Model Driven Architecture in Adaptive Library Generation

Joan De Arcayne Mañá
Inmaculada Jiménez Moral
Abstract

This master thesis is about building an adaptive library based on a platform independent model, which represents a conceptual design realizing the functional requirements being independent of the technologies and software architectures changes.

This adaptive library is a model, which has a higher level of abstraction than the code that we will generate afterwards, generalizes the solution of those problems that can be resolved using different implementations that uses different kinds of data structures.

In addition the higher level of abstraction that this adaptive library is having will give a speed up in developing the application and higher quality solution because of the maintainability and reusability. This solution is going to have the propriety to be changed easily in order to solve different concrete problems.

Keywords: Model driven architecture (MDA), adaptive library, open architecture ware, optimization.
A big thank you to

Mersiha Huseinovic for tighten my nose every night and supporting me in every single step.

Ajla Blesic for giving to me the sister that I have never had and all those fika that we took.

Mi familia, que me apoyan en todo lo que emprendo, y me han ayudado a ser quien soy ahora.

Noel Galindo, por sus grandes consejos, su paciencia, por quererme, cuidarme y darme caprichos =)
Content

Abstract ............................................................................................................................ ii
A big thank you to ........................................................................................................... iii
1. Introduction ................................................................................................................... 1
   1.1 Goal problem ........................................................................................................... 1
   1.2 Goal criteria / Objective ....................................................................................... 2
   1.3 Motivation ............................................................................................................... 2
   1.4 Structure of thesis ............................................................................................... 2
2. Basic definitions and notion .......................................................................................... 3
   2.1 Background ............................................................................................................ 3
   2.2 General definitions ............................................................................................... 4
   Adaptive code ............................................................................................................ 4
   Automatic code generation ...................................................................................... 4
   2.3 Adaptive code ....................................................................................................... 4
   Reusability of the code ............................................................................................. 4
   Maintainability of the code ...................................................................................... 4
   Portability .................................................................................................................. 4
   Interoperability ......................................................................................................... 4
   2.4 Specific definitions ............................................................................................... 4
   MDSD (Model Driven Software Development) .......................................................... 4
   OMG ............................................................................................................................ 4
   MDA (Model Driven Architecture) ........................................................................... 5
   CIM (Computer-Independent Model) ........................................................................ 5
   PIM (Platform-Independent Models) ......................................................................... 5
   PSM (Platform-Specific Models) ............................................................................... 5
   Stereotype ................................................................................................................. 5
   2.5 Tools definitions .................................................................................................... 5
   oAW (openArchitectureWare) .................................................................................. 5
   MagicDraw ................................................................................................................. 6
   Generator-Workflow ................................................................................................. 6
   XPand6 ...................................................................................................................... 6
   Fornax-Platform ....................................................................................................... 6
3. State of the art ............................................................................................................. 7
   3.1 How others have solved this problem ................................................................... 7
   Sculptor ...................................................................................................................... 7
   Gentleware ............................................................................................................... 8
   OOWS Suite ............................................................................................................. 8
4. Towards Model Driven Optimization ........................................................................... 9
   4.1 Introduction .......................................................................................................... 9
   4.2 Which role does MDA have in this thesis ........................................................... 10
   4.3 Analysis of the target platform ......................................................................... 11
   4.4 Generating the code ............................................................................................ 12
5. Evaluation .................................................................................................................. 15
   5.1 Introduction .......................................................................................................... 15
   5.2 Training evaluation ............................................................................................ 16
6. Conclusion and future work .................................................................................... 19
   6.1 Conclusion ........................................................................................................... 19
   6.2 Future work ........................................................................................................ 19
List of figures and tables ............................................................................................ 21
References .................................................................................................................................22
Appendices ..................................................................................................................................23
  A. Installation guide.................................................................................................................23
  B. How to work with the tools...............................................................................................24
  B.1 Introduction.......................................................................................................................24
  B.2 What we have been chosen...............................................................................................24
  B.3 Getting started with the model and the openarchitectureware.......................................24
  B.4 Generating the code .........................................................................................................25
1. Introduction

Since the beginning of computers are searching techniques to improve quality and reduce costs of software solutions. Engineering software is a discipline of computing which provides techniques to develop and maintain quality software.

Optimization is a branch of software engineering that tries to improve a system to work more efficiently and with fewer resources. A program can be optimized to run faster, with fewer memory requirements and using resources efficiently.

Optimization is a “complex” technique that improves the performance of application, which also means that introduces extra work to the programmer and modifies the code to do so, in most cases, more complicated. It is important to find a way to optimize, in which we improve the system is response, without increasing costs, as the programmer time consuming.

