Adaptive code based in Aspect Oriented Programming Paradigm

Noel Galindo
Abstract:
The main objective of this work is to collaborate with the Service Composition Tool investigation in the School of Mathematics and Systems Engineering technology at Växjö University. The school is focusing its efforts in the software engineering technology. Presently one of the most promising ways to improve the software development is to look at various possible solutions to create advanced systems, based in adaptive code theory, that are able to make decisions without the aid of a human user. These advanced systems will be able to change their behavior in function of several extern parameters or conditions. We will collaborate to that target, investigating the Aspect oriented Programming paradigm as a possible tool, implementing an adaptive code application using different technologies. We will then compare them so as to determine which one could help us create this kind of systems.

Key words: Software engineering technology, adaptive code, aspect oriented programming paradigm.
Table of content:

1 Introduction 1
2 Problem statement 3
3 Project Structure 4
  3.1 Process 4
4 Basic Definitions 6
  4.1 Aspect Oriented Programming (AOP) 6
  4.2 What is an Aspect? 7
  4.2 Important Concepts:
    4.2.1 JoinPoint: 8
    4.2.2 Advice: 8
    4.2.3 PointCut: 9
    4.2.4 Introduction: 9
    4.2.5 Target: 9
    4.2.6 Proxy: 9
    4.2.7 Weaving: 9
  4.3 Weaving Types 10
    4.3.1 Static weaving: 10
    4.3.2 Dynamic Weaving: 10
  4.4 Aspects Languages 10
    4.4.1 Spring: 11
    4.4.2 Aspect PHP & Aspect C: 11
    4.4.3 COOL: 11
    4.4.4 More Java Languages AOP: 11
  4.5 Aspect J Technology 11
    4.5.1 Advice types: 11
    4.5.2 Advice before: 11
    4.5.3 Advice after: 12
    4.5.4 Advice catches: 12
    4.5.5 Advice finally: 12
    4.5.6 Advice around: 12
    4.5.7 Hello World Aspect J Example: 12
    4.5.8 Logging Aspect Example: 13
  4.6 Aspect Werkz Technology 13
    4.6.1 HelloWorld Aspect J Example: 15
5. Adaptive Sort 17
  5.1 The Sort Example 17
    5.1.1 Data Structures: 17
    5.1.2 Algorithms: 17
    5.1.3 Size of data and data properties: 17
    5.1.4 Dispatch Table: 18
    5.1.5 Training Learning: 18
    5.1.6 Our Work: 18
    5.1.7 General Schema 18
  5.2 Implementing the First probes 19
    5.2.1 Example 1: The aspect decides which part of code will be executed: 19
    5.2.2 Example 2: The aspect asks the dispatch table to take a decision: 21
1 Introduction

Despite the software engineering achieving its fortieth birthday, it is still relatively a new discipline. The ideas that are in development now could even change the whole perspective in years to come. As an initially chaotic area, with each programmer creating the application using their own criterion, software engineering has become one of the most conflicted areas in the computer science investigation. This uncoordinated way of working made it difficult to reuse codes already programmed, or to improve applications that were becoming obsolete, or fix problems in systems that another programmer developed before when they were not working properly. Having said this, we can be proud of the work already done; creating standards and paradigms that led us in the right direction towards developing applications and systems. The progress made is just that – progress, it has not yet become the final solution. Other areas of computer science have a regular evolution, whilst software engineering, although young, is not evolving as simply as we would wish. There is a long way to go, and a lot of work to do before flawlessness can be achieved. Fortunately universities and various corporations recognize this fact and are willing to spend time and effort investigating and developing new standards, paradigms and ideas.

One of the most interesting ideas developed during last few years is the one called Aspect Oriented Programming Paradigm (AOP). Although based upon the famous and world wide spread Object Oriented Programming Paradigm (OOP), this new concept has some new interesting characteristics that we cannot find in the regular object programming. The promising thing about this new concept is that the AOP allows the code reusability and the code modularization easily, and improves the normal OOP systems. This includes a new concept called Aspect, which is able to crosscut a regular OOP class during execution time, and lets us work with it without changing any line of code of the main application that we are working with. Probably and hopefully, it will be a powerful tool which will be able to help us create more advanced systems based on adaptive code, improving the efficiency of the application, and fully solving some of our actual software deficiencies.

The applications are becoming more and more complex every day, the systems are bigger each time and their functionalities grow with the increase in size; but not everything is as beautiful as we like to think, more capacities and functionalities means more decisions that the user should take every time the machine does not know how to continue working, or which is the best way to solve the task. Pressuring the user with futile questions, or repeatedly asking for extern data seems to be the wrong way to continue developing software. These machines were invented for them to calculate and work for us. The user time is more valuable than that of the machine time; they are, in a way, our ‘slaves’ that we wish to work quickly and efficiently without causing us bother. Therefore we should develop systems which allow this. We need to create systems that are complex enough to decide how to solve a problem, or how to execute a function in the best way possible. Why not create software which takes these decisions by itself in function of some criteria that we could teach them before?

“The adaptive code is an application able to change its own behavior in function of some extern parameters”. The adaptive software application uses information about the changes in its environment to improve its functionalities. The application can adapt its behavior by itself, without the user having to repeatedly and explicitly telling the application. We can find some regular examples really useful, in a procedure, for instance, which finds the best algorithm to execute a task in function of the different data structures that it has to work with. Or in web pages able to change its
representation in function of which kind of user is accessing. Or in the more complex, such as a dynamic distributed system able to change how to execute an algorithm or task in function of how many processors it can use in this moment, or how much free memory it can achieve by the execution.

The AOP paradigm is a recent idea developed during the last few years, meaning it is still a young concept, and so many companies are working hard to develop powerful technologies based in this abstraction. They wish to find a good place in the AOP developer’s desktop, and perhaps finally, they will get a standard based on their technology purpose. Some good examples are Aspect J, suggested by Xerox Corporation, or Aspect Werkz, developed by BEA Systems; but they are not the only ones. Obviously not all of the new technologies will survive after this process, so it will be interesting to discover the different purposes in the software that will be more promising in the development of the kind of application we are looking for. We need to analyze the best and worst aspects of each one, the best implementation should survive with time, hopefully not just the implementations that are supported and funded by big and powerful companies.

If we think about all of these criteria, the next obvious question is how can we help with this challenge? Which small part of our work could be valuable for the investigation that is being developed to solve and improve these kinds of systems? We should find the way to collaborate in this research. If we consider all the things explained before, we can decide that our target will be to investigate the adaptive systems possibilities and capacities using the AOP paradigm.

We want to work with the AOP concepts and investigate how good this paradigm can help to develop adaptive code able to adapt itself to the system necessities, in function of the data, the memory or the system capacities, in this exact moment. We want to know how good the AOP would be in implementing an application based on adaptive code; and decide how good this implementation is working. Also, it will be interesting try to implement the application using different technologies based on the paradigm, and compare both purposes focusing our efforts on the final product, in its behavior and in the whole process of the implementation of the systems. This includes the first documentation, the download process, and the installations of the technology in our machines, the compatibility of them with our chosen system, the developing difficulties and in the final application behavior.
2 Problem statement

This thesis’ problem statement will be to focus on various questions which we will solve during the process of the investigation. From the basics to the most advanced requisites, it is important to decide the bases of the investigation before trying to solve some of the more advanced level questions. If some of the first statements fail we will not be able to continue researching for the AOP capacities developing adaptive code:

“Is it possible to develop adaptive code application based in the Aspect Oriented Programming Paradigm (AOP)?”
“How may we use AOP to automatically include the adaptability code in the application?”
“Is the adaptability aspect possible using aspect Werkz?”
“Which are the important characteristics that we should analyze and take care with the different technology purposes?”
“Which technology should we choose in each different case?”

The first question will be solved implementing several adaptive code applications with Aspect J technology. We will explain the whole previous process to solve the second question. The third part will be solved developing the same examples using Aspect Werkz. The next step will be to decide the different important criteria that we should evaluate with each technology. Finally we will evaluate both technology purposes to decide which one is better.
3 Project Structure

In this chapter we will explain the process that we will follow to answer the questions before mentioned in the chapter 2.

3.1 Process

The first step in this project will be to investigate and learn the Aspect Oriented Programming paradigm. We will learn about the theory supporting the main idea, and read about how this standard is working. We will also learn and explain the basic definition and methods that the AOP uses to be able to work with OOP programs, crosscutting them without modifying any line of code of the main application. Also we will discover why this technology was developed in first case, and which characteristics are different from the traditional OOP. Our responsibility will be to explain the developing process of a basic application as much as we can, showing how the aspects work modifying the main application, and explaining how the new developer could start to work with these ideas.

After all the learning process about the Aspect Oriented Programming Paradigm we will focus our attention on the different aspect technologies we can find in the market, and we will choose the most promising of them to begin to work with, explaining why we have chosen them.

Aspect/J will be the first technology we will work with. It was developed for the Xerox Corporation, and is currently the most spread distribution based in aspect paradigm. With this technology we can do double the work. First we will use it like an AOP standard to start to work with, and try to demonstrate, if the adaptive code is possible when you base the application in aspects. If we can develop this system using aspect/J then we are sure to find that the investigation will continue in the same way. We will then know that to develop adaptive code is possible using AOP Paradigm.

