. More details on the case study implementation of the migration can be found in [20].

6 Discussion

UML vs. custom MOF-based metamodels In retrospect we discovered that, although the Object Model metamodel contains many constructs that are similar to UML metamodel constructs, the mapping is not straightforward. The main problem is that the Object Model metamodel contains constructs that cannot be directly mapped to the UML without using its extension mechanisms, such as, for example, the Object Constraint Language (OCL). This is a problem because those mechanisms are poorly supported by currently available tools.

One such example is the so-called Domain Type in the Object Model metamodel. In the Object Model of the mortgage system 'Sensible Age', defined as an integer between 0 and 150, is an example of such a Domain Type. Without using OCL, it is impossible to specify such types in UML. Furthermore, because MTF currently does not support OCL, an exact mapping cannot be defined. Therefore, we had to map such constraints to UML comments. Several other examples were encountered during the case study.

These examples show the problems that can arise when mapping a DSML onto a generic modeling language. Although we do not claim that such mappings cannot be defined at all, we do question whether a direct mapping of these languages is desirable.

With the rise of the MDA a shift becomes possible from modeling in a 'unified' modeling language to modeling in specialized custom modeling languages aimed at specific problem domains. From a conceptual point of view, these custom languages offer advantages such as inherent model validation (syntactically correct by construction), the possibility to define domain specific model elements and smaller metamodels (compared to UML) resulting in less complicated model transformations.

From a practical point of view, the UML has the advantages of tool availability and familiarity with the language among architects, developers and domain experts. In fact, this was what made Fortis decide to use UML as a target model in the first place.

However, with the rapid adoption of MDA by tool-vendors and, with it, the upcoming availability of more and more MDA tools, the practical limitations of MDA technology seem to disappear. The discussion therefore is whether the UML is the correct target model in a DSML migration, as MOF offers the advantages of modeling in a language

aimed at a specific problem domain (the original DSML advantage), being an industry standard language as well as getting more and more tool-support, by both tool-vendors as well as open-source communities.

For our migration approach, however, it makes no difference whether the target metamodel is UML or a custom-defined MOF metamodel. In fact only the definition of the model transformation step would have to be adapted.

Generalizing the Approach In this paper we mainly considered the migration of only one of the six types of INVINCIBLE's models, Object Models. However, other INVINCIBLE models are defined similar to the Object Model metamodel. In fact, a basic metamodel of the User Interface Model metamodel has already been constructed during the analysis phase of this project. Second, the Application Model metamodel and System Model metamodel are already contained in the currently used MOF-based metamodel.

Because the MOF is a metametamodel in which modeling languages can be defined, (almost) every metamodel can be constructed in the MOF. This implies that a condition for the application of the proposed migration strategy to other software factories is that their metamodels can be defined in the MOF. A second condition for the application of the proposed approach is that the (proprietary) models are accessible, extractable and normalizable.

Phased Migration A complete migration of the INVINCIBLE Software Factory to an MDA-based approach with full code generation is currently impossible. Moreover, a 'big bang' migration of such a complicated development environment, including the business critical applications it supports would poses an unacceptable risk. A phased migration would mitigate this risk.

It is possible to apply our approach to implement such a phased migration strategy. By transforming all INVINCIBLE models to MOF based models, and by implementing a MOF based model repository to replace the current repository, an environment can be created in which the models are stored in an industry standard format, while the current generators and development tools can be preserved. This strategy ensures the continuity of development and support regarding the current (business critical) applications. Second, it creates an opportunity to develop new development tools which use, and are based on, standard technologies (e.g. XMI, JMI, Eclipse, GEF, GMF). Finally, although the current mapping to UML we defined is not complete, our approach can be used to visualize application models using standard UML tools, while other tools (e.g., code generators) might be developed based on the MOF metamodels derived for the different application models used.

Normalization The normalization step applied in this case study has a strong proprietary nature. As mentioned earlier, the question is when to insert this normalization in the migration process. There are basically two options, applying the normalization as part of the bridge or, after that within the MDA technological space. The decision should be made on a pragmatic basis, as the normalization is a necessary intermediate step in the migration process and not the final goal. This decision is highly dependent on the

proprietary data format from which the migration process is started, the final target of this process, the type of normalization required, and the implementation of the bridge.

In our case, we did not base the source metamodel for the transformation step on the relational structure of the database, resulting in two possibilities to insert the normalization step: in the bridge, or as model transformation. Note that, the latter requires the definition of a (MOF-based) relational metamodel as source for the normalization step. Furthermore, the type of normalization required could be conveniently implemented using transformations in XML (using XSLT), the technology of the bridge. Therefore, we assigned the normalization step to the bridge.