Model-Driven Architecture (MDA) is an approach to model-based software development. The aim of MDA is to increase the quality and speed of system development by raising the level of abstraction, using modelling techniques, model transformation, and code generation.

MDA optimize software development, making it possible for the developer to focus on functionality of the application and system behaviour, rather than on the technology.

The use of models in the development process software enables to enhance the reuse of software elements at a high level of abstraction. The transformation of the models in source code improves the quality software developed.

Furthermore this higher quality software generated is easy to maintain. This will decrease the time that the developers will expend developing or in posterior stages of the life cycle of the software modifying it.

Moreover we are going to use adaptive code that is going to reduce the execution time of the application. This adaptive code is able to change its behaviour in function of some intern or extern parameters.

1.1 Goal problem

The goal of the project is a proposal of code optimization, trying to reduce development and execution time.

The previous design has to predict almost all unexpected problems we can find in development time or later, in implantation time, it should be easy to add new requirements.

The objective in a project should be building an application without errors, easy to modify and add new requirements. In software development it is important too the possibility to reuse code, so that it is not necessary rewrite code in other applications. All of these programming “good practices” will make the project stable, easy and cheapest.

The final solution should make a code run faster or at least as fast as using the combination of data structure and implementation that is having a better computational cost. The solution will grant this because it is choosing in every step the combination of data structure and implementation that is best in each case.

The goal of our project is to build an application, based on a model that will be able to solve several problems and adapt itself with a minimal development cost. We will achieve, in any case, minimize the execution time of them.
1.2 Goal criteria / Objective
We will achieve the code optimization basing development in models, with automatic code generation.

It is important not to add extra complexity to the application instead of some kind of improvement. We will provide a technique to generate code from model so that will decrease development time, not to complicate the resulting code.

The challenge that we are going to face is combining this knowledge about model driven architecture optimization with the adaptive library. This means that we are going to create this adaptive library using adaptive code, which is going to change depending on the concrete problem that we are dealing with.

An adaptive code is able to change its behaviour in function of some intern or extern parameters. Current software development approaches specify the functionality of the system at design time.

1.3 Motivation
The objective of the thesis is making a high maintainable and very easy extensible adaptive library that make the whole application run faster in runtime.

Using this adaptive library is going to decrease the developing time since modelling is more important than programming. The developer just needs to change some parts of the UML model and program the concrete part of the problem to solve. Furthermore is going to create a solution that is having a higher quality since is more maintainable and reusable.

However the fact that the execution is going to be probably faster than the execution without using this adaptive library is giving to this library another reason for being implemented.

First of all we have a simple way to obtain the problem implementations code: auto-generation from the UML schema. The code maintenance is easier, because is not necessary to modify the code directly.

Furthermore, we obtain the fast way to execute a problem.

This time execution improvement is because of in runtime the application have been able to choose which is the algorithm that combined with one specific representation of the data structure is running faster.

1.4 Structure of thesis
The objective of the thesis is combining the benefits of the model driven architecture with the adaptive library that we designed. In the section we wrote about how model driven architecture is making a step further in the ancient problem of the optimization; after in the section 3 are explained all the concepts needed for understanding the thesis. In section 4 we deal with how other people solved this problem or similar problems using the same kind of technologies. In the section 5 we explained how we solved the problem combining our knowledge about model driven architectures and our adaptive library. Finally we are going to show the conclusions and the future work in the section 6.
2. Basic definitions and notion

The main goal of this chapter is explain the basic nomenclature and some basics explanations for understanding the technical concepts of this thesis.

2.1 Background

One of the main aims of the model driven optimization is making independent the design from the architecture. Usually the model is designed and followed afterwards for the developers in order to build the system. Frequently in this approach saying that the developer is using the model is not the best way to describe it because what the designer are actually designing are case diagrams; we should use model for a higher levels of abstraction. This way of developing software is probably going to give to the final solution a lower level of software quality.

The approach that we are following for building this adaptive library is much more code centric. This means that the model is the code although that in every concrete problem that the adaptive library is going to face is going to be necessary have some parts where the code is going to be develop handwriting it.

This is going to give to the solution a highest software quality. In this section is going to be possible find some basic information about software quality. However basing the design in the model is giving some disadvantages like the time that the designers need at the beginning of the project.

Figure 2.1, comparison between MDSD and RUP.

In the figure 2.1 we can see a comparison between MDSD and RUP, different approaches for developing software.
2.2 General definitions

Adaptive code
An adaptive code is able to change its behaviour in function of some intern or extern parameters.
Normally a problem can be resolve of many ways. That is it, one problem several implementations. Of those different implementations, one use to be faster than the others depending of a lot of variants.

