Feature Modeling For Adaptive Computing

Bo Tao
Xiaoming Jiang
Acknowledgment

First of all, we want to thank our home university: Wuhan University of Technology and Växjö University, without their agreement, we could not have the chance to study at Växjö University and finish our thesis. We would also like to thank our families for their supports.

We also want to express our gratitude to our supervisor, Jesper Andersson, without his hints and suggestions we couldn’t finish this thesis on time.

We would also like to thank our thesis coordinator, Mathias Hedenborg for his help on how to write a thesis report.
Abstract

This report presents the results of a thesis project that surveys and designs about the issue “Feature Model for Adaptive Computing”. In this project, there are two main issues, first one is about the Feature Modeling, and the second is how to use this Feature Modeling for adaptive computing.

In this thesis report, at the beginning, we present the problem we expected to solve and introduce some background information, including the knowledge of feature model and adaptive computing. Then we explain our solution and evaluate this solution. At the end of this report, we give a short conclusion about our thesis project and feature work.

**Key-words:** Feature Model, Model Driven Architecture, Adaptive Computing, Software Product Lines, DSL, Java Reflection.
# Contents

1. Introduction ................................................................................................................ 1
   1.1 Problem discussion .......................................................................................... 1
   1.2 Problem Statement........................................................................................ 2
   1.3 Method ............................................................................................................. 3
       1.3.1 Design strategy...................................................................................... 3
       1.3.2 Evaluation strategy............................................................................... 3
   1.4 Restrictions ...................................................................................................... 5
   1.5 Structure of the thesis....................................................................................... 5

2. Related Work.............................................................................................................. 6
   2.1 Theoretical framework ..................................................................................... 6
       2.1.1 Adaptive Computing ............................................................................. 6
       2.1.2 Adaptive Computing building blocks ................................................... 7
       2.1.3 Feature modeling .................................................................................. 8
       2.1.4 Application of feature model .............................................................. 10
       2.1.5 Domain-Specific Language (DSL) ..................................................... 11
       2.1.6 Deployments system ........................................................................... 11
   2.2 Implementation technologies ......................................................................... 12
       2.2.1 UML class modeling ........................................................................... 12
       2.2.2 XML .................................................................................................... 14
       2.2.3 XPath ................................................................................................... 14
       2.2.4 Java Reflection API............................................................................. 16

3. Feature models for adaptive computing ................................................................... 19
   3.1 Restriction Features ....................................................................................... 19
   3.2 Application of restriction features .................................................................. 22
   3.3 Restriction Language (RestrictionL).............................................................. 29
       3.3.1 Designing and using the RestrictionL ................................................. 30
       3.3.2 RestrictionL rules ................................................................................ 32
       3.3.3 Designed Java classes ......................................................................... 33
       3.3.4 Comments in RestrictionL ................................................................. 33
       3.3.5 Importing information from other RestrictionL files.......................... 33
   3.4 Methodology .................................................................................................. 34
       3.4.1 Working procedure .............................................................................. 34
       3.4.2 Example for Methodology .................................................................. 35

4. Evaluation ................................................................................................................ 39
   4.1 Evaluation for Feature Model with Restriction Features ............................... 39
   4.2 Evaluation for RestrictionL ........................................................................ 40
   4.3 Generality ....................................................................................................... 41
   4.4 Robustness and Efficiency ............................................................................. 44

5. Conclusions and Future work .................................................................................. 45
   5.1 Conclusions .................................................................................................... 45
List of Figures

Figure 2.0.1: Adaptive computing process ............................................................ 6
Figure 2.0.2: A feature model example ................................................................. 9
Figure 2.0.3: Grouping features by using multiplicities ...................................... 10
Figure 2.0.4: Links between features, requirements, design and implementation ...................................................................................................................... 11
Figure 2.0.5 Deployments map ......................................................................... 11
Figure 2.0.1: Hierarchy of UML 2.0 Diagrams, shown as a class diagram ...... 13
Figure 2.0.2: A class diagram for feature modeling ............................................ 13
Figure 2.0.3: A simple XML file ......................................................................... 16
Figure 3.0.4: Example features .......................................................................... 20
Figure 3.0.5: Restriction relationship between features ...................................... 20
Figure 3.0.6: Decoders and Effects ..................................................................... 21
Figure 3.0.7: Restriction relationship between Decoders and Effects .............. 22
Figure 3.0.8: The final feature tree for a video player ....................................... 24
Figure 3.0.9: Feature model of Decoder and Effects part ................................ 26
Figure 3.0.10: Restriction relationship between Steaming Decoder & Band-width ...................................................................................................................... 27
Figure 3.0.11: Restriction relationship between Decoder and format ............. 28
Figure 3.0.12: The final video player model which can achieve adaptive computing ...................................................................................................................... 29
Figure 0.13 Working procedure .......................................................................... 35
Figure 0.14 Feature model for compression and decompression system ......... 36
List of Tables

Table 1.0.1: Evaluation Form ................................................................. 4
Table 1.0.2: Evaluation Form 2 ............................................................... 4
Table 2.0.1: Some common XPath expressions ..................................... 15
Table 0.2 Class object creating methods .............................................. 17
Table 3.0.1: Table of Effects ................................................................. 23
Table 3.0.2: Table of Decoder ............................................................... 23
Table 3.0.3: Table of Stream Decoder .................................................. 23
Table 3.0.4: Restriction relationship between bandwidth, Decoders & Effects .... 24
Table 3.0.5: Restriction relationship between Media format, Decoders & Effects ................................................................. 24
Table 3.0.6: Restriction relationship Decoders & Effect ............................ 25
Table 3.0.7: Restriction relationship S-Decoder & Effects ................. 25
Table 3.0.8: Bandwidth and Streaming decoder ................................... 26
Table 3.0.9: Media Format and Decoder ............................................. 27
Table 4.13: Evaluation form for theory ............................................. 39
Table 15 .................................................................................................. 40
1. Introduction

The developing of software is a process analyzing the requirements and designing a solution for these requirements. Designer designs the software according to the exact requirement. In modern software industry, some parts of a software must be reusable and extendable in order to reduce the cost of developing software. However, there is a risk that is a variety of different requirements. The feature requirement of a software is unknown, accordingly the risk is high to develop a improper software. Hence, Domain Analysis will be helpful to solve this problem.