The first examples will give to us some good information about how the basic application can be developed, now it is time to start to work with the big ideas, it is time to start work with our main example. The intriguing element of this application is in the sorting of various elements using different algorithms in function to the size of the data, the different data structures where the elements are stored, or in some of the capacities of the system where the application is working. Our part in this big problem is in trying to show that the AOP technology can be used to change the behavior of the application in function to all the extern parameters. Without modifying the main application and without the interactivity of the user, all the properties of the program will be stored in a data structure called ‘dispatch table’. Our task will be modifying the main application during execution time, in function of the data of the dispatch table. It is now the time to evaluate the application. Is the developed program working properly? Is it behaving like it shall for each example in each different case?

The last example we will work with will be a more complex version of this main application, the structure of the whole program will be the same than before, but in this case we will try to implement the aspect functionality introducing inductive theory. We will try to modify some objects of the program using recursion with the aspects, which could be a really ambitious challenge. However, it will open new doors in the AOP possibilities. Finally we will evaluate this purpose and check how good it is working, and if it is behaving as we expected.

The examples are already developed using Aspect/J technology, the next candidate to be investigated is the technology developed by BEA systems, called Aspect Werkz. It is one of the oldest standards based in AOP; some others companies based their
application on the ideas developed by BEA, even Aspect/J share some similitude with this standard.

Aspect Werkz will be the next candidate, and we will try to solve the same examples that we already explained in the case of Aspect/J, from the basics to the main application and implementation based in the recursive model.

Both purposes are already defined, implemented and checked. Now is the time to evaluate both of them in function to certain criteria that we should decide before use. Are both systems working properly? What reasons would a normal developer have in working with one technology and not the other? Reasons are varied and plenty, for instance, how big both applications are, or how difficult the implementation process is. How fast each application is, and which one is the faster? Is one application consistently faster or is it only quicker in certain aspects, and in which case, what aspects? To measure all of this we will have to experiment with the applications that we have already implemented and check how quickly and efficiently they work with different cases in each example.

Finally we will present and inform with reference to the whole process, from the first steps to the conclusion, focusing our attention on the final results, and suggesting some future work that will be interesting to work with.
4 Basic Definitions

The focus of this chapter is on explaining the theory of Aspect Oriented Programming, the basic concepts and important definitions. After which we will have a basic idea about how the AOP works and the reasons behind its development.

4.1 Aspect Oriented Programming (AOP)

The Object Oriented Programming was born out of the necessity to develop a more advanced system, and where the modularity and the maintenance were important problems to consider and overcome. It has been presented as a technology that could help with the software engineering architecture. The OOP allows us to abstract some of the important concepts and create classes which define these objects; we can give to them some properties as fields, and define some functions as functionalities. The inheritance concept allows us to reuse some code and add more meanings. This creates a patron that other classes could use and develop.

However the OOP is not the final solution to our questions. There are some problems where these techniques are not sufficient to clearly capture all the important design criteria that the application must implement. We can find a really good investigation about a number of these deficiencies in which the OOP fails during the support process. This explains why Aspect oriented Programming can be a good response to solve some of these bugs. This work is called “Aspect Oriented Programming” and it was presented during the European Conference on Object Oriented Programming (ECOOP) in Finland in June 1997. [30]

In the first years of the software engineering architecture the code was developed as a mix of all the different functionalities, without any kind of structure or control. The code was together with the data and the functionalities. This kind of programming where each programmer uses their own criteria to develop the application without any kind of modularity patterns was called “spaghetti code”. This was because all the items were mixed everywhere without any control or system, obviously this chaotic process was not the best method to develop software.

If we take a look at the history of software engineering we can discover that the most important progress was based on the idea of dividing the big problems into smaller, easier to solve, problems. Two important examples are the algorithm strategy called “divide and conquer” and the creation of the “Quick Sort algorithm”. We can realize that the next step in the software engineering, the OOP paradigm is just a decomposition of each different abstraction. It divided the program in function of different concepts that we will use to create objects and then work with all of them together to build the final application. It is one of the biggest advances in the last years of the software technology used to construct advanced programs and systems, using the decomposition principle. The OOP is more closed to the real life concept and is easier to implement and design. It introduces new functionalities and a new way to work with the data.

However this implementation does not work well with other aspects that we must consider, for example when we want to create such new functionalities as memory optimization code, security functions like error control and exception code; we have to modify some objects. We have to write and repeat the code many times, and the code can clearly then be affected each time we want to introduce a small change in several places in a large application already developed. We also must consider that the OOP techniques do not work very well in dividing the different basic functionalities of the systems, and this problem has negative consequence in the software quality.

In recent times the software technology investigation is working hard to create a new way to divide the systems: the Aspect Oriented Programming. This paradigm was
developed to increase the modularity skills of a program application, and it allows the separation of crosscutting concerns. The AOP paradigm lets us to work with an already developed application, adding some changes without having to modify any line of the main application.

The Aspect Oriented Programming (AOP) is a new programming paradigm created to make possible the separation between the different aspects previously mentioned. The paradigm tries to separate the components and the aspects. The final target is to implement an efficient and easy way to develop more complex systems.

The AOP is the next step of the Object Oriented Programming paradigm; it means that the AOP can work with objects and classes like OOP programming, with the procedure and the functional decomposition. We cannot, however, think of the AOP like an OOP extension as we can use the aspects in the different programming styles. Both technologies should live together, and we cannot think about AOP as the next successor, just an extra help to improve some of the OOP deficiencies.

4.2 What is an Aspect?
The new programming paradigm is only able over “aspect languages”; these languages have been constructed to capture the elements in all of the system. These elements are called Aspects.

One of the first definitions of an aspect concept is from 1995:
“An aspect is a unit which defines the partial information of other units” [5]

The Wikipedia defines it thus (2007):
“An aspect is a part of a program that crosscuts its core concerns, therefore violating its separation of concerns” [19]

G. Kizacles defines the aspect:
“An aspect is a modular unit which is disseminated over the structure of other functional units. The aspect exists in the design step and in the implementation step. A design Aspect is a modular unit of the design which is mixed in the structure of other parts of the design. A program or code Aspect is a modular unit of the program that we can find in other modules of the program.”[17]

ADRIAN COLYER, ANDY CLEMENT, GEORGE HARLEY & MATTHEW WEBSTER defines the aspect:
“Like a class, an aspect is a unit of modularity, encapsulation, and abstraction: and although classes and aspects have many similarities, there are also some important differences. Unlike classes, aspects can be used to implement crosscutting concern in modular fashion” [2]

We can say, then, that the Aspect is the basic unit in the Aspect Oriented Programming, and it can be defined as the parts of the application which describes the important questions about the semantic and the performance. We can also think about it like the elements scattered over the code, in which we cannot describe the locality in terms of the other components.

If we want an aspect oriented program we will need three different elements:
1. A language to define the basic functionality. This language is called “base language”. It is a general language like C++ or java. Also we can use imperative languages.
2. One or more Aspect Languages. The aspect language defines the form of the aspects.
3. A weaver, responsible to mix the languages. The mix process can be done in execution and compilation time.

![Figure 4.1: Traditional program and Aspect program differences](image)

Figure 4.1 shows the difference of the code between a traditional program, and an aspect oriented program. We can see how the blue squares represent the basic functionality of the program, the green represents the synchronization code and the red mutual exclusion accessing to the memory. In the left program all the codes are mixed during the entire program, making it difficult in terms of maintenance and reusability of the code. However, on the right hand side of figure 4.1 we can see how the codes of the different functionality are modularized in different parts. It is now easier to find and fix problems, add new functionalities or reuse the code in other applications.

In the traditional program the code is mixed during the entire program, but we can see how in the aspect oriented program the different functionalities are in different modules. If you want to fix a problem or add a new method in a concrete functionality, you do not need to look for it, thereby modifying the entire program.

### 4.2 Important Concepts:

#### 4.2.1 JoinPoint:

This is an execution point inside the system where the aspect can be connected, like a method call, throwing an exception or if the field is being modified. The code of the aspect will be inserted in the execution flow of the application adding the functionality.

#### 4.2.2 Advice:

This is the implementation of the aspect. The code that implements the new functionality is inside the advice. The Advice is introduced in the application by the JoinPoint. The advice of the aspect can be applied over different methods at different moments: before, after, around, etc.
4.2.3 PointCut:
The PointCut defines the advice that will be applied at each JoinPoint. It is designed using regular expressions or using names patron (classes, methods or fields). It is even possible to do it during execution time using the value of some parameters.

![Figure 4.2: Advice, JoinPoint & PointCut Structure](image)

In figure 4.2 we show the program structure using aspects, the square triangle represents all the possible JoinPoint where the aspects can be connected. Point Cuts have the target to connect the specific advice in the correct function or exception, and to decide in which moment of the execution the advice has to be executed, after the function, before the function, etc..

4.2.4 Introduction:
The introduction makes possible the adding of new methods or attributes to existing classes.

4.2.5 Target:
The target is the class that is being changed with the aspect.

4.2.6 Proxy:
The proxy is the result of the application of the aspect over the target. The rest of the application will work using the object.

4.2.7 Weaving:
The weaving is the process to transform the target object creating the new proxy objects in the specific JoinPoint. This process can happen during all the target object life. The weaver is responsible for mixing the different mechanism of abstraction and composition in the aspect languages and components using the JoinPoint.
In figure 4.3 we can see the difference between the normal compilers. In the normal process the compiler creates the executable program directly from the source code. The process aspect is different when used before a tool called weaver. That machine is able to work with the source code and the aspect creating a new source code that the compiler will be able to translate to create the executable.

