Effort Reducing in Software Modeling on MDA Approach

José Eduardo Belix
Sérgio Martins Fernandes
Selma Shin Shimizu Melnikoff
Edison Spina

The Department of Computing and Digital Systems Engineering (PCS)
University of São Paulo - Polytechnic School
Av Professor Luciano Gualberto, travessa 3, nº158 - Sala C2-42.
Cidade Universitária - São Paulo - SP
CEP: 05508-900
Brazil
(jose.belix; sergio.martins; selma.melnikoff; edison.spina)@poli.usp.br

Abstract: - The MDA (Model Driven Architecture) is an approach for software development that has the ability to produce applications for virtually every middleware platform from the same base model. All the MDA development projects start with the creation of a Platform Independent Model (PIM), which expresses only business functionality and behavior, abstracting away platform-specific details.

This paper focuses on defining recommendations to help reducing the necessary effort to represent the PIM model, by the adoption of predefined solutions for the software to be generated. The paper also presents a taxonomy for these predefined solutions and the consequences of their use, pertaining to modeling effort.

Key-Words: - MDA, MDD, CASE TOOLS, Generative Programming, Software Architecture, Abstractions.

1 Introduction

The OMG's MDA (Model Driven Architecture) proposes automatic code generation from UML models, independently from the implementation platform. To meet this goal, MDA defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform [1]. The MDA process is divided in 3 models: the PIM (Platform Independent Model), a system's model that has no platform considerations; the PSM (Platform Specific Model) that is generated from PIM and considers the platform's technological issues, and the Source Code, generated from PSM.

For the model-driven software development vision to become reality, CASE Tools must support this automation [2].

This paper proposes that the MDA CASE Tools must be elaborated considering the commonalities between the different products [3], which raises two points:

- Its is advantageous to the developer that MDA Case tools provide solutions that could help defining/implementing the commonalities of these applications, reducing the effort in modeling software;
- Considering that these provided solutions establish some parts of the software design, developers must direct their efforts to what really needs design definitions.

1.1 Taking Decisions

Conscientiously or not, a set of decisions is always taken in a software development effort. These decisions are related to high-level software issues, like the choice of an architectural pattern; or with more specific issues, like data persistence.

These decisions represent solutions with different characteristics. Focus on easiness of maintenance by separation of concerns can take to the decision of adoption of the Layers architectural pattern. An architectural pattern, "describes a particular recurring design problem that arises in specific contexts, and presents a well-proven generic scheme for its solution" [4]. The pattern provides solution, but provides no implementation.

Different decisions can also result in solutions with complete or partial implementation. For example, the decision about the presentation of an application can lead to the adoption of the STRUTS framework (Apache Software Foundation), which provides design and code.

1.2 A Taxonomy for Solutions

[5] presents a taxonomy to classify these issues. The taxonomy is about abstractions, which are defined as "composed new solutions from existing ones to solve problems expected to be encountered during system refinement". The taxonomy is:

- *White-box abstraction*
  Provides a description of the solution, but no implementation. It can be applied in many contexts.
  Patterns are examples of white-box abstractions.
- *Black-box abstraction*
  Provides a fully implemented solution that is completely opaque to the user.
  Web services, components or features supplied by libraries or compiled languages are generally black-box abstractions.
- *Gray-box abstraction*
  Represents the intermediate level: provides source code and design for a partial or complete implementation of the solution, which can be completed or modified when needed.
  Frameworks are examples of gray-box abstractions.

1.3 The Impact of the Presented Abstractions on Modeling Effort

The presented taxonomy basically classifies abstractions in a spectrum that ranges from no implementation at all to fully implemented solution.

By providing a completely implemented solution, a black-box abstraction is inflexible and can only be applied to problems that require that exact solution. On the other hand, white-box abstraction provides no implementation but a description of the solution. The user of this latter type of abstraction studies the provided description of the solution and applies this solution when necessary. The user supplies its implementation and can tailor this implementation in any way necessary for the problem at hand. So, the less implementation an abstraction provide, the more generic it is and can therefore be applied in many contexts, supporting variations [5].