The requirements, the system resource and available resource are changing all the time within a runtime and the demand is changing all the time. Therefore, there should be a way to make certain adjustment to achieve a balance in certain level. The solution for hitting the balance between requirement and resource is "Adaptive Computing". Obviously when requirement and resource combine in certain way, the whole system will get a point which the performance is maximal. And then the question is coming how to achieve adaptation, to achieve adaptation, the presupposition is that there should be a variability model. In that case, dynamic adaptation can be achieved.

This project is about feature modeling for adaptive computing. In this project, the aim is to investigate existing feature modeling notation and design a formal feature model for adaptive software systems. Moreover, we need to design a Domain-Specific Language (DSL) to express Feature Model. This feature model contains the information which will be used for adaptive computing. For software or a system, the ability of adaptive computing is reconfiguration. This reconfiguration will lead to the designed performance or aim, or simply saying it can self-improve.

Feature Model, Generally speaking it is a way that expresses requirements and demand in the level of abstraction. It generally describes the variables and properties and it derive configurations of software system. If requirements and solutions are isolated island, then Feature Model can be bridge which connected these islands together. It's a way to connect them. With an exactly precise definition of features, there can be a more robust model. Because precise features provide a more precise description of the relations between the elements in the model, that is very useful information for adaptive computing.

1.1 Problem discussion

In this project we will investigate several principal issues. To find a generic solution to these issues, we plan to study several specific instances. By analyzing these special cases, we may be able to generalize. Therefore the examples used in this thesis project are very important.

Moreover, adaptive computing is the key point of this project. The feature model should be able to dynamic adapt to meet the changed requirement. The requirement includes space, speed, time and so forth. When to make this re-configuration? How to make this re-configuration? What to do with this re-configuration? That's the
information to include in feature model to achieve adaptive computing.

Feature models are frequently used in domain analysis for describing common and different requirements for software systems as instances of a product line [14]. Feature models describe properties distinguishing common and variable requirements in several similar products or software systems. They structure requirements by generalizing them to concepts. The models should bridge the gap between requirements and the solution. They provide a very flexible means of description. Every feature covers a set of requirements.

Domain-Specific are used to express the feature model. It’s an important part of this project, because it connects the feature model and the program. The program need to get the adaptive-coordination information, and it is specified in that language. With this information, the program can make the right configuration or re-configuration and adapt to changes in its execution environment, for instance balance load to fulfill a performance requirement.

Adaptive applications are able to adapt their properties and resource requirements at runtime in response to dynamically varying user needs and resource constraints. Since these properties vary at runtime, there should be a solution to find the balance between the needs and available resources. Accordingly, the approach is to build an adaptive system. Adaptation can be achieved by reconfiguring the available components or the component composition. For example, a weather forecast system, it will require huge calculation work. Consequently, it will be very necessary to adaptive the algorithm for calculation at runtime to save time. We need to find the balance between time and veracity.

The first issue of this project will be the design of a suitable Feature Model, after that the second issue of this project is to use a DSL to express the Feature Model in a suitable way.

1.2 Problem Statement

Using a feature model to approach adaptive computing, first and foremost there should be a well designed feature model to make sure what will be built is able to adaptive. The solution of this feature model should be a universal solution, which means that it should be robust, adaptable and available.

The main issues for this project we worked on are:

1. **How to provide information about the configuration within the Feature Model for Adaptive Computing?**
2. **Design a Domain Specific Language (DSL) which is used to express dynamic adaptations to the feature model.**

Adaptive applications require information about configuration at runtime. Consequently, the task is to investigate existing models for formal feature modeling. And then develop a way to provide necessary configuration information within the Feature Model. To develop a way to provide necessary configuration information means to extend the formal Feature Model. Extend on this theory and design a formal feature model that may be used to express dynamic adaptations to the feature model.
Consequently, this thesis will investigate: “How to extend the Feature Model to provide necessary configuration information for adaptive computing”.

1.3 Method

Because there are three steps in this project, first to design the Feature Model for adaptive computing, second to design the suitable DSL to express this Feature Model and the last step is evaluation. The strategy is to find examples of small adaptive computing systems, and then analyze the requirement for adaptive computing and what is needed to express the structure and information in a feature model using these subject systems as a starting point.

Therefore, the method is “Design from examples and improve the solution until it provides everything required to satisfy the example systems.

1.3.1 Design strategy

Our work is based on Feature modeling and Adaptive computing. So first we build a model for feature trees. Then we investigate existing models and draw their feature trees according to the model. We select one of these models which have dynamic adaptations, and it is also fit to our later work. After that we start to analysis the model we chose and develop a language by which we may access and modify feature models of the feature model at run-time. By working at this specific example, we summarize a conclusion, and then generalize it to make it applicable to various entities.

The strategy of designing the DSL, first considering what kind of information should be contain in this DSL, second thinking about what technology will be used in the program which is related to DSL, moreover the most important strategy is to keep improving the theory by studying several example.

1.3.2 Evaluation strategy

To evaluate the work, the work should fulfill all requirements on a theoretical level. In the second step of the project we will develop several implementations that use the solution to achieve adaptiveness in the software. The first part will be considered a success, if the solution works in theory. For the second part, the criterion for success is to demonstrate that the solution works in an implementation. Successful achievements at the theoretical level and the implementation level, is a solid set of argument for considering the project a success.