7 Concluding Remarks

In this paper, we have addressed the issues surrounding the migration of a successful yet proprietary model-driven development environment based on a set of domain-specific modeling languages towards an implementation based on the MDA-approach. We specifically focussed on the migration of the application models defined in one of those languages. The migration itself was implemented using the very same MDA technologies. We consider the following as our most important contributions:

- We described the problems a model-driven approach, such as the INVINCIBLE Software Factory, can face.
- We identified the necessary steps for the migration of a proprietary MDE environment towards the MDA, including a bridge between the involved technological spaces.
- We applied MDA itself in a migration context, which requires a normalization step. We demonstrated how the migration steps can be implemented in the context of an an industrial-strength application built using INVINCIBLE.
- We discussed and analyzed the decisions to be taken when designing and implementing a migration of a proprietary MDE approach to the MDA.

Future work consists of extending our work with respect to the case study we presented by applying our approach to the other application models as well. Furthermore, we will address the migration of the business logic implemented in COOL code. Finally, we are studying to what extent generic tool support for the migration steps we provide can be implemented. Naturally, some steps will remain context specific. Model extraction for instance is largely dependent on the type of model repository. For the model population step on the other hand it might be possible to implement a model populator that can instantiate arbitrary MOF models.

Acknowledgements We thank NWO for sponsoring part of the research described in this paper via the Jacquard Reconstructor project. Furthermore, we thank Piet van Horssen of Fortis for sharing his knowledge on the INVINCIBLE Software Factory

References

1. OMG: MOF 2.0 Query/Views/Transformation RFP. `http://www.omg.org/docs/ad/02-04-10.pdf` (2002)

2. Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A.: Transformation: The missing link of MDA. In: Proc. 1st Int'l Conf. Graph Transformation (ICGT 2002). vol. 2505 of Lecture Notes in Computer Science., Springer-Verlag (2002) 90–105
3. Mansurov, N., Campara, D.: Managed architecture of existing code as a practical transition towards mda. In: UML Modeling Languages and Applications: ≪UML≫ 2004 Satellite Activities. vol. 3297 of Lecture Notes in Computer Science., Springer-Verlag (2005) 219–233
4. Architecture-Driven Modernization Task Force: Architecture-Driven Modernization (ADM). http://adm.omg.org/ (2006)
5. Architecture-Driven Modernization Task Force: Architecture-Driven Modernization scenarios. Technical report, OMG (2006) http://adm.omg.org/ADMTF_Scenario_White_Paper(pdf).pdf.
6. Kurtev, I., Bézivin, J., Aksit, M.: Technological spaces: an initial appraisal. In: Confederated Int'l Conferences CoopIS, DOA, and ODBASE 2002, Industrial Track, Springer-Verlag (2002)
7. Staikopoulos, A., Bordbar, B.: A metamodel refinement approach for bridging technical spaces, a case study. In: Proc. 4th MoDELS Workshop in Software Model Engineering (WiSME 2005). (2005) http://www.planetmde.org/wisme-2005/.
8. Wimmer, M., Kramler, G.: Bridging grammarware and modelware. In: Satellite Events at the MoDELS 2005 Conference: MoDELS 2005 Int'l Workshops. vol. 3844 of Lecture Notes in Computer Science., Springer-Verlag (2006) 159–168
9. Graaf, B., Weber, S., Van Deursen, A.: Migration of supervisory machine control architectures using model transformations. In: Proc. 10th European Conf. Software Mainteenance and Reengineering (CSMR), IEEE CS (2006)
10. Lämmel, R., Verhoef, C.: Cracking the 500-language problem. IEEE Software **18**(6) (2001) 78–88
11. Van Deursen, A., Klint, P.: Little languages: Little maintenance? J. Software Maintenance **10** (1998) 75–92
12. Frankel, D.S.: Model Driven Architecture: Applying MDA to Enterprise Computing. OMG Press (2003)
13. Eclipse Foundation: Eclipse Modeling Framework (EMF). http://www.eclipse.org/emf (2005)
14. Netbeans.org: Metadata repository (MDR) project home. http://mdr.netbeans.org/ (2006)
15. IBM alphaWorks: Model transformation framework. http://www.alphaworks.ibm.com/tech/mtf (2006)
16. ATLAS Group: The ATL home page. http://www.sciences.univ-nantes.fr/lina/atl (2005)
17. Gerber, A., Raymond, K.: MOF to EMF: there and back again. In: Proc. OOPSLA workshop on Eclipse Technology eXchange, ACM Press (2003) 60–64
18. Duddy, K., Gerber, A., Raymond, K.: Eclipse modelling framework (EMF): import/export from MOF / JMI. Status report, Co-operative Centre for Enterprise Disributed Systems (DSTC) (2003) http://www.dstc.edu.au/Research/Projects/Pegamento/publications/MOF2EMF-Tech-Report.pdf.
19. McFarland, G., Rudmik, A., Lange, D.: Object-oriented database management systems revisited. Technical Report DACS-SOAR-99-4, DoD Data & Analysis Center for Software (DACS) (1999) http://www.dacs.dtic.mil/techs/oodbms2/oodbms2.pdf.
20. Doyle, D.G.: Model transformation of domain-specific models. Master's thesis, Delft University of Technology (2005) http://swerl.tudelft.nl/twiki/pub/Main/DuncanDoyle/DuncanDoyleMsc2005.pdf.