Because of its intrinsic differences, this paper states that the different abstractions perform different roles on reducing modeling efforts.

1.3.1 Effort Reducing Caused by White-Box Abstractions

Being only specification, not implementation, white-box abstractions have a great scope of application and are mainly concerned with architecture solutions.

Architecture patterns, for example, support the construction of software with defined properties, providing a scheme for a generic solution to a family of problems, rather than a prefabricated module that can be used "as is" [4].

The modeling effort economy caused by the white-box abstractions would be provided by tool's automatic application of them, resulting in software structure without manual intervention of the user. Of course, user's reasoning is required to determine if the tool's predefined white-box abstractions are suitable for the software to be built.

Being concerned with high-level architecture definitions, the white-box abstractions greatly constraints the possible black and gray-box abstractions.

1.3.2 Effort Reducing Caused by Black-Box Abstractions

As mentioned, black-box abstractions are inflexible and must be used "as is". They allow no further design or behavioral consideration, but on the other hand, they require no modeling effort because all the structure and behavior have been already defined.

Components are example of Black-box abstractions and can be used for business rules solution or technical issues solution, this latter inside a specific architecture context.

- *Business Rules Solutions*
  When modeling software, architects may want to use existing built component, which may address project requirements, e.g., a component to calculate an asset value. In this case it is not necessary to model the component (it already exists), but only to model the software to use this component.
  Some platform problems can occur in this reuse attempt. For example, during a .Net technology software development, there could exist a component that exactly fulfills a particular service, but is written in CORBA. In this case, it is necessary to construct appropriate wrapper code to adapt the component to the actual environment [7]. The CASE tool could provide this wrapper.
  Appropriated interface also could be provided by the tool for web services connections.
- *Technical Issues Solutions*
  Following the idea of provide the maximum number of ready-to-use solutions to the user of a MDA CASE Tool, and also focusing on reducing effort to model software, the tool can provide abstractions to solve technical issues pertaining to specific development strategies. For example, DAO-based classes normally

require a component to provide DB connections and SQL Statement execution. Such component could be supplied and automatically applied to the software by the MDA CASE tool.

In the same way, a CASE Tool can also apply features provided by programming languages. For example, Disconnected Recordset (ADO in .Net projects or CachedRowSet in Java projects) is a feature supplied by programming languages that enable the access of persistent data. With no need of extra implementation, it also can be used as a medium to transport data between layers or to help preventing problems caused by uncontrolled multi-access. However, it is only possible to take advantage of these two latter characteristics if the application's architecture allows it. This restriction is caused by the limited scope of black-box abstractions.

The point is that the use of black-box abstraction saves on modeling effort because it is not necessary to model the abstractions – they already exist beforehand. More important, the use of black-box abstractions represents an economy of modeling not only for structural aspects of the software, but also for the behavioral aspects.

The representation of behavioral aspects is the weak point of UML [8].

In the specific case of black-box abstractions to solve business rules, the user must know the abstractions and must know the context in which that abstraction works.

For technical issues black-box abstractions, the scenario changes: the tool must automatically apply these abstractions, based on the architecture context, in a way that the user does not have to know the abstractions and does not have to verify if the architecture choices are suitable. It is axiomatic that the overall decisions taken by the tool must be convenient for all the applied abstractions.

1.3.3 Effort Reducing Caused by Gray-Box Abstractions

Being a partially implemented solution, the gray-box abstractions allow code and design changes.

As mentioned, frameworks are examples of Gray-box abstractions. A framework is a reusable design of all or part of a system. It is the skeleton of an application that can be customized by an application developer. A framework provides reuse of design and reuse of code [6].

Most commercially available frameworks seem to be for technical domains, such as user interfaces or distribution [6].

Similar to the black-box abstractions, this paper states that, based on its own embedded architecture strategy, a CASE tool can automatically apply these technical domains gray-box abstractions. This represents an opportunity of modeling effort reduction, but even more interesting, having a CASE tool automatically applying a framework is a way to avoid the costs of the learning curve of that framework. "Frameworks are more customizable than most components, and have more complex interfaces. Programmers must learn these interfaces before they can use the framework" [6].