Below we show an evaluation form that includes relevant information for evaluation activities.
Information type | Solution
---|---
System current state | -
Resource condition | -
Components combination information | -
Components restriction information | -
Parameter value | -
Priority for combination | -
Sequence for combination | -

Table 1.0.1: Evaluation Form

System current state: Including Hardware state information, available resources information and runtime state information etc.

Resource condition: the state of resource, available or not and how much resource is available.

Components combination information: Information about the combination among components.

Components restriction information: Information about the limitation of components, such as in a certain condition which component can be used.

Parameter value: The value of parameter.

Priority for combination: Information about the priority of component.

Sequence for combination: Information about the sequence of the combination.

Furthermore, there also something that would be considered into the evaluation:
- Efficiency
- Robustness
- Generality

Since the second step of this project, it is to express the Feature Model in a Domain-Specific Language (DSL); therefore it is also necessary to give an evaluation for this DSL.

<table>
<thead>
<tr>
<th>DSL function</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction Feature</td>
<td>-</td>
</tr>
<tr>
<td>&quot;AND&quot; Relation</td>
<td>-</td>
</tr>
<tr>
<td>&quot;OR&quot; Relation</td>
<td>-</td>
</tr>
<tr>
<td>&quot;==&quot; Comparison</td>
<td>-</td>
</tr>
<tr>
<td>&quot;&gt;=&quot; Comparison</td>
<td>-</td>
</tr>
<tr>
<td>&quot;&lt;=&quot; Comparison</td>
<td>-</td>
</tr>
<tr>
<td>&quot;&gt;&quot; Comparison</td>
<td>-</td>
</tr>
<tr>
<td>&quot;&lt;&quot; Comparison</td>
<td>-</td>
</tr>
<tr>
<td>&quot;!=&quot; Comparison</td>
<td>-</td>
</tr>
<tr>
<td>Dynamic function call</td>
<td>-</td>
</tr>
<tr>
<td>Notation</td>
<td>-</td>
</tr>
<tr>
<td>Import file</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1.0.2: Evaluation Form 2
1.4 Restrictions
This work is mainly theoretical. The focus is on developing a notation for expressing sufficient information for the integrated adaptiveness in an adaptive software package. Some implementation work will be carried out but especially demonstrators used in evaluation are sketchy and do not fully implement the applications described.

1.5 Structure of the thesis
Chapter 2 gives an overview about the theory and background information that are used in this thesis, here we explain what a feature model and adaptive computing is in theory and the practical implications from these frameworks. Chapter 3 is the main part of this thesis; it presents the final solution and its rationale. Chapter 4 is the evaluation of the solution. And the last chapter: Chapter 5 includes conclusions and future work.
2. Related Work

In this chapter, we introduce the theories and practical implementations that are relevant for the scope of this thesis. First we will introduce the theoretical framework and for the second part we will introduce the implementation technologies.

2.1 Theoretical framework

The theoretical framework mainly contains two parts: Adaptive computing and Feature Modeling. First we will introduce these two theories and then show the deployments system.

2.1.1 Adaptive Computing

Adaptive computing or Self-Adaptive software systems have the ability to adapt themselves at the runtime. Adaptive computing focuses on the methodology and implementation of systems that adjust to different situations. "An adaptive system may change its own behavior to the goals, tasks, interests, and other features of individual users and the environment. Adaptively is important for ubiquitous and pervasive computing." [1]

The purpose of the adaptation is to meet the changing requirements. As the development of software technology reaches new heights with more and more complex requirements and solutions, there is an increasing interest in letting the software have the ability to adapt dynamically to changes in resources, requirements, errors, and the whole runtime environment.

![Adaptive computing process](image)

*Figure 2.0.1: Adaptive computing process*

An adaptive computing system works generally as shown in Figure 2.1. A monitor gathers information from the runtime system. The information monitored can, for instance, be a certain value of some parameters or state of a specific piece of hardware.
After the monitor gets the information, the evaluation part begins. This is based on the information gathered from the runtime. The evaluation and analysis will provide more detailed information, for instance about where to enact the reconfiguration.

When developing system or software with Adaptive Computing, questions below question should be brought into consideration. “Under what conditions does the system undergo adaptation? Should the system be open-adaptive or closed-adaptive? What type of autonomy must be supported? Under what circumstance is adaptation cost-effective? How often is adaptation considered? What kind of information must be collected to make adaptation decisions?”[2]

It's certainly that it is unnecessary to do adaptation all the time; these adoptions should be done only under certain conditions. There also must be some information about resources, runtime and so forth to help to make the decision to do adaptation or not. Some parameters may very helpful when making decisions. The overhead of an adaptation must be considered too, because to manage adaptation may cost more overhead than without adaptation. What make the system/software most effective? That is the aim for adaptive computing. It is certainly that adaptive computing should bring every ingredient into consideration.

2.1.2 Adaptive Computing building blocks

There are several technologies for adaptive computing to achieve the goals.

● Monitoring:
The first technology for Adaptive computing is monitoring. As appear in most articles about adaptive computing, monitoring is core or very important technology for adaptive computing. Only based on the value that the monitor gets, the adaptation can be made. Monitoring will provide necessary information that the adaptive computing system need.

1. Performance monitoring. Performance monitoring will give the information about the overhead of memory, CPU time, periphery equipments communication states, network connection conditions and so on.
2. System events monitoring. Some events will trigger the diversification of the runtime.

This project is focus on the architecture level not code level, so we'll skip some detail on code level.

● Evaluation:
An evaluation should be making after get the information after monitoring ingredients which will be brought into the consideration of this whole program. Evaluation is a necessary to avoid the happening of inconsistent.

The evaluation will make the decision upon the one or several values that provides by the monitoring part.

