A Framework for the Development and Deployment of Evolving Applications

Georgios Voulalas and Georgios Evangelidis

University of Macedonia, Thessaloniki, GR-54006, Greece
{*voulalas*, *gevan*}@*uom.gr*

Abstract. Software development is an area in which there exist many major problems one has to struggle with. Model-driven development improves productivity by introducing formal models that can be understood by computers. Through these models the problems of portability, interoperability, maintenance, and documentation are also successfully addressed. However, the problem of evolving requirements, which is more prevalent within the context of business applications, additionally calls for efficient mechanisms that ensure consistency between models and code, and enable seamless and rapid accommodation of changes without interrupting severely the operation of the deployed application. This paper introduces a framework that supports rapid development and deployment of evolving web-based applications and is based on an integrated database schema.

1 Introduction and Motivation

Software development is an area in which we are struggling with a number of major problems. The most important ones are [7]:

The Productivity, Documentation, and Maintenance Problem. The software development process includes a number of phases: (a) Conceptualization and requirements elicitation and gathering, (b) Analysis and functional description, (c) Architectural specification and design, (d) Implementation, (e) Testing, and, (f) Deployment. Whether we use an incremental and iterative process or the traditional waterfall process, documents and diagrams are produced during phases (a) through (c). Practically speaking, analysis and design artifacts are required, but to be really productive they should not be just static, paper representations. They have to stay in high cohesion with the code throughout the software lifecycle, elevate technologists above the lower level complexities and be eligible as input in forward-engineering operations.

The Portability Problem. Each year, and sometimes even faster, new technologies are being invented and becoming popular. The new technologies offer concrete benefits for companies and many of them can not afford to lag behind.

The Interoperability Problem. Software systems rarely live isolated. Most systems need to communicate with other, often legacy, systems.

The Evolution Problem The management of evolution in information systems is a dominant requirement. This is even stronger in business applications, due to the dynamic nature of business domains [11].

In order for evolution to be handled efficiently the following objectives should be met: (a) changes should be seamlessly incorporated without the need of restructuring the existing application, (b) analysis and design artifacts should be updated in order for changes to be reflected, (c) the operation of the deployed application should not be interrupted, or at least interruption should be minimal, and, (d) access to old business objects within their right context should be supported, i.e., at any time one should be able to easily retrieve and examine an old business object through the specific version of the application that produced and manipulated it, in order for a user to be able to trace back to former business data.

The remainder of the paper is structured as follows. In Section 2, we present the Model Driven Architecture (MDA) and the modern practices brought out by Microsoft. In Section 3, the deficiencies of the MDA are discussed along with some key thoughts. In section 4, a new framework for the development and deployment of web-based business applications is introduced. The last section provides a conclusive summary of the paper and identifies our future research plan.

2 MDA & Microsoft Software Factories

The Model Driven Architecture [8, 9, 7] is a framework for software development defined by the OMG. The MDA development lifecycle is not very different from the traditional lifecycle; they both involve the same phases. One of the major differences has to do with the nature of the artifacts that are produced during the development process. The artifacts are formal models, i.e., models that can be understood and processed by computers. The following three models are at the heart of the MDA.

Platform Independent Model (PIM). This model is the first to be defined and is a model with a high level of abstraction that is independent of any implementation technology. Within a PIM, the system is modeled from the aspect of how it best supports the business requirements. Whether a system will be implemented using the .NET or the J2EE framework plays no role.

Platform Specific Model (PSM). In the next step, the PIM is transformed into one or more PSMs. A PSM specifies the system (or part of the system) in terms of the implementation details that are defined by one specific implementation technology.

Code. The final step in the development is the transformation of each PSM to code. Because a PSM fits its technology rather closely, this transformation is relatively straightforward.

MDA transformations, in contrast to traditional development, are always executed by tools. Many tools are able to transform a PSM into code; there is

nothing new to that. What is innovative in MDA is that the transformation from PIM to PSM is automated as well.

Let us now clarify how MDA responds to the challenges presented in the previous section.

Productivity, Documentation and Maintenance: In MDA the focus for a developer shifts to the development of a PIM. The PSMs and the code are generated automatically. The PIM fulfills the function of high-level documentation, and is not frozen after writing, since changes made to the system will eventually be realized by changing the PIM and regenerating the PSMs and the code.

