MDA – Model Driven Architecture
Improving Software Development Productivity in Large-Scale Enterprise Applications

Master Thesis

Pedro De Almeida
December 2008

Thesis supervisors:
Prof. Dr. Jacques Pasquier–Rocha & Dr. Patrik Fuhrer
Software Engineering Group

Slah Eddine Driissi
Tecost SA

te cost
technology consulting studies

Software Engineering Group
Department of Informatics
University of Fribourg
Switzerland
“Productivity is never an accident. It is always the result of a commitment to excellence, intelligent planning, and focused effort.”

Paul J. Meyer
American entrepreneur and author
Abstract

Software development productivity has always been one of the most important centers of interest in the Software Engineering domain. After Object Oriented Programming (OOP), Designs Patterns and Model Driven Development (MDD), a new vision proposes to enhance interoperability, portability and productivity of computing systems by means of abstract models. This is the promise of the Model Driven Architecture (MDA) strived by the Object Management Group (OMG).

This thesis focuses on the MDA approach by highlighting the fundamental aspects of this development process. Particularly, it introduces the notion of model and the standards behind the MDA scene. These concepts are first applied in an introductory example, following the AndroMDA methodology to demonstrate how logic implementation can be reduced to the minimum when core business functionality is addressed during the modeling phase. In order to evaluate the benefits of this methodology, lessons learned are then confronted to a concrete MDA based application, namely SIEMS. This Electronic Health Record (EHR) system, which has been developed during 4 years by Tecost SA, a mid-sized software and consulting company established in Fribourg (CH), stands as a perfect example of a MDA integration in a large-scale enterprise solution and confirms that MDA principles are decisive in software development productivity.

Finally, this work exposes a critical point of view about the MDA approach, from development processes to productivity and performance, and suggests some directions that should help the reader on realizing an MDA based application.

Keywords: Model Driven Architecture (MDA), Software Engineering, Software Development, model transformation, code generation, Object Management Group (OMG), AndroMDA, SIEMS
Foreword

The first time I heard about the Model Driven Architecture was in a teaching course in the University of Fribourg. I was obviously fascinated by this new vision of software development but I never thought that few months later, I could be working in a company that precisely developed a MDA based solution. Wandering around the MDA world for 3 years now helped me to improve my understanding in the Software Engineering domain and in that sense, I would recommend this paper to all software developers that prefer to focus on business modeling, rather than on the details of writing code.

Acknowledgments

With these few lines, I would like to acknowledge all the persons that have been involved in the achievement of this project or that supported me during this long year of research and parallel work.

In particular I wish to thank my supervisors, Dr. Jacques PASQUIER–ROCHA for giving me the opportunity to achieve my Master Thesis in the Software Engineering Group and especially Dr. Patrik FUHRER for his availability and skilled precisions that improved the quality of this document. Next to them, I would like to express my gratitude to Slah Eddine DRISI for letting me accomplish this work in his company and for providing me with useful information about the SIEMS project.

From the Tecost SA team, a big thanks to my friends and work colleagues, Stéphane LE PEUTREC, Peter RUCH, Pierrick TERRETTAZ, Christophe SKULTETY and Nicolas JUILLERAT for their great help on trying to answer to my unsolvable questions and for supplying the coffee machine with water on critical periods.

I am also grateful to Michèle COURANT from the Pervasive Artificial Intelligence research group, who introduced me to Tecost SA and without whom this thesis would never exist. And obviously to all those who reviewed this documentation and helped me in their own manner.

Finally, I would like to dedicate these last words to my loving family and especially to my parents for their encouragements and for showing me the way of life. Obrigado.
Notations and Conventions

- **Bold** and *italic* are used for emphasis and to signify the first use of a term.
- **Courier New** is used for web addresses.
- **Code** is used for all the source codes and generally for anything that would be typed literally when programming, including keywords, constants, method names, variables, class names, and interface names.
- **AUTHOR** names are formatted in small capitals.
- The present report is divided in *chapters*. Chapters are broken down into *sections*.
- Where necessary, sections are further broken down into *subsections* and subsections may contain some *paragraphs*.
- **Figures, Tables** and **Listings** are numbered inside a chapter. For example, a reference to *Figure j* of chapter *i* is noted *Figure i.j*.
- **Author references** [Mel04] as well as **web references** [Sof08] (note the *italic* style) are tagged inside square brackets.
- As a respect to both genders, *he* and *she* are used interchangeably in this document.
- Source code is displayed as follows:

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

- Finally, useful hints are highlighted as shown below:

  *Hints provide complementary information in the corresponding section.*

About This Documentation

This documentation is based on a template created by Pedro De Almeida and Patrik Führer (Software Engineering Group, University of Fribourg, Switzerland). Citations and bibliography references are formatted through a XSL stylesheet, tailor-made for the purposes of this thesis.

Both resources are open source and freely available under the *Creative Commons Attribution 3.0 License* [Cre08] from:

- [http://diuf.unifr.ch/softeng/student-projects/templates/soften_en_msword_bib.xsl](http://diuf.unifr.ch/softeng/student-projects/templates/soften_en_msword_bib.xsl)
# Table of Contents

1 Introduction .......................................................................................................... 1  
  1.1 Motivation ......................................................................................................... 2  
  1.2 Research Questions ........................................................................................... 3  
  1.3 Workspace Environment .................................................................................... 3  
    1.3.1 Hardware ....................................................................................................... 3  
    1.3.2 Software ........................................................................................................ 4  

### Part I. Model Driven Architecture  
5  
2 The MDA Approach ................................................................................................. 6  
  2.1 Using Models in Software Development ............................................................. 7  
  2.2 Models and Metamodels ..................................................................................... 8  
    2.2.1 Computation Independent Model (CIM) ......................................................... 9  
    2.2.2 Platform Independent Model (PIM) ............................................................... 10  
    2.2.3 Platform Specific Model (PSM) .................................................................... 11  
    2.2.4 Models Transformation ................................................................................ 12  
  2.3 Modeling Standards ........................................................................................... 13  
    2.3.1 Unified Modeling Language (UML) ............................................................... 13  
    2.3.2 Meta Object Facility (MOF) .......................................................................... 13  
    2.3.3 XML Metadata Interchange (XMI) ............................................................... 14  
  2.4 MDA Benefits ................................................................................................... 14  
    2.4.1 Interoperability ............................................................................................. 14  
    2.4.2 Portability ..................................................................................................... 15  
    2.4.3 Productivity ................................................................................................... 15  

3 HelloWorld goes (Andro-)MDA ........................................................................... 16  
  3.1 About AndroMDA ............................................................................................... 17  
    3.1.1 Generated Application Architecture ............................................................ 17  
    3.1.2 Cartridges ...................................................................................................... 18  
  3.2 HelloWorld Overview ....................................................................................... 19  
  3.3 Preparing Workspace ......................................................................................... 19  
    3.3.1 Application Generation ................................................................................ 19  
    3.3.2 Datasource Configuration ............................................................................ 23  
  3.4 Model Design ..................................................................................................... 24  
    3.4.1 Class Diagram ............................................................................................... 25
# Table of Contents

3.4.2 Activity diagram ........................................................................................................ 26
3.5 Build and Deploy ........................................................................................................... 31
3.6 Logic Implementation ................................................................................................. 34
   3.6.1 Data Access Object (DAO) .................................................................................. 36
   3.6.2 Service ................................................................................................................. 36
   3.6.3 Controller .............................................................................................................. 37
3.7 Application Testing and Development Analysis ....................................................... 39

## Part II. MDA for Large-Scale Enterprise Applications 42

4 SIEMS: the Electronic Health Record System .............................................................. 43
   4.1 The SIEMS Project .................................................................................................. 43
      4.1.1 Simplifying Healthcare Management ............................................................... 44
      4.1.2 Application Integration ..................................................................................... 44
   4.2 End-User Point of View ........................................................................................... 45
      4.2.1 Modules Overview ............................................................................................ 48
      4.2.2 System Accessibility ......................................................................................... 49

5 SIEMS: the Concrete MDA Application ....................................................................... 52
   5.1 Development Context ............................................................................................. 53
      5.1.1 Application Statistics ...................................................................................... 54
      5.1.2 Infrastructure and Deployment ......................................................................... 54
   5.2 Extending a SIEMS Use Case .................................................................................. 55
      5.2.1 Model Adaptation ............................................................................................ 56
      5.2.2 Business Logic Implementation ....................................................................... 59
      5.2.3 Presentation Tier Actualization ........................................................................ 62
      5.2.4 Deployment and Extension Analysis ................................................................ 63

## Part III. Lessons Learned 65

6 Evaluation ...................................................................................................................... 66
   6.1 Development Process .............................................................................................. 67
      6.1.1 MDA Tool Chain ............................................................................................. 68
      6.1.2 Code Generation ............................................................................................ 70
   6.2 Development Productivity ....................................................................................... 71
      6.2.1 Business Modeling ........................................................................................... 72
      6.2.2 Code Patterns .................................................................................................. 72

7 Conclusion .................................................................................................................... 74
   7.1 Durability of Models ............................................................................................... 75
   7.2 Software Systems Development ............................................................................. 75

## Part IV. Appendix 76

A Environment Setup ....................................................................................................... 77
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B Eclipse Classpath Configuration</td>
<td>79</td>
</tr>
<tr>
<td>C SIEMS Source Statistics</td>
<td>81</td>
</tr>
<tr>
<td>Overall Resources</td>
<td>81</td>
</tr>
<tr>
<td>Hand-Coded Resources</td>
<td>82</td>
</tr>
<tr>
<td>D Project Websites</td>
<td>83</td>
</tr>
<tr>
<td>SIEMS Application</td>
<td>83</td>
</tr>
<tr>
<td>SIEMS Portal</td>
<td>83</td>
</tr>
<tr>
<td>Master Project</td>
<td>84</td>
</tr>
<tr>
<td>E CD-ROM</td>
<td>85</td>
</tr>
<tr>
<td>Acronyms</td>
<td>87</td>
</tr>
<tr>
<td>References</td>
<td>89</td>
</tr>
<tr>
<td>Referenced Web Resources</td>
<td>91</td>
</tr>
<tr>
<td>Index</td>
<td>94</td>
</tr>
</tbody>
</table>
List of Figures

Figure 2.1 *The Model Driven Architecture (adapted from [Mda08])* ........................................ 7
Figure 2.2 *The MDA models transformation process* ............................................................ 9
Figure 2.3 *Models, metamodels and platforms relationships* .............................................. 9
Figure 2.4 *The CIM model defines application requirements through use case and activity diagrams* ......................................................................................................................... 10
Figure 2.5 *The PIM model defines data, dependencies and architectural realizations* ........... 11
Figure 2.6 *The PSM model describes platform and language specific elements* ................... 11
Figure 2.7 *Platform independent model (PIM) to platform specific model (PSM) transformation* ............................................................................................................................. 12
Figure 3.1 *Architecture and data propagation in AndroMDA generated applications* ............ 17
Figure 3.2 *Code generation through AndroMDA cartridges* ................................................. 18
Figure 3.3 *Overview of the HelloWorld use case* ............................................................... 19
Figure 3.4 *Console output of application generation with AndroMDA* ................................... 20
Figure 3.5 *Building the HelloWorld application* .................................................................. 22
Figure 3.6 *Generating an Eclipse-compliant project* ............................................................ 22
Figure 3.7 *The HelloWorld application as Eclipse project* .................................................. 22
Figure 3.8 *The HelloWorld class diagram with dependencies and associations* ............... 25
Figure 3.9 *The HelloWorld use case diagram* ................................................................... 27
Figure 3.10 *The HelloWorld activity diagram* .................................................................... 28
Figure 3.11 *The enabled tagged values of the messagesDetails parameter* ....................... 29
Figure 3.12 *Tagged values applied to the birthday and personId parameters* ..................... 29
Figure 3.13 *The HelloWorld model elements overview* ....................................................... 30
Figure 3.14 *Generated files after building* ............................................................................ 31
Figure 3.15 *The web page of the HelloWorld application after modeling* ......................... 34
Figure 3.16 *The HelloWorld web page without sample data* ............................................. 40
Figure 3.17 *Saving persons and messages with the HelloWorld application* ....................... 40
Figure 4.1 *The application login interface of the SIEMS application* .................................. 45
Figure 4.2 *The home interface with news, events and system messages* ......................... 46
Figure 4.3 *The patient’s summary interface with medical-related information* .................... 46
List of Figures

Figure 4.4 The user interface common layout ................................................................. 47
Figure 4.5 Overview of the SIEMS modules ................................................................. 48
Figure 4.6 Exportation of the patient’s administrative folder ................................. 49
Figure 4.7 Dialog box of an automatic system message .............................................. 49
Figure 4.8 Institution configuration and accessibility restrictions ............................ 50
Figure 4.9 The Profiles Management matrix interface with roles grouped in categories .... 50
Figure 4.10 The User Edition interface with group assignment ............................... 51
Figure 4.11 The Employee Edition interface with professional abilities .................... 51
Figure 5.12 The Feedback interface regrouping users remarks from all SIEMS instances ... 53
Figure 5.13 The infrastructure of the SIEMS execution platform ............................... 55
Figure 5.14 The initial EditUserGroup interface .......................................................... 56
Figure 5.15 The initial UserGroups activity diagram .................................................... 56
Figure 5.16 The initial UserGroups class diagram ....................................................... 57
Figure 5.17 Overview of the user class diagram and related value objects ................. 58
Figure 5.18 The UserGroups activity diagram after modeling ..................................... 58
Figure 5.19 The UserGroups class diagram after modeling .......................................... 59
Figure 5.20 The user class diagram with the UserGroupFull value object ................. 59
Figure 5.21 The Eclipse Problems view showing compilation errors ......................... 59
Figure 5.22 The final EditUserGroup interface with the ability to select multiple users at once ........................................................................................................... 64
Figure 6.1 A simplified overview of the MDA development life cycle ......................... 67
Figure 6.2 The MDA tool chain of the SIEMS application ........................................... 69
Figure 6.3 A model validation error caused by a mistyped transition parameter .......... 69
Figure A.1 AndroMDA plugin download using Maven ............................................... 78
Figure B.1 Contents of the Eclipse .classpath file ......................................................... 79
Figure B.2 The Classpath Variables section of the Preferences dialog box under Eclipse .... 80
Figure D.1 The welcome page of the SIEMS application ............................................ 83
Figure D.2 The information portal of the SIEMS project ............................................. 83
Figure D.3 The project page of this Master Thesis ....................................................... 84
Figure E.1 Contents of the CD-ROM ........................................................................... 85
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1.1</td>
<td><em>Hardware used for development and documenting</em></td>
<td>4</td>
</tr>
<tr>
<td>Table 1.2</td>
<td><em>Software used under the scope of the SIEMS application</em></td>
<td>4</td>
</tr>
<tr>
<td>Table 4.3</td>
<td><em>Description of web interface components</em></td>
<td>47</td>
</tr>
<tr>
<td>Table C.1</td>
<td><em>Overall resources statistics from the SIEMS application</em></td>
<td>81</td>
</tr>
<tr>
<td>Table C.2</td>
<td><em>Estimation of hand-coded resources from the SIEMS application</em></td>
<td>82</td>
</tr>
</tbody>
</table>
List of Source Code

Listing 3.1 Contents of the schema-create.sql file ................................................................. 33
Listing 3.2 Contents of the schema-drop.sql file ................................................................. 33
Listing 3.3 Skeleton of the init() method controller ............................................................ 35
Listing 3.4 Example of dummy data for filling results tables ............................................... 35
Listing 3.5 Skeleton of the Message entity to MessageDetailsVO transform ....................... 36
Listing 3.6 Implementation of the entity to value object transform ...................................... 36
Listing 3.7 Implementation of service operations .................................................................. 37
Listing 3.8 Implementation of controller operations ............................................................ 39
Listing 5.1 Implementation of the toUserGroupFull() DAO method .................................. 60
Listing 5.2 Implementation of the handleGetUserGroupAsUserGroupFull() service method .......................................................................................................................... 60
Listing 5.3 Implementation of the handleSaveUserGroup() service method ...................... 61
Listing 5.4 Implementation of the saveUserGroup() controller method ............................ 61
Listing 5.5 Implementation of the loadUserGroup() controller method ............................ 62
Listing 5.6 Initial contents of the edit-the-user-group-save-user-group.jspf file .... 62
Listing 5.7 HTML code for displaying the users checkboxes and labels ............................ 63
1 Introduction

1.1 Motivation ......................................................................................................................... 2
1.2 Research Questions .......................................................................................................... 3
1.3 Workspace Environment .................................................................................................. 3
  1.3.1 Hardware ..................................................................................................................... 3
  1.3.2 Software ..................................................................................................................... 4

“Rien ne se perd, rien ne se crée, tout se transforme”.