Automatic code generation
A lot of problems have common code; it is useful to share this code, and find the way to execute them with the same application. That makes easier the code maintenance and the reuse of those programs.

2.3 Adaptive code

Reusability of the code
It is the likelihood a segment of source code can be used again to add new functionalities with slight or no modification. Reusable modules and classes reduce implementation time, increase the likelihood that prior testing and use has eliminated bugs and localizes code modifications when a change in implementation is required.

Maintainability of the code
It is the ease with which a software product can be modified in order to correct defects, meet new requirements, make future maintenance easier, or cope with a changed environment.

Portability
It is the ease with which a system or component can be transferred from one hardware or software environment to another [IEEE 90].

Interoperability
It is the ability of two or more systems or components to exchange information and to use the information that has been exchanged [IEEE].

2.4 Specific definitions

MDSD (Model Driven Software Development)
It refers to the systematic use of models as primary engineering artifacts throughout the engineering lifecycle. MDSD can be applied to software, system, and data engineering. Models are considered as first class entities.

OMG
OMG™ is an international, open membership, not-for-profit computer industry consortium. OMG Task Forces develop enterprise integration standards for a wide range of technologies, and an even wider range of industries. OMG’s modeling
standards enable powerful visual design, execution and maintenance of software and other processes.

**MDA (Model Driven Architecture)**
OMG’s Model Driven Architecture (MDA) provides an open, vendor-neutral approach to the challenge of business and technology change. Based on OMG’s established standards, the MDA separates business and application logic from underlying platform technology. Platform independent models of an application or integrates systems business functionality and behaviour, built using UML and the other associated OMG modeling standards, can be realized through the MDA on virtually any platform, open or proprietary, including Web Services, .NET, CORBA, J2EE, and others. These platform-independent models document the business functionality and behavior of an application separate from the technology-specific code that implements it, insulating the core of the application from technology and its relentless churn cycle while enabling interoperability both within and across platform boundaries. No longer tied to each other, the business and technical aspects of an application or integrated system can each evolve at its own pace – business logic responding to business need, and technology taking advantage of new developments – as the business requires.

**CIM (Computer-Independent Model)**
It represents the Computationally-Independent Model that characterizes the domain of the problem. Such models arise primarily on modeling business processes and ideally conceived before making requirements for a particular application.

**PIM (Platform-Independent Models)**
Represents models that describe a software solution that does not contain details of the concrete platform on which the solution will be implemented. These models are the result of the analysis and design.

**PSM (Platform-Specific Models)**
Models that are derived from PIM, that contain details about platform or technology with which is the solution implemented. These models are constructed between designs and coding.

**Stereotype**
It is an extension mechanism that defines a new and more specialized element of the model based on an existing element. Stereotype can be based on all types of elements including classes, packages, components, and notes, as well as such relationships as associations, generalizations, and dependencies.

2.5 Tools definitions

**oAW (openArchitectureWare)**
oAW is a modular MDA/MDD generator framework implemented in Java(TM). It supports parsing of arbitrary models, and a language family to check and transform models as well as generate code based on them. Supporting editors are based on the Eclipse platform.
MagicDraw
MagicDraw is a visual UML modelling and CASE tool with teamwork support. Designed for Business Analysts, Software Analysts, Programmers, QA Engineers, and Documentation Writers, this dynamic and versatile development tool facilitates analysis and design of Object Oriented (OO) systems and databases.

MagicDraw supports MDA/MDD development with a sophisticated UML 2 platform with interfaces to major MDA tools. In addition, MagicDraw has an Open API to create your own custom models, or use our export to EMF (Eclipse Modelling Framework to use Eclipse’s powerful model transformation tools).

Generator-Workflow
It is defined in generator.oaw. It describes workflow of code generation.

XPand
The openArchitectureWare framework contains a special language called Xpand that is used in templates to control the output generation.

Templates are stored in files with the extension xpt.

Fornax-Platform
The Fornax-Platform is an Open Platform to develop components and tools related to the Model-Driven-Software-Development - short MDSD. Based on the powerful openArchitectureWare Generator Framework we provide a great number of components. We strongly recommend check the webpage for upgrades of this platform where new tools for MDSD are being constantly developed.
3. State of the art

So far the uses of models merely serve as documentation or as a guide for implementation. Involve the design with software development, so that we obtain source code from the model, leads us to achieve a high level of abstraction, so that makes it easy portability, interoperability and reusability.

MDA (Model Driven Architecture) based in OMG’s standards, separates business and application logic from underlying platform technology. It is based on the principles of abstraction, automation and standardization.