Portability: Portability is achieved by focusing on the development of PIMs that are by definition platform independent.

Interoperability: When PSMs are targeted at different platforms, they can not directly talk to each other. Concepts from one platform should be transformed into concepts used in another platform. MDA addresses this problem by generating not only the PSMs, but the necessary bridges between them as well.

Evolution Management: The PIM is a live artifact that depicts precisely the system throughout its lifecycle. Changes to the system are made by changing the PIM and regenerating the PSMs and the code.

At the same time Microsoft is working on Domain Specific Languages (DSLs). DSL's [4] are programming languages dedicated to specific problems and consisting of their own built-in abstractions and notations. DSLs underpin Microsoft's concept of software factories, that are planned modules of tools, content and processes used to build applications in specific domains like healthcare, human resources or enterprise resource planning. Narrowing the domain enables to more precisely define the features of the target family and facilitates the definition of languages, patterns, frameworks and tools that automate the development of its members. One early backer of the DSL and Software Factories approach is Borland.

3 Rethinking MDA

MDA is a complete framework that enables organizations to respond efficiently to the augmentative requirements of modern software projects. However, the current status of the framework, as it is mainly shaped by the availability of support tools, presents the following deficiencies [7]:

- Though OMG has defined the mapping standards between the three models (the PIM, the PSM and the code), it has yet to define how to implement the models.
- Current tools are not sophisticated enough to fully provide the transformations from PIM to PSM and from PSM to code.
- The extent to which portability can be achieved depends on the automated transformation tools that are available. For popular platforms, a large number of tools will undoubtedly be available. For less popular platforms, the user may have to use add-on tools, or write proprietary transformation definitions.

– Cross-platform interoperability can be realized by tools that not only generate PSMs but the bridges between them as well. Existing tools are not so advanced to cope with this requisite.

Undoubtedly, it is a matter of time before software vendors overcome the above-mentioned limitations. However, there exist a number of shortcomings that are inherent with the core principles of MDA. Specifically, MDA suffers from:

– **Inability to ensure consistency between the produced code and the preceding models**. Even if vendors succeed in building transformation tools that fully generate the required code based on the specifications modeled in the PSMs, one can not guarantee that developers will not interfere manually with the generated code. Consequently, the consistency between the three cornerstone models is unstable.
– **Weakness to cope efficiently with the problem of evolving requirements**. In MDA, every new change requires code to be regenerated and recompiled, and the final application to be redeployed. Even worse, the arbitrary realization of changes may create gaps between the three models. Last but not least, MDA can provide access to data that have been manipulated by previous versions of the application, only by maintaining different installations of the applications, approach that is a neither practical, nor elegant.

4 The Proposed Framework

Motivated by the deficiencies of the MDA paradigm, its core principles, and the latest practices adopted by Microsoft and Borland, we introduce an innovative alternative for the realization of a development and deployment framework targeted to web-based business applications. The proposed framework will be structured on the basis of a universal database schema (meta-model). Development will be supported by components that will elicit functional specifications from users and transform them in formal definitions, and by data structures (part of the meta-model) that will be utilized for the storage of the definitions. Deployment will be supported by generic components (meta-components) that will be dynamically configured at run-time according to the functional specifications provided during development, and by application-independent data structures (part of the meta-model) that will hold all application-specific data. The following two statements outline the philosophy of the proposed solution:

– No code (SQL, Java, C++, JSP, ASP, etc.) will be generated for the produced applications; just run-time instances of generic components will be created.
– There will always exist one deployed application, independently of the actual number of running applications. Application-specific behavior will be rendered by this universal application according to the functional definitions

that are maintained in the database. In other words, functional and presentation specifications are shifted from the middle and front tier respectively to the database tier (we take as basis the 3-tier approach that is the most outstanding architectural paradigm). Response to business changes is instant, simply through the manipulation of data tuples.

More specifically, the proposed framework includes the following models.

4.1 Domain Model

The Domain Model is a business-oriented model that maps to the MDA Platform Independent Model (PIM). It defines the structure of the data that the application is working on (objects, attributes, and associations), along with their behavioral aspect (methods) and business rules. It is mainly structured on the basis of the Object-Oriented paradigm, augmented with the extensions introduced by the Object Constraint Language [10, 3] for the description of constraints that govern the modeled objects, plus elements from an acceptable business rules classification scheme [1, 2, 6], with the Ross method [1] being the prevalent. Therefore, its main entities are: Business Object, Status, Attribute, Method, Association, Argument, Term, Fact, Computation Rule, Pre-condition, Post-condition, Guard, Invariant Constraint.