**4.3 Weaving Types**
The classes and the aspect can be entwined in two different ways.

**4.3.1 Static weaving:**
The static weaving means to modify the base code of a class inserting sentences in the JoinPoint. The code of the aspect is introduced in the class; an example of this kind of weaver is the AspectJ weaver.

The best advantage of this kind of weaving is that we don’t have to use the abstraction level introduced with the OOP paradigm, producing a negative impact over the application. However, identifying the different aspects in the application when the application is already weaved can become extremely difficult or even impossible.

**4.3.2 Dynamic Weaving:**
One of the most important prerequisites is that the aspects have to exist in compilation time and in execution time.

The aspects and the weaver structures have to be modeled as objects. Then the weaver will be able to add, adapt and delete the aspects in dynamic way during execution time. An example of this kind of weaver is the AOP/ST weaver, because it does not modify the base code in the classes when the aspects are being entwined. This weaver uses the inheritance to add the specific aspect to the classes.

**4.4. Aspect Languages**
We can find a number of different AOP Languages like:
4.4.1 Spring:
This language is probably the most closed to the AOP paradigm, but it works more than the necessary using XML pipes to work.

4.4.2 Aspect PHP & Aspect C:
These technologies have not got the same development time of Aspect J and Spring. Therefore they are not an ideal support to implement AOP, but they can complete some characteristics of the paradigm.

4.4.3 COOL:
This language is usually used to synchronize threads. The base language is a restricted version of Java, without the reserved words: wait, notify notifyAll and synchronized.

4.4.4 More Java Languages AOP:
The following is a list of tools that support AOP with Java:

- Hyper/J
- JAC
- JMangler
- MixJuice
- PROSE
- ArchJava

4.5 Aspect J Technology
Aspect/J is an extension of Java™ aspect oriented programming developed by Xerox Corporation in Xerox Palo Alto Research Center (PARC). The Xerox purpose is a subset of Java, like an extension of it. Xerox introduces new reserved words and new constructors for develop Aspects, point cuts, joint points and so on.

Aspect/J is available in the open source projects of Eclipse Foundation. The distribution is using its own JDK incorporating the AOP paradigm support. We can find a plug-in in Eclipse distribution oriented to Aspect/J and also to Netbeans. The IDE Eclipse works over Windows and GNU/Linux; also we can find a plug-in called AJDT.

In Aspect/J, an aspect is a class, like the Java class but with an extra ability - it can have cut constructors. The Aspect/J PointCut catches the constructor events in the execution of a program. These events can be methods invoked constructor’s events and management and signal exceptions. The PointCut does not define actions, but describes events.

Right now Aspect/J is the most spread AOP technology in the world, the favorite distribution for many users, and the main reference of this new paradigm. The IDE of Eclipse is famous because it is both easy to use and powerful. This is probably its main advantage because the community is adopting this technology quickly and easily.

4.5.1 Advice types:
In the advice body, we can define different possibilities using the following keys words:

4.5.2 Advice before:
This executes just before the actions associated to the PointCut.

*Practical Example:* it could be useful to control the login process in a specific set on function where you need to have some privileges to access inside information.
4.5.3 Advice after:
This executes just after the actions associated to the PointCut.
Practical Example: we can use it to store the information of the function just after the execution of that function creating a report with all the movements of commodities such as bank accounts.

4.5.4 Advice catches:
This executes when, during the execution, the actions associated to the PointCut throws an exception of type which is the same defined in the catch clause.
Practical Example: It could be an exceptional tool to work with exceptions detecting problems and solving in execution time.

4.5.5 Advice finally:
This executes just after the actions associated to the PointCut. If the program throws an exception during the execution, the advice will be executed.

4.5.6 Advice around:
This is the most powerful form of advice. It is executed after, following the associated function to the JoinPoint. It can decide if the program execution should continue into the specific JoinPoint, and then what context of the JoinPoint should be executed.

4.5.7 Hello World Aspect J Example:
The classical example to begin:

```java
1. public class HelloWorld{
2.     String cadena;
3.     public HelloWorld ( String c){
4.         cadena = c;
5.     } 
6.     public String toString(){
7.         return "HelloWorld, with [" + cadena + "]";
8.     }
9.     public static void main( String args[]){
10.            HelloWorld h = new HelloWorld ( "this is an example");
11.            System.out.println( h );
12.     }
}
```

Figure 4.4: Aspect J Hello world example, main class

In figure 4.4 the aspect is going to write “Before of:” just before to enter in the function to String which write “Hello World, with this is an example”. The PointCut is applying over the method to String of the class called “HelloWorld”, this is the one which introduces the code just before the function.
1. public aspect Aspect2 {
2.  
3.    pointcut in HelloWorld () : execution(public * HelloWorld.toString(..));
4.    before() : in HelloWorld () {
5.      System.out.println("Before of: " + thisJoinPoint);
6.    }
7.  }

Figure 4.5: Aspect J Hello world example, aspect class

Then the console will show the figure 4.6:

Before of: Hello World, with this is an example

Figure 4.6: Aspect J Hello world example, console output

4.5.8 Logging Aspect Example:

A good example to understand the powerful of the Aspect possibilities is the logging example; we have wanted to create a system controlled with a password security. Then we will create an aspect which controls the access, and will win two important things: we will not to have to implement the aspect in all the methods and classes, and we will want to modify all the logging process that is implemented and centralized in the same place.

1. public aspect LoggingAspect {
2.  
3.    pointcut log() : execution(void Client.save())
4.    after() : log() {
5.      System.out.println("…end log");
6.    }
7.    before() : log() {
8.      System.out.println("…begin log");
9.    }

Figure 4.7: LoggingAspect class

4.6 Aspect Werkz Technology

Aspect Werkz was developed by BEA systems with the purpose that their users would be able to use the AOP capacities in their application servers even when their users don’t have any idea about this paradigm or about other Aspect languages, and it compatible with all the versions of Web Logic from 6.1.

Aspect Werkz is a framework developed to be used under AOP paradigm in our programs. That Framework represents a new philosophy different from that of the famous Aspect/J because it lets the configuration of all the different aspects using XML Files, and because “Aspect Werkz has focused its efforts largely on close integration of loadtime weaving into J2EE application environments while Aspect/J has focused more on (static) compilation and weaving, runtime performance, and tools support”. [31]
However BEA Systems is not developing new versions or tool support like other technologies do, and it is logical to think that maybe Aspect Werkz could not survive after 10 or more years. However, this does not mean that we should not investigate this option. Some developers are betting hard on it, and even when BEA is not taking care of it. It is still one of the favorite options for the developers. This landscape can remember how COBOL language is still one of the most useful tools today, used by banks all over the world, despite a lack of recent new versions.

In figure 4.8 and 4.9 we can how many questions had been asked by the users during the last 65 months, the users are asking about Aspect Werkz 0.9 and 1.0. These data was taken from an article presented in the biggest portal for programming developers in Castellán (Spanish) [39]

**Figure 4.8 Aspect Werkz 0.9 consults**

In figure 4.8 we can see the number of consults per month from the last 65 months. We can see how in the beginning, when the technology was presented, there are the maximum number of consults. We can also see how the level of consults is stable 65 months after. From February 2004 to May 2009 [38]
Figure 4.9 Aspect Werkz 1.0 consults

Just as in figure 4.8, figure 4.9 lets us see how some developers are still choosing Aspect Werkz as a tool to develop AOP systems. From January 1999 to May 2009 [37]

We could see how the first versions of Aspect Werkz are still useful and used. However we are not going to focus our studies in these ones. We are going to investigate over the last version created by BEA Systems, Aspect Werkz 2.0.

Aspect Werkz offers two ways to define the aspects over a concrete breakpoint in our code. The first one is the simplest; it uses an xml file to define these relations. The second one needs to be precompiled and it uses attributes in execution time, the relations are defined using headlines before each function like Javadoc. So we will explain and use the first one for this small example. During the implementation of our examples we will use the more powerful, thereby taking decisions in execution time.

4.6.1 HelloWorld Aspect J Example:
Like we did with Aspect /J we are going to present a basic example developed with this technology. The classical example that we can find in the example part of the main web page of Aspect Werkz [35] will let us see the difference between the implementation using Aspect / J and Aspect Werkz. We can see how the main application can be quite similar, it is because both are developed using Java language. They are not exactly the same, in this example we are adding two aspects instead of just one. One aspect is executed just before the call of the function and another just after it. We can see how in the Aspect Werkz implementation we are using one class more called aop.xml, this class is used to define which aspects are active and to decide the specific advices for each different pointcut.

```java
1. package testAOP;
2. public class HelloWorld {
3.       public static void main(String args[]) {
4.         HelloWorld world = new HelloWorld();
5.         world.greet();
6.     }
```
7. public String greet() {
8.     System.out.println("Hello World!");
9. }
10. }

Figure 4.10: Aspect Werkz Hello world example, main class

The main class is not presenting anything new. We can see in figure 4.10 that it is a regular Java Class.

1. package testAOP;
2. import org.codehaus.aspectwerkz.joinpoint.JoinPoint;
3. public class MyAspect {
4.     public void beforeGreeting(JoinPoint joinPoint) {
5.         System.out.println("before greeting...");
6.     }
7.     public void afterGreeting(JoinPoint joinPoint) {
8.         System.out.println("after greeting...");
9.     }
10. }

Figure 4.11: Aspect Werkz Hello World example, aspect class

We are defining two aspects. The first one will be executed just before the call of the function while the second one will be executed just after it. It is interesting to see that we can implement more than one device in each Aspect. The Aspect will be linked using the JoinPoints over the pointcuts using the xml file.