Models are divided in Platform-Independent Models (PIM) and Platform-Specific Models (PSM). MDA transforms PIM, higher level of abstraction, in one or more PSM, from which it is possible to generate the source code of the system.

MDA is the answer to two problems in software development:

- The diversity of platforms and technologies, and
- The rapid technological developments.

The aim of MDA and its related concept is to increase the quality and speed of system development by raising the level of abstraction, using modelling techniques, model transformation, and code generation.

The core concept of MDA is to formulate the business logic in a PIM that uses stereotypes such that elements in the model are explicitly related to the architecture concepts of the system. Platform-Specific properties and the mapping from architecture concepts to the technical realization come in via model transformation and code generation steps.

However we need a UML tool in order to build the model, we used Magic Draw tool. This tool is allowing us to export the model to EMF UML2 XMI, which is the support that we need to import it with the Eclipse framework.

Once we have the model imported in the Eclipse we can start to define model transformations and finally generate the code from the model. The openarchitectureware tool is the one that we used in order to make this model transformations and code generation.

3.1 How others have solved this problem

There are many solutions that use MDA to generate automatic code from the model. In general, these solutions that we have found focus the goal criteria in optimize the development time, building almost the entire application with automatic code generation.

These are some of the solutions that we found:

Sculptor

Is an open source tool that provides a "best practice design" for Hibernate and Spring. It purposes is to improve developer productivity and quality. Uses oAW to parse the DSL and generate high quality Java code.

Using sculptor the developer can focus on the business domain instead of technical details. Also, reduces the amount of hand-written code with 50% compared to a manually coded application with a similar design. It raises the level of abstraction and automates a lot of otherwise repetitive manual coding.
The main goal criterion is to reduce the development time, automating part of the code based on the model.

**Gentleware**

It is a modeling tool manufacturer and expert service provider for model-driven software development.

The goal criterion is focus the development process in the design time.

Transforming the model in business, so the programmer does not have to know technical details, focus in business logical.

**OOWS Suite**

Is a development platform for web applications based in MDA. It provides a set of tools to support the development process to generate web applications automatically.

The goal criterion is generating web applications automatically, saving time to the developer.
4. Towards Model Driven Optimization

In this chapter we explain the essence of the problem, how we approached the problem to arrive to a solution that reaches our goals. And also, how the adaptive library and the model driven architecture get together, complementing each other, although is not so obvious how to do it.

4.1 Introduction

In order to solve any problem we can choose different ways to implement it and more over we can chose between different ways on representing the data structure. Normally, for choosing which implementation or representation of the data structure we need, we will think in the usual size of the data structure that we will have to face and which algorithm is going to run faster knowing this info and knowing the computational cost of this algorithm in average.

The problem that we will face is choosing the correct representation of the data structure and the algorithm in every single stage in the running time. It means that for every single step the application will choose the representation of the data structure and the algorithm that will make run it faster.

The first problem that we will have to face is knowing, in runtime, which is the implementation that is going to run faster in the concrete conditions of this executing moment. The most important factor for deciding which is the best implementation for this concrete executing moment is going to be the size of the data structure. For being able to choose in runtime we are going to use a table that contains, for every size, which is the best implementation to use, that in training time we build.

However, the actual problem is in building this table. To build this table we have to check which implementation is faster. We do this in a training phase. In this phase, every implementation is executed and the faster one is noted in the table. If we can run every implementation once at the beginning later we only have to use the faster implementation.

The table will look like:

<table>
<thead>
<tr>
<th>Size</th>
<th>Best Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 100</td>
<td>Impl1</td>
</tr>
<tr>
<td>100 - 200</td>
<td>Impl1</td>
</tr>
<tr>
<td>200 - 300</td>
<td>Impl2</td>
</tr>
<tr>
<td>300 - 400</td>
<td>Impl2</td>
</tr>
</tbody>
</table>

Table 4.1 Size based dispatch table

With this information we are going to fill this table in runtime and this table it is going to make that the whole application run faster. It means that the extra time that we are using to fill the table should not be big comparing to the time that the applications expends to run.

Once we have this we will start to face the second part, choosing the best representation of the data structure. Now we have to train for every algorithm implementation that we have with every representation of the data structure that is compatible with this implementation.
This result of this new training is going to be the definitive dispatch table, which is going to look like this:

<table>
<thead>
<tr>
<th>Size</th>
<th>Data structure</th>
<th>Implementation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D1</td>
<td>I1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>D2</td>
<td>I2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2 Size – Data structure based dispatch table

The last thing before considering which combination of data structure representation and algorithm we should choose is the cost, in case we need to do it, of changing from one data representation to the another one. For this we will have another table where we will find the cost of changing to one data structure to another.