Antoine-Laurent de Lavoisier
French intellectual and chemist, 1743-1794

Every decade has brought a new set of programming techniques in order to complete the previous efforts made in software engineering. The increasing power of programming languages assisted by the significant evolution of IT hardware [Dub05] in the last years allowed rich software development sometimes to the expense of quality or interoperability.

Recently, Information Systems have been a turning point in business strategy as their easy-to-use functionalities, performance and reliability allow companies to remain competitive. Software engineering actors have focused their efforts in developing such platform systems while improving quality and reducing implementation difficulties.

Nevertheless, these technological evolutions have negative consequences for the software industry as, in most cases, the architecture of the new execution platform is drastically different from existing ones. Companies have to make a crucial choice between affording the migration costs or bypass the evolution and take the risk to see competitors take the advantages of the opposing choice.

In order to improve software development productivity, the complexity of these technologies has to be handled in a generic way in order to guarantee the quality and
the longevity of the created software [Fra03], independently of the execution platform. This is the goal of the Model Driven Architecture (MDA).

Launched in 2001 by the Object Management Group\(^1\) (OMG), Model Driven Architecture is a relative interesting approach to application design and implementation [Ala04] as it separates the business logic from the complexity of the execution platforms. This separation is done through the efficient use of models in the software development process where enterprise architectures are supported by automated model transformations.

This project analyzes software development productivity through the OMG model driven standards and terminology. First, we give the key principles of the MDA development process (Chapter 2) and then, we highlight these concepts in an example application (Chapter 3). Development results are afterwards evaluated under the context of SIEMS (Chapter 4 and 5), a concrete application of a model driven architecture project. Finally, we expose some critics about the MDA approach and reveal the importance of models and modeling in computer industry (Chapter 6 and 7).

1.1 Motivation

Over years of programming, developers have acquired personal methodologies in developing software applications for specific platforms. Nevertheless, these code-centric methodologies become obsolete when you produce business solutions in a multi-developer environment.

Alan Brown, an IBM distinguished engineer, formulates in this way business functionality implementation [Ala04]: “Developing enterprise-scale applications today requires an approach to software architecture that helps architects evolve their solutions in flexible ways.” This point of view expresses the necessity to consider independently the “[...] system specification from the implementation technology or platform” [Joh01].

This paper precisely covers the integration and interoperability issues through the design- and architecture-centric paradigms that MDA tries to promote. Addressing separately functional, respectively infrastructure requirements are certainly the most expected improvement that the overall software development community waits for.

---

\(^1\) The Object Management Group [Obj08] is a nonprofit consortium focused on modeling and model-based standards since 1989.
1.2 Research Questions

One of the most important questions in software engineering is to know if you can integrate “[…] what you’ve built, with what you’re building, with what you’re going to build.” [Ric00]

Taking into account that enterprise systems constantly require more functionality and that scalability, reliability and performance should not be ignored, this paper tries to address the following questions:

- Based on the OMG well-defined standards how can the MDA approach be integrated to enterprise systems?
- In case of successful integration how does this approach enhance software development productivity and what are the consequences for the software engineer and the business infrastructure?

1.3 Workspace Environment

Realizing an MDA process does not require specific hardware but software is relevant to carry out successfully an MDA-enabled application. Nowadays, a lot of commercial and open source solutions have emerged on the market but only few of them can be considered as serious challengers. Further, some of available solutions are only partial ones and do not handle the complete design development process [Mik05].

In the field of complete solutions, we could cite ArcStyler [Arc08] from Interactive Objects Software GmbH, Blu Age [Blu08] from Netfective Technology SA, OptimalJ [Opt08] from Compuware Corp., or Rational Rose Enterprise [Rat08a] from IBM. On the hand of partial solutions, we could find MagicDraw UML [Mag08] from No Magic Inc., Rational Software Modeler [Rat08b] from IBM and some open source tools like AndroMDA [And08] or OpenMDX [Ope08a]. Two of these partial solutions are particularly discussed on Part II of this work.

1.3.1 Hardware

Two workstations have been used during this project: a desktop computer for software development and a notebook computer for documentation purposes. Furthermore, both computers have contributed to establish benchmarks during development process. Computers components are listed on Table 1.1.
### Workspace Environment

<table>
<thead>
<tr>
<th>Desktop Computer</th>
<th>Notebook Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>Intel Core 2 Duo E6700, 2.6Ghz, 4MB Level 2 Cache</td>
</tr>
<tr>
<td><strong>Hard disk</strong></td>
<td>Western Digital 74Go, 10000RPM, SATA</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>NVIDIA GeForce 7300GT, 512 Mo</td>
</tr>
<tr>
<td><strong>Monitor</strong></td>
<td>Samsung SyncMaster 205BW, 20” LCD</td>
</tr>
<tr>
<td><strong>OS and RAM</strong></td>
<td>Microsoft Windows XP SP3, 3GB</td>
</tr>
</tbody>
</table>

Table 1.1 Hardware used for development and documenting

#### 1.3.2 Software

Software listed on Table 1.2 has been specifically selected to achieve development requirements of this project. As of December 2008, this software was fully compliant with the SIEMS application introduced in Part II of this document.

<table>
<thead>
<tr>
<th>Name</th>
<th>Freely available from</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>Java JDK 1.6.0u10</td>
</tr>
<tr>
<td>IDE</td>
<td>Eclipse SDK 3.4.1</td>
</tr>
<tr>
<td>Project management</td>
<td>Maven 1.0.2</td>
</tr>
<tr>
<td>Code generation</td>
<td>AndroMDA 3.1 RC1</td>
</tr>
<tr>
<td>Servlet/JSP Server</td>
<td>Tomcat 6.0.18</td>
</tr>
<tr>
<td>Database Server</td>
<td>PostgreSQL 8.3.4</td>
</tr>
<tr>
<td>JDBC Driver</td>
<td>PostgreSQL 8.2-504 JDBC 4</td>
</tr>
</tbody>
</table>

Table 1.2 Software used under the scope of the SIEMS application

Note that software with bold italic versions is outdated and may contain unfixed bugs. If you plan to create an MDA application based on the AndroMDA methodology, consider to update software with the last available releases.
Part I.

Model Driven Architecture
2 The MDA Approach

“Now is the time for the Model Driven Architecture.”

Richard Soley in [Ric00]
Chairman and CEO of the Object Management Group (OMG)

It is worth to precise that Model Driven Architecture is not a framework but simply an approach (supported by tools) to application design and implementation. The OMG is confident that the use of models provides an efficient solution for vendor- and language dependencies as models separate “[…] the specification of the operation of a system from the details of the way that system uses the capabilities of its platform” [Mil03].

Chapter 2 provides only a technical overview of the Model Driven Architecture. It first introduces the necessity of models in software development and then explains some terms and technologies behind the world of MDA. At last, Section 2.4 examines the expected advantages that the OMG promised with the MDA approach.
2.1 Using Models in Software Development

Until the late 1990’s, software developers relied on a code-only approach and the use of models was perceived as an expense instead of a productivity gain. Unfortunately, they ignored consciously the facts that “[…] every application ever built must be built to last, to be integrated and to be updated […]” [Mil03].

Today, application implementation faces another problem: the “middleware proliferation” [Ric00]. The twentieth century has seen powerful middleware platforms like CORBA, Java EE, .NET, COM+ emerge from the IT industry but we need to admit that these technologies swift continually over time. Therefore, as companies’ requirements increase in the same way and interoperability with other enterprises becomes a necessity, the OMG managed to resolve this situation using models to derive code for platform-specific architectures.

The MDA approach gives the facility to understand complex and real-world systems while providing an abstraction of the physical system as shown on Figure 2.1. This abstract view of the system is represented through the OMG’s modeling standards (Section 2.3) namely the Unified Modeling Language (UML), the Meta-Object Facility (MOF) and the XML Metadata Interchange (XMI). These three well-established modeling standards ensure an open, vendor-neutral approach to system interoperability and allow transformations to major open or proprietary execution
platforms [Ala04]. Furthermore, **automatic code generation** from models transformation\(^1\) (Section 2.2.4) efficiently increases productivity in the software development process.

As a matter of fact, MDA does not reinvent another software development process but it regroups the knowledge of the past twenty years in software engineering to express enterprises architectures. In fact, the MDA approach is mainly based on the *Model Driven Development (MDD)* [Sta06] which suggests that a **system must be first modeled** before it can be transformed to suit the final execution platform [Gas06]. But MDA goes further in raising the level of abstraction and increasing the **reusability** [Mel04] such that applications become “independent of implementation and can be recombined with other technologies”.

### 2.2 Models and Metamodels

“Models play a major role in MDA.” [Gas06] The OMG has defined the MDA standard with this objective: **describe business solutions through abstract models**.

For every development life cycle phase of an application, MDA suggests to elaborate a corresponding model through a well-defined notation that facilitates system **understandability**. Each model is a **system viewpoint** with regard to the architectural concepts and structuring rules that it tries to abstract [Mil03].

To visually represent the MDA approach, the OMG has established a conceptual framework that relies on “a specific set of layers and transformations” [Ala04] (Figure 2.2). Precisely, the *Computational Independent Model (CIM)*, the *Platform Independent Model (PIM)* and the *Platform Specific Model (PSM)* are the three model types that have been largely adopted by the software community.

---

\(^1\) Producing code from abstract is commonly named *forward engineering*. 
Models are expressed in UML (Section 2.3.1), the actual reference language for modeling. However, analyzing, automating and transforming models require an unambiguous way to describe the semantics of a model. This ability is achieved with metamodels, a model of a modeling language, which defines the structure, semantics and constraints for the subsequent model. For example, the UML metamodel specifies that UML models may contain packages, packages may contain classes, classes their attributes and operations, etc.

While MDA approach highlights platform independence, a metamodel can also describe some properties of a specific platform and, in turn, a specific platform can be described by more than one metamodel. In this context, platform is expressed as “[…] the specification of an execution environment for a set of models” [Mel04]. The relationships between models, metamodels and platforms are illustrated on Figure 2.3.

2.2.1 Computation Independent Model (CIM)

Building large-scale software needs that companies’ demands are clearly defined, especially the services provided by the future application or the integration with third party extensions. Even if this step seems to be far away from software implementation, the OMG recommends defining a computation independent model in order to represent the application in its global environment.
The aim of CIM is to link the needs of the application client with the others models that are built in the next development life cycle phases. In other words, this model could be viewed as a *contractual element* that acts as a reference to check if *client requirements* have been correctly fulfilled.

![Diagram of CIM model and its components](image)

*Figure 2.4 The CIM model defines application requirements through use case and activity diagrams*

From its terminology, a *computation independent* model does not include information about the final application neither about the programming or platform technologies used to implement this latter. As illustrated on Figure 2.4, the CIM model describes the application's business functionality and behavior through use case and activity diagrams and the *actors* that interacts with the application.

While hiding the details of the system’s structure, the CIM plays another role in “[…] bridging the gap between those that are experts about the domain and its requirements […] and those that are experts of the design and construction.” [Mil03]

### 2.2.2 Platform Independent Model (PIM)

The *platform independent model* deals with the constraints of execution platforms. Ideally, application design should be considered as independent from the underlying technology (operating system, programming language, hardware, network performance, etc.) and hence, the PIM should give only a view of the system that is suitable for any type of execution platform.
As proposed in [Mil03], platform independence can be achieved using a “[…] technology-neutral virtual machine”. Services and operations are defined independently of any specific platform but transformation tools (see Section 2.2.4) with knowledge of the target platform ensure that application functionality is successfully performed.

In other words, PIM is a view of a system without any knowledge of the implementation details. Figure 2.1 describes the main purposes of the PIM model: realizing logical data, establishing dependencies and defining workflows processes. Nevertheless, the PIM model requires that model elements contain enough information so that programming APIs for logic implementation can be concretely defined and that code generation is conceivable accordingly.

### 2.2.3 Platform Specific Model (PSM)

Once the platform independent model has been accurately defined, platform specific details can be handled in the platform specific model. The role of this model type is to ease code generation from the PIM through code models that fit the underlying execution platform. Figure 2.6 shows some of PSM code models, like interface code, class or schema models.
It is important to distinguish *source code*, a succession of textual lines, and *code models* that are a structured representation of the source code. Using these code models to transform a PSM into code particularly creates corresponding source files as well as methods and properties skeletons. Note that MDA suggests using UML profiles\(^1\) to create language-specific code models. For example, an UML profile for a Java EE platform allows code models creation that ensures Java EE development after code generation.

### 2.2.4 Models Transformation

Another important aspect of the MDA approach is the transformation between models as suggested by Figure 2.2. Using different model types to “[…] express each aspect of a system at an appropriate level of abstraction” [Mel04] requires that models remain synchronized in some way in order to support **iterative and incremental development**.

![Figure 2.7 Platform independent model (PIM) to platform specific model (PSM) transformation](image)

Models transformation relies on a set of *mapping rules* between models that inform the *transformer tool* about the patterns it needs to follow to output the target model. Figure 2.7 shows the transformation process between a platform independent model (PIM) and a platform specific model (PSM) where the transformer can eventually be supplied with some *transformation knowledge* that brings specialized information about the business domain or the implementation technology. Consistency and validation of models can be automated during this transformation process, improving the efficiency and quality of generated applications.

\(^1\) An UML profile is an extension of basic UML constructs to match a specific purpose.
2.3 Modeling Standards

Models and transformation between them use open-standards that are supported by OMG for their flexibility and expressiveness. The base MDA technologies for modeling are the Unified Modeling Language (UML), the Meta Object Facility (MOF) and the XML Metadata Interchange (XMI).

The MDA core standards aim to formalize the specification of models through metamodels to allow “[…] a common understanding among all involved parts” [Mel04] and simplify the communication about models. These standards can be considered as metamodels for the models that provide the abstraction of the physical system.

2.3.1 Unified Modeling Language (UML)

The Unified Modeling Language is a language “[…] for visualizing, specifying, and documenting software systems” [Mil03]. As the 962 pages of the last UML specifications ([OMG07a] and [OMG07b]) may suggest, UML is a result of the best practices in modeling engineering and has successfully proven his abilities in modeling successfully complex systems [Gas06].

Even if UML is a language independent of any graphical aspect, it remains a powerful language for representing graphically models of applications. These graphical representations of models are commonly named UML diagrams. Use Case diagrams, Class diagrams and Activity diagrams are some views of popular models but Interaction (sequence, collaboration, etc) or Implementation (component, deployment, etc) diagrams can also be taken into account when realizing advanced software systems.