1. <aspectwerkz>
2.     <system id="AspectWerkzExample">
3.         <package name="testAOP">
4.             <aspect class="MyAspect">
5.                 <pointcut name="greetMethod" expression="execution(*
                   testAOP.HelloWorld.greet(..))"/>
6.                 <advice name="beforeGreeting" type="before"
                   bindto="greetMethod"/>
7.                 <advice name="afterGreeting" type="after" bindto="greetMethod"/>
8.             </aspect>
9.         </package>
10.     </system>
11. </aspectwerkz>

Figure 4.12: Aspect Werkz Hello world example, aop.xml

Finally we can see the xml file which is defining the pointcut “greetMethod” and it is applying both advices for this example. “beforeGreeting” and “afterGreeting”

1. before greeting...
2. Hello World!
3. after greeting...

Figure 4.13: Aspect Werkz Hello world example, console output

The output presents that the execution of our aspect are being executed in the right order. We can see it in figure 4.13
5. Adaptive Sort

This chapter will explain the goals and specifications that the main example should implement. We will explain all the different parts in detail, and then we will have a general idea about the problem requirements, finally we will show why it would help us to solve the questions of this investigation.

5.1 The Sort Example

The perfect example to understand this landscape is the sort problem example: in this example our target is to sort certain elements. These elements can be presented in different data structures, over various computers and using different algorithms. Our task is to sort them using the best way in function of:

5.1.1 Data Structures:

We will have our elements in different kind of data structures; vectors, trees, lists, set…. And we will be able to decide which will be the best form to work with them. Thinking in the complexity of the different functions: insert, delete, search and so on. In some cases it will be better to change the structure of the elements, sort them and return to the initial structure. However, this will not be our task in this thesis. We will assume that the elements are presented always in an array, as it is not our target to try and work with all the different data size implementation.

5.1.2 Algorithms:

We will be able to use some different algorithm to sort our elements, and we will have to decide in real time which algorithm is the best in function of the data structures, the collocation of the elements of the data structure, size of data…… We will work with four of the most famous sort algorithms; QuickSort, InsertionSort, BubbleSort and SelectionSort. All of them will be implemented in its own class using the inheritance concept. All of them will be sons of the class Sort, which implements the methods, sort and print. The principal class will use a sort object which will be instanced like a current son class in function of the decision taken. We can find some good definitions of the different algorithms in [40], these definitions were used so many times in lectures and reports that we can find them almost everywhere.

5.1.3 Size of data and data properties:

It is important to consider the data size of elements, as the different element could be stored in some distinct data structures, like vectors, maps, sets, trees and so on. Obviously, it is not the same to work with one structure as with another, the bubble sort may be good working with trees, whilst the insertion sort may be good when we have our elements stored in a stack. The number of elements that we have to sort is also important; some algorithm could be really good working with not so many elements, while QuickSort it is better in the average case with many elements. The collocation of them in the data structure is a difficult skill to decide, but it is important if we realize that SelectionSort could be even better than QuickSort when the elements are almost sorted, or maybe they are sorted in the reverse order.

For the moment we will not have a dispatch table able to know all the possibilities in function of all the parameters, and then we will create our own simple dispatch table just to decide randomly which algorithm is better. The construction of the dispatch table is a hard and long work, and it is not the meaning of the thesis. We will work just with the size of the vector and in function of it; it returns the best option in each case.
5.1.4 Dispatch Table:
This table will know which algorithm is the best to use in this concrete case, in function of all the characteristics explained before. We will accede to this table in run time to decide in run time how to work with our data.

In our implementation we will not use all the possible parameters which the dispatch table has to work; in our simple dispatch table it will use only the size of the vector.
If the number of elements (n) is:
- n < 25: BubbleSort
- (n >= 25) & (n< 50): InsertionSort
- (n >=50) & (n< 75): SelectionSort
- (n >=75: QuickSort

We will assume that these algorithms are the best options in these cases.

5.1.5 Training Learning:
Using all the concepts explained before we will be able to implement and create the dispatch table that will contains all the information about the application that we will use to obtain the best way to solve the actual problem.

5.1.6 Our Work:
We will center our investigation in the last part of the process, when the dispatch table is already created using the training learning work, the general application and the aspect points are going to mix the different codes to create the final code, which is going to solve the solution of the problem in a new application.

Then we will create application based in the AOP paradigm which will be able to change the code in execution time in function of the dispatch table decision.

5.1.7 General Schema
We can see how the data structures (green), the different algorithms (orange) and the data properties (purple) are used in the Training Learning (blue). To create the dispatch table (yellow) the system’s properties (red) influences over the decisions of the table. Finally, using the dispatch table the application (green) and the aspects (red) we can create the new application able to solve the problem.
5.2 Implementing the First probes

5.2.1 Example 1: The aspect decides which part of code will be executed:

In this example we will have different elements inside of an array. In function of the value of the different elements, we are going to create an aspect which decides which kind of paint algorithm has to be executed. To be sure that the decision is taken in execution time the values of the vector will be created using a random function also in the execution time:

The class principal has five different print methods: print1 (), print2 (), print3 () and print4 (). Each algorithm prints its name in the console, also, the method print () takes the decision about which of the others print methods has to print in function of a variable called printVar.
1. **public class** principal
2. {
3.     **public** **void** print()
4.     {
5.         count = (count +1)%10;
6.         if (printVar == 0)
7.             print0();
8.         else if (printVar == 1)
9.             print1();
10.            else if (printVar == 2)
11.                print2();
12.                else if (printVar == 3)
13.                    print3();
14.     }
15.     **public static void** main(String[] args)
16.     {
17.         principal p = **new** principal();
18.         int n;
19.         Integer nAux;
20.         **for**(int i = 0; i < 10 ; i++)
21.             {
22.                 n = (int) Math.round((Math.random()* 10)% 4 );
23.                 nAux = **new** Integer(n);
24.                 p.myArray[i] = n;
25.             }
26.         **for**(int j = 0; j < 10 ; j++)
27.             {
28.                 p.print();
29.             }
30.     }
31. }

**Figure 5.2: SortExample 1, main class**

There is a vector with different elements, we are going to visit all the elements of the array and calling the method print() with the different values of the different elements. Just before the print() method, the aspect is going to check the element value and decide which is the correct value of the variable printVar in function of the value of the actual element, then the print() method will call the correct printX() in function of the decision taken by the aspect.
To be sure that the decision is taken in execution time, and the code change in execution time, the elements of the vector will be created also in execution time, using a random function.

```
1. package firstProgramPackage;
2. public aspect firstAspect
3. {
4.   pointcut print(principal p) :call(void print()) && target(p);
5.   before (principal p): print(p)
6.   {
7.     beforePrint(p);
8.   }
9.   protected synchronized void beforePrint(principal p)
10.   {
11.     System.out.print("Inside the aspect : ");
12.     System.out.println(p.myArray[p.count]);
13.     if (p.myArray[p.count] == 0)
14.       p.printVar = 0;
15.     else if (p.myArray[p.count] == 1)
16.       p.printVar = 1;
17.     else if (p.myArray[p.count] == 2)
18.       p.printVar = 2;
19.     else if (p.myArray[p.count] == 3)
20.       p.printVar = 3;
21.   }
22. }
```

Figure 5.3: SortExample 1, aspect class

We are working in a second example (package secondProgramPackage) where the aspect implements the code. Although it can be useful in other applications, we thought it is better using the aspect only to take a decision, and implement the different codes outside the aspect.

5.2.2 Example 2: The aspect asks the dispatch table to take a decision:

The second one creates a vector, the size of the vector is chosen randomly, and we will want to decide how big the vector is, smallest, small, medium or biggest. After the creation of the vector we will call the function print, in this moment the aspect will get the data relative of the size of the vector, and ask to a database called dispatch table. In function of the answer of the database it will change the value of the size, and print smallest, small, medium or biggest.
5.3 Implementing the Sort Example

The adaptive Sort example uses the ideas explained before but in this case we will use the Aspect to create an instance of the object Sort. Sort is the parent of all the different sorting algorithms. And in function of the decisions of the dispatch table one kind of child or another one will be created like an instance of the Sort Object.