<table>
<thead>
<tr>
<th>Size</th>
<th>Data structure</th>
<th>Implementation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D1</td>
<td>I1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>D2</td>
<td>I2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3 Final dispatch table generated.

To sum up in run time we will have this two tables, one giving us the result of which is the combination of data structure representation and algorithm that are going to run faster under this concrete size of the data structure and another one with the cost of changing from one data structure representation to another. With all this information the application is going to be able to choose the best option to improve the run time.

4.2 Which role does MDA have in this thesis

However this solution should be integrated in the adaptive library that we have been modeling. The first model that we take from the lesson of the subject Selected topics in compilers, was the next one.
This design was made only thinking in choosing the best implementation not taking a step farther where the different data structures are taking an important role in the solution. This is why we designed a new model based in this one.

What we are going to do now is generating code from this model. To do this we are going to use model driven architecture techniques. Modelling instead of programming is going to give us a higher abstraction level than the code has. In addition we are not going to be bound to a technology so we will be able to change to another platform.

This is going to give us, once we now how to use it, a faster time for developing because of this higher abstraction level and a higher quality of the code.

Now we are going to analyze the basic steps that of Model-Driven Software Development.

### 4.3 Analysis of the target platform

We will base the development on a platform independent model, which represents a conceptual design realizing the functional requirements being independent of the technologies and software architectures changes.

Our project uses an automatic code generation tool that generates classes from a UML model. That is it, add a new problem solution to the project will be to add to the UML schema the new classes that describe the problem, and we only need to include in hand-code the different solution implementations.

In order to design this new model we first questioned which is application parts can be automatically generated from the model. The code that we can generate is the schematic code that is going to be different for every application but structurally is going to be the same.

The part that we can automatically generate is the more abstract, is the part that is not affected about which kind of algorithm we are going to work with or which kind of data structures we will use, this is the generic code of the adaptive library.
These ones are the classes and the methods and the relations between them that are generic. Only the method (execute) is going to change the parameters depending on the problem that the library is facing. When the parameters are changed and the code is generated all the calls to this method will be automatically generated, saving time to the developer and forcing this person to model instead of programming.

Following the example that we are using we will see now how looks like the model when we are having a couple of concrete data structures and algorithm implementations.

### 4.4 Generating the code

Once the model is representing the problem to solve, the abstract classes that are representing the concrete algorithm and the concrete data structures are modelled; we can proceed to export the model to EMF UML2 XMI.

This exported model is easy importable with Eclipse. Once imported in the Eclipse frameworks is possible transform the model and generate code based on them with the openArchitectureWare model-driven software development tool.

For transforming the model to the code we decided to use the Fornax-Platform in order to take profit from the use of the Xpand templates that they already made. With these templates we generated the code.
In figures number 4.4 and 4.5 we can see the process of generating the code and in figure number 4.6 the result with the java files generated.

The part of the problem that is not automatically generated and that is belonging to the individual code are in different files for avoiding the use of protected regions, which where creating some problems in the earlier parts of the project.
Figure 4.7 Detail of the concrete and generate code
5. Evaluation

In this chapter we analyze the solution that we implemented and also test it in order to know if we reached the goals that we specified in the section number one.

This chapter is divided in two parts, the first one measures how difficult is change the problem that the adaptive library have to face, while the second part is focused in how much speed up we get using the adaptive library facing the concrete example that we have choose.

5.1 Introduction

One of the most important goals is reducing the developing time drastically. Its one of the goals that is more difficult to measure, we are going to show how to adapt the library to be able to face another problem.

The first step a one of the more important, since it is showing how important is the model, is to open the modeling tool and import the project. Once we have the model the execute method should be modified with the parameters that the concrete algorithm need. It is important to say that the parameter size is understood that always is going to be needed so is not necessary to put it as a parameter although is not problematic putting it.

![Figure 5.1 Detail of the execute operation in the model](image)

The next step is exporting the model to EMF UML2 XMI in the model folder of the project.

Once we have the model correctly exported and we should generate the code from the model. We are going to do this from Eclipse, executing the file oaw/generator.oaw as oAW Workflow.

In the folder src/Problem specific should be copied the code there the concrete algorithm that are going to be used, those concrete algorithm should extend the abstract algorithm class, from the src-gen. In the same folder should be copied the generator files, typically one for every data structure and they should extend the abstract class generator that have been generated and is in the folder src-gen.