● Reconfiguration:
After evaluation there will be an answer to the question "Do it or not?” If evaluation
gives a positive answer, then the system/software will take action to make necessary change in configuration that means reconfiguration to replace some code with some other code, or re-distribute the resources.

2.1.3 Feature modeling

Feature modeling plays an important role in our thesis project, thus it is necessary to know the knowledge of feature model.

1. Basic conceptions of feature modeling

In order to understand feature models, you should first understand the concept of a feature. A well formed software system can be divided into many parts according to their characteristics. Then each part could be described as feature. There are four types of features:

- **Capability feature:**
  - Service, Operation, Nonfunctional characteristics.
- **Domain technology feature:**
  - Domain method, Standard, Law.
- **Operating environment feature:**
  - Hardware, Software.
- **Implementation technique feature:**
  - Design decision, Communication, ADT [7].

Take a software video player for example, it can be divided into two central parts: Play and Media, for the part Play we can divide it into two parts: Decoders and Effects, and for the part Media, it can also be divided into Local Media and Stream Media. Instead of calling them parts, we call them features. They describe properties of a video player.

Feature model is an abstract illustration which is formed by a series of features; each feature connects to other features according to the specific relations between them. It can be used to describe characteristics of products in a product line and internal configuration of a software system. (A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.) For this usage, feature model becomes more and more popular in recently years; it is widely used in software exploitation. Figure 2.4 is a simple feature model for video player.
For feature model looks like a ‘tree’, then we can also call a feature model as a feature tree.

2. **Grouping features in a feature model**

In a feature model, features are grouped with other features according to the relations between them. There are several ways of how to connect features; one method is just connecting a feature and its sub-features by using a line. However, this way cannot show their relations clearly, it cannot show some details, such as whether the sub-feature is mandatory or optional. Another way is using specific signs to describe the details; this method can figure out the disadvantages in the first method. Three main notations are:

1: \[ \text{optional features} \]

2: \[ \text{mandatory features} \]

3: \[ \text{alternative features} \]

This method can achieve most feature models, but when it comes to the case that one feature has three or more sub-features, and only two features from the set of sub-features can be chosen, the above way to connect features could not achieve this case. So an advanced way is necessary, that is using multiplicities. Figure 2.5 shows the example of using multiplicities [8].

![Feature Model Example](VideoPlayerDiagram.png)
Different variants of multiplicities of the groups are possible:

- 0..1 at most one feature has to be chosen from the set of the sub-features,
- 1 exactly one feature has to be chosen from the set,
- 0..* an arbitrary number of features (or none at all) have to be chosen from the set,
- 1..* at least one feature has to be chosen from the set [8].

This method is the best way to connect features; it can achieve all common cases of feature models.

### 2.1.4 Application of feature model

The industrial importance of feature model is increasing rapidly in recent years. “Feature models should fill the gap between requirements specifications and design models and architectures.” [8]

Feature models play an important role in the activities of software product line development. To the people who handle software more as a black box, such as sellers and customers, a feature model can provide an overview over requirements; distinguish between common and variable properties, show dependencies between features and support decision about evolution of a product line. To the people who work on the development of a product line, such as architects, software developers, feature models can help them to define reusable components and separate them according to the separation of Concern principle, assign reusable components to variable features, describe dependencies and constraint between components and features, control the configuration of products out of reusable components [8].

By linking features to elements of design and implementation and requirements, we can see details of the product line [8]. Figure 2.6 is the linking diagram between features, requirements, design and implementation [8].

---

**Figure 2.0.3: Grouping features by using multiplicities**

Alternative of B, C and D

---
2.1.5 Domain-Specific Language (DSL)

A Domain-Specific language is most often a declarative that is designed for certain specific/particular one kind of task. In this project, the task is to design a DSL to express a Feature Model for Adaptive Computing.

There is no specific language or rule for this language, which means that designer should design their own rule about this language for their own domain requirements. In this project a DSL will be designed for Feature Model which can achieve adaptive computing.

2.1.6 Deployments system

In this project the deployments system basically contain two parts, one is the software or system, the other part is the configuration file or files.

Accordingly, the structure is a software/system contains a part that can deal the configuration file and make an interaction with it, and then make necessary re-configuration at runtime.

The configuration file/files will provide lots of information. What kind of object should be created at runtime? Which method should be called? And which parameters should be used in the method. The following figure shows this process.

![Figure 2.0.4: Links between features, requirements, design and implementation](image-url)

Figure 2.0.4: Links between features, requirements, design and implementation
2.2 Implementation technologies

The implementation technologies used in this thesis are: Unified Modeling Language (UML), Extensible Markup Language (XML), XPath and Java Reflection API.

2.2.1 UML class modeling

Unified Modeling Language (UML) is widely used in many fields of computer nowadays, especially in the part of building models.

1. Basic conceptions of UML class modeling

Obviously, model plays an important role in this thesis. So what is a model?

“A model is an abstraction of something for the purpose of understanding it before building it.” [3] For example, when an architect wants to build a building, the first thing he should do is to make a model for the building, once he get this model, he can show his idea to other people. Model is very common in our lives, for instance, a tellurion is a model of the earth.

Modeling means to build a model. It is very useful in development of software. Here are 4 main uses:

- Testing a physical entity before building it.
- Communication with customers.
- Before building it, the designers can show this model to their customers to see whether the customers were satisfied with it.
- Visualization.
- Reduction of complexity.

“Models reduce complexity by separating out a small number of important things to deal with at a time.” [4]

The class model describes the structure of objects in a system—-their identity, their relationships to other objects, their attributes, and their operations.

“Class diagrams express the class model. In the Unified Modeling language (UML), a class diagram is a type of static structure diagram that describes the structure of a system by showing the system’s classes, their attributes, and the relationships between the classes.” [5] The following is an example of class diagram [6]:

---

[12]
2. Class model for feature modeling
The following figure is a class diagram for feature modeling.