2.3.2 Meta Object Facility (MOF)

The Meta Object Facility originated from an adaptation of the UML core to suit the requirement of the MDA approach. The MOF is also an OMG standard that defines “a common, abstract language for the specification of metamodels” [Joh01]. In other words, MOF could be considered as a metametamodel, i.e. a model of a metamodel.

From its object-oriented nature, MOF defines the essential elements, syntax and structure of metamodels but also the rules defining the life cycle, composition and semantics of MOF-based metamodels. As a matter of fact, UML is precisely a perfect example of a metamodel that is captured in the MOF formalism.
2.3.3 **XML Metadata Interchange (XMI)**

In addition to UML and MOF, the MDA is based upon the *XML Metadata Interchange*, a standard that defines mappings of MDA-based metamodels and models to XML documents and XML schemas. That means that the XMI main purpose is to “[...]
define how XML tags are used to represent serialized MOF-compliant models in XML” [Joh01].

XMI tries to ease the interchange of metadata between UML-based modeling tools and MOF-based metadata repositories in distributed heterogeneous environments. As XMI is based on XML, both metadata (tags) and the instances they describe (element content) can be packaged together in the same document. This enables applications to represent consistently objects and their associations via their metadata.

2.4 **MDA Benefits**

From the beginning of the software engineering era, the market endured a succession of software technologies. Nevertheless, few from these could be considered as a significant revolution, in terms of “fundamental advances in efficiency or effectiveness for any given application” [Fla08]. Regardless of the causes of technology shift, change inexorably occurs and standardization of a particular technology is most of time outdated as the process to define, test and revise a standard lasts more than the lifetime of the technology itself.

Given this statement, the MDA approach actually represents a significant and interesting resource in the software development field in the sense that it hides the “technological and engineering details that are irrelevant to the fundamental functionality of a software component” [Arc01]. Focusing on the *conceptual design* of a system, the OMG expects to bring some important benefits in the long run, particularly through removing the recurrent problem of changing realization technologies and software architectures.

2.4.1 **Interoperability**

Interoperability has always been a major objective in any OMG technology or standard. In the case of MDA, the interoperability is accessible within the system design itself, enabling interoperability definition and verification between applications. In fact, the MDA approach introduces a strict formalization of mappings between metamodels. As applications are built around models that are defined by metamodels, which in turn describe a platform (see Figure 2.3), interoperability between two applications is ensured by these mappings provided by the respective metamodels of models.
More precisely, the base specification of an application, in terms of services and facilities, is a platform independent model. If links to another application have to be specified, then they are part of the model itself. Transforming the PIM to a platform specific model and then into code is done accordingly to these links “which are implemented properly regardless of the target’s native middleware”[Mda08]. In this way, interoperability with other applications is inherent to the design, as long as the quality and characteristics of these links, provided by the metamodels mappings, are respected.

2.4.2 Portability

Separating system architecture from platform architectures is one of the main enhancements of the MDA approach that will lead to a transparent support for new or different technologies. Within the MDA approach, *portability* is achieved by integrating platform descriptions during PIM to PSM transformation.

The platform independent property of the PIMs allows producing multiple PSMs for different platforms, regardless to their implementation. Therefore, everything that is specified “at the PIM level is therefore completely portable” [Kle03].

2.4.3 Productivity

As explained in the last sections, software developers particularly focus on the development of the PIM model when modeling an application. Technical details and specificities of the target platforms are only added during the PIM to PSM transformation and, therefore, developers work independently without taking in account platform technology constraints.

Focusing on the PIM improves software development *productivity* in three ways. First, developers do not need to handle platform specific details as they are addressed in the PIM to PSM transformation definition. Secondly, PSM and code model generation reduce considerably the amount of source code to write. And, finally, developers get more time to implement business functionalities and deliver them more quickly to the end user.

Productivity gain is based on a fully automation of code generation. Nevertheless, transforming models requires defining specialized transformation tools for targeting the platform technology. **This implies that high level models should not only be complete and consistent but also contain enough information so that generation tools are able to produce the platform specific code accurately.**
In order to illustrate the MDA approach introduced in Chapter 2, a HelloWorld example describes step-by-step the tasks needed to create a MDA application based on the AndroMDA methodology. This example also highlights the fundamental aspects that should be observed when modeling and implementing such applications, so that hand-coded logic is reduced to the minimum.

Reproducing this tutorial requires that Environment Setup has been carried out as explained in appendix A. Finally, to ensure continuity with the advanced use case extension under the SIEMS context (see Chapter 5), the corresponding technology and software listed on Section 1.3.1 are used to perform the development process.

"Rules make the learner’s path long, examples make it short and successful."

_Seneca_
Roman rhetorician and writer, mid-1st century AD
3.1 About AndroMDA

From the developers’ web site [And08], AndroMDA is “an extensible generator framework that adheres to the Model Driven Architecture (MDA) paradigm”. In fact, AndroMDA establishes a development methodology to generate deployable components for several platforms and technologies with a definitive goal: write less code.

The HelloWorld example follows the guidelines recommended by AndroMDA to generate a MDA based application for a specific execution platform, namely Java EE. Before starting development, the architecture of generated applications as well as the role of cartridges shipped with AndroMDA is clarified in the next sections.

3.1.1 Generated Application Architecture

The architecture generated by AndroMDA is relatively standard to modern enterprise applications. This latter is composed of four stacked layers, each one containing several components providing some specific functionality. Each component, represented by an AndroMDA cartridge (see Section 3.1.2), can communicate with other components of the same layer or directly with layers below it.

![Figure 3.1 Architecture and data propagation in AndroMDA generated applications](image)

Figure 3.1 shows the layer structure and which types of objects are propagated through layers:
- the **presentation layer** manages interaction between the user and the application by providing the needed component like web pages, rich-client forms, etc. It uses **value objects** (see Section 3.4.1) to retrieve and propagate data from the **business layer**.
- the **business layer** encapsulates the core business functionality by providing service interfaces in order to hide complexity of business logic. It communicates with the **data access layer** through **entities** and value objects.
- the **data access layer** provides a simple **application programming interface** (API) for accessing and manipulating **database records** from **data stores**.

### 3.1.2 Cartridges

The primary resources of AndroMDA are certainly **cartridges**. A cartridge provides “the ability to process model elements that have specified **stereotypes** (i.e. <<Entity>>, <<Enumeration>>, etc.) or model elements that meet certain conditions inferred by the model” [And08], as dependencies or associations. As shown by Figure 3.2, code is generated from validated elements accordingly to **template files** defined within AndroMDA cartridges.

For the HelloWorld example that follows, a set of cartridges that rely on Open Source libraries are used to handle database persistence (**Hibernate** [Hib08]), core business functionality (**Spring** [Spr08]) and front-end processing (**Bpm4Struts** [Str08][Bpm08]). Other cartridges like **Query** or **UML** could also be cited but they are not relevant in the development process of our example.
3.2 HelloWorld Overview

This introductory example consists of a basic messaging system that handles users management and messages sending. For a conventional HelloWorld example, managing users is obviously an unnecessary feature but it will serve to underline dependencies and data retrieval during the modeling phase.

As illustrated by Figure 3.3, the HelloWorld use case consists of one interface that manipulates the entities Person and Message. Each entity is represented by a set of properties, name and birthday for the former and text, sender and date for the latter. Further to the saving functionality, the HelloWorld application should also display a list of sent messages with details in a tabular manner.

3.3 Preparing Workspace

3.3.1 Application Generation

1. The first step consists on generating an MDA based application through the AndroMDA plugin that has been downloaded during Environment Setup. Open a command line window on an empty directory (e.g. c:\MDA, avoid blank spaces in directory path) and type:

   ```
   » maven andromdapp:generate
   ```

2. Maven calls the generate goal on the AndroMDA plugin and starts a wizard for generating the application. This step requires some specific information in order
to download application dependencies and create the project folder (see Figure 3.4).

```
---
\[\text{\ LaTex}\]  \text{intelligent projects} \v 1.0.2
---

... Please enter your first and last name (i.e. Chad Brandon):
Pedro De Almeida

Please enter the name of your J2EE project (i.e. Animal Quiz):
HelloWorld

Please enter the id for your J2EE project (i.e. animalquiz):
helloworld

Please enter a version for your project (i.e. 1.0-SNAPSHOT):
1.0

Please enter the base package name for your J2EE project (i.e. org.andromda.samples):
ch.unifr.diu.fase

Would you like an EAR or standalone WAR (enter 'ear' or 'war')? war

Would you like to use the jBpm workflow engine, it uses Hibernate3 (enter 'yes' or 'no')? no

Please enter the hibernate version number (enter '2' for 2.1.x or '3' for 3.0.x): 3

Would you like to be able to expose your services as web services? (enter 'yes' or 'no'): no

build:start:
andromdapp:init:
andromdapp:generate:
  \[\text{echo}\] + \text{GENERATING AndroMDA J2EE PROJECT} +
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld
  \[\text{copy}\] Copying 1 file to C:\MDA\helloworld

andromdapp:init:
andromdapp:generate-spring-subproject:
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\core

andromdapp:init:
andromdapp:generate-core-subproject:
andromdapp:generate-module:
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\core\src\java
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\core\target\src

andromdapp:init:
andromdapp:generate-common-subproject:
andromdapp:generate-module:
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\common
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\common\src\java
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\common\target\src

andromdapp:init:
andromdapp:generate-mda-subproject:
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\mda\src\uml
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\mda\conf
  \[\text{copy}\] Copying 1 file to C:\MDA\helloworld\mda\conf\mappings

andromdapp:init:
andromdapp:generate-web-subproject:
andromdapp:generate-module:
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\web
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\web\src\java
  \[\text{mkdir}\] Created dir: C:\MDA\helloworld\web\target\src

[\text{echo}] + New J2EE project generated to: 'C:\MDA\helloworld'
[\text{echo}] + For more information on how to proceed from here read 'C:\MDA\helloworld\readme.txt'

BUILD SUCCESSFUL
Total time: 1 minutes 23 seconds
Finished at: Mon Dec 01 11:07:18 CET 2008
```

Figure 3.4 Console output of application generation with AndroMDA
3. Once the application has been generated, it is worth to examine the folder structure and the files that have been created under the helloworld directory. Particularly, a blank UML model, HelloWorldModel.xmi, has been generated under the mda/src/uml folder and will be intensely manipulated in Section 3.4. Furthermore, the file readme.txt on the root of project folder contains valuable information about generated files and Maven goals.

4. Actually, the HelloWorld application is divided into four sub-projects, each represented by its own folder:
   - **common**: this subproject collects resources and classes that are shared among other subprojects. These include value objects, enumerations and other utility classes.
   - **core**: the core sub-project collects resources and classes that use the Spring framework, optionally making use of Hibernate and/or EJBs. These include entity classes, data access objects, hibernate mapping files and services.
   - **mda**: the mda folder contains the files needed to generate and assemble the application. Particularly, it holds the HelloWorld UML model (HelloWorldModel.xmi), which can be found under the src/uml directory.
   - **web**: the web sub-project handles resources and classes that make up the presentation layer. These rely mainly on the Struts framework and controller classes to ensure data validation.

5. Now that project resources are available, it is time to compile the application by simply running the maven command, as it calls the default goal install specified in the maven.xml configuration file. Note that all subprojects have a configuration file, which means that each sub-project can be compiled independently:

   ```
   » maven [common|core|mda|web]
   To view all available project goals, type the following command: maven -u
   ```

6. This step first loads and validates the HelloWorld model, build schemas for creating and dropping database, compiles each subproject and, finally, builds the war archive for deployment. Figure 3.5 shows some selected entries from the console output.
**Figure 3.5 Building the HelloWorld application**

```plaintext
[echo] Building WAR helloworld
[jar] Building jar: C:\MDA\helloworld\web\target\helloworld.war
...
INFO [App] BUILD SUCCESSFUL
INFO [App] Total time: 2 minutes 52 seconds
```

7. At this stage, the HelloWorld project could be deployed using the *Tomcat Manager* for example. Obviously, as nothing has been modeled yet, accessing the application through a web browser will display a blank page.

8. As some logic will be implemented later, we now ask Maven to turn our newly created application into an Eclipse compliant project, so that it can be imported into the IDE (see Figure 3.6):

```plaintext
» maven eclipse
```

```
build:start:
eclipse:generate-project:
  [echo] Creating C:\MDA\helloworld/.project ...
eclipse:generate-classpath:
  [echo] Creating C:\MDA\helloworld/.classpath ...
Plugin 'cactus-maven' in project 'The HelloWorld' is not available
  [echo] Setting default output directory to target/classes
eclipse:
  [echo] Now refresh your project in Eclipse (right click on the project and select "Refresh")
BUILD SUCCESSFUL
Total time: 3 seconds
Finished at: Mon Dec 01 11:45:21 CET 2008
```
9. Simply import the HelloWorld project into Eclipse and you should have a fully configured and ready to use project\(^1\), which should look like Figure 3.7.

3.3.2 **Datasource Configuration**

In order to use the relational database *PostgreSQL*, we will provide the necessary information to Maven and Tomcat such that they will be able to link the application with the *datasource* service and the *database schema*.

1. Open the `project.properties` file located in the project root folder and replace contents with the following lines:

```properties
# PostgreSQL JDBC driver:
postgresql.jdbc.version=8.2-504.jdbc4

# ANTLR (Another Tool for Language Recognition) for HSQL queries processing:
antlr.version=2.7.2

# Properties for management of the database schema:
dataSource.driver.jar=${maven.repo.local}/postgresql/jars/postgresql-${postgresql.jdbc.version}.jar
dataSource.driver.class=org.postgresql.Driver
dataSource.url=jdbc:postgresql://localhost:5432/helloworld
dataSource.user=helloworld
dataSource.password=helloworld
dataSource.sql.init=core/target/schema-create.sql
dataSource.sql.drop=core/target/schema-drop.sql
dataSource.sql.load=core/target/db/create-dummy-load.sql

# What schema related goals should do when an error occurs.
dataSource.sql.onError=continue

# Change this to generate to the correct MDA database mappings
sql.mappings=PostgreSQL
hibernate.db.dialect=org.hibernate.dialect.PostgreSQLDialect
```

2. Edit the `project.xml` file located in the same folder and add the following dependencies under the `dependencies` element:

```xml
<dependency>
  <groupId>postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>${postgresql.jdbc.version}</version>
</dependency>

<dependency>
  <groupId>antlr</groupId>
  <artifactId>antlr</artifactId>
  <version>${antlr.version}</version>
  <type>jar</type>
  <properties>
    <war.bundle>true</war.bundle>
  </properties>
</dependency>
```

---

\(^1\) Depending on your environment and/or project path, imported project may signal errors due to incorrect dependencies. Replace contents of `.classpath` file on the root directory with those of appendix B and refresh the project on *Eclipse*. 

3. Inform Tomcat about the datasoure by adding a new resource under the GlobalNamingResources element, which is located in the server.xml file under Tomcat configuration directory:

```xml
<Resource
    name="/jdbc/HelloWorld"
    type="javax.sql.DataSource"
    driverClassName="org.postgresql.Driver"
    username="helloworld"
    password="helloworld"
    maxIdle="2"
    maxWait="5000"
    url="jdbc:postgresql://localhost/helloworld"
    maxActive="4"/>
```

4. Start pgAdmin (installed with PostgreSQL) and create a new login role:

<table>
<thead>
<tr>
<th>Role name:</th>
<th>helloworld</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password:</td>
<td>helloworld</td>
</tr>
<tr>
<td>Role privileges:</td>
<td>Can create database objects</td>
</tr>
</tbody>
</table>

5. Finally, create a new database from pgAdmin:

| Name: | helloworld |
| Owner: | helloworld |

### 3.4 Model Design

Now that the HelloWorld workspace has been completely configured, the next step consists on modeling the use case presented on Section 3.1. An empty UML model has been created during the Application Generation phase and will serve as template to start modeling our entities, value objects, services and controllers and the concrete use case. In order to have a better insight of the current modeling progression, a class diagram and an activity diagram are used to dispose elements and visualize their relationships.