The basics for running the training are done, is only needed some set up in the trainImplementation java file.

```java
public void testTraining() {

    //Creating the instances of the DataStructures
    AdjacencyMatrix adjMat = new AdjacencyMatrix();
    NodeList listaNodos = new NodeList();

```
//Creating the instances of the Generator
Generator gen = new GenAdjacencyMatrix(new MyBoolean());
Generator gen2 = new GenNodeList(new MyBoolean());

//Association between Generator and dataStrucutre
gen.setDataStrucutre(adjMat);
gen2.setDataStructureImpl(adjMat); 

//Association between Data Structure and generator
adjMat.setGenerator(gen);
listaNodos.setGenerator(gen2);

//Adding the instances of the data structure to the list
adjMat.addDataStructureImpl(
   new DataStructure[]{adjMat,listaNodos});
listaNodos.addDataStructureImpl(
    new DataStructure[]{adjMat,listaNodos});

// Training data structures
bestComb = adjMat.trainDataStructures();
}

The last step for executing the train is run as Junit Test. With this all the training is going to be done.

Figure 5.2 How to run the application

The process for changing which problem the adaptive library is going to face is quite simple and is showing how highly easy adaptable the library is.

5.2 Training evaluation
The final solution that we are having is easily adaptable to face the different problems that are following the requirements for being able to take benefit out of this library. This means that problems that can be solve using different algorithm, probably supported by different data structures, that the efficiency of the algorithm in combination with the data structure is different when the size of the data structure is changing.

The example that we chose for showing how the adaptive library is working is the graph reachability in directed graph. We are having two data structures for representing the graph. One of them is using an adjacency matrix for representing it and the other one is using a linked list.

The algorithms that the first of the implementations is using are searching the nodes following a strategy in depth and width, while the second data structure is making a width search in the linked list.

With those two data structures and the three algorithms we are going to evaluate the benefits of the training in this concrete example.

Before starting to comment the test is important to talk about the nature of this kind of algorithm. For searching reachability between to nodes in a graph we generated randomly the graph, which is going to be the same in both data structures for each
execution and for the nodes that should be checked if there is reachability we decided to use the first and the last node in the graph. One of the problems that we think that we can have for analysing the results of the training is the possibility that one execution is going to be done so fast because the two nodes are directly connected and in the next size its possible that the nodes are not connected at all and the execution time is going to be much more bigger.

We are going have six variables to follow the track, which are the three algorithm plus the conversions from one data structure to the other one. In the legend of the plot we can see the six combinations.

As we can see in the plot there are two very different regions. The ones that are having the highest time cost are the ones corresponding to the adjacency matrix data structure with the algorithm using the strategy of the search in depth after changing the data structure from the linked list, the other one is corresponding to the node list data structure after changing the data structure from the adjacency matrix.

In order to analyse the results with the other four variants we need another plot.
In the figure 5.4 we have the detail to compare the three algorithms that have not been changing from one data structure to another one. The adjacency matrix with the strategy of search based in depth is growing following an exponential progression. The best option is between adjacency matrix with the strategy of search based in width and the node list. Almost for all the values the node list is better, which is having sense knowing the implementation that is using with java generic data types.

In this concrete example the benefit of using the training for having a final speed up in the moment of running the application is difficult to see because of the best option in almost every single case is using the data structure with linked list with the only algorithm that is implemented for this data structure. The adaptive library is going to choose always this implementation that is the same as running the problem with only one data structure and implementation.

However for other kind of problems or even for the same problem using another implementations, specially using data structures where the time for changing from one data structure to another one is not going to be O(size), is going to give a final speed up.

To sum up this example is showing us when this adaptive library is going to be useful for speeding up the solution. Those parameters are:

- Temporal cost of changing from one data structure to the other ones
- Comparing the theoretical time cost of the algorithms

The optimization in execution time that this adaptive library is bringing is because of the adaptive code that is using depending from the results that in training time. This means that the library is choosing the best option depending from some parameters but is not optimizing the data structures or the algorithms, so if the data structures in combination with the algorithms implemented for solving the problem are having a huge time cost, the solution is going to choose the combination with less cost, in other words; the limit in the speed up is the limit that is giving the combination of data structure and algorithm better for this execution moment, depending of the size of the data structure and which is the data structure given.
6. Conclusion and future work

In this chapter are explained the benefits of using this adaptive library that we designed and implemented but as well is debated the impact that the model driven architecture can impact in the software industry.

In the future work we purpose some two different ways for upgrading this adaptive library, one is about the multicore revolution that is happening nowadays and the other one is about adapting this library in a online environment.

6.1 Conclusion

The benefits of using a model driven software developing are clear, and the software industry should take advantage of this knowledge as soon as possible.

However, training is needed for many developers that nowadays are not used to modeling. They are instead more used to handwriting code and having the model only as a reference that they should follow, but not as most important part in the project. For following this new conception of developing software is going to be needed effort for part the software industry and as well from the universities where usually the technologies are growing faster than capacity of change of the enterprises or the universities can afford.

