A FRAMEWORK FOR THE DEVELOPMENT AND DEPLOYMENT OF EVOLVING APPLICATIONS

Elaborating on the Model Driven Architecture Towards a Change-resistant Development Framework

Georgios Voulalas, Georgios Evangelidis
Department of Applied Informatics, University of Macedonia, 156 Egnatia St., Thessaloniki, Greece

Keywords: Model-driven Development, Meta-Models, Evolving Business Applications, Application Generators, Application Deployment Platforms, Reflectional Programming.

Abstract: Software development is an R&D intensive activity, dominated by human creativity and diseconomies of scale. Current efforts focus on design patterns, reusable components and forward-engineering mechanisms as the right next stage in cutting the Gordian knot of software. 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, based on an integrated database schema. The proposed framework can be seen as an extension of the Model Driven Architecture targeting a specific family of applications.

1 INTRODUCTION

Software development is an area in which we are struggling with a number of major problems. The most important problems are (Kleppe & Warmer & Bast, 2003):

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 the first three phases. The connection between those artefacts and the code fades away as implementation progresses. Changes widen the gap, since they are usually done at the code level only, due to time restrictions. The idea of Extreme Programming (XP) has rapidly become popular, since it is built upon the fact that the code is the driving force of software development and thus the phases that should accumulate the major effort are coding and testing. However, having just code and tests makes maintenance of a software system very difficult. Practically speaking, analysis and design artefacts 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, they should elevate technologists above the lower level complexities that are imposed by the available (with continuously increased complexity) technologies, and they need to be eligible as input in forward-engineering operations.

The Portability Problem. The software industry has a special characteristic that makes it stand apart from most other industries. Each year, and sometimes even faster, new technologies are being invented and becoming popular (e.g., Java, CORBA, UML, XML, J2EE, .NET, and Web Services). The new technologies offer concrete benefits for companies and many of them cannot afford to lag behind. As a consequence, the investments in previous technologies lose value, and existing

Voulalas G. and Evangelidis G. (2006).
A FRAMEWORK FOR THE DEVELOPMENT AND DEPLOYMENT OF EVOLVING APPLICATIONS - Elaborating on the Model Driven Architecture Towards a Change-resistant Development Framework.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 22-29

systems have to be ported to the new technology in order for interoperability (with systems built with the new technology) restrictions to be completely wiped out.

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. In (Roddick & Al-Jadir & Bertossi et al., 2000) the following factors that drive information system evolution are listed:

"*A change in the universe of discourse*": The application world is continually evolving. A viable application system should accommodate these changes.

"*A change to the interpretation of facts about the universe of discourse and the manner in which the task is realized in a system*": People are not able to precisely express the desired functionality of a large-scale application system. Only experience from using the system will enable them to properly formulate the needs and requirements.

"*Changes in the form of updates to effect upgrades to the functionality or scope of a system*": People do not know in advance all the desired functionality of a large-scale application system. Only experience from using the system will enable them to realize and express all needs and requirements.

"*Changes in the form of updates to effect efficiency improvements*". For example, the restructuring of database elements in order for faster information retrieval to be achieved.

In order for evolution to be handled efficiently the following objectives should be met:

- changes should be seamlessly incorporated without the need of restructuring the existing application,
- analysis and design artefacts should be updated in order for changes to be reflected,
- the operation of the deployed application should not be interrupted, or at least interruption should minimized, and
- access to old business objects within their right context should be supported, i.e., at any time an old business object should be able to be easily retrieved and examined through the specific version of the application that produced and manipulated it, in order for user to be able to trace back to former business data.

This paper introduces a new framework for the development and deployment of web-based business applications. In Section 2, we present the Model Driven Architecture (MDA) and the modern practices brought out by Microsoft. In Section 3, we discuss the areas of the MDA that will take advantage of the proposed framework. In section 4, the framework is introduced. The last section provides a conclusive summary of the paper and identifies our future research plan.

2 MDA & MICROSOFT SOFTWARE FACTORIES

MDA (Kleppe & Warmer & Bast, 2003; Miller & Mukerji, 2001; Miller & Mukerji, 2003) 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 artefacts that are produced during the development process. The artefacts are 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 modelled from the aspect of how it best supports the business requirements.

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 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.

For many specifications, PIM and PSMs are defined in UML, making OMG's standard modelling language a foundation of the MDA.

In contrast to traditional development, MDA transformations are always executed by tools. Many tools are able to transform a PSM into code; there is nothing new to that. What's 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. The focus for a developer shifts to the development of a PIM. The PSMs and the code are generated automatically. The PIM fulfils the function of high-level documentation, and is not frozen after writing, since changes will eventually be made 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 cannot 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 artefact that depicts precisely the system throughout its lifecycle, since all changes made to the system are eventually made by changing the PIM and regenerating the PSMs and the code.