Even if Section 2.2 considers that application architecture should be separated between different types of models in order to perform successive transformations, most of available MDA tools do not strictly follow OMG recommendations and combine CIM and the PIM purposes in a unique model. Furthermore, PIM to PSM transformation is often skipped and code generation is performed directly after modeling. This does not mean that the MDA approach is lost but that models are simply represented by different views inside the same model.

Opening the generated model with MagicDraw UML will automatically complain about broken dependencies that can be found in the andromda/xml.zips folder under the Maven repository. In fact, the blank model is already linked to AndroMDA types and profiles, which enables stereotyping of attributes and classes. Stereotypes are the most important aspect to take into account when modeling as they determine the code patterns generated by AndroMDA for every model of element.

Before starting modeling, a package named ch.unifr.diuf.ase is created under the Data folder from the Containment Tree view of MagicDraw UML. This will serve as
reference for our future model elements and for targeting output folder when transforming the model into source code. Note that next sections only focus on special operations to ensure conformity with AndroMDA methodology and do not deal about UML modeling. Refer to MagicDraw UML help and maybe [OMG07a] or [OMG07b] to learn how to create standard UML models.

### 3.4.1 Class Diagram

The *logical view* depicted by Figure 3.8 is composed with five class elements that are disposed inside a *class diagram* (HelloWorldClassDiagram). First, our data model requires defining two *Entity* stereotyped classes, namely *Person* and *Message*, which will contain corresponding attributes as introduced in Section 3.2. Note that attribute types are platform independent types and should not be confused with Java types. For example, the type String of *name* attribute does not correspond to the java.lang.String type from the Java EE framework. Furthermore, entities are assumed to have an *id* attribute and, therefore, it is not necessary to specify an identifier unless it has a different name. Entities *Person* and *Message* are besides linked with an association element, which indicates that a *Message* has exactly one *sender* and that one *Person* can have sent none or multiple *Messages*.

![Figure 3.8 The HelloWorld class diagram with dependencies and associations](image)

**Figure 3.8 The HelloWorld class diagram with dependencies and associations**
In addition to these entities, a MessageDetailsVO class with the ValueObject stereotype ensures data propagation between layers. A value object is “a special packaging of attributes from one or more entities, with the purpose of shielding the entities from other tiers of the application and/or external applications” [Nar06]. Of course, a simpler implementation could use real entities to pass information through layers but it is recommended that business logic remains off limits to the presentation layer. We will see later that AndroMDA supports some kind of translation between entities and value objects and, therefore, a dependency from the Message entity to the MessageDetailsVO value object is drawn to perform this transformation.

Business logic is handled in the HelloWorldService class, which receives the Service stereotype. As service operations manipulate entities, drawing a dependency from the HelloWorldService to both Person and Message entities ensures that the service have an access to the data access objects (DAOs) that allow entity reading from and writing to the database.

Finally, the HelloWorldController class contains the interface operations that can be called from the activity diagram. In order to gain access to business logic, a dependency element links the controller with the previously defined service. Note that the controller class does not require to be stereotyped but needs to be explicitly linked to a use case as explained in the next Section.

3.4.2 Activity diagram

Activity diagrams detail the use case workflow from the server and client perspective. That means that an activity diagram needs to be encapsulated in a UseCase element, generally defined in a requirements model as illustrated in Figure 2.4. As this HelloWorld example only contains one use case and that no actors need to be represented, the use case diagram could be skipped without any consequence to the modeling process. Nevertheless, we could imagine that accessing the HelloWorld use case requires some privileges for viewing and managing resources as represented by the generalization on Figure 3.9.
Taking into account previous considerations, a UseCase element is added to our reference package and receives the FrontEndApplication and FrontEndUseCase. If the latter is explicit enough, the former simply tells AndroMDA about the application’s entry point! The HelloWorldActivityDiagram can now be created inside the UseCase element and, in order to access controller operations, we need to link them together by assigning the controller to the activity diagram¹.

¹ To assign a controller to an activity diagram in MagicDraw UML, right-click on the controller, select State/Activity Diagram and choose Assign.
Figure 3.10 shows server and clients activities starting from the initial state. From a workflow point of view, this activity diagram has been designed to load messages in the first action state and send them to the unique front-end view, in order to be displayed. From there, the user can either save a person or a message. Whatever action he chooses, the load action state will be called back, reloading messages and persons to ensure that front-end view is always synchronized.

Action states within the server swimlane have no specific stereotype but those in the client swimlane require to be stereotyped with FrontEndView. This particularly informs AndroMDA to use the Bpm4Struts cartridge to generate Struts web pages. Server action states can defer to controller operations but they require that ingoing transitions trigger a method signal, which signature must be identical to those of controller operations.

Navigation is performed through transitions that may carry parameters. As we want to style elements inside web pages, transition parameters are tagged in order to define rendering of corresponding parameter in the web page. For example, the messagesDetails parameter in the ingoing transition of the display action state is tagged with andromda.presentation.view.table and andromda.presentation.view.table.columns. The former tagged value indicates that the parameter should be rendered as a table and the latter specifies which bean properties are to be called in order to retrieve values of each column as presented in Figure 3.11.
Outgoing transitions from FrontEndView action states have another important role. In fact, they are used to build submission forms into web pages as they trigger controller methods with specific parameters. Each transition parameter causes the creation of a form field, whose rendering can be customized if parameter has received some tagged value as mentioned before.

For example, the outgoing transition savePerson builds a complete form containing two inputs fields. As the birthday parameter has been tagged with the andromda.presentation.web.view.field.calendar, the corresponding field will be attached to a date selector widget. In the saveMessage transition, the personId parameter is tagged with the andromda.presentation.web.view.field.type value in order to be rendered as a select field and with the andromda.presentation.web.view.field.required Boolean value. This last tagged value
HelloWorld goes (Andro-)MDA

will add a red asterisk to the field label and tell Struts that field value should not be empty.

Figure 3.13 The HelloWorld model elements overview

Figure 3.13 shows all elements of the HelloWorld model from the containment tree view of MagicDraw UML. As modeling is now completed, next phase consists on building the entire application and implementing some logic before the final deployment.
3.5 Build and Deploy

Application building is done by running maven on a command prompt window. During this task, model is validated and files are generated accordingly to model elements. Figure 3.14 gives a partial list of the most important files that are mainly created in two groups of folders. Target folders contain source files that do not need manual editing and that are always overwritten. On the other hand, src folders collect files that should be edited manually and are only be generated once.

![Generated files after building](image)

Generated source code could be regrouped in 13 types of files:

- `<Entity>.hbm.xml` files define the Hibernate mappings which bind entities and database tables.
- `<Entity>.java` files are abstract classes that implement the corresponding entity. They have to be instantiated using their own entity factory.
<Entity>Impl.java files are concrete implementations of the corresponding <Entity> classes. If entity operations have been modeled, they should be implemented here.

<Entity>Dao.java files are interfaces for classes that access data of the corresponding object (DAO).

<Entity>DaoBase.java files implement methods of the *Dao interfaces and delegate business logic to concrete implementations.

<Entity>DaoImpl.java files are concrete extensions of *DaoBase classes. They will contain custom code to extend the <Entity> data access object. Particularly, they define how entities are transformed into value objects.

*VO.java files represent embedded value objects as described in Section 3.4.1.

*Service.java files are interfaces that specify service methods.

*ServiceBase.java files implement interface methods by performing first validation on parameters and then delegating concrete implementation to the “handle”-starting methods of *ServiceImpl classes.

*ServiceImpl.java files are concrete extensions of *ServiceBase classes. Business logic should be implemented here.

*ControllerImpl.java files are intermediate classes to propagate data between presentation and business layer.

*.sql files allow dropping and creating database schema.

*.jsp, *.jspx and *.css files are front-end relative resources and form web pages.

Note that all files of the presentation tier, i.e. inside the web folder, are generated in the target folder. In fact, web resources do not need necessarily to be modified unless you want to customize rendering or provide more advanced Javascript functionalities. In that case, files have to be imported to the src folder before being edited in order to avoid overwriting next time the application is compiled. A source file that is present in the src folder will no more be re-generated in the target folder. Browse the HelloWorld folder located in the CD-ROM resource of Appendix E to view the source code of the HelloWorld application at different development stages.

Database can now be initialized by running the maven create-schema\(^1\) command. This takes as input the schema-create.sql file and creates the necessary tables for the entities defined before. Inversely, database schema can be dropped using the maven drop-schema\(^1\), which executes the schema-drop.sql script. The contents of these two files are reported by Listing 3.1 and Listing 3.2.

\(^1\) The drop sequence instruction inside sql files does not seem to be compliant with latest PostgreSQL JDBC drivers. Removing corresponding line should solve the problem.
At this stage, the HelloWorld application can be deployed into Tomcat using the created war archive. Accessing application URL through your web browser will display the web page shown in Figure 3.15. As you may have noticed, the browser has been automatically redirected to the /HelloWorldUsecase/HelloWorldUsecase.do action. This is because the HelloWorld ucase has been specified as the application entry point with the FrontEndApplication stereotype in Section 3.4.2.

The reader can observe how the tagged parameters have been rendered in the web page and also that some field have been filled with sample data. Especially, the messages table has been completed with extra functionalities, like sorting or data exporting. In fact, this rendering is performed by the Display tag library [Dis08] which has been integrated in the Bpm4Struts cartridge. We will see in next Section that skeletons of method controllers have already been implemented to send dummy data to the client.
3 Hello World goes (Andro-)MDA

3.6 Logic Implementation

Even if the HelloWorld application is operational, currently it does not handle business functionality. Trying to save a person or a message probably results in a validation error handled in the back-end and another error message displayed in the front-end. Looking at source code, we find that methods to implement are documented and contain coding techniques that can be helpful to spare time during development process. For example, the snippets of source code from Listing 3.3 and Listing 3.4 (taken from HelloWorldControllerImpl.java) show how to fill objects intended to be rendered as table results or select fields.

```java
/**
 * @see ch.unifr.dii.ase.HelloWorldController#init(org.apache.struts.action.ActionMapping,
 *      ch.unifr.dii.ase.InitForm, javax.servlet.http.HttpServletRequest,
 *      javax.servlet.http.HttpServletResponse)
 */
public final void init(ActionMapping mapping, InitForm form,
                       HttpServletRequest request, HttpServletResponse response)
    throws Exception {
    // populating the table with a dummy list
    form.setMessagesDetails(messagesDetailsDummyList);
    // this property receives a default value,
    // just to have the application running on dummy data
    form.setPersonId(new Long((long) 443163472));
    form.setPersonIdValueList(new Object[] {
      "personId-1", "personId-2", "personId-3",
      "personId-4", "personId-5"
    });
```

Figure 3.15 The web page of the HelloWorld application after modeling
form.setPersonIdLabelList(form.getPersonIdValueList());

Listing 3.3 Skeleton of the init() method controller

/**
 * This dummy variable is used to populate the "messagesDetails" table.
 * You may delete it when you add your own code in this controller.
 */
private static final java.util.Collection messagesDetailsDummyList =
    java.util.Arrays.asList(
        new MessagesDetailsDummy("date-1", "text-1", "sender-1", "birthday-1"),
        new MessagesDetailsDummy("date-2", "text-2", "sender-2", "birthday-2"),
        new MessagesDetailsDummy("date-3", "text-3", "sender-3", "birthday-3"),
        new MessagesDetailsDummy("date-4", "text-4", "sender-4", "birthday-4"),
        new MessagesDetailsDummy("date-5", "text-5", "sender-5", "birthday-5"));

/**
 * This inner class is used in the dummy implementation in order to get the web application
 * running without any manual programming. You may delete this class when you add your own code
 * in this controller.
 */
public static final class MessagesDetailsDummy implements java.io.Serializable {
    private String date = null;
    private String text = null;
    private String sender = null;
    private String birthday = null;

    public MessagesDetailsDummy(String date, String text, String sender, String birthday) {
        this.date = date;
        this.text = text;
        this.sender = sender;
        this.birthday = birthday;
    }

    public void setDate(String date) {
        this.date = date;
    }

    public String getDate() {
        return this.date;
    }

    public void setText(String text) {
        this.text = text;
    }

    public String getText() {
        return this.text;
    }

    public void setSender(String sender) {
        this.sender = sender;
    }

    public String getSender() {
        return this.sender;
    }

    public void setBirthday(String birthday) {
        this.birthday = birthday;
    }

    public String getBirthday() {
        return this.birthday;
    }
}

Listing 3.4 Example of dummy data for filling results tables

It is now time to implement business logic of DAO, service and controller that have
been modeled in Section 3.4.1. Note that implementation order of different resources
is not imposed and, depending of developers’ conveniences, some would prefer to implement data propagation from the presentation tier to the data access layer. All source files that are mentioned in the next sections can be found easily by referring to Figure 3.14.

3.6.1 Data Access Object (DAO)

As no entity operation has been defined, the only method to implement in the data access layer is the transformation from the Message entity to the MessageDetailsVO value object. This method can be found in the MessageDaoImpl.java source file and looks initially like Listing 3.5.

```
/**
 * @see ch.unifr.diuf.ase.MessageDao#toMessageDetailsVO(
 * ch.unifr.diuf.ase.Message)
 */
public ch.unifr.diuf.ase.MessageDetailsVO toMessageDetailsVO(finally ch.unifr.diuf.ase.Message entity) {
    // put your implementation here
    return null;
}
```

Listing 3.5 Skeleton of the Message entity to MessageDetailsVO transform

This step consists simply in filling a new value object instance that needs to be returned by the toMessageDetailsVO() method. As the method is called with the corresponding entity and that this latter has an access to Person, mapping values is as trivial as presented in Listing 3.6.

```
/**
 * @see ch.unifr.diuf.ase.MessageDao#toMessageDetailsVO(ch.unifr.diuf.ase.Message)
 */
public ch.unifr.diuf.ase.MessageDetailsVO toMessageDetailsVOfinally ch.unifr.diuf.ase.Message entity) {
    final ch.unifr.diuf.ase.Message entity) {
        MessageDetailsVO messageDetails = new MessageDetailsVO();
        messageDetails.setDate(entity.getDate());
        messageDetails.setText(entity.getText());
        messageDetails.setSender(entity.getSender().getName());
        messageDetails.setBirthday(entity.getSender().getBirthday());
        return messageDetails;
    }
```

Listing 3.6 Implementation of the entity to value object transform

3.6.2 Service

Implementing service methods does not require advanced programming knowledge as it consists mainly in delegating core business functionality to the data access layer that is handled by Hibernate. The HelloWorldServiceImpl.java file (see Listing 3.7) contains the four methods that have been specified during the Class Diagram modeling process. Each method has been prefixed with the “handle” keyword such that the base class can first perform validation on parameters, e.g. checking not null values.
package ch.unifr.diuf.ase;

/**
 * @see ch.unifr.diuf.ase.HelloWorldService
 */
public class HelloWorldServiceImpl extends ch.unifr.diuf.ase.HelloWorldServiceBase {

/**
 * @see ch.unifr.diuf.ase.HelloWorldService#savePerson(
 * java.lang.String, java.util.Date)
 */
protected java.lang.Long handleSavePerson(java.lang.String name, java.util.Date birthday) throws java.lang.Exception {
    Person person = Person.Factory.newInstance();
    person.setName(name);
    person.setBirthday(birthday);
    return this.getPersonDao().create(person).getId();
}

/**
 * @see ch.unifr.diuf.ase.HelloWorldService#saveMessage(
 * java.lang.String, java.util.Date, java.lang.Long)
 */
protected java.lang.Long handleSaveMessage(java.lang.String text, java.util.Date date, java.lang.Long personId) throws java.lang.Exception {
    Message message = Message.Factory.newInstance();
    message.setText(text);
    message.setSender(this.getPersonDao().load(personId));
    message.setDate(date);
    return this.getMessageDao().create(message).getId();
}

/**
 * @see ch.unifr.diuf.ase.HelloWorldService#getPersons()
 */
protected java.util.Collection handleGetPersons() throws java.lang.Exception {
    return this.getPersonDao().loadAll();
}

/**
 * @see ch.unifr.diuf.ase.HelloWorldService#getMessage()
 */
protected java.util.Collection handleGetMessages() throws java.lang.Exception {
    return this.getMessageDao().loadAll(MessageDao.TRANSFORM_MESSAGEDETAILSVO);
}
}

Listing 3.7 Implementation of service operations

As the HelloWorldService has dependencies to both entities, a PersonDao and a MessageDao instance have been injected into the service class pattern during code generation. DAO instances create, load, update and even transform entities into value objects by specifying a transforming rule as shown in the handleGetMessages() method of Listing 3.7 (line 43). Note that the handleGetPersons() method on line 36 returns a collection of entities without any transformation. Even though this approach is not recommended, it can be useful to quickly retrieve bean property values without having to define a new value object as we will see in next Section.