![Diagram](image)

**Figure 2.0.7:** A class diagram for feature modeling

There are 6 classes: Feature model, Features, Relationships, Mandatory, Alternative and Optional. The last three classes are inherited from class Relationships.
One instance of class Feature Model has one or more instances of class Features, an instance of class Features has 0, 1, or more instances of class Features. There are 3 kinds of relationships between feature and feature. Class Relationships is a super class, class Mandatory, Alternative and Optional are inherited from it. We will build feature models by following this class diagram.

### 2.2.2 XML

Extensible Markup Language (XML) is a markup language for general-purpose. It’s is an extensible language that users can define their own elements. The main purpose of XML is to share data or information between different types of systems. Considering the character of this language, it is convenient to use XML to store and transfer the information. The basic rule of this language is there should be a start-tag and an end-tag.

Well-formed: A well-formed document conforms to all of XML's syntax rules.

"As long as only well-formedness is required, XML is a generic framework for storing any amount of text or any data whose structure can be represented as a tree. The only indispensable syntactical requirement is that the document has exactly one root element (alternatively called the document element). This means that the text must be enclosed between a root start-tag and a corresponding end-tag. The root element can be preceded by an optional XML declaration. This element states what version of XML is in use (normally 1.0); it may also contain information about character encoding and external dependencies."

The following is a "well-formed" XML document:

```xml
<Thesis> Isn't this a good one? </Thesis>
```

As shown that is the basic necessary well-formed XML document.

The basic syntax for one element is:

```xml
<name attribute="value">Content</name>
```

The comment format is:

```xml
<! content >
```

The user should fill the “content” part with their information.

Above are the basic rules of XML. Obviously, it is a very extensible language. Due to this character, we will extend the rules of it and use it in our project.

### 2.2.3 XPath

**XPath** is a language for finding information in an XML document. XPath is used to navigate through elements and attributes in an XML document. [11] It can be seen as a kind of query language. XPath is a W3C standard. There are two versions now: XPath 1.0 and XPath 2.0. However, XPath 1.0 is still the more widely-used version, so here only describes XPath 1.0.

The XPath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria.
In popular use (though not in the official specification), an XPath expression is often referred to simply as an XPath. [10]

- **XPath expression:**

XPath uses path expressions to select nodes or node-sets in an XML document. The following is a simple XML example:

```xml
<author>
  <book/>
  <book/>
  <book/>
</author>
```

The simplest XPath takes a form such as: `/author/book`, which selects book elements that are the children of the element author, it will select 3 book elements of the XML file.

Expression: `/author//book/*[1]`, selects the first element (`'[1]'`), whatever its name (`'*'`), that is a child (`'/'`) of a book element that itself is a child or other, deeper descendant (`'///'`) of an author element that is a child of the current context node (the expression does not begin with a `'/`'). If there are several suitable book elements in the document, this actually returns a set of all their first children.

XPath can also select attribute node, text node, comment node and processing instruction node, table 2.1 gives how to select these four kinds of nodes by using XPath expressins.

<table>
<thead>
<tr>
<th>Expressions</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/author/book/@id='2'</code></td>
<td>Select the book element whose attribute id=2</td>
</tr>
<tr>
<td><code>/author/book/text()</code></td>
<td>Select the text node of element book</td>
</tr>
<tr>
<td><code>/author/book/comment()</code></td>
<td>Select all the comments in element book</td>
</tr>
<tr>
<td><code>/author/book/processing-instruction()</code></td>
<td>Select all the processing instructions in element book</td>
</tr>
</tbody>
</table>

*Table 2.0.3: Some common XPath expressions*

- **Examples:** [10]

Given a simple XML document:
The XPath expression:

/wikimedia/projects/project/@name
Selects name attributes for all projects, and

/wikimedia/editions
Selects all editions of all projects, and

/wikimedia/projects/project/editions/edition[@language="English"]/text()
Selects addresses of all English Wikimedia projects (text of all edition elements where language attribute is equal to "English"), and the following

/wikimedia/projects/project[@name="Wikipedia"]/editions/edition/text()
Selects addresses of all Wikipedias (text of all edition elements that exist under project element with a name attribute of "Wikipedia").

Using XPath API:
The following Java code shows using Xpath API to select one or several nodes from a XML document.

XPath xpath = XPathFactory.newInstance().newXPath();
String expression = "/author/book";
InputSource inputSource = new InputSource("books.xml");
NodeSet nodes = (NodeSet) xpath.evaluate(expression, inputSource, XPathConstants.NODESET);

2.3.4 Java Reflection API

“Reflection is a powerful tool. It lets you build flexible code that can be assembled at run time without requiring source code links between components.” [13]

“Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine.”
It gives the Java code access to internal information for classes loaded into the JVM and allows the programmer to write code that works with classes selected during execution, not in the source code. This makes reflection a great tool for building flexible applications.

By using reflection, first build a **class object** for a given class A, and using this class object to call the method in class A.