5.3.1 The Principal class code:

```java
1. package example3Package;
2. import example3Package.Example3Principal;
3. public class Example3Principal {
4.     static DispatchTable myTab;
5.     int[] myArray;
6.     sort_enum printVar;

7.     enum sort_enum {
8.         Bubble, Insertion, Selection, Quick
9.     }
10.    static Sort mySort;
11.    public Example3Principal()
12.    {
13.        printVar = sort_enum.Bubble;
14.        mySort = new Sort();
15.    }
16.    public void sort()
17.    {
18.        mySort.sortElements();
19.    }
```

**Figure 5.4: PrincipalSortExample, main class head**

The data type enum called sort_enum is created to indicate the different names of the algorithm that we can use. If we want to insert a new type of algorithm we will have to modify this data type, adding the name to the type definition.

```java
20.    public static void main(String[] args)
21.    {
22.        //create a new Example3Principal
23.        Example3Principal p = new Example3Principal();
24.        //create the dispatch table
25.        myTab = new DispatchTable();
```

**Figure 5.5: PrincipalSortExample, main class sort method**

The method sort() only call the method sort of the object mySort(), and the aspect will be applied over this method.

```java
26.    public static void main(String[] args)
27.    {
28.        //create a new Example3Principal
29.        Example3Principal p = new Example3Principal();
30.        //create the dispatch table
31.        myTab = new DispatchTable();
```

**Figure 5.6: PrincipalSortExample, main class main method**
Like we did with the previous examples, we will create a new vector example, using a random value to decide the number of elements to sort and the values of them. Now we can be sure that the decision is not taken in compiler time.

```java
//create a new Array with random number of elements
//and random values
int n;
int max = (int) Math.round((Math.random() * 100));
//max = 80;
p.myArray = new int[max];
for(int i = 0; i < max; i++)
{
    n = (int) Math.round((Math.random() * 10));
    p.myArray[i] = n;
}
```

**Figure 5.7: PrincipalSortExample, main class new dynamic TAD**

Finally we will call the method sort() and print the vector after it has been sorted.

```java
// sort and print the array
p.sort();
System.out.println("The Sorted Array:");
Example3Principal.mySort.print();
```

**Figure 5.8: PrincipalSortExample, main class sort call**

### 5.3.2 The dispatch table code:

```java
package example3Package;
import example3Package.Example3Principal.sort_enum;
public class DispatchTable {
    sort_enum[] table;
    int sortNum = 4;
    public DispatchTable() {
        table = new sort_enum[sortNum];
        table[0] = sort_enum.Bubble;
        table[1] = sort_enum.Insertion;
    }
}
```
14.     
15.     public sort_enum getSortAlg(int size)  
16.     {  
17.         if (size < 25) //smallest, Bubble  
18.             return table[0];  
19.         else if (25 <= size && size< 50)//small, Insertion  
20.             return table[1];  
21.         else if (50 <= size && size< 75)//medium, Selection  
22.             return table[2];  
23.         else if (75 <= size)//biggest, Quick  
24.             return table[3];  
25.     return table[0];  
26.     }  
27. }

Figure 5.9: PrincipalSortExample, dispatchTable

The dispatch table has four different fields, in each field there is a “sort_enum” variable with the name of one of the algorithms; the method “getSortAlg(int size)” return the different fields of the table in function of the number of elements, in theory, it will return the “correct” algorithm.

5.3.3 The Aspects code:

```java
1. package example3Package;
2. import example3Package.Example3Principal;
3. import example3Package.Example3Principal.sort_enum;
4. public aspect Example3Aspect {
5.     int[] a;
6.     int b;
7.     int c;
8.     // pointcut over the method print in Principal3example
9.     pointcut sort(Example3Principal p) :call(void sort()) && target(p);
10.     //before to execute print of Principal3Example we execute beforePrint
```

Figure 5.10: PrincipalSortExample, Aspect class pointcut

In the first part of the aspect we will define the pointcut, it will be applied over the method void sort() of the Example3Principal class.
Figure 5.11: PrincipalSortExample, Aspect class advice

Now we will setup the aspect, indicating which function we want to execute just before to execute the method sort of the Example3Principal, we will execute the function called beforeSort().

protected static void beforeSort(Example3Principal p)
{
    System.out.println("Inside the aspect: ");
    p.printVar = Example3Principal.myTab.getSortAlg(p.myArray.length);
    Example3Principal.mySort = new Sort();
    if (p.printVar == sort_enum.Bubble)
        Example3Principal.mySort = new BubbleSort(p.myArray);
    else if (p.printVar == sort_enum.Insertion)
        Example3Principal.mySort = new InsertionSort(p.myArray);
    else if (p.printVar == sort_enum.Selection)
        Example3Principal.mySort = new SelectionSort(p.myArray);
    else if (p.printVar == sort_enum.Quick)
        Example3Principal.mySort = new QuickSort(p.myArray,0,p.myArray.length);
}

Figure 5.12: PrincipalSortExample, Aspect class advice implementation

The function beforeSort will consult the dispatch table asking for the best algorithm in function of the size of the vector, and it will create a new Sort object which executes the correct code.

5.4 The second aspect (the fourth example)

We were also working on a second aspect, proof over recursive algorithms, and it is applied over the recursive calls of the QuickSort algorithm called quickSort1. Our target implemented the aspect which had to be able to use different algorithm in the same example, changing the algorithm when the calls are over smaller vectors. The application works and it sorts the algorithm, but it does not change the algorithm in function of the vector’s size.

The challenge of this part of the investigation is to develop an application able to sort some elements using the best algorithm in each case. In function of the number of elements that we should sort, the application should use the right algorithm. This
application starts sorting the vector using an algorithm, by instance QuickSort. When the QuickSort algorithm split the vector in two parts, the application checks again the size of each part of the vector and it sorts each part using the best option for this size. These decisions will be taken during execution time, and we will modify the Sort Object in function of which kind of vector we will be sorting.
6 Evaluation

This chapter will explain the different criteria that we will use to evaluate the different purposes, and we will explain why these criteria could be interesting. During the second part of this chapter we will evaluate both technologies and compare them.

6.1 Evaluation Criteria

The next step in our work will be to begin to think about all the important skills that each technology should offer to us and the different arguments that could make a difference between one technology and the other. Our next task will be to decide the interesting points that can give to us interesting information about the different purposes. Once the implementation process sends this information then the final products investigating all of these different skills of each technology option focuses our attention on the elements that are more interesting for our final challenge. What arguments could we make to a programmer, developer or a user to choose one option over another?

After all this work we have finally defined all the criteria and made an adequate investigation of them. We will have some good information to understand the best and worst points of each technology option, making our final decision easy to make, and finally choose one technology over another.

The next few pages will explain all the different criteria that we are going to use to evaluate the different technologies purposes. Some of these skills that will be observed over the different technologies will be based in measurements, like for example completion times, or the size of the programs, even when we will have to make calculations, graphics and experiments over the machine and the implementation of each program. This part is easy, because the number will show us without any doubt which one is better.

The bigger problem is in the rest of the patterns, because with some of them it is going to be difficult in finding a completely objective argument which will help us decide. For instance, a number could be bigger or smaller than other one, but how should we measure how difficult it is to do something, or how well it will work? All this criteria have subjective parts, based on the personal experience of the programmer, and in their experiences with each program and technology. One of the best ways to not be involved in the final results is in watching the big problems; using small points of view (divide and conquer). We will have to try to solve our big questions splitting them in small cases easier to evaluate it and without the pressure of our own idea.

6.1.1 How good this technology could solve our tasks? Is it working properly?

We want to investigate how the applications are working and if they are running correctly in function of the behavior that we are expecting.

6.1.2 Performance

Completion times: During this moment of the investigation we will focus our attention in measuring the different times that each technology purpose spends for each different example, increasing the number of elements in each case. We will use both algorithms in each different case, and then compare the results of each one. This part is going to be the final evaluation from all the results that we received from the different experiments that we achieved during the previous part. All the experiments are measured in milliseconds.

All the experiments will be developed using the same Hardware system and software configuration.
Experiments over example1: The target during the first example measurements is to execute the Aspect calls a certain number of times and decide which technology is working faster in this sense. We are going to focus these experiments just in this part. The experiments will start creating vectors from 100 elements until 2900 increasing each step in 100 more elements. After creating each element the Aspect should recognize each element, and answer it correctly as fast as possible. Each experiment with each size vector will be repeated 100 times and after it we will calculate the average. Finally we are going to present them in a matrix and create a graphic to compare them easily.

Experiments over example2: The second example that we already have implemented is not giving new information that we cannot already take from the first example.

Experiments over example3: The main example takes a vector where the elements are stored, and examines the system. It then asks to the dispatch table which is the best way to execute this task in function of the data system. The answer allows the creation of one object or another, and then sorts the elements in the way that the dispatch table decided. We are going to measure how fast the application sorts the elements, with AspectJ with Aspect Werkz and with a regular application using just QuickSort algorithm. After it captures the final results and compares them using a graph.

We are going to create arrays with some elements randomly, the size of them will be from 100 elements to 2900, each example size will be executed 100 times and the final result will be the time that the algorithm needs to sort 100 vectors.

Experiments over example4: The recursive version of the main example is not working properly, or not as good as we would like. It could be of use to know how fast these implementations are working, and how fast the recursion is. We are going to use the same method than the main example and compare with the recursive version of the algorithm QuickSort.

We will create arrays with some elements chosen randomly, and the size of them will be from 100 elements to 2900, each example size will be executed 100 times and the final result will be the time that the algorithm needs to sort 100 vectors of this size.

6.1.3 How difficult to implement

Lines of code: How many lines of code we had to implement to be able to develop each example using both technologies.

Number of classes: How many classes had each program, for each different purpose.
Number of different technologies: Some technologies could be based in just one programming language like Java, C++ or C#, while some must use some additional tools, like XML or HTML.

Number of programming languages: The number of programming languages that we should learn before.

Documentation we can find: We will measure this part of the investigation checking on the internet different articles, developers support, written books or number of web pages dedicated to these technologies. We will use Google Searcher to look for the number of web pages related to each technology, and Amazon.com to find all the books in the market talking about these technologies.

Tool support: Is this technology able to be implemented using plug-in for some of the most common programs to develop software like Eclipse or Neat Beans?

Subjective Opinion: Finally we will give our honest and subjective judgment. Remember that the second implementation is always easy because we already have some previous knowledge from the technology developed before.

6.1.4 Installation properties

Number packages to download and install: Some technologies could be installed using plug-in from other programs created to develop software, but some of them should be installed manually downloading different packages and modifying the Class Path.

Easy to find and download? This is probably one of the most subjective criteria that we will investigate. However, it is helpful to know how easily we can find and get the tool we are working with. Can we easily find all the packages? Can we download all the different things we will need from the same place? Is the main Web Page the only place where we can download from? Or can we find some another places or servers to download it?