Besides business rules and data processing logic, every business application incorporates mechanisms for enterprise modeling, business relationships establishment, role assignment, and personnel administration. Thus, the Domain Model embraces an additional component, named Enterprise Model, which covers inter-organizational and intra-organizational aspects. The main entities of this sub-model are: Business Role, Enterprise, Business Units, Partnership, Partner, Employee, Role, User.

Although the entities included in the Enterprise model can be implemented as instances of the meta-entities of the core Domain Model, we have selected to handle them separately for reasons of performance. Thus, instead of dynamically configuring the meta-entities to render the desired functionality, we utilize standard entities. This differentiation stems from the fact that the mechanisms implemented by the Enterprise Model can be specified in advance, as they are common among all business applications.

Specifications included in the Domain Model will be stored in a database. The database schema should embrace the proposed structure and include all identified entities (Business Object, Method, Rule, etc.).

4.2 Application Model

The Application Model maps to the MDA Platform Specific Model (PSM) and focuses on the targeted platform. The Application Model contains the following three sub-models:

Presentation Model: It pictures the overall structure of the presentation elements. Display pages are defined for every business object based on the identified attributes. Input pages that elicit the information required for the execution

of the methods are defined based on the specified methods and arguments. Pages are interrelated according to the identified object associations.

Business Logic Model: Suppose that we select J2SE as target platform. All objects and terms will be mapped to the 'java.lang.Object' class. Alphanumeric attributes will be mapped to 'java.lang.String' class. A method (or piece of a method) that returns part of an alphanumeric will be mapped to the 'substring' method that is implemented by the 'java.lang.String' class. Similarly, a computation rule will be mapped to a set of primitive methods supported by the target platform that will be invoked in specific order in order for the rule to be propagated. In general, all elements included in the Domain Model will be mapped to fundamental elements of the target programming language.

Data Model: Based on the identified objects, their attributes and the way they are associated, a data model is structured. Only persistent objects (i.e. objects that need to "survive") are mapped to database structures. The discrimination between persistent and transient objects is captured in the domain model.

4.3 Operation Model

The Operation model consists of the following building blocks.

Presentation Model Instance: Run-time instances of generic presentation elements (e.g., Java Server Pages or Active Server Pages that obey to specific Cascading Style Sheets).

Business Logic Model Instance: Run-time instances of the generic functional components (meta-objects) that render the behavior of an application-specific object. The exact process is the following: application specifications are retrieved from the database at run-time and the generic components are configured dynamically in order to expose the specified functionality by utilizing reflectional adaptation techniques (reflection is the process by which a program can modify its own behavior and is supported by many object-oriented programming languages). For each different technology utilized at the Application Level (J2SE, .NET, J2EE), different components should exist. Practically speaking, every programming language that supports reflectional behavior can be utilized.

Data Model Instance: This is the part of the unified database schema that will hold the realizations of the business object instances (e.g., realizations of the travel applications, orders, products, etc.). The database schema will be independent of the applications, i.e., its structure will be fixed. In [14, 15, 13] a framework for dynamically evolving database environments is introduced. Similar to our approach it is based upon a database structure that is independent of applications. Changes to the data structure of the application result to record modifications, instead of changing the schema itself. In comparison to our approach the specific research effort focuses only to the data side of applications.

Note that the three sub-models included in the Application Model are not transformed to code at operation level, except for the part of the Business Logic Model that originates from the Enterprise Model. Instead, the definitions that they include are coupled with the generic components (presentation elements,

functional components, and database) in order for the required functionality to be rendered.

The proposed framework responds to the challenges identified in Section 3 as follows:

Consistency between the produced code and the preceding models. Since no code is generated and the middle model is generated automatically in its entirety, all changes are realized through the Domain Model.

Efficient handling of evolving requirements. Having shifted the functional and presentation specifications from the middle and front tier respectively to the database tier we can easily achieve evolution management by applying standard data versioning techniques. In case the static (attributes) or dynamic (methods) definition of a business object is modified this results in modifications to the underlying data instances, i.e., we can deal with changes at deployment time without recompiling and redeploying the application. What's more we can, at anytime, refer to a previous version of an application and examine old data in their real context by retrieving the corresponding data instances from the database, without the need of maintaining multiple installations.