- **How to create a class object:**

There are many ways to create class object in Java, the following table shows several common ways to create a class object;

<table>
<thead>
<tr>
<th>Class object creating methods</th>
<th>Examples</th>
</tr>
</thead>
</table>
| Using `getClass()`          | String str = "abc";  
|                              | Class c1 = str.getClass(); |
| Using static method `Class.forName()` (This is the most common way.) | Class c1 = Class.forName ("Java.lang.String");  
|                              | Class c2 = Class.forName ("Java.awt.Button");  
|                              | Class c3 = Class.forName ("Java.util.LinkedList$Entry");  
|                              | Class c4 = Class.forName ("I");  
|                              | Class c5 = Class.forName ("[I");  
| Using syntax `.class`       | Class c1 = String.class;  
|                              | Class c2 = Java.awt.Button.class;  
|                              | Class c3 = Main.InnerClass.class;  
|                              | Class c4 = int.class;  
|                              | Class c5 = int[].class;  
| Using `Class.getSuperClass()` | Button b = new Button();  
|                              | Class c1 = b.getClass();  
|                              | Class c2 = c1.getSuperclass(); |
| Using syntax: `.TYPE`       | Class c1 = Boolean.TYPE;  
|                              | Class c2 = Byte.TYPE;  
|                              | Class c3 = Character.TYPE;  
|                              | Class c4 = Short.TYPE;  
|                              | Class c5 = Integer.TYPE;  
|                              | Class c6 = Long.TYPE;  
|                              | Class c7 = Float.TYPE;  
|                              | Class c8 = Double.TYPE;  
|                              | Class c9 = Void.TYPE;  

Table 0.4 Class object creating methods

- **Using reflection:**

In this thesis project, the technology of Java reflection is useful and the following example demonstrates the referent technology that will be used to solve the thesis problem.

```java
import Java.lang.reflect.*;
public class method2
{
    public int add(int a, int b)
```
public static void main(String args[])
{
    try {
        Class cls = Class.forName("method2");
        Class partypes[] = new Class[2];
        partypes[0] = Integer.TYPE;
        partypes[1] = Integer.TYPE;
        Method meth = cls.getMethod("add", partypes);
        method2 methobj = new method2();
        Object arglist[] = new Object[2];
        arglist[0] = new Integer(37);
        arglist[1] = new Integer(47);
        Object retobj = meth.invoke(methobj, arglist);
        Integer retval = (Integer)retobj;
        System.out.println(retval.intValue());
    }
    catch (Throwable e) {
        System.err.println(e);
    }
}

In this example, there is a class named method2 which has a method: add(), in the main method, it shows how the reflection works: first build a class object cls for class method2, and then using this class object to call the method add().
3. Feature models for adaptive computing

In this chapter we give a detailed explanation about the solution we find to solve the problems. In order to provide information about the configuration within the Feature Model for Adaptive Computing, a new feature type the **Restriction Feature** is introduced. We have also designed a Domain Specific Language (DSL) which can be used to express dynamic adaptations and constraints on these in the feature model.

3.1 Restriction Features

In this project the main task is to find a solution for expressing the restriction relation between features in a feature model. To achieve adaptive computing, there must be some rules to put into the feature models.

As discussed in chapter 2, for Adaptive computing, whether an adaptation should be made or not must be evaluated. Evaluation is an indispensability part of Adaptive computing. What is the most important thing for Evaluation? The answer is information. All evaluations are based on information. Therefore, for a Feature model which can achieve Adaptive computing must provide the system or software necessary information to do adaptive computing.

In building a feature model, things are considered in the aspect of “Feature”. What the information for adaptive computing is some information about the restriction relationship. That is “under what conditions what should be done”, “under what conditions how to do adaptation” and so forth. During the designing of the feature model of this server system, certainly this restriction should be contained in the model.

For a feature model, obviously restriction rules are features too. Simply we consider these restrictions are features. Consequently, we create a new kind of feature call “Restriction Feature” for feature model.

To give a definition to "restriction feature", restriction features are features for features. As an Adaptive system, some features must be optional. In that case it is necessary to make it clear that which feature should be chose during the executing of this program. The key issue here is to find the right way to express this "restriction rules" suitably, accurately, universally.

"Restriction features", means to add a feature to the feature which may need some restriction rules. What can be containing in this feature is that the feature the parent feature should obey. It's like the relationship between keys and door, "Restriction feature" Is the key, it can open some doors but not all the doors. These can be what else features it can be combine or what features it can not combine. In this way, the relationship between features can be expressed, and they are connected by an invisible line.

**Here is an example which expresses how this solution works.**

For a piece of software feature model, as figure 3.1, shows, it has three features 'A', 'B', 'C' and 'D'. Features 'B', 'C' and 'D' are three optional features. When feature 'A' was selected, one of 'B', 'C' and 'D' should be picked up for 'A', and there is a restriction
feature of 'A' to express what rules the 'A' should obey when selecting from 'B','C' and 'D'.

Figure 3.0.9: Example features
As shown in Figure 3.2 below, when feature 'A' is selected, then there should be a feature chosen to fit the feature 'A'. The system should choose one feature from 'B','C' and 'D'. The system will check the restriction feature of feature to figure out which rules to by obey to select the suitable features. In this example, system will read the restriction features of feature 'A', then it will know that only the Feature 'B' and Feature 'D' is available for Feature 'A', and the Feature 'C' is not suitable for 'A'.

In this way relation between features was built.

Figure 3.0.10: Restriction relationship between features
As this example shows, “restriction feature” can contains lot of information.

For a more common example in our model for a Video Player. In a video player program, there may be several decoders in one player. Here is a player with to two decoders, "Decoder 1" and "Decoder 2". There also are several effects that can be chosen for the decoders, but these effects are not suitable for every decoder. For
instance, there are certain effects that can be only used for a MEGP decoder and some effects can be only used for a RMVB decoder, which means they need to under the control of some restriction rules. When the system decide which decoder to use, then it will select one effect from effect 'E1', 'E2', 'E3' or 'E4'.

![Figure 3.0.11: Decoders and Effects](image)

When the system comes to the step to choose which effect for the decoder, it reads the restriction features of this decoder. If it the Decoder1, the restriction feature of Decoder1 shows that "Decoder1 should use Effect1 or Effect4 not Effect2 or Effect3". The Restriction Feature of Decoders also contains the information that it can use E2 or E3. As shown in figure 3.4, several invisible lines connect the features. In fact, these lines are Restriction Features.