3.6.3 Controller

Implementation of controller methods can be considered as the most time-expensive part of the implementation process as it requires validating data before delegating core business functionality to services. Typed transitions parameters as well as
tagged values enable to define explicitly what kind of value we expect and if a parameter is required. These validations are supported by the Spring cartridge and errors will be thrown before accessing controller methods if passed parameters do not suit specified requirements. Nevertheless, some parameters need custom checking and have to be validated manually. For example, before saving one person, it should be controlled that this latter is not born in the future as implemented in the savePerson() method of Listing 3.8 (line 50).

```java
package ch.unifr.diuf.ase;
import java.util.Date;
import org.apache.struts.action.ActionMapping;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**< *
 * @see ch.unifr.diuf.ase.HelloWorldController
 */
public class HelloWorldControllerImpl extends HelloWorldController {

/** *
 * @see ch.unifr.diuf.ase.HelloWorldController#init( *
 * org.apache.struts.action.ActionMapping, *
 * ch.unifr.diuf.ase.InitForm, *
 */
public final void init(ActionMapping mapping, ch.unifr.diuf.ase.InitForm form, HttpServletRequest request, HttpServletResponse response) throws Exception {
 form.setMessagesDetails(this.getHelloWorldService().getMessages());
 // This method takes a collection of objects, as well as the
 // property names to query on these objects in order to find
 // the corresponding values and labels:
 form.setPersonIdBackingList(this.getHelloWorldService().getPersons(), "id", "name");
}

/**< *
 * @see ch.unifr.diuf.ase.HelloWorldController#saveMessage( *
 * org.apache.struts.action.ActionMapping, ch.unifr.diuf.ase.SaveMessageForm, *
 */
public final void saveMessage(ActionMapping mapping, ch.unifr.diuf.ase.SaveMessageForm form, HttpServletRequest request, HttpServletResponse response) throws Exception {
 form.setText(form.getText());
 form.setPersonId(form.getPersonId());
 this.getHelloWorldService().saveMessage(form.getText(), new Date(), form.getPersonId());
 this.saveSuccessMessage(request, "Message has been successfully saved!");
}

/**< *
 * @see ch.unifr.diuf.ase.HelloWorldController#savePerson( *
 * org.apache.struts.action.ActionMapping, ch.unifr.diuf.ase.SavePersonForm, *
 */
public final void savePerson(ActionMapping mapping, ch.unifr.diuf.ase.SavePersonForm form, HttpServletRequest request, HttpServletResponse response) throws Exception {
 Date today = new Date();
 if(form.getBirthdayAsDate().after(today)){
 this.saveWarningMessage(request, "Birthday must not be in the future!");
 }
 else{
 form.setBirthdayAsDate(form.getBirthdayAsDate());
 form.setName(form.getName());
 this.getHelloWorldService().savePerson(form.getName(), form.getBirthdayAsDate());
 this.saveSuccessMessage(request, "Person has been successfully saved!");
 }
```
The Struts cartridge also provides *syntactic sugar* to pass information to the presentation layer. For example, every controller is able to store messages using `saveSuccessMessage()`, `saveWarningMessage()` or `saveErrorMessage()`. Furthermore, as the `personId` parameter has been tagged to be rendered as a select field, a convenient method for defining default values and labels to display as select options is available inside the `init()` operation. Indeed, calling the `setPersonId-BackingList()` method with a collection of objects as well as properties names will retrieve corresponding information.

### 3.7 Application Testing and Development Analysis

The HelloWorld application should now be fully operational but requires another build before running some tests. After deploying application, accessing project’s web page displays now a similar interface without sample data as shown by Figure 3.16. As all dependencies have been downloaded, using the `maven -o` switch tells Maven to work offline without checking dependencies, which will speed up project compilation.

Some basic tests can be run in order to check application behavior, particularly when data has not been correctly validated. For example, submitting any of the forms should display one error message as all fields have been tagged as required. Entering some text and saving a message should also return an error message. Furthermore, saving a person that has her birthday in the future should lead to a warning message. Finally, if data has been correctly validated, it will be confirmed by a success message.

Remember that you have to register at least one person in order to be able to submit one message. Once a person has been saved, the select field from the save message form is updated as the `savePerson` state in the activity diagram automatically forwards to the initial state (see Section 3.4.2). Identically, every time a message is sent, a new record is inserted in the results table. Figure 2.1 shows the application state after saving some persons and sending some messages.
3 HelloWorld goes (Andro-)MDA Application Testing and Development Analysis

Figure 3.16 The HelloWorld web page without sample data

Figure 3.17 Saving persons and messages with the HelloWorld application
This concludes our HelloWorld example of creating a MDA based application using the AndroMDA methodology. Developing this basic application from scratch takes about 2 hours for an experienced developer, who masters the usage of the AndroMDA plugin and knows how to use the MagicDraw UML interface to build models, as it remains the most time-expensive part of this example.

For a deeper analysis, business logic implementation has been achieved in only 32 lines of code! Database persistence handling, automatic parameters validation and web pages generation are some aspects that increase seriously software development productivity. Nevertheless, this assumption should be verified in a more complex application, as the HelloWorld example provides only limited functionality. Furthermore, the presentation tier remains “as-is”, which is rarely the case as the recent Web 2.0 applications may suggest.

Therefore, the results returned by the HelloWorld example will be exposed to a concrete MDA application. Chapter 5 evaluates how real applications can take advantage of a MDA development process and, on the other hand, what are the weaknesses of this approach under large-scale applications, principally those under production like the SIEMS project.
Part II.

MDA for Large-Scale Enterprise Applications
4

SIEMS: the Electronic Health Record System

4.1 The SIEMS Project

The SIEMS project has been initiated in June 2004 by Tecost SA and several other professional actors from the health domain in the French part of Switzerland. In a common initiative to provide electronic management of healthcare processes, particularly those involving seniors, and enhance communication between health organizations, this project lead to the creation of a non-profit association and a web-based application. The former, known as the SIEMS Association [Sie08b] and regrouping actually 19 medical and health related institutions in canton of Fribourg (CH), served as project guideline and established business functionality and specifications of the latter.


“Health is not valued till sickness comes.”

Dr. Thomas Fuller
British physician, 1654-1734

4.2 End-User Point of View

4.2.1 Modules Overview

4.2.2 System Accessibility

---

1 SIEMS [Sie08a] is a French acronym for “Système d’Information pour les Etablissements Médicalisés ou de Soins”, which could be literally translated as “information system for medical or health institutions”.

2 Tecost SA [Tec08] is a company specialized on technology, consulting and studies, particularly in the eHealth domain.
Induced by the economical and political context of the health sector, the SIEMS application has been proposed as an Electronic Health Record (EHR) system for medical and social institutions. The main idea was to provide an efficient platform in order to coordinate the storage and retrieval of patients’ records while preventing human error risks and reducing health costs. Furthermore, SIEMS aimed to promote care standardization in Switzerland by integrating some international healthcare standards, particularly nursing classifications like ICD10\(^1\), NIC\(^2\) or NOC\(^3\) and nursing theories like the Virginia Henderson [VHL08] or the Roper, Logan & Tierney [Rop00] models of nursing.

### 4.1.1 Simplifying Healthcare Management

As stated in [Hew05], “healthcare industry has been very slow to adopt information technologies”. This observation can be confirmed by the lack of information accessibility and availability that patients face every time they need to provide a lifetime summary of their health state.

In fact, an internal study over 40 medical or health related institutions revealed that only 20 percent of these had a fully integrated EHR system and most of others were still working with paper records. Furthermore, if some institution could provide an individual Electronic Medical Record (EMR) of some patient, it is probably inconsistent or without any compatibility for third-party application integration.

SIEMS was the response to the chaotic infrastructure in the healthcare business environment. Addressing complexity of health processes and health event communication, the system provides real-time access to patient’s information and offers the ability to manage care treatments, either from a medical or a human resources perspective. As a consequence, medical services monitoring enables to target specialized treatments for individual patients, decreasing considerably the risk of medical errors and contributing to reduce health costs.

### 4.1.2 Application Integration

The main aspect highlighted by the SIEMS Association was to assess integration of the future application with the existing institution’s infrastructure. Some institutions already relied on well-established proprietary solutions for accounting, invoicing or human resources management and planning. Nevertheless, as mentioned in Section 4.1.1, paper was the principal media for health information interchange and

\(^1\) International Statistical Classification of Diseases and Related Health Problems [ICD08]
\(^2\) Nursing Interventions Classifications [Cen08a]
\(^3\) Nursing Outcome Classifications [Cen08b]
further researches established that several institutions did not have a computer network or even an Internet connection.

This lack of computerization needed to be seriously considered as integrating the SIEMS application on computer-independent institutions could lead to a complete reorganization of business processes and unaffordable initial investments. From this statement, the technical committee focused on merging institutions’ requirements in order to provide suitable specifications that ensure a successful transition. Furthermore, operational interfaces to third-party software had to be implemented to avoid redundant information.

4.2 End-User Point of View

SIEMS is a web-based application shared by a medical or health related institution, which means that the application is accessible from a simple computer terminal. As explained in Section 5.1, access to the SIEMS interface has been restricted to a specific web browser, currently Firefox, and available languages are limited to French and German. Consequently, screenshots of front-end parts are taken from the French version but they should be explicit enough to be understood.

![The application login interface of the SIEMS application](Image)

**Figure 4.1** The application login interface of the SIEMS application
Figure 4.2 The home interface with news, events and system messages

Figure 4.3 The patient’s summary interface with medical-related information

Figure 4.1 to Figure 4.3 introduce the look and feel as three informative interfaces of the SIEMS application. The login interface provides user authentication based on a pre-registered account and an eventual IP address verification (see Section 4.2.2). The home page lists the active news or events and offers the ability to process system
alerts. Finally, the patient’s overview interface provides a complete summary of his current health status.

Except special pages like the login or the help system, all interfaces are structured with a common layout in order to enable fast navigation and information accessibility. Figure 4.4 shows how web pages are organized, following the common pattern of header, body and footer. Each document part may contain one or more components, serving different purposes as detailed in Table 4.3.

<table>
<thead>
<tr>
<th>Component</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
<td>Patient’s Card</td>
</tr>
<tr>
<td></td>
<td>Application Menu</td>
</tr>
<tr>
<td></td>
<td>User’s Card</td>
</tr>
<tr>
<td><strong>Body</strong></td>
<td>Plugins</td>
</tr>
<tr>
<td></td>
<td>Data</td>
</tr>
<tr>
<td><strong>Footer</strong></td>
<td>Status Bar</td>
</tr>
<tr>
<td></td>
<td>Notifications</td>
</tr>
<tr>
<td></td>
<td>Alerts</td>
</tr>
</tbody>
</table>

Table 4.3 Description of web interface components
4.2.1 Modules Overview

Within the SIEMS application, the term “module” is used to describe a set of correlated interfaces and their corresponding functionalities in order to achieve a specific process. As mentioned in Section 4.1 and without entering into technical or medical explanations, modules are mainly health oriented and provide the requested tools to manage the life cycle of a patient inside the institution.

![Diagram of SIEMS modules](Image)

Figure 4.5 Overview of the SIEMS modules

Figure 4.5 shows the SIEMS modules floating around the main Care module, which is composed of no more than 34 interfaces as of December 2008. This latter is the most important module of the SIEMS application as it supplies other modules with specific information and serves as data source for generating system alerts about patients necessities. For example, the Administration module needs to access information gathered by the Care module in order to export patient-related documents, like the patient’s administrative folder or a transfer folder in case of hospitalization. In the same way, the system processes care resources in order to alert users of unchecked necessities, like a vital sign that has not been measured for a long time. These interactions are illustrated on Figure 4.6 and Figure 4.7.

Others modules like Technical Service, Animation, Evaluation, Statistics, Stewardship and Feeding operate in the same manner to specialize information that is displayed, particularly when they are accessed under the patients context. For further information about modules functionalities, visit the regularly updated web portal [Sie08b] of the SIEMS application.
Amongst all business concerns that are handled by the SIEMS application, a particular effort has been made to address *security* and *privacy* issues in order to restrict access to confidential data. The sensible and personal nature of information stored in EHR systems is subject to legal considerations and therefore, actors that interact with the application need to be authenticated and provided with the necessary abilities to access an interface.
SIEMS can be configured to define the *accessibility* scope of the application. This consists on specifying if users are allowed to login from outside of the institution network and if so, which IP addresses can connect to the system. These parameters can be set in the institution configuration interface as illustrated by Figure 4.8.

Furthermore, SIEMS relies on a complex system of *groups*, *profiles*, *roles*, and *professional abilities* in order to grant access on interfaces. Furthermore, these credentials allow employees to create and/or validate *medical acts* as well as other *administrative* or *technical tasks*. All these attributes can be retrieved on the *Profiles Management*, the *User Edition* and the *Employee Edition* interfaces captured respectively in Figure 4.9, Figure 4.10 and Figure 4.11.
SIEMS: the Electronic Health Record System

End-User Point of View

Figure 4.10 The User Edition interface with group assignment

Figure 4.11 The Employee Edition interface with professional abilities
5 SIEMS: the Concrete MDA Application

5.1 Development Context ........................................................................................................................................53
  5.1.1 Application Statistics ........................................................................................................................................54
  5.1.2 Infrastructure and Deployment ........................................................................................................................54

5.2 Extending a SIEMS Use Case ................................................................................................................................55
  5.2.1 Model Adaptation ................................................................................................................................................56
  5.2.2 Business Logic Implementation ..........................................................................................................................59
  5.2.3 Presentation Tier Actualization ..........................................................................................................................62
  5.2.4 Deployment and Extension Analysis ....................................................................................................................63

“As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications.”