Is the technology free? Some technologies find a good place in the market just because they are free, MySql is a good example. It could not compete against distribution like Oracle which is designated to big corporations and business, but so many users found in this technology the help that they needed. Because of that MySql improved its capacities, and now we can consider it as one of the most important Data Bases Server that we can find in the market. A free technology will be studied in schools and universities easily, and in crisis moments the corporation bet for this kind of purposes to keep their profits. That is why we consider this skill as valuable and a skill to be taken care of.

Tool Support: Can we install it using a tool support like Eclipse or NeatBeans, or should we download each package independently, and install the technology by ourselves? In this case is it difficult? Is it counting with an executable file to install? Should we change the configuration for the Path or Class Path?

6.1.5 Support online:
Is it possible to find information in the main Web page of each technology about how to install and download?

6.2 Aspect J Evaluation

6.2.1 How good this technology could solve our tasks? Is the application working properly with each example?

Example 1: The first example is working properly.
Figure 6.1: First Example Aspect J

We can see in figure 6.1 how when the aspects is called it first prints the value of the vector, and after it, it takes the decision printing an string with the value repeated some times.

Example 2: The second example is working properly as well.

Inside the aspect:
Size : 7
smallest

Inside the aspect:
Size : 38
small

Inside the aspect:
Size : 62
medium

Inside the aspect:
Size : 91
biggest

Figure 6.2: Second Example

During the second example the aspect asks the dispatch table for the right way to call a vector in function of its size, if the number of elements (n) is < 25 then it is Smallest; if n >= 25 and n < 50 then it is Small; if n >= 50 and n < 75 then it is medium and if n >= 75 then it is biggest. We can see that aspect is taking modifying the execution correctly in function of the database data.

Example 3: In the following tables we can see the console output in different cases. Each sort implementation shows in the beginning of the execution the name of the algorithm which is going to be used in that specific moment. Also it shows the size of the target vector and values that the vector stores.

Inside the aspect:
The original Array : BubbleSort 17 elements
3,8,2,5,7,9,9,9,6,4,8,6,8,7,3,5,3,
The Sorted Array: BubbleSort 17 elements
2,3,3,3,4,5,5,6,6,7,7,8,8,8,9,9,9.

Figure 6.3: PrincipalSortExample, console output BubbleSort (1-25) elements

Our first figure shows a BubbleSort implementation, we can see in figure 6.3 that the algorithm is working correctly sorting the elements in down to up order, also the
Aspect is choosing the correct kind of algorithm for that specific conditions; When the size is $1 < 17 < 25$.

<table>
<thead>
<tr>
<th>Inside the aspect:</th>
<th>The original Array :insertionSort 35 elements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8,6,1,8,9,3,4,9,8,5,4,7,7,8,4,0,6,8,4,10,7,4,7,4,6,1,7,1,7,3,6,5,9,1,7,</td>
</tr>
<tr>
<td></td>
<td>The Sorted Array: insertionSort 35 elements</td>
</tr>
<tr>
<td></td>
<td>1,3,4,4,4,5,5,5,5,6,6,6,6,7,7,7,7,8,8,8,8,8,8,8,8,9,9,9,9,10,10,</td>
</tr>
</tbody>
</table>

**Figure 6.4: PrincipalSortExample, console output InsertionSort (26-50) elements**

The InsertionSort algorithm works with a vector between 25 and 50. In this case the vector is 35, and then the aspect is deciding correctly which algorithm it has to use. The vector is also correctly sorted.

<table>
<thead>
<tr>
<th>Inside the aspect:</th>
<th>The original Array :selectionSort 69 elements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9,0,8,9,4,8,3,6,6,1,1,1,3,4,7,5,3,8,3,7,9,9,10,2,5,5,0,5,9,7,4,4,1,0,2,4,3,6,6,7,10,1,6,0</td>
</tr>
<tr>
<td></td>
<td>The Sorted Array: selectionSort 69 elements</td>
</tr>
<tr>
<td></td>
<td>0,0,0,0,0,1,1,1,1,1,1,1,2,2,2,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,5,5,5,5,6,6,6,6,6,6,6,6,6,</td>
</tr>
<tr>
<td></td>
<td>7,7,7,7,7,7,8,8,8,8,9,9,9,9,9,9,9,9,10,10,10,</td>
</tr>
</tbody>
</table>

**Figure 6.5: PrincipalSortExample, console output SelectionSort (51-75) elements**

The Selection sort implementation is not an exception. It is working correctly sorting the correct order and using the algorithm that the dispatch tables told the aspect.

<table>
<thead>
<tr>
<th>Inside the aspect:</th>
<th>The original Array :quickSort 98 elements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8,7,8,9,3,1,10,3,0,8,4,5,6,7,6,9,2,1,1,4,0,4,0,8,0,2,5,9,10,8,10,4,6,1,10,6,9,10,7,3,6,2,8</td>
</tr>
<tr>
<td></td>
<td>The Sorted Array: quickSort 98 elements</td>
</tr>
<tr>
<td></td>
<td>0,0,0,0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5</td>
</tr>
<tr>
<td></td>
<td>5,5,5,5,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,10,</td>
</tr>
<tr>
<td></td>
<td>10,10,10,10,10,10,10,</td>
</tr>
</tbody>
</table>

**Figure 6.6: PrincipalSortExample, console output QuickSort (>75) elements**

We can observe in figure 6.6 how the program sorts the algorithm correctly from the minimum element to the maximum element, and also how the application works perfectly using the correct algorithm in each case. The aspect manages and chooses the exact instance of the different sort algorithms using the dispatch table information on the different vector size.

**Example 4:** The application is able to sort the elements after the whole process, using the Aspect to decide which algorithm is the best to use in each case in function of the size of the data. However the application doesn’t behave like we expected, even when the elements are sorted at the end. The idea is to develop a mechanism able to split the vector into different parts, and create different sort objects able to sort each part of the vector. The application is not able to change the object during the execution; it just uses the first object created. Hopefully, we can use another kind of weaver. Indeed, the dynamic weaver of Aspect Werkz could be the answer.
6.2.2 Performance
Experiments over example1:

<table>
<thead>
<tr>
<th>Example1</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
<th>1100</th>
<th>1200</th>
<th>1300</th>
<th>1400</th>
<th>1500</th>
<th>1600</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect J</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Experiments over example3:

<table>
<thead>
<tr>
<th>Example3</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
<th>1100</th>
<th>1200</th>
<th>1300</th>
<th>1400</th>
<th>1500</th>
<th>1600</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect J</td>
<td>234</td>
<td>313</td>
<td>375</td>
<td>469</td>
<td>734</td>
<td>671</td>
<td>829</td>
<td>906</td>
<td>985</td>
<td>1077</td>
<td>1235</td>
<td>1343</td>
<td>1406</td>
<td>2000</td>
<td>1609</td>
<td>1796</td>
</tr>
</tbody>
</table>

Experiments over example2:

<table>
<thead>
<tr>
<th>Example2</th>
<th>1600</th>
<th>1700</th>
<th>1800</th>
<th>1900</th>
<th>2000</th>
<th>2100</th>
<th>2200</th>
<th>2300</th>
<th>2400</th>
<th>2500</th>
<th>2600</th>
<th>2700</th>
<th>2800</th>
<th>2900</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect J</td>
<td>7</td>
<td>28</td>
<td>228</td>
<td>570</td>
<td>785</td>
<td>869</td>
<td>884</td>
<td>944</td>
<td>1038</td>
<td>1032</td>
<td>1089</td>
<td>1226</td>
<td>1235</td>
<td>1192</td>
</tr>
</tbody>
</table>

All the experiments are measured in milliseconds.

6.2.3 How difficult to implement

**Lines of code:**
- Example1: 102 +23 = 125
- Example 2: 83+79+31 = 193
- Example 3: 87+30+55+73+70+101+89+24+14 = 543
- Example 4: 88+30+62+69+71+102+90+24+14 = 550
- Tests: 80

**Number of classes:**
- Example1: 2 classes
- Example 2: 3 classes
- Example 3: 9 classes
- Example 4: 9 classes
- Tests: 1 class

**Number of different technologies we should use:** We will need to use j2sdk for the compilation of the java classes and JRE System Libraries. Also we will need to use the ASPECT JRT_LIB, which counts with 10 packages which enables us to work with the aspects. These packages add all the reserved words, constructors for JoinPoint, PointCut, and all the different kinds of advice.

**Programming languages we should learn before:** Java and the special extension of Java Language called Aspect/J that introduces new reserved words, and new syntax to define the JoinPoint and PointCut.

**Documentation we can find:**
- Web Pages: 517.000
- Books: 50

**Tool support:** Aspect/J has plug-in for all the versions of Eclipse from 3.0 to 4.1 and for Netbeans.

**Subjective Opinion:** At the beginning the implementation process was quite tricky, whilst the Java part was easy. However to implement the JoinPoint, and discovering how to link it with the concrete advice we wanted to apply with, is the most difficult part. When we had all the concepts clear and we were able to execute codes before and after the function we were working with everything started to go well. The next problem was when we were trying to modify part of the fields and objects of the main application when it is being executed. It is not that easy to access the main class and
modify it as we want, we should be careful with what we are trying to change. Finally we should add that the Debugger of Eclipse is not working adequately, this makes it even more difficult to fix problems, or follow the execution in conflictive moments.