Generally speaking, "Restriction Feature" is what we create to achieve adaptive computing. "Restriction Feature" expresses the information that how should the system to do adaptive computing. It connects optional features in an invisible way.
Generally, in our solution there are 4 steps to build a Feature Model which can achieve adaptive computing:

I. Analysis all the features of that certain system/software.
II. List all the ingredients will be used in adaptive computing.
III. Design the Restriction Features, use these features to provide necessary information for adaptive computing.
IV. Design the whole architecture and build the model.

3.2 Application of restriction features

For what we analysis in this project is a model of video player (Figure 3.2.1). This Video Player is able to play both local media file and stream media file. It has several decoders to decode different types of media or different band-width, which means that this video player system should be able to adjust different decoder for different media type. Moreover, there are also some visual effects that add to the decoder when decoding. These visual effects have the function to improve the quality or add special visual effects. Generally, the whole video player system is designed to be able to adaptive computing in two parts. First, it should be able to choose the right decoder to decode the media. Second, it will select one or several suitable visual effect to add to the media which is decoding. Could be used to choose the best fit stream decoder to decode in order to get the best quality under that bandwidth condition.

Generally, there are two types of media, local media and stream media. The
definition of "Local media" in this project is a visual media file that exists in the local machine. "Stream media" is a media transmit via certain network.

"Streaming media is multimedia that is constantly received by, and normally displayed to, the end-user while it is being delivered by the provider. The name refers to the delivery method of the medium rather than to the medium itself. The distinction is usually applied to media that are distributed over telecommunications networks, as most other delivery systems are either inherently streaming (e.g. radio, television) or inherently non-streaming (e.g. books, video cassettes, audio CDs). The verb 'to stream' is also derived from this term, meaning to deliver media in this manner."[9]

For a streaming media, the quality and the size of visual area is most lie on the band-width of the network. Obviously, adaptive computing should automatically adapt its decoder to fit the band-width condition.

<table>
<thead>
<tr>
<th>Effect1(E1)</th>
<th>Quality Improve Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effect2(E2)</td>
<td>Middle Visual Effect</td>
</tr>
<tr>
<td>Effect3(E3)</td>
<td>Interpolation Improve Effect</td>
</tr>
<tr>
<td>Effect4(E4)</td>
<td>Smoke Effect</td>
</tr>
<tr>
<td>Effect5(E5)</td>
<td>Firework Effect</td>
</tr>
</tbody>
</table>

*Table 3.0.5: Table of Effects*

<table>
<thead>
<tr>
<th>Local Media Decoder 1(D1)</th>
<th>RMVB format decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Media Decoder 2(D2)</td>
<td>AVI format decoder</td>
</tr>
<tr>
<td>Local Media Decoder 3(D3)</td>
<td>WMV format decoder</td>
</tr>
</tbody>
</table>

*Table 3.0.6: Table of Decoder*

<table>
<thead>
<tr>
<th>Stream Media Decoder 1(SD1)</th>
<th>Low band-width Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream Media Decoder 2(SD2)</td>
<td>High Quality Decoder</td>
</tr>
<tr>
<td>Stream Media Decoder 3(SD3)</td>
<td>Middle Band-width Decoder</td>
</tr>
</tbody>
</table>

*Table 3.0.7: Table of Stream Decoder*

According to the tables of feature above, the feature tree can be built, as Figure 3.5 shows.
Video Player Feature Tree

![Feature Tree Diagram]

*Figure 3.0.13: The final feature tree for a video player*

Restriction relationship between features:

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Decoders</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=0.5Mbps</td>
<td>SD1. Low band-width Decoder</td>
<td>E1;E3</td>
</tr>
<tr>
<td>&gt;0.5Mbps &amp; &lt;=1.0Mbps</td>
<td>SD2. High Quality Decoder</td>
<td>E2;E4;E5</td>
</tr>
<tr>
<td>&gt;1.0Mbps</td>
<td>SD3. Middle Band-width Decoder</td>
<td>E1;E2;E3;E4;E5</td>
</tr>
</tbody>
</table>

*Table 3.0.8: Restriction relationship between bandwidth, Decoders & Effects*

<table>
<thead>
<tr>
<th>Media Format</th>
<th>Decoder</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMVB</td>
<td>D1</td>
<td>E1;E4</td>
</tr>
<tr>
<td>AVI</td>
<td>D2</td>
<td>E1;E2;E3;E5</td>
</tr>
<tr>
<td>WMV</td>
<td>D3</td>
<td>E2;E4</td>
</tr>
</tbody>
</table>

*Table 3.0.9: Restriction relationship between Media format, Decoders & Effects*
In fact there is not direct relationship between Media Format/Band-width and Effects. Therefore, the work for next step is to figure the restriction features for 1: Decoders and Effects, 2: Band-Width and Decoders and 3: Media Format and Decoders

1. Relationship between Decoders and Effects feature

Restrictions relationship:

<table>
<thead>
<tr>
<th>Decoder</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>E1; E4</td>
</tr>
<tr>
<td>D2</td>
<td>E1; E2; E3; E5</td>
</tr>
<tr>
<td>D3</td>
<td>E2; E4</td>
</tr>
</tbody>
</table>

*Table 3.0.10: Restriction relationship Decoders & Effect*

<table>
<thead>
<tr>
<th>Decoders</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD1. Low band-width Decoder</td>
<td>E1; E3</td>
</tr>
<tr>
<td>SD2. High Quality Decoder</td>
<td>E2; E4; E5</td>
</tr>
<tr>
<td>SD3. Middle Band-width Decoder</td>
<td>E1; E2; E3; E4; E5</td>
</tr>
</tbody>
</table>

*Table 3.0.11: Restriction relationship S-Decoder & Effects*

As the table shown above, for each one decoder, there are only some of the effects can be add to the decoder during playing media.

Therefore, take D1 as an example, its Restriction Feature is "It's can only work with Effect 1 and Effect 4".