David Lorge Parnas
Canadian professor and pioneer of Software Engineering

Initially, the SIEMS application was not based on the AndroMDA methodology. In fact, the open source OpenLaszlo platform\(^1\) [Las08] has been first suggested as development framework to build a Rich Internet Application (RIA), relying on the interactive and dynamic Flash\(^2\) technology.

Nevertheless, this execution platform was not adapted to the complexity of the medical business model and did not offer enough development flexibility, particularly in a multi-developer environment. In that sense, a different approach based on models has been proposed to address these requirements: generate a MDA-based application from an UML model transformed by AndroMDA. Furthermore, in order to allow multiple developers working simultaneously on the same model, the

\(^1\) OpenLaszlo is an open source platform for the development and delivery of rich Internet applications developed by Laszlo Systems Inc. [Las08].

\(^2\) Flash is a multimedia platform developed and distributed by Adobe Systems Inc. [Ado08].
MagicDraw’s Teamwork Server [Mag08] has been integrated to the development process so that elements can be locked to avoid modeling conflicts.

## 5.1 Development Context

SIEMS development started in January 2005 as a pilot project that involved initially 14 medical and health related institutions. In order to realize the intended final solution requested by the partners of SIEMS Association, the first application snapshot has been deployed into production after few months of development to confront the system with the expected business requirements.

From 2005 to December 2008, an average of 5 developers have been extending the SIEMS application, creating new use cases, improving continuously usability as well as user experience and fixing bugs from users feedback. Indeed, as the production application was constantly upgraded, a dedicated interface (Figure 5.12) has been set up in order to provide a communication channel between developers and final end-users. This latter collects users remarks from all SIEMS instances (see Section 5.1.2) and allows developers to answer directly by giving a processing priority and specifying if the bug or the improvement will be released in a future version.

![Figure 5.12 The Feedback interface regrouping users remarks from all SIEMS instances](image-url)
5.1.1 Application Statistics

In order to evaluate the MDA approach under production applications, some development statistics are reported below and will be taken as input in Part III of this document. Since the SIEMS application is entering into its ending phase of development, these data should give a sufficient overview of the final performances.

At the moment, SIEMS has been released to version 2.0.7 after 38 deployments. A clean version of the application weights about 175 MB with 13’782 files in 2’246 folders and contains more than 3’000’000 lines of code, whereof only 20% has been manually coded (see Appendix C). Regarding the SIEMS model (SIEMSModel.xmi), it weights 2.35 MB as an archive (xml.zip file) and reaches 26.7 MB in its standard form (xml file). More than 16’000 elements have been manually modeled and the 357 model entities produce exactly 400 tables in the database when the model is transformed.

Finally, building the entire application in offline mode (i.e. without checking dependencies with Maven) takes about 24 minutes using the relatively powerful desktop computer from Table 1.1. This considerable amount of time rises easily to half an hour after creating database schema and deploying application in a development context. As we will see in Section 6.1.1, model validation and compilation times are some of the weak points of this development process.

5.1.2 Infrastructure and Deployment

From a physical perspective, it is worth to precise that each medical or health related institution is proprietary of its own SIEMS instance. This was a mandatory constraint to fulfill security and privacy concerns explained in Section 4.2.2. Obviously, managing multiple application instances results in a more complex system of deployment that needs to be carefully organized in order to avoid application errors or data corruption.

Figure 5.13 schematizes the infrastructure of the SIEMS execution platform. The Java EE platform handles multiples instances of Apache Tomcat services which in turn manage multiple instances of the SIEMS application. Each SIEMS instance is linked to a connection pool that communicates with the PostgreSQL [Pos08] server through a specific JDBC driver.

The interested reader could also notice that the Tomcat server is directly connected to the PostgreSQL server through the Realm component. Realm is “a "database" of usernames, passwords, and roles […] assigned to those users” [Rea08] that enables Tomcat to identify valid actors of the web application. Furthermore, each SIEMS instance communicates through a socket with the OpenOffice.org (OOo) server [Ope08b]. This latter provides a language independent API to access functionalities
of the office suite in order to generate XML-based documents and export them to other file formats, like PDF or Word for example.

![Diagram]

**Figure 5.13 The infrastructure of the SIEMS execution platform**

### 5.2 Extending a SIEMS Use Case

In Chapter 3, we have seen how the AndroMDA methodology has been applied with a HelloWorld example to create a MDA based application from scratch. Yet, this experience does not really reflect the development process in a multi-user environment as use cases are often extended when business requirements change or when user experience is not met. To illustrate this possible scenario, a SIEMS use case will be extended with some extra functionality in order to evaluate the programming consequences, particularly from the logic implementation and the presentation tier perspective.

Figure 5.14 shows the EditUserGroup interface as currently represented in SIEMS. This really simple interface only provides a text field for entering the users group name and the actual way to assign or change a user from a group consists on editing each user account separately as the Figure 4.10 may suggest. Extending the UserGroups use case tries to address this lack of usability, i.e. provide an easier and faster method to assign multiple users during group edition.
In that sense, the SIEMS model is adapted accordingly to propagate a possible list of users. Business logic is then completed to handle this new functionality and the presentation tier actualized to display a list of checkboxes. Finally, the database schema may eventually be altered before deploying the application. These steps are detailed in the next sections with the knowledge acquired during the HelloWorld example.

### 5.2.1 Model Adaptation

Figure 5.15 depicts the actual activity diagram of the UserGroups use case, located in the `ch.tecost.siems.uscusecase.institution.system:editUserGroup:editUserGroups` package of the SIEMS model. At this stage, we particularly focus on the “edit the user group” action state from the User swimlane as all modifications are performed on ingoing and outgoing transitions of this element.

Actually, adding a `users` parameter to the three transitions is enough to achieve our requirements exposed on previous section. On ingoing transitions, this parameter
carries the users that have been assigned to the current group and on the outgoing transition, this latter collects users that have been selected on the front-end interface. Furthermore, this parameter is typed as `List` so that list elements can be sorted more easily.

Note that displaying the list of available users on the front-end does not require an additional incoming parameter. As mentioned in Section 3.4.2, outgoing transitions allow specializing input rendering and therefore, the `users` parameters is tagged by setting the `andromda.presentation.web.view.field.type` property to `checkbox`. This will consequently create a useful method on the controller class to initialize default values of the users list.

Next step consists on adapting controller and service methods in the `UserGroups` class diagram (Figure 5.16), which is available in the same package than the activity diagram. As we could expect, this diagram contains the controller class as well as a dependence to the service class and consequently to all its available operations. Thus, adding a `users` parameter to the `loadNewUserGroup()`, `loadUserGroup()` and `saveUserGroup()` operations of the controller (Figure 5.19) allows transitions to carry or defer correctly the parameters that have been specified in the previous paragraphs.

Looking further, we discover that most service operations related to user groups manipulate a `UserGroupSimple` value object. Searching the SIEMS model for this latter, MagicDraw UML tell us that the value object is actually referenced in other use cases and that it is integrated in the amazing class diagram dedicated to the principal entity of the system, namely the `user` itself, as shown on Figure 5.17. Luckily, this diagram also states that our value object can be transformed from the `UserGroup` entity, which in turn has already an association to eventually many `SiemsUser` entities. This means that the database schema does not require any alteration as the `junction table` between `UserGroup` and `SiemsUser` already exists.
In order to propagate the users list from controller to the DAO while avoiding conflicts, the UserGroupSimple element is extended with a new value object as shown on Figure 5.18. This latter, let’s call it UserGroupFull, simply embeds a users list and inherits others attributes from the parent element by linking them together with a generalization element. Furthermore, a dependency element is drawn from the UserGroup entity to the UserGroupFull value object to ensure that a transform will be available in the DAO implementation class.

Accordingly, the signature of the saveUserGroup() service method is changed to take an UserGroupFull as input parameter. Finally, as we do not want to overload other use cases with a useless list of users, a new service operation is created in order to load a UserGroupFull as shown by the getUserGroupAsUserGroupFull() operation on Figure 5.19.

At this stage, our modeling process is completed after less than half an hour of research and design. The result of the modeling adaptation is shown on Figures 5.18 through 5.20.
5.2.2 Business Logic Implementation

Before beginning implementation, the model needs to be validated and transformed in order to create or update existing resources in target folders, which requires running a full build of the application. As explained in Section 3.5, concrete implementation Java files are extensions of abstract classes, which are always re-generated in these target folders. This means that refreshing the SIEMS project under Eclipse automatically leads to errors summarized in the Problems view as shown on Figure 5.21.

Business logic is implemented following the bottom-up approach introduced in Section 3.6, i.e. propagating data from the data access layer to the presentation layer. As no entities operations have been defined, the unique DAO task consists on completing the transformation between the UserGroup entity to the new UserGroup-
Full value object as shown by the snippet code from Listing 5.1. As users names could be displayed in the front-end, the users attribute is initialized with a list of SiemsUserSimple value objects (lines 7 and 15), which contains the requested data as shown on Figure 5.17.

```java
@override
public ch.tecost.siems.vo.user.UserGroupFull toUserGroupFull(final ch.tecost.siems.dao.user.UserGroup entity) {
    SiemsUserDao siemsUserDao = ServiceLocator.instance().getSiemsUserDao();
    List<SiemsUser> siemsUsers = new LinkedList<SiemsUser>(entity.getSiemsUsers());
    siemsUserDao.toSiemsUserSimpleCollection(siemsUsers);
    // Sort users using fullName property:
    Collections.sort(siemsUsers, new BeanComparator("fullName"));
    UserGroupFull userGroupFull = new UserGroupFull();
    userGroupFull.setId(entity.getId());
    userGroupFull.setName(entity.getName());
    userGroupFull.setUsers(siemsUsers);
    return userGroupFull;
}
```

Listing 5.1 Implementation of the toUserGroupFull() DAO method

From the service perspective, implementing the handleGetUserGroupAsUserGroupFull() simply consists on delegating data retrieval to the DAO as shown by Listing 5.2. Moreover, the dependency that has been modeled from the UserGroup entity to the UserGroupFull value object produces a transform rule (line 4), which can be applied to the entity just after loading.

```java
@override
protected UserGroupFull handleGetUserGroupAsUserGroupFull(Long groupId) throws Exception {
    return (UserGroupFull) this.getUserGroupDao().load(
            UserGroupDao.TRANSFORM_USERGROUPFULL, groupId);
}
```

Listing 5.2 Implementation of the handleGetUserGroupAsUserGroupFull() service method

Implementing the handleSaveUserGroup() method requires more knowledge about how Hibernate handles many-to-many relationships in the database. In fact, current version of the Hibernate cartridge does not provide a direct way to update the list of users from a group as the length of the source code from Listing 5.3 may suggest. Indeed, users that could have been unselected in the front-end need to be first removed (lines 13 and 23) and then, those who are actually selected can finally be added to the group (line 45). Furthermore, as navigability is allowed from both sides, we need to explicitly remove or add the current group from all concerned users (lines 19 and 40) before delegating physical storage to the DAO layer.

```java
@override
protected void handleSaveUserGroup(UserGroupFull userGroup) throws SiemsUserServiceException {
    UserGroup ug = null;
    if (userGroup.getId() == null) {
        ug = UserGroup.Factory.newInstance();
    } else {
        ug = this.getUserGroupDao().load(userGroup.getId());
    }
    // Collect users that have been removed from group:
    List<SiemsUser> usersToRemove = new LinkedList<SiemsUser>();
    ```
Controller methods do not need a particular attention. In fact, the name parameter is validated by Struts through its corresponding tagged value in the activity diagram and the group users list can be emptied as stated by the class diagram of Figure 5.17. Therefore, implementation of the `saveUserGroup()` method (Listing 5.4) is trivial and only requires to fill an `UserGroupFull` value object with provided form values before sending it to the `saveUserGroup()` service method defined in previous Section.

```
@Override
public final void saveUserGroup(ActionMapping mapping, SaveUserGroupForm form, HttpServletRequest request, HttpServletResponse response) throws Exception {
    UserGroupFull userGroupFull = new UserGroupFull();
    userGroupFull.setId(form.getId());
    userGroupFull.setName(form.getName());
    userGroupFull.setUsers(PropertyUtils.asLong(form.getUsers()));
    this.getUserGroupService().saveUserGroup(userGroupFull);
}
```

Listing 5.4 Implementation of the `saveUserGroup()` controller method

The `loadUserGroup()` controller method (Listing 5.5) simply prepares data in order to synchronize checkboxes state in the front-end view. Particularly, it retrieves all active users by calling the `getActiveSiemsUsersSimple()` method available in the `SiemsUserService` (see Figure 5.19) and sends the list of groups users in the form of ids (line 8). This last task allows checking group users in the presentation tier, as it
will be shown in the next section. Implementation of the remaining `loadNewUserGroup()` method follows the same logic and therefore, no further explanations are given here.

```java
@override
public final void loadUserGroup(ActionMapping mapping, LoadUserGroupForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception {
UserGroupFull userGroupFull = this.getSiemsUserService().getUserGroupAsUserGroupFull(form.getId());
List<SiemsUserSimple> siemsUsers = this.getSiemsUserService().getActiveSiemsUsersSimple();
form.setName(userGroupFull.getName());
form.setUsers(PropertyUtils.asList(userGroupFull.getUsers(), "id"));
form.setUsersBackingList(siemsUsers, "id", "fullName");
}
```

Listing 5.5 Implementation of the `loadUserGroup()` controller method

### 5.2.3 Presentation Tier Actualization

It is worth to precise that web resources for the EditUserGroup interface have already been imported to web folder, may it be for layout or Javascript purposes. This means that web files have to be manually actualized to adopt the changes performed in the previous Sections. Listing 5.6 presents the initial source code of the `edit-the-user-group-save-user-group.jspf` file, which is responsible for displaying the EditUserGroup form.

```html
<html:form method="post"
styleId="editUserGroupsEditTheUserGroupSaveUserGroupForm"
action="EditUserGroups/EditTheUserGroupSaveUserGroup">
<table class="formComponent">
<tbody>
<tr>
<td class="label">
<bean:message
key="edit.user.groups.[…].user.group.param.name"/>
</div>
</td>
<td class="field">
<html:text name="form" property="name"
styleId="saveUserGroupName" styleClass="formText"
onmouseover="hints.show('editUserGroupName')" onmouseout="hints.hide()"
styleId="saveUserGroupName" styleClass="formText"
onmouseover="hints.show('editUserGroupName')" onmouseout="hints.hide()"
</td>
</tr>
</tbody>
</table>
</html:form>
```

Listing 5.6 Initial contents of the `edit-the-user-group-save-user-group.jspf` file

HTML coding mainly relies on tag libraries, particularly the html [Htm08] and the bean [Bea08] libraries shipped with the Struts cartridge. Furthermore, the JSP expression language (EL) [Exp08] technology is used to dynamically evaluate bean
attributes values. These features will be significantly useful for actualizing the current interface.

Listing 5.7 shows the source code that has been added to the `tbody` element (line 8 of Listing 5.6) in order to display the list of checkboxes and some extra elements for visual representation. The `forEach` tag from the `core` JSTL tag library [Jst08] allows to iterate over users using the `backing list` that was initialized in the `loadUserGroup()` controller method. Finally, the `multibox` tag enables to define the default state of the current checkbox. That is, if the passed `value` can be located in the list specified by the `property` and `name` attributes, then this latter will be checked by default.