6.2.4 Installation properties

Nº packages to download and install: The easy way to install it is to use the Eclipse Tool, and then from the update section look for The Aspect/J technology.

Easy to find and download? It is really easy to find everything you need in the main page of Eclipse, and you can find many other servers where you can download the technology. The easiest way, I found, was to start work using the plugin of eclipse tool, and download from the software updates.

Is the technology free? The technology is free.

Subjective Opinion: Probably one of the main secrets of AspectJ is how easy it is to start to work with. That is why it became famous and used all over the world. Eclipse is making a great job in this sense, and it is working.

6.2.5 Support online

Because Aspect J is the most popular distribution to work with AOP paradigm, there are so many places blogs and pages to ask for help.

6.3 Aspect Werkz Evaluation

6.3.1 How good this technology could solve our tasks? Is it working properly with each example?

Example 1: The first example working with Aspect Werkz is working properly. The first line: “AspectWerkz INFO Preprocessor org.codehaus.aspectwerkz.transform.AspectWerkzPreProcessor loaded and initialized”, means that the dynamic mode is already working.

```
AspectWerkz  INFO  Preprocessor
org.codehaus.aspectwerkz.transform.AspectWerkzPreProcessor loaded and initialized
inside The aspect example1 :0
0000000000000000000
inside The aspect example1 :0
0000000000000000000
inside The aspect example1 :2
22222222222222222
inside The aspect example1 :2
22222222222222222
inside The aspect example1 :1
11111111111111
```

Figure 6.7: First Example Aspect Werkz

The first line of figure 6.7 says “AspectWerkz INFO Preprocessor org.codehaus.aspectwerkz.transform.AspectWerkzPreProcessor loaded and initialized” Means that the dynamic mode is already working. We can see that the aspect is choosing the print method correctly in function of the value of the element in the vector.
Example 2: The second example is working well.

```
AspectWerkz INFO Preprocessor
org.codehaus.aspectwerkz.transform.AspectWerkzPreProcessor loaded and initialized inside The aspect example2
Size : 2
smallest 1-25
```

```
AspectWerkz INFO Preprocessor
org.codehaus.aspectwerkz.transform.AspectWerkzPreProcessor loaded and initialized inside The aspect example2
Size : 40
small 26-50
```

```
AspectWerkz INFO Preprocessor
org.codehaus.aspectwerkz.transform.AspectWerkzPreProcessor loaded and initialized inside The aspect example2
Size : 62
medium 51-75
```

```
AspectWerkz INFO Preprocessor
org.codehaus.aspectwerkz.transform.AspectWerkzPreProcessor loaded and initialized inside The aspect example2
Size : 87
Biggest: > 100
```

**Figure 6.8: Second Example Aspect Werkz**

The aspect is able to ask to the dispatch table and answer correctly in each different case. Also in the first line of figure 6.8, we can see that the dynamic mode is activated.

Example 3: The main example is sorting the elements of the vector using the right algorithm in each case. It uses the aspect to decide the best way to sort them, using the data from the size of the vector and the decisions from the dispatch table.

```
AspectWerkz INFO Preprocessor
org.codehaus.aspectwerkz.transform.AspectWerkzPreProcessor loaded and initialized inside The aspect example3
Size : 15
Bubble:
The original Array :BubbleSort 15 elements
1,10,10,6,4,5,3,4,7,1,10,8,7,4,2,
The Sorted Array:BubbleSort 15 elements
1,1,2,3,4,4,4,5,6,7,7,8,10,10,10,
```

**Figure 6.9: Third Example Aspect Werkz, BubbleSort**
Example 4: Probably this is the most disappointing part of the whole thesis, even when we found good answers for all the previous problems. The recursion case is not part of them. We wished that a different kind of weaver could be the answer of the wrong behavior that we found using the recursive option with AspectJ. But it is not. The application is not able to change the instance of the objects during execution time, even when we are using a different kind of weaver. We can apply an aspect over a class, but not over another aspect. It makes the recursion more difficult and tricky. Also “The Aspects” in aspect Werkz are not able to work with variables from the main class that are not defined as static. It means that all the objects should work with the same vector at the same time; it will create many problems in mutual exclusion, making the application slower. The vector is not being sorted at the end of the algorithms, even when the behavior of this technology is being clearer.
The possible answer is to be able to develop an application which uses different Sort objects in the different part of the vector to create a different object in each different case. To give to it a copy of the part of the vector that it must sort, and after it copy again the sorted part of the vector, in the original one. But it means that we will have so many different sort objects in memory at the same time that we risk blocking the system, and spending more time copying the elements from one vector to another one whilst sorting.

6.3.2 Performance

Experiments over example1:

<table>
<thead>
<tr>
<th>Example1</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
<th>1100</th>
<th>1200</th>
<th>1300</th>
<th>1400</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect Werkz</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>107</td>
<td>529</td>
<td>878</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example1</th>
<th>1600</th>
<th>1700</th>
<th>1800</th>
<th>1900</th>
<th>2000</th>
<th>2100</th>
<th>2200</th>
<th>2300</th>
<th>2400</th>
<th>2500</th>
<th>2600</th>
<th>2700</th>
<th>2800</th>
<th>2900</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect Werkz</td>
<td>866</td>
<td>655</td>
<td>913</td>
<td>962</td>
<td>1055</td>
<td>974</td>
<td>1088</td>
<td>1123</td>
<td>1249</td>
<td>1252</td>
<td>1239</td>
<td>1218</td>
<td>1268</td>
<td>1249</td>
</tr>
</tbody>
</table>

All the experiments are measured in milliseconds.

Experiments over example3:

<table>
<thead>
<tr>
<th>Example3</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
<th>1100</th>
<th>1200</th>
<th>1300</th>
<th>1400</th>
<th>1500</th>
<th>1600</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect Werkz</td>
<td>219</td>
<td>375</td>
<td>453</td>
<td>594</td>
<td>609</td>
<td>750</td>
<td>844</td>
<td>968</td>
<td>984</td>
<td>1156</td>
<td>1641</td>
<td>1328</td>
<td>1484</td>
<td>1563</td>
<td>1671</td>
<td>2313</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example3</th>
<th>1600</th>
<th>1700</th>
<th>1800</th>
<th>1900</th>
<th>2000</th>
<th>2100</th>
<th>2200</th>
<th>2300</th>
<th>2400</th>
<th>2500</th>
<th>2600</th>
<th>2700</th>
<th>2800</th>
<th>2900</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspect Werkz</td>
<td>2313</td>
<td>1873</td>
<td>2047</td>
<td>2577</td>
<td>2205</td>
<td>2719</td>
<td>2468</td>
<td>2985</td>
<td>3015</td>
<td>2844</td>
<td>3298</td>
<td>3766</td>
<td>3578</td>
<td>4002</td>
</tr>
</tbody>
</table>

All the experiments are measured in milliseconds.

6.3.3 How difficult to implement

Lines of code:

- Example1: 15 + 28 + 124 = 167
- Example 2: 15 + 22 + 79 + 80 = 196
- Example 3: 15 + 32 +87 +87 +30 +75 +101 +89 +25 +24 = 548
- Example 4: 15 + 64 +88 +30 +71 +74 +104 +91 +25 +14 = 576
- Tests: 78

Number of classes:

- Example1: 3 classes
- Example 2: 4 classes
- Example 3: 10 classes
- Example 4: 10 classes
- Tests: 1 class

Number of different technologies we should use: We will need to use j2sdk for the compilation of the java classes and JRE System Libraries. Also we will need to use the aspectwerkz2.0.RC2.jar, aspectwerkzcore2.0.RC2.jar, aspectwerkzjdk142.0.RC2.jar, concurrent1.3.1.jar, dom4j1.4.jar, jarjar0.3.jar, jrexx1.1.1.jar, piccolo1.0.3.jar, qdox1.4.jar, torve1.0.2.jar; These packages add all the reserved words, constructors for JoinPoints, pointCuts, and all the different kinds of advice.

Programming languages we should learn before: Java, Aspect Werkz, XML

Documentation we can find:
- Web Pages: 128.000
Books: 38

**Tool support:** Aspect Werkz has tool support just for the version 3.0, 3.1 of Eclipse. And there is not any plug-in available for the versions after it or for Netbeans.

**Subjective Opinion:** Like with AspectJ the main application implementation was quite easy, the main language is Java as well, so this part was large, but not really difficult. The tricky part, once again, was linking the aspects and the main application. There are two different ways to do this, using headlines like javadoc, or using the xml files to decide which advice should be applied over each PointCut and then over each JoinPoint. Initially it is confusing, as in each example you read the JoinPoint and point cuts are linked in a different way. There are two types of execution as well, static or dynamic, and to work with the second one we should enable some special properties before to compile or execute. Also we had to start to work with XML to define the aspects relationships, it is not really difficult, but it is still a new effort. Like with AspectJ, Eclipse is not working properly when we want to debug the program using Aspect Werkz, and it is quite tricky to fix problems or follow the execution of the program especially in recursive functions or complex algorithms.

### 6.3.4 Installation properties

**Nº packages to download and install:** There are some options to download the complete aspect Werkz; the easy one, as always, is to use a tool to do it. Downloading the plug-in for eclipse was the fastest way to find everything we needed to start work. But Aspect Werkz is not working with all the versions of Eclipse, and not with all the versions or jre and jdk. So it is necessary to download the concrete version of each one if we want everything to work properly.

**Easy to find and download?** There are not so many places to find everything we will need to start work. The main Web page is offering the main libraries and .jar. There are not so many other places or servers to download it, except from the main web page.