In fact, the user does not have to know about which gray-box abstractions are supposed to be applied by the CASE tool to cope with technical issues.

Also similar to black-box abstractions, gray-box abstractions (frameworks) can be used to help the user in business modeling task, but in this case the framework understanding is a pre-requisite for evaluating its applicability [9].

Frameworks are derived from business domain and are the key to successful design and code reuse in object-oriented software development [10].

Business domain gray-box abstractions can be used for PIM's pre-configuration.

2 A Practical Example

A MDA CASE Tool can be configured to generate business applications from a set of design decisions. As an example of solution that does not provide implementation (white-box abstractions), the tool must generate software based on the Layers Pattern [4] in three-tier configuration, using DAO Pattern [12] for data persistence. The separation of concerns in layers must be done in the following manner:

- Uppermost layer: business entities;
- Intermediated layer: DAO classes;
- Lowest layer: access to the persistent data.

DAO classes must implement a CRUD-based interface [13], and the Use-Case Controller pattern [14] [15] must be used to map the requirements specifications to the implementation.

2.1 Applying the White-Box Abstractions

The above design example will be used for the development of online retail sales software. In the Use Case "Customer Maintenance", there is a customer information query, which selects customer by ID and validates the dealer's password (two business objects interacting to allow this query). The conventional representation of this behavior is:

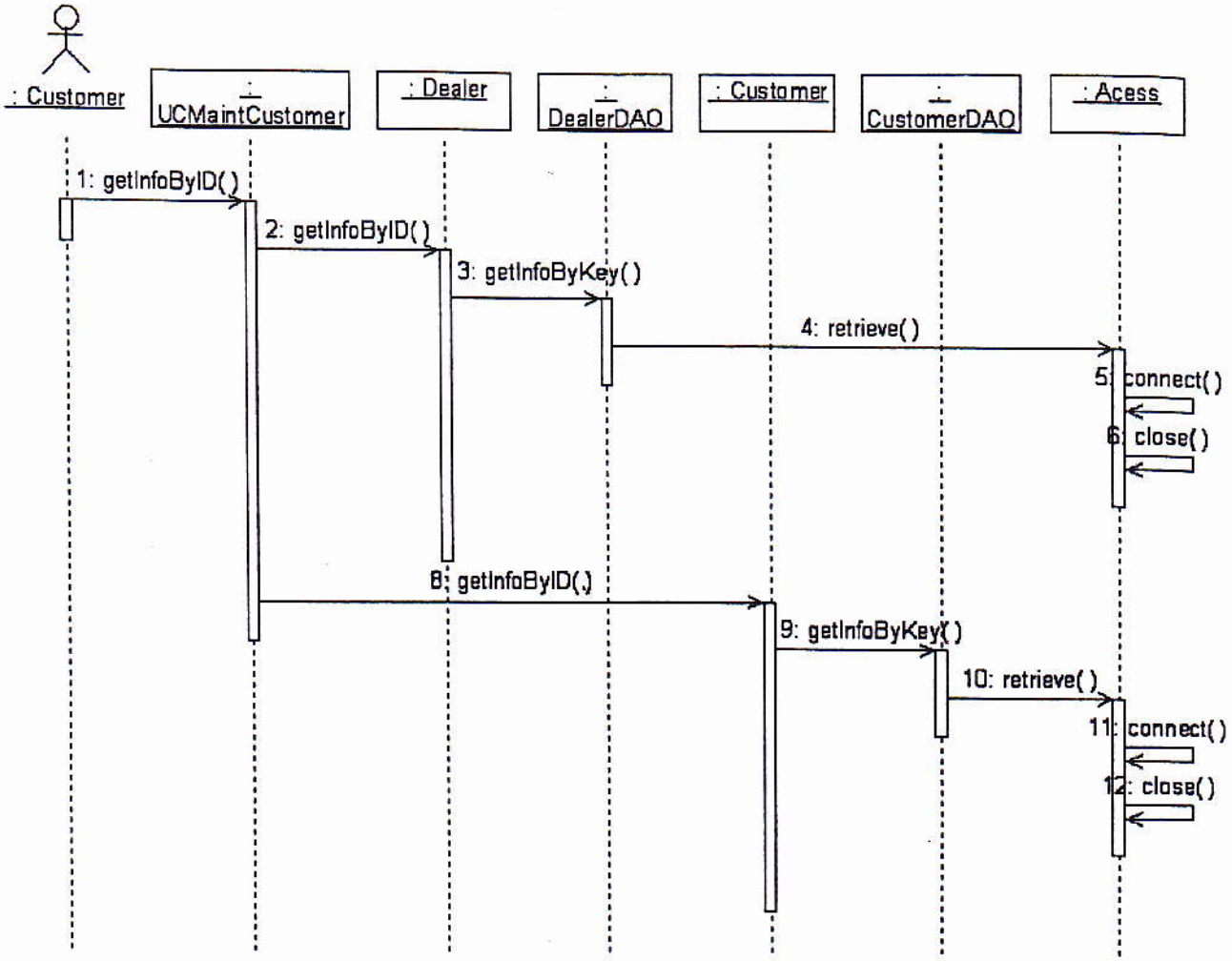

Figure 1 - A Use-Case path

The above sequence diagram (figure 1) represents the necessary messages for this query within the design context. The DAO and ACCESS classes, for example, are consequence of the adopted decisions.

Now, suppose a MDA CASE tool able to automatically apply the mentioned set of white-box abstractions. In this same example, the tool would be prepared to understand that every persistent class would have a corresponding DAO class, responsible for any request related to persistent data. Also defined by abstractions, the DAO classes would inherit a CRUD interface and would use a class (ACCESS class) for SQL statements execution.

Despite the fact that the software architecture remains the same, having a MDA CASE tool that automatically applies this set of white-box abstractions will allow the user to only specify the following PIM behavior for the same sequence diagram:

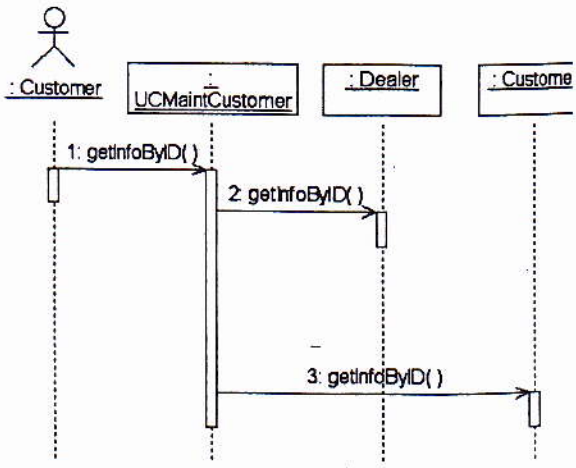

Figure 2 - The necessary PIM representation for sequence diagram

With the behavior presented in figure 2, and applying the defined white-box abstractions, the

MDA CASE Tool would be able to generate a PSM sequence diagram similar to figure 1.

But even such diagram does not contain all the information for a CASE Tool to generate the necessary code. Despite being representative to the humans' cognition, a method called "getInfoByID" represents nothing to a computer. With only the white-box abstractions, it would be necessary to represent the detailed behavior of the method in UML.

Another possible way to solve this problem is by applying solutions that really implement source code, e.g. black and gray-box abstractions.

2.2 Applying the Black and Gray-Box Abstractions

Once the CRUD principles were applied, the source code for persistence method implementation could be partially written and be automatically configured for the CASE Tool to any specific class (the implementations are all very similar), in a gray-box abstraction case.

To configure the variants in this partially written code, parameters could be provided by the CASE tool and XVCL (XML-Based Variant Configuration Language – a general-purpose language for configuring variants in programs and other textual assets) [11] could be used to generate the code.