On the other side, Microsoft has recently introduced Domain Specific Languages (DSLs) with its own modelling environment, Visual Studio 2005 Team System (VSTS). DSLs (Greenfield, 2004) 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. Microsoft has chosen the term “software factory” in order to emphasize upon reusable assets and tooling for supporting them. The software industry welcomed the new approach, however many are still cautious, mainly due to the displacement of the UML and the fact that since software is an R&D and not a production activity, it is difficult to apply manufacturing principles. Undoubtedly, 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 for 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.

The current status of the framework is mainly shaped by the availability of support tools and therefore presents the following deficiencies (Kleppe & Warmer & Bast, 2003):

- 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 requires 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’s a matter of time before software vendors overcome the above-mentioned limitations. However, there exist a number of areas that can be improved. More specifically, MDA fails 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 modelled in the PSMs, one cannot guarantee that developers will not interfere manually with the generated code. Consequently, the consistency between the three cornerstone models is unstable.
- **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. What’s more, 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.

Those limitations are inherent to the MDA's comprehensiveness, since it is very difficult to elaborate on a more sophisticated solution while in parallel coping with all types of applications.

4 THE PROPOSED FRAMEWORK

Motivated by the above-mentioned findings related to the MDA paradigm, its core principles, and the latest practices adopted by Microsoft and Borland, we introduce an innovative extension for the realization of a development and deployment framework targeted to web-based business applications. The proposed framework (depicted in figure 1) will be structured on the basis of a universal database schema (meta-model). Development will be supported by components (modelling tools) 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 behaviour 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 (taking as basis a 3-tier approach that is the most outstanding architectural paradigm). Response to business changes is instant, simply through the manipulation of data tuples.

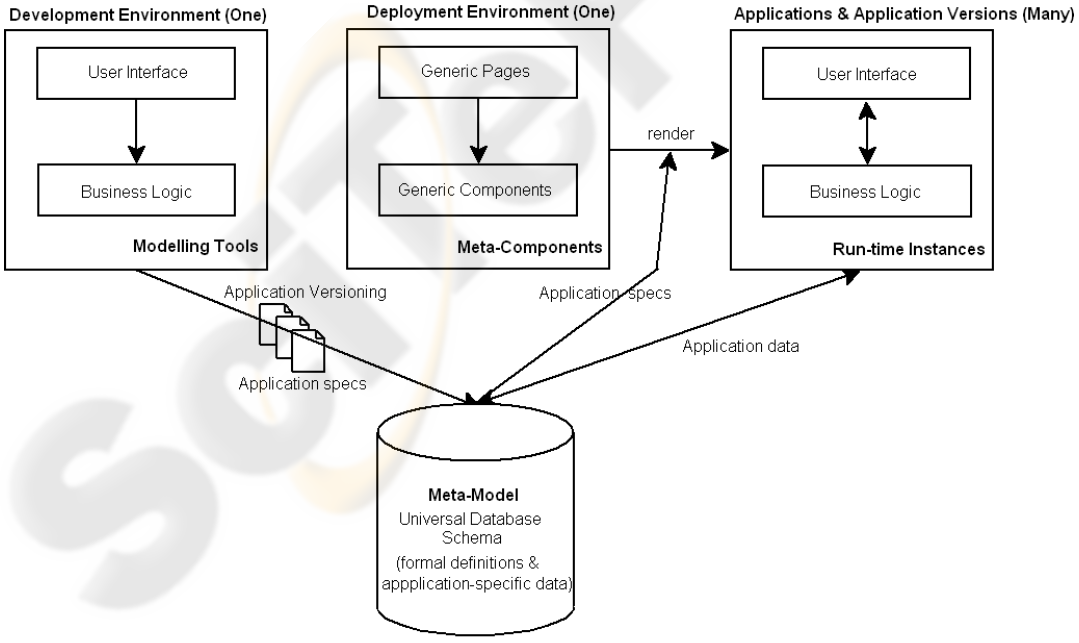

Figure 1: Structure of the Proposed Development and Deployment Framework.

More specifically, the proposed framework includes the models that are described below.