Furthermore this new approach for developing software is taking a lot time in the first steps of the project. Our personal experience in other projects following another kind of software developing methods as for example RUP, is showing to us that the beginnings with MDSD can be hard and disappointing, specially because not so many people is choosing this way of developing and this make that the tools are not very mature although are improving so fast and more amore people are behind this projects.

However seeing the result of the thesis and the lots of new concepts that we learned we really believe in that the values that are behind the model software driven development are the next step that the software engineering is going to take because the benefits that the whole project is taking out of it are bigger than the initial effort that the developing team have to do at the beginning. The quality of the software is progressing one step further and the time for the developing applications is going to be substantially decreased.

The adaptive library that we designed with the example that we took is showing that can give to the final solution a speed up but sometimes is not worth it only because of improvement in the execution time as we saw in the evaluation. However the adaptive library is giving the chance to decrease the developing time of the application, since it is easy to change to use in another problem and the concrete modifications that should be done in the code are so small.

6.2 Future work

The multicore revolution that we are experiencing nowadays is giving the opportunity to run faster our applications but also this constitutes a challenge for programmers. Multicore architectures involve multiprocessing, and to take advantage of that, parallel programming is needed.

Using these processors in an efficient way it is going to be important. Depending the number of processors available one implementation is going to be faster than another, these is because there are implementations more parallelizable than another’s. These implementations that can be more parallelized are going to faster when we can use a
bigger number of processors. If we are able to choose the implementation that we are going to use in runtime knowing the number of processors available in runtime the whole application is going to run faster.

An extension of this project could be extending the test that are being done in training time for knowing in runtime which is the optimal number of processors needed for running faster the application.

Another way for going further with the work is rethink the basics of this model and make it adaptable for on-line environment. This means that the different data structures and algorithm would be added in runtime and the training is going to be recalculated depending the new combinations of data structure and algorithm that have been added or deleted from the system.
List of figures and tables

TABLE 4.1 SIZE BASED DISPATCH TABLE ........................................................................................................... 9
TABLE 4.2 SIZE - DATA STRUCTURE BASED DISPATCH TABLE ........................................................................ 10
TABLE 4.3 FINAL DISPATCH TABLE GENERATED .................................................................................................. 10

FIGURE 2.1, COMPARISON BETWEEN MDSD AND RUP ........................................................................................ 3
FIGURE 4.2 PREVIOUS DESIGN ............................................................................................................................. 11
FIGURE 4.2 UML MODEL ........................................................................................................................................ 12
FIGURE 4.3 DETAIL OF THE MODEL FILES ............................................................................................................ 12
FIGURE 4.4 DETAIL OF THE TEMPLATES USED ....................................................................................................... 13
FIGURE 4.5 DETAIL OF THE CONSOLE GENERATING .......................................................................................... 13
FIGURE 4.6 JAVA FILES GENERATED .................................................................................................................... 13
FIGURE 4.7 DETAIL OF THE CONCRETE AND GENERATE CODE ......................................................................... 14
FIGURE 5.1 DETAIL OF THE EXECUTE OPERATION IN THE MODEL ................................................................... 15
FIGURE 5.2 HOW TO RUN THE APPLICATION .................................................................................................. 16
FIGURE 5.3 PLOT WITH THE RESULTS OF EXECUTIONS WITH CHANGES IN DATASTRUCTURE .... 17
FIGURE 5.4 PLOT WITH THE RESULTS OF EXECUTIONS ...................................................................................... 18
References

Appendices

A. Installation guide

1. Install Eclipse OpenArchitectureWare modular MDSD generator framework
   - Downloads of the openArchitectureWare MDSD generator framework integrated into a complete Eclipse distribution as a courtesy to the oAW community.

2. Install Magic Draw
   - Version must be Magic Draw community edition 14.0 or later version (needed on-line registration)

3. Set the used JDK on 5.0 or 6.0 on Eclipse
   - In file properties java compiler
   - This JDK must be installed in your machine

4. Import the project
   - File import projects into the workspace
   - Browse to the project location

5. The UML model
   - Open Magic Draw
   - File open project
     i. /model/myAdaptative.xml
B. How to work with the tools

B.1 Introduction
This how to pretends to be a guide for being able to work with this adaptive library and not getting lost with a way of working that probability is not familiar.

In this document we want also to show the advantages and disadvantages of working following the model driven software development approach, which since is a quite new approach of developing software is not that well documented as others are. Furthermore the versions of the software required are changing so fast and there is no very well know framework to work with.