```html
<tr>
  <td>
    <div class="formLabel">
      <bean:message key="edit.user.groups.[…].param.users"/>
    </div>
  </td>
  <td>
    <div class="checkboxesCollection">
      <c:forEach items="${form.usersBackingList}" var="user">
        <html:multibox name="form" property="users" value="${user.value}" styleId="user_${user.value}"/>
        <label for="user_${user.value}">${user.label}</label>
      </c:forEach>
    </div>
  </td>
</tr>
```

Listing 5.7 HTML code for displaying the users checkboxes and labels

5.2.4 Deployment and Extension Analysis

At this stage, only the `common`, `core` and `web` subprojects need to be compiled before deploying the SIEMS application. As the model did not change since last build, validating and transforming this latter is simply a waste of time as the `mda` subproject takes no less than 17 minutes to compile.

This relative short use case extension did not create or change entities during the modeling process and consequently, database schema does not need to be altered. Thus, it is important to clarify that AndroMDA is not able to generate SQL scripts to update database accordingly. Nevertheless, developers can make use of the generated SQL files, namely `schema-create.sql` and `schema-drop.sql`, to extract the required statements in order to create an incremental SQL script.

Accessing again our `EditUserGroup` interface should now offer the ability to define multiple users for a group as depicted by Figure 5.22. Note that some checkboxes are already selected as the old and painful way to assign users to a group consisted on editing each user separately. Saving the group after checking or unchecking some users should lead to one success message, confirming that data correctly propagated from the presentation tier to the data store as schematized on Figure 3.1.
The aim of this short use case extension was to give an overview of the development process of a large-scale application in a multi-developer environment. Particularly, it tried to show that developers do not require a complete knowledge of the business model and that development focus is no more centered on code but rather on design. Moreover, the architecture of AndroMDA based applications enables to intuitively locate altered resources and provides faster business logic implementation with the help of comprehensible model views.

Consequences caused by the functionality extension of the EditUserGroup interface are relatively minimal as deploying the SIEMS application into production only consists on replacing the war archive and restarting the Tomcat server. Nevertheless, this scenario rarely occurs and important deployments lead to significant database alterations or data shifts. Consequently, updating productions databases requires a particular attention in order to ensure data integrity. Moreover, the presentation tier actualization was reduced to the editing of a single file without any kind of extra functionality, like filtering or sorting. All these aspects are analyzed in Part III of this document in order to evaluate how the MDA approach enhances software development productivity.
Part III.

Lessons Learned
6 Evaluation

6.1 Development Process
6.1.1 MDA Tool Chain
6.1.2 Code Generation

6.2 Development Productivity
6.2.1 Business Modeling
6.2.2 Code Patterns

“There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.”

Frederick P. Brooks, Jr.
Software engineer and computer scientist

Part I and Part II of this work introduced and put into practice the technologies and techniques behind the Model Driven Architecture approach. Particularly, they showed how to follow the AndroMDA methodology to build and extend MDA-based applications. Even if the HelloWorld example from Chapter 3 revealed the efficiency of a MDA development process, the concrete extension of the SIEMS use case (Section 5.2) undoubtedly remains a more representative scenario in enterprise environments.

In order to evaluate the contribution of the OMG vision to software development, we consider some critical aspects of the Model Driven Architecture and determine if they have been comprehensively applied with the AndroMDA methodology. Especially, we examine how MDA supports the development process of large-scale systems and analyze performance and productivity consequences, particularly from a multi-developer point of view.
6.1 Development Process

In opposition to traditional software development which relies on low-level design and coding, the MDA development process is driven by the activity of modeling an application system through the abstraction of models. Even if the development life cycle of a MDA-based application does not look very different from a traditional one, the main difference relies on the nature of the artifacts that are created during the development process.

As shown on Figure 6.1, the design phase produces an executable model, which automatically generates corresponding source code during the transformation process. The logic implementation phase overrides fragments of generated source code with business implementation, which serves as input for the application deployment. As a matter of fact, the final application remains closely linked to implementation and generated source code and thus, to the high-level specifications of the formal models.

This development process has been intrinsically observed during the extension of the SIEMS use case presented in Section 5.2. The existing source code composed of
implemented and generated fragments has been automatically adapted to take into account the new business requirements defined during model design. This states that the transformer tool has sufficient knowledge for updating source files and inversely, that source files remain synchronized with classes and diagrams of the SIEMS model to allow an efficient update.

Nevertheless, if this last assumption is valid for Java implementation classes and for XML descriptors, it becomes obsolete for presentation tier resources. Indeed, the lack of markers inside web files prevents any possible adaptation once they have been imported to the src folder and manually edited. As the OMG standards only define the architecture of MDA applications, generating and updating code for a specific execution platform relies on the capabilities of the transformer tool, particularly when it executes the PSM to code transformation.

In order to allow an iterative and incremental development process, the tools that realize the model-driven architecture are relevant to ensure consistency during code generation while providing enough flexibility for developers (see developers shortcuts on Figure 6.1). These tools form the commonly named MDA tool chain and are either fully integrated within a complete and mostly commercial solution or loosely sequenced with partial and often Open Source solutions (see Section 1.3). As we will see below, this tool chain is critically important with respect to performance and productivity.

### 6.1.1 MDA Tool Chain

The MDA tool chain that defines the model-driven environment of the SIEMS application, as well as the one of the HelloWorld example, is not only represented by the AndroMDA framework as we may believe. Even if this latter constitutes the main piece of the tool chain, the MDA development process is performed through different software products and components built on top of Software Engineering standards and technology specifications. Ranging from CASE tools to model or code compilers, software is sequenced such that the output of each tool becomes the input of the next one as schematized on Figure 6.2.

An obvious question to be answered is how efficiently the sequenced tools integrate the development process of the SIEMS application. More precisely, how does a software product handle the output of the previous tool in an iterative process? Unfortunately, we have to admit that using partial solutions lead to increasing compilation times as the model evolved to handle the complexity of the medical business domain.

---

1 Computer-Aided Software Engineering
For example, AndroMDA parses and validates the XMI model outputted by MagicDraw UML before performing code generation. Even if each model element is provided with a timestamp, AndroMDA is not able to detect model changes and consequently, the transformation process is always applied to the whole model or to a specific subset of elements in the best case. As a result, building the SIEMS application can reach half an hour as long as no validation error is thrown.

Filters can be specified when building the mda subproject, responsible for model transformation. For example, running the maven mda command tells AndroMDA to only process resources related to entities.

In fact, MagicDraw UML is supplied with AndroMDA datatypes and profiles to specialize elements but without any constraint definitions required to achieve model transformation. This means that MagicDraw UML cannot check model validity on-the-fly and errors are only outputted in the console during the validation phase performed by AndroMDA. One of the most recurrent and frustrating error concerns mistyped transitions parameters that defer to controller methods as shown on Figure 6.3.

![Figure 6.2 The MDA tool chain of the SIEMS application](image-url)

**Figure 6.2 The MDA tool chain of the SIEMS application**

![Figure 6.3 A model validation error caused by a mistyped transition parameter](image-url)

**Figure 6.3 A model validation error caused by a mistyped transition parameter**
Note that AndroMDA only reported the error after 17 minutes of validation! In the context of a consistent use case creation (i.e. defining new entities, services, controllers, diagrams, etc.), it would require to validate the whole model again (and again) until all model errors have been corrected. Indeed, the lack of integration between MagicDraw UML and AndroMDA results in repetitive and time consuming tasks that could be avoided if model validation was addressed during design.

There are other secondary but perfectible processes in this MDA tool chain that could reduce development time between model design and application deployment. For example, the SIEMS project inside the Eclipse workspace needs to be refreshed, cleaned and rebuilt every time AndroMDA generates source code. Within the SIEMS application, Eclipse becomes inefficient when handling a considerable amount of files (see Table C.1) and cannot manage to complete these tasks in less than 3 minutes. We could also mention the fact that AndroMDA does not create update scripts for database alteration, which would appreciably facilitate application testing and deployment, but this has already been discussed on Section 5.2.4.

6.1.2 Code Generation

Previous Section has revealed some integration problems in the MDA tool chain that degrades to some extent the development process of the SIEMS application. Nevertheless, the MDA paradigm focuses mainly on standardized models in order to allow vendor-neutral transformer tools to generate source code for a specific execution platform. In that sense, we will examine how automation of code generation leverages the development process and how efficiently does the generated code fit the underlying technology.

As mentioned in Section 3.1.2, generating source code is achieved through the specific code patterns contained within the AndroMDA cartridges. From database persistence to presentation tier layout, AndroMDA addresses all business requirements by producing highly targeted code for the specific platform, namely Java EE in our case. Actually, the HelloWorld example showed the efficiency of the model transformation process by reducing logic implementation to some dozens of lines.

Obviously, large-scale applications demand a more considerable effort from developers in order to implement business logic. Measuring this involvement can be done by comparing the estimated hand-coded resources with the overall resources of the SIEMS application, as illustrated respectively on Table C.2 and Table C.1. These statistics show clearly that files requiring manual edition represent less than 32% of the overall resources and that hand-coded lines constitute only 20%\(^1\) of the global

\(^1\) Surprisingly, this value also reflects perfectly the 80/20 rule followed by code generator systems [Cod08].
lines of code. Note that this last measure is an upper bound since manually edited files usually contain some source code (e.g. documentation, method skeletons, HTML, etc.) and that hand-coded XML files are mainly sample data, probably generated in an automatic manner.

A deeper look into the hand-coded statistics also reveals that code on front-end is significantly more important than in the back-end. Precisely, lines of code from the presentation tier represent about 64% of the hand-coded resources if we exclude database related files. As a matter of fact, the Bpm4Struts cartridge has not been designed to handle the advanced features and functionality of Rich Internet Applications. Nevertheless, customized code can be generated by overriding the velocity [Vel08] templates (VSL files), located inside AndroMDA cartridges. Defining custom patterns for the presentation tier addresses partially the requirements for rendering dynamic web pages (DHTML) but interaction and animation remains a specialized task that is handled locally by scripting languages.

Another critical aspect of code generation concerns application performance. Without talking about excessive memory and CPU usages, some situations show that generated code is not optimal, particularly when dealing with a large amount of data. In fact, using entities relationships to retrieve data requires sometimes loading an incredible schema of values, which results in questionable processing times. Again, marking these relationships for lazy evaluation or using HQL [Hql08] finders can significantly improve system performance.

### 6.2 Development Productivity

Skepticism around code generation always prevails as long as some productivity gain in software development has not been demonstrated. Those who have used XDOCLET [XDO08] or Java5 Annotations [J5a08] may have experienced the benefits of reducing repetitive tasks through javadoc tags. Nevertheless, the basis for automating code generation remains in the Java code and business functionality is scattered over multiple source files.

The MDA approach takes a step further in enhancing code generation through model transformations. Defining business requirements through the abstraction of models allows code generators to create and maintain coherent application structures while freeing developers from the complexity of the underlying technology. In 2003, a case study [TMC08a] from the Middleware Company [TMC08b] observed that an MDA team has able to develop a complete Java EE application 35% faster than a traditional team and encourages “organizations that wish to improve their developer productivity to evaluate MDA-based development tools for their projects.”

Even if the MDA paradigm offers others architectural benefits (see Section 2.4), business modeling and code patterns aspects will be particularly discussed in the
next sections to examine how software development productivity is enhanced when realizing large-scale applications.

### 6.2.1 Business Modeling

As explained in Section 2.4.3, business functionality and behavior are modeled in the high-level PIM model, independently from the execution platform. Moving the developer focus from code to an abstract view forces this latter to concentrate on business model rather than diving into implementation details. Taking into account the complexity of the SIEMS model, this model-driven approach improves software development productivity in two ways.

First, a complete knowledge of the *business model* is no more required to understand complex applications. Extending the SIEMS use case on Section 5.2 proved that developers rely on *readable class and activity diagrams* to gather information about the system in order to design requested requirements. Furthermore, addressing business requirements during the modeling phase also reduces “the possibility of introducing architectural flaws into your system later on in the development” [TMC08a].

Then, developers make a more intelligent use of their time by focusing on business logic rather than on the details of writing code. As AndroMDA handles platform specific details and produces source code based over confirmed *design patterns*, logic implementation shifts to a secondary level. This also aims to promote architectural principles in the application while reducing the number of hand-coded source lines as shown in the next Section.

### 6.2.2 Code Patterns

The quality of source code is directly related to the quality of *code patterns* that finalize the model transformation process. In the MDA context, code patterns are not only bounded to business design patterns but to all potential resources that realize the application system. From application descriptors to *interface definitions* and presentation resources, the code base that is produced by generators ensures *consistency* and *maintainability* in opposition to a hand-coded approach.

Consistency has been particularly observed in the interface definitions of some AndroMDA components, such as services and controllers. Implementation of these generated APIs is clearly a straightforward task as the complexity of the component is hidden in the base Java classes. Particularly, using consistent design patterns to address business logic reduces the risk of implementation errors and consequently, the number of hand-coded lines.
Using code patterns to generate code has another significant advantage when talking about maintainability. Updating a unique pattern to fix a bug or adapt source code instantly reflects changes on corresponding files. If most IDEs allow code refactoring on Java and even XML files, source code of the web resources remains complicated to maintain as it does not rely on a base architecture. Taking into account the thousand of files from the SIEMS application (see Table C.1), code patterns express all their usefulness in these kind of situations.
7 Conclusion

7.1 Durability of Models 75
7.2 Software Systems Development 75

“The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency.”

*Bill Gates*
Co-founder and Chairman of Microsoft

“MDA is about using modeling languages as programming languages rather than merely design languages” [Fra03]. These are the words of David S. Frankel to clarify the nature of the Model Driven Architecture as strived by the Object Management Group. Raising the level of abstraction of execution platforms by modeling business requirements through a Platform Independent Model (PIM) offers a new and promising dimension to the Software Engineering domain.

Part I of this work has emphasized the necessity of models in Software Development and introduced the modeling standards and expected benefits behind the MDA approach. Following the AndroMDA methodology, a HelloWorld example has revealed the efficiency of code generation through model transformations. In order to evaluate the MDA integration in large-scale enterprise systems, Part II first introduced SIEMS, an Electronic Health Record (EHR) system and a concrete real-world MDA application; and then detailed a chosen use case extension. Finally, the development process and productivity of MDA-based applications have been evaluated on Part III. Particularly, it analyzed some critical aspects of this development approach, such as the inefficiency of the MDA tool chain, the automation of code generation, the interest of modeling business requirements and the benefits of using code patterns.
Eight years after the OMG released the first MDA guide (currently in version 1.0.1 [Mil03]), many MDA tools have emerged from the market and existing ones tend to become MDA-compliant. In 2005, the Tecost SA company leap on the MDA opportunity to develop an EHR application. And despite the lacking maturity of partial solutions used in the MDA tool chain, costs of development, maintenance and integration have been undoubtedly reduced with the model-driven paradigm.

### 7.1 Durability of Models

Even if business requirements evolve slower that the technologies used to build their software systems, enterprises will face the middleware proliferation problem as soon as they will need to migrate their application to a new execution platform. Conscious of this reality, the OMG proposes to extend the longevity of computing systems by encapsulating the business functionality and behavior inside standard UML models. As a consequence, enterprises will be able to target any platform and technology by transforming business model using the appropriate code patterns.

Nevertheless, this vision of software durability shifts development skills from programming to design. In that sense, enterprises should evaluate the skill levels of their modelers in order to eventually perform platform models (PM) updates accordingly to targeted technology.

### 7.2 Software Systems Development

Radically opposed to Extreme Programming (XP) [Bec00] but compatible with the Agile [Agi08] method, the Model driven Architecture defines a new approach for building software systems using the ability to derive code from models. After four years of development, the SIEMS application has shown the efficiency of this development process, particularly by handling the complexity of the medical domain through abstract models and powerful software components. At the time of writing these lines, a new project based on the same but updated AndroMDA methodology has already been initiated by Tecost SA.

Obviously, commercial solutions have been considered to overcome the poor integration of partial solutions in the development process. Nevertheless, acquiring a complete MDA framework may represent an unaffordable investment for a mid-sized company like Tecost SA. For example, one year user license for the IBM Rational Rose Enterprise costs about 7’500 CHF and a floating license reaches 14’500 CHF per year. With such prices, enterprises need to evaluate the return on investment from multiple MDA-based solutions in order to adopt the software processes which are right for them.
Part IV.

Appendix
Environment Setup

Environment setup is done accordingly to technology and software used within the SIEMS application context (see Chapter 5). Furthermore, tasks described below are relative to Microsoft Windows but they can easily be translated to other operating systems.

1. Install Java JDK, Maven, Tomcat and MagicDraw UML (refer to Section 1.3.2 for software versions and download locations).

1. Set environment variables:
   - JAVA_HOME: <Java installation directory>
   - MAVEN_HOME: <Maven installation directory>
   - PATH: %PATH%;JAVA_HOME\%bin;%MAVEN_HOME\%bin

   To check if Maven installation successfully succeeded, type `maven -v` in a command line window.

2. Define Maven’s repository locations for dependencies retrieval:
   - Create a file called `build.properties` within your home directory
   - Append the following line to the previously created file:

   ```
```

   For more plugin properties, see: http://maven.apache.org/maven-1.x/plugins/java/properties.html

3. Download AndroMDA plugin using Maven:

   ```
maven plugin:download -DgroupId=andromda -DartifactId=maven-andromdapp-plugin -Dversion=3.1
```

   Under Microsoft Windows Vista, also add the `maven -Dos.name="Windows.XP"` switch to the preceding command to avoid compatibility errors.

4. During this phase, Maven will regenerate its plugin cache and create the repository directory for dependencies download. Make sure the task completes with a BUILD SUCCESSFUL message.
Plugin cache will be regenerated
le répertoire C:\Users\Devel\maven\repository n'existe pas. Tentative de création.
Tentative de téléchargement de commons-ic-20030203.000550.jar.
59K downloaded
Tentative de téléchargement de commons-net-1.1.0.jar.
130K downloaded
Tentative de téléchargement de commons-httpclient-2.0.jar.
217K downloaded
Tentative de téléchargement de commons-lang-2.0.jar.
160K downloaded
Tentative de téléchargement de commons-logging-1.0.3.jar.
30K downloaded
Tentative de téléchargement de jsch-0.1.5.jar.
79K downloaded
Tentative de téléchargement de commons-jelly-20030902.160215.jar.
150K downloaded
Tentative de téléchargement de commons-jelly-tags-velocity-20030303.205659.jar.
217K downloaded
Tentative de téléchargement de velocity-1.4-dev.jar.
500K downloaded
Tentative de téléchargement de commons-jelly-tags-xml-20030211.142705.jar.
31K downloaded
Tentative de téléchargement de commons-jelly-tags-interaction-20030211.143817.jar.
4K downloaded
build:start:
plugin:download-artifact:
[mkdir] Created dir: C:\Users\Devel\maven\repository\andromda\plugins
[echo] repo is 'http://repo1.maven.org/maven'
47K downloaded
plugin:download:
[delete] Deleting 1 files from C:\Devel\Software\Maven_1.0.2\plugins
[delete] C:\Users\Devel\maven\plugins not found.
[delete] Deleting 39 files from C:\Users\Devel\maven\cache
[delete] Deleting 21 directories from C:\Users\Devel\maven\cache
[copy] Copying 1 file to C:\Devel\Software\Maven_1.0.2\plugins
BUILD SUCCESSFUL
Total time: 37 seconds
Finished at: Mon Dec 01 09:21:21 CET 2008

Figure A.1 AndroMDA plugin download using Maven
Note that variable `MAVEN_REPO` must be set in Eclipse preferences and point to `/.maven/repository` inside your home directory as shown by Figure B.2.
Figure B.2 The Classpath Variables section of the Preferences dialog box under Eclipse
Source statistics of Table C.1 and Table C.2 report effective file and line count for the most representative file types and patterns of the SIEMS application. In that sense, binary (images, legacy application files, etc.) as well as archive file types (jar, war, zip, etc.) have been excluded from the results exposed below.

Overall Resources

Overall resources represent unique files that have been generated by AndroMDA and those which have been imported to src folders or created manually.

<table>
<thead>
<tr>
<th>File Count</th>
<th>Line Count by Subproject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>common</td>
</tr>
<tr>
<td>*.java</td>
<td>6'586</td>
</tr>
<tr>
<td>*.jsp</td>
<td>606</td>
</tr>
<tr>
<td>*.jspx</td>
<td>1'415</td>
</tr>
<tr>
<td>*.js</td>
<td>37</td>
</tr>
<tr>
<td>*.css</td>
<td>406</td>
</tr>
<tr>
<td>*.xml</td>
<td>564</td>
</tr>
<tr>
<td>*.sql</td>
<td>226</td>
</tr>
<tr>
<td>*.vsl</td>
<td>18</td>
</tr>
<tr>
<td>*.properties</td>
<td>337</td>
</tr>
<tr>
<td>Totals</td>
<td>10'198</td>
</tr>
</tbody>
</table>

Table C.1 Overall resources statistics from the SIEMS application

For your information, files and lines have been counted under a Linux desktop computer using the following command, where <fileTypes> is a space-separated list of the corresponding file types:

```
for i in <fileTypes>; do echo "[*.$i] Files: `find . -name "*.\$i" | wc -l` && for j in common core db mda web; do echo "[*.$i \$j] Lines: `find \$j -name "*.\$i" | xargs cat | wc -l`; done; done
```
Hand-Coded Resources

Hand-coded resources represent all files that are no more generated by AndroMDA. These include utilities, implementation, initialization data, templates and descriptors files as well as web resources from the src folder. Results below are an estimation of all possible hand-coded source files.

As before, counting files and lines has been achieved with the following command, where <filePatterns> is a list of the corresponding file patterns:

```
  » for i in <filePatterns>; do echo "[$i] Files: `find . -wholename "*$i" | wc -l" Lines: "`find . -wholename "*$i" xargs cat | wc -l`; done
```

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Files</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>/common/src/*.java</td>
<td>119</td>
<td>15'350</td>
</tr>
<tr>
<td>/core/src/*.java</td>
<td>554</td>
<td>81'060</td>
</tr>
<tr>
<td>/db/src/*.java</td>
<td>57</td>
<td>9'112</td>
</tr>
<tr>
<td>/db/src/*.xml</td>
<td>103</td>
<td>319'047</td>
</tr>
<tr>
<td>/db/src/*.sql</td>
<td>216</td>
<td>14'104</td>
</tr>
<tr>
<td>/mda/src/*vsl</td>
<td>18</td>
<td>2'971</td>
</tr>
<tr>
<td>/web/src/*.java</td>
<td>376</td>
<td>80'912</td>
</tr>
<tr>
<td>/web/src/utils/*.java</td>
<td>53</td>
<td>7'249</td>
</tr>
<tr>
<td>/web/src/*.tld</td>
<td>3</td>
<td>1'048</td>
</tr>
<tr>
<td>/web/src/*.jsp</td>
<td>437</td>
<td>22'841</td>
</tr>
<tr>
<td>/web/src/*.jspf</td>
<td>767</td>
<td>75'875</td>
</tr>
<tr>
<td>/web/src/*.js</td>
<td>34</td>
<td>10'431</td>
</tr>
<tr>
<td>/web/src/*.css</td>
<td>165</td>
<td>10'729</td>
</tr>
<tr>
<td>/web/src/*.properties</td>
<td>326</td>
<td>55'806</td>
</tr>
<tr>
<td><strong>Totals</strong></td>
<td>3'228</td>
<td>706'535</td>
</tr>
</tbody>
</table>

Table C.2 Estimation of hand-coded resources from the SIEMS application
Project Websites

SIEMS Application

http://www.siems.ch/siems/

Figure D.1 The welcome page of the SIEMS application

SIEMS Portal

http://www.siems.ch

Figure D.2 The information portal of the SIEMS project
D Project Websites

Master Project

http://diuf.unifr.ch/softeng/student-projects/completed/dealmeida/

Figure D.3 The project page of this Master Thesis
Main resources related to this work, excepting SIEMS source code, can be located on the attached CD-ROM. Particularly, this latter contains:

- *Documentation*, presentations and images related to this Master Thesis.
- Three different development phases of *HelloWorld* example.
- Useful *resources* to setup the HelloWorld development environment.
- *Software* for the MDA tool chain as listed on Table 1.2.
- *Templates* used to create this documentation and to handle bibliographic references.

![Contents of the CD-ROM](image.png)

*Figure E.1 Contents of the CD-ROM*
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer-Aided Software Engineering</td>
</tr>
<tr>
<td>CEO</td>
<td>Chief Executive Officer</td>
</tr>
<tr>
<td>CHF</td>
<td>CH (Swiss) Franc</td>
</tr>
<tr>
<td>CIM</td>
<td>Computer Independent Model</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DAO</td>
<td>Data Access Object</td>
</tr>
<tr>
<td>DHTML</td>
<td>Dynamic HyperText Markup Language</td>
</tr>
<tr>
<td>EAR</td>
<td>Enterprise ARchive</td>
</tr>
<tr>
<td>EHR</td>
<td>Electronic Health Record</td>
</tr>
<tr>
<td>EMR</td>
<td>Electronic Medical Record</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>HQL</td>
<td>Hibernate Query Language</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines</td>
</tr>
<tr>
<td>ICD10</td>
<td>International Statistical Classification of Diseases and Related Health Problems</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>ISBN</td>
<td>International Standard Book Number</td>
</tr>
<tr>
<td>JAR</td>
<td>Java ARchive</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JRE</td>
<td>Java Runtime Environment</td>
</tr>
<tr>
<td>JSP</td>
<td>JavaServer Pages</td>
</tr>
<tr>
<td>JSTL</td>
<td>JavaServer Pages Standard Tag Library</td>
</tr>
<tr>
<td>LCD</td>
<td>Liquid Crystal Display</td>
</tr>
<tr>
<td>MDA</td>
<td>Model Driven Architecture</td>
</tr>
<tr>
<td>MDD</td>
<td>Model Driven Development</td>
</tr>
<tr>
<td>MOF</td>
<td>Meta Object Facility</td>
</tr>
<tr>
<td>NIC</td>
<td>Nursing Interventions Classifications</td>
</tr>
<tr>
<td>NOC</td>
<td>Nursing Outcome Classifications</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>OOP</td>
<td>Object Oriented Programming</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>PDF</td>
<td>Portable Document Format</td>
</tr>
<tr>
<td>PIM</td>
<td>Platform Independent Model</td>
</tr>
<tr>
<td>PSM</td>
<td>Platform Specific Model</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RIA</td>
<td>Rich Internet Application</td>
</tr>
<tr>
<td>ROM</td>
<td>Read Only Memory</td>
</tr>
<tr>
<td>SATA</td>
<td>Serial Advanced Technology Attachment</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SIEMS</td>
<td>Système d’Information pour les Etablissements Médicalisés ou de Soins</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>UTF</td>
<td>Unicode Transformation Format</td>
</tr>
<tr>
<td>WAR</td>
<td>Web Application Archive</td>
</tr>
<tr>
<td>XMI</td>
<td>XML Metadata Interchange</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSL</td>
<td>Extensible Stylesheet Language</td>
</tr>
</tbody>
</table>
References


References


Referenced Web Resources


[Cre08] **Creative Commons.** *Attribution-Noncommercial-Share Alike 3.0 Unported.* [http://creativecommons.org/licenses/by-nc-sa/3.0/](http://creativecommons.org/licenses/by-nc-sa/3.0/) [Accessed December 17, 2008].


**Referenced Web Resources**

[Str08] **Apache Software Foundation.** Struts. *Apache Tomcat.*


[VHL08] **VHL.** *Virginia Henderson International Nursing Library.*

[Las08] **Laszlo Systems Inc.** *OpenLaszlo.*

[Pos08] **PostgreSQL Global Development Group.** *PostgreSQL.*

[Rea08] **Apache Software Foundation.** The Realm Component. *Apache Tomcat.*

[Ope08b] **OpenOffice.**

[Htm08] **Apache Software Foundation.** HTML Taglib Guide. *Struts.*


[Exp08] **Sun Microsystems Inc.** *Expression Language.*

[Jst08] **Sun Microsystems Inc.** *JavaServer Pages Standard Tag Library.*

[Vel08] **Apache Software Foundation.** *Velocity.*

[Hql08] **Red Hat Inc.** Hibernate Query Language. *Hibernate.*


A

abstract model, 8
Accessibility, 49
action state, 28
activity diagram, 13, 24, 26
actor, 10, 26
AndroMDA, 16
API See Application Programming Interface
Application Programming Interface, 18
artifact, 67
automatic code generation, 8

B

backing list, 63
Bpm4Struts, 28, 71
business functionality, 10, 18
business layer, 18
business logic, 72
business logic implementation, 41
business model, 72

C

cartridge, 17, 18, 28, 70
CASE See Computer-Aided Software Engineering
CIM See Computational Independent Model
class diagram, 13, 24
class model, 11
code model, 11
code pattern, 70, 72
Computation Independent Model, 9

D

computer-Aided Software Engineering, 68
conceptual design, 14
consistency, 68, 72
constraint definition, 69
contractual element, 10
controller, 26, 57

E

efficiency, 66
EL See expression language
entity, 25
executable model, 67
expression language, 62
Extreme Programming, 75
Index

F
Firefox, 45
Flash, 52
front-end view, 28

G
generalization, 58

H
Hibernate, 36
HQL, 71

I
implementation diagram, 13
interaction diagram, 13
interface code model, 11
interface definition, 72
interoperability, 14

J
Java5 annotations, 71
davascript, 32

K
knowledge, 72

L
lazy evaluation, 71
level of abstraction, 74
logic implementation, 67
logical data, 11
logical view, 25

M
MagicDraw UML, 24
maintainability, 72
many-to-many relationships, 60
mapping rule, 12
marker, 68
MDA tool chain, 68
MDD See Model Driven Development
Meta Object Facility, 13
metametamodel, 13
metamodel, 9
Meta-Object Facility, 7
model, 8
Model Driven Development, 8
model transformation process, 72
MOF See Meta-Object Facility

O
OpenLaszlo, 52
OpenOffice.org, 54

P
performance, 66
PIM See Platform Independent Model
Platform Independent Model, 10
Platform Specific Model, 11
plugin, 19
portability, 15
PostgreSQL, 23
presentation layer, 18
productivity, 15, 66, 72
PSM See Platform Specific Model

R
Realm, 54
requirements, 10
return on investment, 75
reusability, 8
RIA See Rich Internet Application
Rich Internet Application, 52, 71
Index

S

- schema model, 11
- service, 18, 26, 36, 37, 57
- Spring, 38
- stereotype, 18, 24
- swimlane, 28
- syntactic sugar, 39

T

- tag library, 62
- template file, 18
- transformation knowledge, 12
- transformer tool, 12
- transition, 28
- translation, 26

U

- Unified Modeling Language, 7, 13
- use case, 19, 24
- use case diagram, 26

V

- validation, 36
- value object, 18, 26
- velocity, 71

W

- Web 2.0, 41
- workflow, 11

X

- XDOCLET, 71
- XMI See XML Metadata Interchange
- XML Metadata Interchange, 7, 14

UML See Unified Modeling Language