The ACCESS class (responsible for DB connection and SQL statements execution) would not demand further parameterization and could be automatically implemented (black-box abstraction). With these two latter abstractions, the CASE Tool would be able to completely implement all the persistence methods.

3 Modeling and Constraints

3.1 Reasoning about Modeling

In the example presented in this paper, (figure 2) the developer would only describe the Use Case to the point where he/she could really decide about the application design. It is important to note that – in the approach presented here - many design decisions are left to the tool. There is no reason to model what is already implemented/defined and no further consideration is required. "When deciding how to model something, determining the correct level of abstraction and detail is critical to providing something that will be of benefit to the users of the model. Generally speaking, it is better to model the artifacts of the system - those 'real life' entities that will be constructed and manipulated to produce the final product. Modeling the internals of the web server, or the details of the web browser is not going to help the designer and architects of a web application" [16].

3.2 Constraints are Useful

A MDA CASE Tool customized to work within the design decisions presented in this paper would not work when modeling a digital signal processing system, which uses the Pipes & Filters architecture pattern [17]. On the other hand, the adopted solutions greatly facilitate the business application modeling task, as suggested in this paper.

Aided by the present abstractions, the tool user can develop a PIM strictly focused on business, and the tool would provide the technical solutions, automatically creating a PSM based on best practices.

The tool could provide a default configuration, which could be customized by the user. It would be useful if the tool had its own decision tree - configured beforehand - that could offer consistent choices. There is no point in choosing J2EE with container-managed persistence and Hibernate for the same project, but there is some room for reasoning.

It is recommended that tools be elaborated with different templates, each one customized for different software development conditions. Even in business applications, it would be useful to have templates for .Net with DAO strategy, or J2EE applications, etc...

4 Conclusions

This paper supports the need of MDA CASE Tools to incorporate predefined solutions, which can be classified in a spectrum varying from just specification to fully implemented code. These solutions constraint the scope of possible domain problems where tools can work but, on the other hand, they aim to specialize the tools in a way to reduce the amount of necessary efforts to model software.

References:

[1] OMG. "Model Driven Architecture (MDA). Document Number ormsc/2001-07-01" Object Management Group, july 2001.

[2] Sendall, Shane; Kozaczynski, Wojtek. "Model Transformation: The Heart and Soul of Model-Driven Software Development" IEEE SOFTWARE, Volume20, Issue5, pg 42-45, Set/Oct 2003.

[3] Bosch, Jan. "Product-line architectures in industry: a case study". Proceedings of the 21st international conference on Software engineering. May, 1999.

[4] Buschmann Frank; Meunier Regine; Rohnert Hans; Sommerlad Peter; Stal Michael. "Pattern-Oriented Software Architecture. A System of Patterns". John Wiley & Sons Ltd., Chichester, UK, 1996.

[5] Greenfield Jack; Short, Keith. "Software Factories: Assembling Applications with Models, Frameworks, and Tools". Wiley Publishing, 2004.

[6] Johnson, Ralph E. "Frameworks = (Components + Patterns)". Communications of the ACM, Vol. 40, No. 10. October 1997.

[7] Billig, A; Busse, S; Leicher, A; Süß J. "Platform Independent Model Transformation Based on Triple". Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, 2004.

[8] Kleppe Anneke; Warmer, Jos; Bast, Win. "MDA Explained: The Model Driven Architecture. Practice and Promise". Addison-Wesley – Object Technology Series, 2003.

[9] Bosch, J; Molin, P; Mattsson, M; Bengtsson, P. "Object-Oriented Framework-Based Software Development: Problems and Experience". ACM Computing Surveys (CSUR). March 2000.

[10] Bäumer, D; Gryczan, G; Knoll, R; Lilienthal C; Riehle, Dirk; Züllighoven , H. "Domain-driven framework layering in large systems". ACM Computing Surveys (CSUR). March 2000.

[11] Jarzabek, S; Zhang, H; Swe, S. "XVCL: A Tutorial". Proceedings of the 14th international conference on Software engineering and knowledge engineering. 2002.