B.2 What we have been chosen
We chose magic draw for modelling because of the change to export the model to EMF UML2 XMI. This means that we are going to be able to import the model from Eclipse and moreover transform from model to model and finally generating code from the model. Probably there are more modelling software solutions for modelling and we recommend before starting make some research about which alternative is possible to choose.

The problems that we had with magic draw especially have to do with thin line that the model driven architecture approach is separating the design from the architecture. This is having lots of advantages, we strongly believe that the software development is going in this direction, even if the tools that we are having nowadays are not very mature.

The eclipse framework is giving to us all the power that we knew before plus all the benefits of having integrated all the bundle of technologies for being able to go further with MDSD. These technologies are openarchitectureware, EMF, GMF and UML2. We had so many problems trying to fit all this plug-ins in the normal version of eclipse, finally we found this version which is having all those technologies already installed and have been much more easier until that moment.

All this technologies are improving so much and so fast, specially the openarchitectureware that as a MDA/MDD generator framework is changing the version so fast. Its important to be in the last version of this concrete technologies because in every version they are fixing so many things that are really making the model transformation and the code generation from the model much more easier. Also all the community that is behind are high qualified professionals which means that they are always following the last updates and if you want to be in you should do it too otherwise no one is going to face the same problems that probably you will have because you are working in another version, and probably this problem have been already fixed.

B.3 Getting started with the model and the openarchitectureware
The first steps for understand the project from a practical viewpoint is opening the model with the magic draw.

In order to open the project in the magic draw go to file open and /model/myAdaptative.xml.
The model is not so different from the model that we studied in the subject selected topics in compilers although there are some changes in order to be able to face the new problem where the thesis is focusing. These changes as we explained in the main document are made for being able to handle having different data structures that are having one or more different concrete algorithm implementations.

The next step is exporting the model to EMF UML2 XMI. For avoiding posteriors problems the model should be exported to the folder called model in the folder where your project is inside your workspace.

Once the exportation process is done, the project is already updated in the project in Eclipse and is ready to be transformed to another models or generating the code.

We experienced some delay between the export process is done and when the Eclipse is updating the new model. Be sure that Eclipse updated yet the model checking the generated files, if is not updated is needed to generate again.

**B.4 Generating the code**

For being able to transform the model into another model or generating the code from the model we are using the oaw generator framework with the huge advantage that is integrated in Eclipse.

Researching what the opeanarchitecture community have been doing we found the Fornax-Platform, which is an Open Platform to develop components and tools related to the Model-Driven-Software-Development. We used some of the templates and extensions of one of the cartridges that they are offering to the community, specially the Java Basics cartridge. After we had to change some of the configuration parameters and adapt some of the templates to our specific problem but it have been really helpful.
For understanding how it works we strongly recommend reading the oaw documentation, which is having different tutorials, even that not all off them are in the UML approach, and some useful video-tutorials about the 4.3 version. One of the Xpand files that is having a very important role is the Root.xpt.

```
«DEFINE Root FOR uml::Model»
«EXPAND Report::Root FOR this»
«EXPAND Class::class FOREACH allOwnedElements().typeSelect(uml::Class)
«ENDDEFINE»
```

In this peace of code is important realize that we are starting to go through the model. We are going to “expand” every uml class that is in the model. This means that the xpand file Class.xpt is going to expand every class that is finding in the model.

```
«EXPAND Generalization::generalization»
«EXPAND Realization::realization»
«EXPAND Attribute::attribute»
«EXPAND Association::attribute»
«EXPAND Attribute::staticAttribute»
«EXPAND Attribute::getterSetter»
«EXPAND Operation::operation»
«EXPAND Generalization::operation»
«EXPAND Association::accessors»
```

From the Class.xpt is going to expand every attribute, association, operation etc that the class that we are dealing with needs. This also means that we are basically changing from the model to the code in one single step even if we are adapting the model to a concrete platform that is Java.
The operation class is one of the xpand files that we have been adapting to our adaptive library. In this piece of code we can see that the parameters of the operation `executeTraining` are being modified in generation time depending of the parameters of the `execute` operation. The parameters of the `execute` operation must be done in the model using the parameters that the concrete algorithm need and later on this is going to be generated in other parts of the code. This parts are the methods `execute performance aware` and the `execute training`.

For starting all this process from the Eclipse is needed run as oAW Workflow the `generator.oaw` in the folder oaw/validator.

```xtext
implVariant.executeTraining(
  «FOREACH class.getAllOperations().select(ele.name == "execute") AS p»
  «FOREACH p.ownedParameter.select(ele.name != "res" && e.direction != "return") AS o SEPARATOR "," » «o.name»«ENDFOREACH»
  «ENDFOREACH», size);
```