5 Conclusions and Further Research

In this paper we examine the development and deployment of web-based business applications through a different perspective: our main aim is to elaborate on and limit the side-effects that are induced by the continuously changing requirements, while conforming to the principles introduced by the MDA paradigm and retaining its advantages. For this reason, we suggest transferring the functional specifications of the application from the components to the database and utilizing them at run-time in order to configure generic components that will render the application-specific functionality. The development and deployment platform will be based upon a unified database schema. The generic components will be built with the use of a programming language that supports reflection. Dynamic functional specifications will let end-users deal with changes at deployment time. The operation of previous versions of an application will be feasible through the same, unique installation. Last but not least, the consistency between the three cornerstone models will not be compromised.

It should be clear that our goal is to present an interesting alternative that it could somehow be absorbed by the MDA initiative. Reviewing the proposed framework against the MDA framework, we identify the following drawbacks:

- The proposed framework has narrower scope, since it focuses on web-based business applications. This constraint is enforced by the fact that is practically infeasible to create a generator that can potentially produce any application [5, 13] and is in compliance with the latest developments as pictured by the initiatives undertaken by major software players.
- MDA handles efficiently cross-platform interoperability in terms of integration with other systems, while the current formulation of the proposed framework supplants the specific coordinate.

– Indisputably, a solution that is build upon a meta-model and extensively utilizes reflection requires increased computational resources compared to a traditional one. However, the availability of powerful computational resources encourages the elaboration of sophisticated solutions. Working towards a "lighter" solution, we will consider adopting partial behavioral reflection [12].

Future research will focus on extending the framework with a coordinate that will cover the need for cross-platform interoperability and implementing the required infrastructure.

References

1. Business Rules Forum 2004 Practitioners' Panel: The DOs and DON'Ts of Business Rules. Business Rules Journal. **6**(4) (April2005). http://www.BRCommunity.com/a2005/b230.html
2. Butleris, R., Kapocius, K.: The Business Rules Repository for Information Systems Design. ADBIS Research Communications (2002) 64–77
3. Coronato, A., Cinquegrani, M., Giuseppe, D.P.: Adding Business Rules and Constraints in Component Based Applications. CoopIS/DOA/ODBASE (2002) 948–964
4. Greenfield, J: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. (November 2004). http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/softfact3.asp
5. Guerrieri, E.: Case Study: Digital's Application Generator. IEEE Software, **11**(5) (1994) 95-96
6. Herbst, H.: Business Rules in Systems Analysis: a Meta-Model and Repository System. Inf. Syst. **21**(2) (1996) 147–166
7. Kleppe, A., Warmer, S., Bast, W.: MDA Explained. The Model Driven Architecture: Practice and Promise (Chapter One). Addison-Wesley. (April 2003)
8. Miller, J., Mukerji, J.: Model Driven Architecture A Technical Perspective. http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01
9. Miller, J., Mukerji, J.: Technical Guide to Model Driven Architecture: The MDA Guide v1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01
10. OMG: Object Constraint Language Specification. http://www.omg.org/cgi-bin/doc?ptc/2003-10-14
11. Roddick, J.F., Al-Jadir, L., Bertossi, L.E., Dumas, M., Estrella, F., Gregersen, H., Hornsby, K., Lufter, J., Mandreoli, F., Mannisto, T., Mayol, E., Wedemeijer, L.: Evolution and Change in Data Management - Issues and Directions. SIGMOD Record **29**(1) (2000) 21–25
12. Tanter, E., Noye, J., Caromel, D., Cointe, P.: Partial behavioral reflection: spatial and temporal selection of reification. OOPSLA (2003) 27–46
13. Wu, Jen-Her, Hsia, Tse-Chih, Chang, I-Chia, Tsai, Sun-Jen: Application Generator: A Framework and Methodology for IS Construction. 36th Annual Hawaii International Conference on System Sciences (IEEE - HICSS) (2003) 263–272
14. Yannakoudakis, E. J., Tsionos, C. X., Kapetis, C. A.: A new framework for dynamically evolving database environments. Journal of Documentation. **55**(2) (1999) 144–158
15. Yannakoudakis, E. J., Diamantis I. K.: Further improvements of the framework for dynamically evolving database environments. HERCMA (2001) 213–218