4.1 Domain Model

The Domain Model is a business-oriented model that maps to the MDA Platform Independent Model. It defines the structure of the data that the application is working on (objects, attributes, and associations), along with their behavioural 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 (OMG, 2003; Coronato & Cinquegrani & Giuseppe, 2002) for the description of constraints that govern the modelled objects, plus elements from an acceptable business rules classification scheme (Business Rules Forum 2004 Practitioners' Panel, 2005; Butleris & Kapocius, 2002; Herbst, 2002), with the Ross method (Business Rules Forum 2004 Practitioners' Panel, 2005) being the prevalent. Therefore, its main entities are:

- Business activities. They carry the information that is necessary for the execution of a process. *Example: Travel Application, Accommodation Proposal, Air Ticket, and Traveller.*
- Status: Each business object passes through different statuses during its lifecycle. *Example: Un-submitted, Submitted, and Rejected (for the travel application).*
- Attributes: Define the static aspect (information) of a business object. *Example: Cost (numeric), Notes (alphanumeric), and Check-out date (for the Accommodation Proposal).*
- Methods: Define the dynamic aspect (behaviour) of a business object. *Example: Submit, Approve, and Reject (for the Travel Application).*
- Association: Represents structural relationship between business objects that exist for some duration (in contrast with transient links that, for example, exist only for the duration of an operation). *Example: A Travel Application is associated with one or more Accommodation Proposals.*
- Argument: A parameter required for the execution of method. *Example: Submission notes and priority are arguments of the 'submit' method.*
- Term: A noun or noun phrase with an agreed upon definition. A term is essentially an object or attribute that is included in a business rule. *Example: Air Ticket, fare.*
- Fact: A complete statement connecting terms (via verbs or prepositions) into sensible, business-relevant observations. A fact is essentially a business-significant association. *Example: A Travel Application is associated with at least one Traveller.*
- Computation Rule: Provides an algorithm for arriving at the value of a term. A computation rule is essentially a business-significant method. *Example: The total cost of a Travel Application is computed as the air tickets fare plus the accommodation cost.*
- Pre-condition: A condition that must hold before executing an operation. It typically evaluates one or more attributes. *Example: The 'submit' method can only be executed upon those travel applications that are un-submitted.*
- Post-condition: Defines either the return value of a method or modifications on the value of component attributes that must be performed. *Example: The status of a Travel Application changes to 'submitted' after the execution of the 'submit' method.*
- Guard: Force the execution of operations anytime triggers (i.e. all attributes involved in the guard condition) get a specific state. *Example: Each time an Accommodation Proposal gets approved by the travellers (i.e., its status changes to 'approved') the status of the associated Travel Application is updated.*
- Invariant Constraint: A condition that must always hold as long as the system operates. It typically constraints the value of an attribute. *Example: The value of the attribute 'numberOfPassengers' should always be greater than zero.*

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

- Business Role: In each process, one or more business roles are identified. Example: Corporation, Travel Agency.
- Enterprise: The organization that participates in the process by undertaking a specific business role. In the case of business applications limited to the enterprise scope, only one organization exists. In the case of business networks or e-

marketplaces multiple organizations exist. Example: Corporation X, Travel Agency Y.

- Business Units: Departments, branches or affiliated companies of an enterprise. Example: The accounting department of corporation X
- Partnership: Cooperation relationships established between enterprises (applies only to business networks and e-marketplaces). Example: The Partnership that has been established between corporation X and travel agency Y within the CTP (supposing that an e-marketplace that enables the cooperation of travel agencies with corporate customers exists).
- Partner: An enterprise that participates in a partnership by playing an undertaking business role. Example: The travel agency Y in the previous partnership.
- Employee: A person employed by an enterprise. Employees usually belong to business units. Example: Mr. X.
- Role: Represents the responsible actor for the fulfilment of a set of activities (methods implemented by business objects). An activity can be optionally associated with more than one role. Example: Traveller, Travel Arranger, Travel Agent, and Travel Administrator.
- User: An employee that has access to the business application. A user is associated with one or more roles. Example: Mr. X that access the business application as traveller.

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.).

As for modelling language, UML including OCL will be extensively utilized within the Domain Model. However there is need for a specialization of UML for modelling inter- and intra-organizational aspects, which means that a new UML profile focused on the Enterprise Model should be defined.

4.2 Application Model

The Application Model maps to the MDA Platform Specific Model 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 arguments. Pages are interrelated according to the identified object associations.
- **Business Logic Model**: Suppose that we select the Java 2 Standard Edition 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 part of it) that returns part of an alphanumeric will be mapped to the 'substring' method that is implemented by the 'java.lang.String' class. 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 associated, a data model is structured. Only persistent objects 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 behaviour 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 behaviour and is supported by many object-oriented programming languages). For each different technology utilized at Application Level (J2SE, .NET, J2EE), different components should exist. Practically speaking, every programming language that supports reflectional behaviour can be utilized.