**Is the technology free?** The technology is free.

**Subjective Opinion:** The installation of Aspect Werkz was a hard and slow task; we found so many problems between different versions of each technology. These problems made the installation process much harder than AspectJ installation. The different ways to compile and execute were more complex to understand with Aspect Werkz, because it includes special options to enable or disable the dynamic weaver. There are some opinions on the internet saying that the secret of AspectJ against Aspect Werkz is in how easily you can start to work with one technology, and how hard is start with the other one. This is not the main different between both purposes, but there is an agreement that it is an important aspect in terms of success in being able to arrive to the users as fast and easy as possible.

### 6.3.5 Support online:

It is easy to find some manual explaining how to install and start to work with Aspect Werkz. And there are some blogs and web pages working quite fast to solve doubts about how to develop and implement systems based in aspect Werkz. Since Aspect J offers much more possibilities and so to compare both in this sense is difficult.
6.4 Aspect / J versus Aspect Werkz

Experiment 1:

Figure 6.13 Comparing times during example1

Figure 6.13 shows the measurements in milliseconds in function number of elements that the system is sorting.

During this experiment we are comparing how fast the technologies are able to use the point cuts over the joint point to choose the right advice. And how the advices of the aspects take decisions and change the behavior of the main application. We can observe in figure 6.13 that AspectJ is being only a little faster in average from the vector bigger 1200 elements, in these moments the application has to link 1200 times the point cut with the joint point and the advice. This means that AspectJ is faster linking the main application and the aspects. It is probably because the weaver of Aspect Werkz is weaving dynamically while the AspectJ does it statically.

Experiment 3:

<table>
<thead>
<tr>
<th>Example3</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
<th>1100</th>
<th>1200</th>
<th>1300</th>
<th>1400</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuickSort</td>
<td>203</td>
<td>250</td>
<td>503</td>
<td>500</td>
<td>671</td>
<td>794</td>
<td>846</td>
<td>951</td>
<td>889</td>
<td>1125</td>
<td>1655</td>
<td>1328</td>
<td>1878</td>
<td>1658</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example3</th>
<th>1600</th>
<th>1700</th>
<th>1800</th>
<th>1900</th>
<th>2000</th>
<th>2100</th>
<th>2200</th>
<th>2300</th>
<th>2400</th>
<th>2500</th>
<th>2600</th>
<th>2700</th>
<th>2800</th>
<th>2900</th>
</tr>
</thead>
<tbody>
<tr>
<td>QuickSort</td>
<td>1996</td>
<td>1952</td>
<td>2363</td>
<td>2162</td>
<td>2548</td>
<td>2764</td>
<td>2701</td>
<td>3280</td>
<td>3187</td>
<td>3503</td>
<td>3645</td>
<td>3827</td>
<td>3826</td>
<td>4312</td>
</tr>
</tbody>
</table>

All the experiments are measured in milliseconds.
Figure 6.14 Comparing times during example3

Figure 6.14 shows the measurements in milliseconds in function number of elements that the system is sorting.

During the experiments over the third example we are going to compare how fast both technologies are able to sort all the different elements stored in the vectors, from 100 elements to 2900 elements. Both implementations will be compared with the regular execution of the QuickSort algorithm. The meaning of this study is not just to decide which is faster in the general example. We will also be observing whether the application that is based over aspects is wasting more time. This means we will see if in the general computation the difference between the regular application and the AOP one is enough significant to decide that we are wasting time. The general example can really show an important difference between AspectJ and Aspect Werkz, the speed of each technology is eclipsed by the rest of the application. The three lines are always in the same values; actually we can see that, in the end, the QuickSort implementation looks slower than the AOP purposes. We can then decide that it is in the big applications when we have to do big process, like sort the algorithms that the AOP necessities are not significantly wasting the computer time.
<table>
<thead>
<tr>
<th></th>
<th>Aspect J</th>
<th>Aspect Werkz</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Is it working properly?</strong></td>
<td>example1</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>example2</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>example3</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>example4</td>
<td>No</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>example1</td>
<td>Faster</td>
</tr>
<tr>
<td></td>
<td>example3</td>
<td>No significant</td>
</tr>
<tr>
<td><strong>Implementation process</strong></td>
<td>Lines of Code</td>
<td>1511</td>
</tr>
<tr>
<td></td>
<td>Number of classes</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Nº technologies</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Tool support</td>
<td>plugin Eclipse for all versions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and for Netbeans</td>
</tr>
<tr>
<td></td>
<td>Nº Prog. Lang</td>
<td>2</td>
</tr>
<tr>
<td><strong>Documentation</strong></td>
<td>Sites</td>
<td>517.000</td>
</tr>
<tr>
<td></td>
<td>Books</td>
<td>50</td>
</tr>
<tr>
<td><strong>Installation</strong></td>
<td>nº packages down</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Easy to find?</td>
<td>Really easy</td>
</tr>
<tr>
<td></td>
<td>Free?</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Support online?</td>
<td>so much</td>
</tr>
</tbody>
</table>

**Figure 6.15 Comparing criteria**

In figure 6.15 we can see a schema of the different criteria analyzed for both technologies. The most subjective aspects are explained with words like ‘Easy’, or ‘so much’. ‘No significant’ means that the difference between them is not significant enough to decide that one technology is noticeably better than the other one. For more information we recommend to focus in the evaluation sections from 6.1 to 6.3.
7 Conclusions and future work

This chapter is focus on the final conclusions of the thesis, and it evaluates how good the thesis is answering the problem statement. The second part of the chapter is going to purpose different options to investigate in the future.

7.1 Conclusions

Our tasks in this thesis were answer to the following questions:

“Is it possible to develop adaptive code application based in the Aspect Oriented Programming Paradigm (AOP)”

“How may we use AOP to automatically the application including the adaptability-code?”

“Is the adaptability aspect possible using aspect Werkz?”

“Which are the important characteristics that we should analyze and take care with the different technology purposes?”

“Which technology should we choose in each different case?”

It is important to start solving the first of our questions. We should spend some efforts showing how this new paradigm could be the answer to develop systems based in adaptive code. Then it is important to decide if this is a practical possibility. The first part of the work is the investigation process. We had to learn everything about this new paradigm. We learned to understand why this new paradigm was developed, which are the new capacities and advantages that it can offer, and look how it is being used right now.

After this, we had to look for a good technology based in the AOP paradigm which can respond to our tasks. Aspect J based in java language programming is the most famous technology based in AOP. It is easy to understand, and to start to work with. So we decided to start the implementation process with it.

We programmed two simple examples showing the basics ideas, and demonstrating that AspectJ allows implement programs which are able to take decisions in execution time. It means the Adaptive code is possible using the AOP paradigm.

During this whole process we are explaining how to use the new concepts introduced, and we showed how to implement an application which is able to take decision in execution time. This means the second question was already solved and explained.

The next step was to find another technology based in AOP to work with. The answer was Aspect Werkz, developed by BEA systems, which is one of the most important distributions based in AOP in recent years. It offers an interesting skill called dynamic weaver. Aspect J spent some long time developing tools to create a friendly way to arrive to the developers; meanwhile BEA focused its efforts improving the weaver.

The development process was more difficult using this new technology, the installations, and online support is not as good as we could wish. But we succeed implementing the basic examples and showing that the code adaptability is possible using Aspect Werkz.

Not all the examples worked perfectly. There were a number of problems with the recursive version of both technologies for different reasons that are already explained during the evaluation. We found a possible way to implement a system based inductive theory. But this solution will sacrifice the final challenge. It is not really answering the
wishes that we expect developing this adaptive code over the main application using the recursive concepts.

Ignoring this last example we found that both technologies were working correctly for the iterative version of the main application; both of them were able to implement the system we wished and both of them are a possible tool to answer our necessities. But the work is not finish here. We should investigate more. Which one is better? Which are the reasons that we should take care to choose one technology or the other one? Is it the Speed? Or how easy it is to program? Or maybe the installation facilities? We made a list of some important criteria that a new developer should take care, and examined all of them comparing both technologies.

AspectJ and Aspect Werkz are both of them powerful tools to develop systems. Both of them are capable to create adaptive code based in AOP. However AspectJ is number one in the world market, and for good reason. It is easy to work with, easy to download and to install, and powerful enough to answer the necessities of almost everybody. Meanwhile although Aspect Werkz is improving, it is still not enough to create an important difference.

However the AOP paradigm was developed and created with a different purpose. The main purpose was to help with the maintenance and the reusability of the code. The paradigm is a wonderful way to increase the modularity of our applications. It was not created to develop the systems that we are exploring in this thesis. The AOP paradigm could be a good tool to use for its main purpose, but is still probably not the best on the market.

7.2 Future Work

The future work in this landscape can by split in a number of ways. I believe it could be interesting to continue investigating in the AOP possibilities, maybe not in the development of adaptive software, but rather it looking at is as a good technology to develop distributed systems and large applications.

The self-adaptive applications are becoming capable to develop more functionalities and capacities obviously we should continue trying to implement this kind of software with different kind of technologies and paradigms. One of the best candidates from our point of view is Spring Framework; it is becoming popular during recent years and offers a powerful purpose to investigate with.
Bibliography:

[17] HTTP://WWW.CS.UBC.CA/~GREGOR/PAPERS/KICZALESICSE05AOPMR.PDF (DECEMBER 2007)
[34] http://www.programacion.net/java/noticia/1177&id=noticia/(MAY 2009)