[12] Alur, Deepak; Crupi, John; Malks, Dan. "Core J2EE Patterns", Sun Microsystems Press. 2001.

[13] Brandon, Daniel. "CRUD Matrices for Detailed Object Oriented Design". Journal of Computing Sciences in Colleges. December 2002.

[14] Aguiar, Ademar; Souza, Alexandre, Pinto, Alexandre. "Use-Case Controller". EuroPLoP 2001.Sixth European Conference on Pattern Languages of Programs. 2001.

[15] Evans, Gary. "Getting From Use Cases to Code Part 1: Use-Case Analysis". IBM – The Rational Edge. July 2004. Available on <http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/jul04/5383.pdf>.

[16] Conallen J. "Modeling Web Application Architectures with UML". Communications of ACM, October 1999, Vol 42, nº 10, pgs 63-70.

[17] François, Alexandre R.J. "Software Architecture for Computer Vision: Beyond Pipes and Filters". July, 2003. Available on <http://iris.usc.edu/~afrancoi/pdf/sacv-tr.pdf>.

WSEAS TRANSACTIONS on COMPUTERS

Issue 6, Volume 4, June 2005

ISSN 1109-2750 . http://www.wseas.org

ISSN 1109-2750

EDITORIAL BOARD

EDITOR-IN-CHIEF

ASSOCIATE EDITORS

TOPICS: Computer Languages, Software Engineering, Data Structures, File Structures and Design, Data Bases, Compilers, Knowledge and Data Technology, Pattern Analysis and Machine Intelligence, File Structures for on-line Systems, Operating Systems, Parallel and Distributed Systems, Information Systems, Complexity Theory, Computing Theory, Numerical and Semi-Numerical Algorithms, Object-Oriented Programming, Parallel Programming, Computerised Signal Processing, Computer Graphics, Computational Geometry, Machine Vision, Computer Elements, Computer Architecture, Computer Packaging, Fault Tolerance Computing, Mass Storage Systems, Microprocessors and microcomputers, Multiple valued Logic, Numerical Analysis, Finite Element, Genetic Algorithms, Game Theory, Operations Research, Optimization Techniques, Real Time Systems, Virtual Reality, Computer Algebra, Symbolic Computation, Simulation, Pattern Analysis, Machine Intelligence, Adaptive and Learning Systems, Classification, Identification, Chaos Fractals and Bifurcations, Analysis and design tools, Simulation, modelling, Emulation, Visualization, Digital Libraries, Hardware Engineering, Programming Techniques in Communications, Networks, Management and Economic Systems, Multimedia, Video technologies, Simulation Techniques Tools, Software for Communications Development and Simulation, Social Implications of Modern Communications, Soft Computing and Communications, Smart Interfaces, Intelligent Systems, Supercomputers and Supercomputing, Internet and Internet Computing, Intelligent Agents, Mobile Computing, E-commerce, Privacy Problems, Hardware/Software Codesign, Cryptography, Computer/Communications Integration, Education.

HOW TO SUBMIT: http://www.wseas.org, http://www.worldses.org

SUBSCRIPTION: The subscription rate for each journal is 100 Euros (per year) for individuals and 200 Euros (per year) for institutions or companies.

FORMAT OF THE PAPERS: http://www.worldses.org/journals

ISSN: 1109-2750

WSEAS

USA Office	European Office I	European Office II
Prof. George Antoniou, Montclair State University, Computer Science Department, New Jersey, USA. http://www.worldses.org	WSEAS Ag. I. Theologou 17-23, 15773, Zographou, Athens, GREECE. Tel: (+30) 210 7473313 Fax: (+30) 210 7473314 http://www.wseas.org	Prof. Nikos Mastorakis, Military Institutes of Univ. Education, Hellenic Naval Academy, Department of Computer Science, Terma Hatzikyriakou, 18539, Piraeus, GREECE. http://www.wseas.org

WSEAS E-LIBRARY: http://www.wseas.org/data

WSEAS CHAPTERS: http://www.wseas.org/chapters

Each paper of this issue was published after review by 3 independent reviewers