- **Data Model Instance**: 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 (Yannakoudakis & Tsionos & Kapetis, 1999) 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.

4.4 Discussion

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 4 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.

What's more, in full compliance with the MDA principles, the framework enhances productivity by incorporating application generation features through the elicitation of high-level, formal definitions that are automatically transformed to low-level technical specifications, and supports portability through the Application Model that can be theoretically supported by any programming language that supports reflection and by any database system.

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 undisputable advantages, i.e., improved productivity, efficient documentation, effective maintenance, production, portability, and interoperability. For this reason, we suggest transferring the functional specifications of the application from the components (code) to the database and utilizing them at run-time in order to configure generic components. 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. These meta-components will be configured at run-time in order to render the application-specific functionality. Dynamic functional specifications will let end-users deal with changes at deployment time without recompiling and redeploying the application. What's more, with simple data versioning techniques that enable the retrieval of previous specifications, the operation of previous versions of an application will be feasible through the same, unique installation. Last but not least, since all changes pass through the Domain Model, the consistency between the three cornerstone models will not be compromised.

It should be clear that our goal is to present an interesting perspective that could somehow extend the MDA framework and not replace it. Besides, one can easily identify a set of drawbacks in comparison with the MDA framework:

- The proposed framework has narrower scope, since it focuses on web-based business applications.
- MDA handles efficiently 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.

The first constraint is enforced by the fact that is practically infeasible to create a generator that can produce any application (Guerrieri, 1994; Wu & Jen-Her & Hsia et al., 2003) and is in compliance with the latest developments as pictured by the initiatives undertaken by major software players. This is the main reason for considering and evaluating this framework as an extension of the MDA that targets on a specific group of applications. The third drawback is minor, since the availability of powerful computational resources encourages the elaboration of sophisticated solutions. Working towards a 'lighter' solution, we will consider adopting partial behavioural reflection (Tanter & Noye & Caromel et al., 2003). We also plan to address the issue of interoperability.

Future research will focus on:

- **Extending the framework** with a coordinate that will cover the need for cross-platform interoperability. This coordinate will be structured on the basis of the Web Services paradigm.
- **Elaborating on a new UML Profile** for the modelling of business entities.
- **Implementing the required infrastructure.** After finalizing the structure of the framework and identifying all main entities, we have to elaborate on the database schema. Performance issues should be seriously taken into account in the selection of the adopted data-modelling paradigm (relational, object-relational, object). The next step will be the specification and implementation of the meta-components along with the components that will support the development process. The derived prototype will verify the viability and efficiency of the proposed solution.

REFERENCES

Business Rules Forum 2004 Practitioners' Panel, 2005. The DOs and DON'Ts of Business Rules. Business Rules Journal, Vol. 6, No. 4, http://www.BRCommunity.com/a2005/b230.html

Butleris, R., Kapocius, K., 2002. The Business Rules Repository for Information Systems Design. ADBIS Research Communications: 64-77

Coronato, A., Cinquegrani, M., Giuseppe, D.P., 2002. Adding Business Rules and Constraints in Component Based Applications. CoopIS/DOA/ODBASE: 948-964

Greenfield, J., 2004. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/softfact3.asp

Guerrieri, E., 1994. Case Study: Digital's Application Generator. IEEE Software, 11(5) 95-96

Herbst, H., 1996. Business Rules in Systems Analysis: a Meta-Model and Repository System. Inf. Syst. 21(2) 147-166

Kleppe, A., Warmer, S., Bast, W., 1996. MDA Explained. The Model Driven Architecture: Practice and Promise (Chapter One). Addison-Wesley.

Miller, J., Mukerji, J., 2001. Model Driven Architecture – A Technical Perspective. http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01

Miller, J., Mukerji, J., 2001. Technical Guide to Model Driven Architecture: The MDA Guide v1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01

OMG, 2003. Object Constraint Language Specification. http://www.omg.org/cgi-bin/doc?ptc/2003-10-14

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., 2000. Evolution and Change in Data Management - Issues and Directions. SIGMOD Record 29(1) 21-25

Tanter, E., Noye, J., Caromel, D., Cointe, P., 2003. Partial behavioral reflection: spatial and temporal selection of reification. OOPSLA 27-46

Wu, Jen-Her, Hsia, Tse-Chih, Chang, I-Chia, Tsai, Sun-Jen, 2003. Application Generator: A Framework and Methodology for IS Construction. 36th Annual Hawaii International Conference on System Sciences (IEEE - HICSS) 263-272

Yannakoudakis, E. J., Tsionos, C. X., Kapetis, C. A., 1999. A new framework for dynamically evolving database environments. Journal of Documentation, 55(2) 144-158