Follow the restriction table above, the tree below can be built. These different decoders have their own restriction features, and they should obey these restriction features.
Decoders and Effects

Figure 3.0.14: Feature model of Decoder and Effects part

2. Relationship between Band-Width and decoder

<table>
<thead>
<tr>
<th>Band-width</th>
<th>Decoders</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=0.5Mbps</td>
<td>SD1. Low band-width Decoder</td>
</tr>
<tr>
<td>&gt;0.5Mbps&amp;&lt;=1.0Mbps</td>
<td>SD2. High Quality Decoder</td>
</tr>
<tr>
<td>&gt;1.0Mbps</td>
<td>SD3. Middle Band-width Decoder</td>
</tr>
</tbody>
</table>

Table 3.0.12: Bandwidth and Streaming decoder

The table above shows the decoder that the player should use under different Band-Width conditions. Therefore, these Band-Width conditions have "Restriction Features". For each Band-width, its restriction will point to the Stream decoder it should use.

In this part, Restriction Feature shows the restriction of what Stream Decoder the player should use under that band-width condition. For example, when the bit rate is under 0.5Mbps, the restriction features will give the system the information that only the SD1 can be used to decode. Restriction feature provide the information that the configuration or re-configuration need.
3. Relationship between Local media format and decoders

<table>
<thead>
<tr>
<th>Media Format</th>
<th>Decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMVB</td>
<td>D1</td>
</tr>
<tr>
<td>AVI</td>
<td>D2</td>
</tr>
<tr>
<td>WMV</td>
<td>D3</td>
</tr>
</tbody>
</table>

Table 3.0.13: Media Format and Decoder

In the section, when the Player detects the format of the media type, it needs to choose a right decoder to decode. Therefore, it's easy to figure out that there should be a feature for each Media Format leading to the right decoder. In that case the Player software/system will adapt to right decoder.

As figure 3.8 shows, each Restriction feature will point to a feature.
4. **The final feature model for this video player**

After analyzing each section which will use adaptive computing and designing the restriction features, then the work is to put each ingredient together.
3.3 Restriction Language (RestrictionL)

Basing on the DSL we designed a language called Restriction Language (RestrictionL) to express the Feature Model for Adaptive Computing. The RestrictionL we designed is based on XML document, since there is a package named Java.XPath in the Java library, this package is used for reading XML documents. Therefore by using Java.XPath it is very convenient to read the RestrictionL files.

Since the RestrictionL file is a XML document, so it should obey all the syntax of XML document. Besides, in order to achieve the adaptive computing, we define some syntax in addition. The additional rules will be listed after in section 3.2.2.
3.3.1 Designing and using the RestrictionL

We choose a simple example to design and improve the RestrictionL. It is a sorting and saving system:

First, Sort a given sequence data X:

X can be an integer Array, or a string Array, or an integer Array-List, or a string Array-List.

Then, after sorting, save the sorting result as:

If Size ()>10, save as a Vector
If Size () <=10, save as an Array-List.

Process

The feature tree of this system is like this:
The restriction features are in the dashed frame.
The RestrictionL file of this System is like this:

```
<restriction>
  <sort>
    <rule>
      <if>restrict condition</if>
      <restrict>sort1()</restrict>
    </rule>
    <rule>
      <if>restrict condition</if>
      <restrict>sort2()</restrict>
    </rule>
    ......
  </sort>
  <save>
    ......
  </save>
</restriction>
```
3.3.2 RestrictionL rules

This RestrictionL file should obey the rules we defined as follows:

♦ These tag names: restriction, rule, if and restrict are mandatory. They can not be changed.

As shown in the example mentioned in 3.3.1, the tag name sort correspond to a method named sort () which has been implemented in the program, it can be changed according to different systems, for instance, if you apply RestrictionL on a Mobile-phone-Show-Pictures system, you can change the tag name sort to showPicture, hence there must be a method named showPicture (), which has been implemented in the program, correspond to the tag name: showPicture.

♦ The restrict condition: it is used to determine which feature (method) to use at runtime.

It may contain one or two restrictions that contain logical connectors: and, or. Each restriction is a logical condition which contains attributes or calls combined with values and logical operators (<, >, ≤, ≥, =, ≠).

Since it is illegal to contain these symbols: <, >, ≤, ≥, as text in XML documents. So we use other symbols to substitute them:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Substitutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>AND</td>
</tr>
<tr>
<td>or</td>
<td>OR</td>
</tr>
<tr>
<td>&lt;</td>
<td>LESS</td>
</tr>
<tr>
<td>&gt;</td>
<td>MORE</td>
</tr>
<tr>
<td>≤</td>
<td>LESS=</td>
</tr>
<tr>
<td>≥</td>
<td>MORE=</td>
</tr>
<tr>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>≠</td>
<td>!=</td>
</tr>
</tbody>
</table>

When using these symbols in restrict condition, you should obey the following 3 rules:

1. When it has method call in restriction, just add “()” after the called method name, for instance size() = 10. If there is no method call, just write the name of the attribute, it should be exactly the same in the code, such as screenWidth = 320, the screenWidth here is just an attribute of certain class.

2. It is mandatory to add a space before and after the symbol, for example size() = 10 not size()=10.

3. Do not use inverted comma for a string value, for instance type() = Array not type() = “Array”.

♦ The text in tag restrict stands for the feature (method) to select if the correspond restrict condition is true. In this part, just write the method name and add a pair of parentheses, such as: <restrict>sort1()</restrict>.
3.3.3 Designed Java classes

Accompany with the RestrictionL, two Java classes should be implemented, named as: RestrictionL and StringParser. They are used for parsing RestrictionL files and make calls for restriction. Class RestrictionL is used to parse RestrictionL files, pick up the restriction condition and judge whether the condition is true or false, if true then pick up the text of tag restrict from the RestrictionL file. These two classes can be applied on other adaptive computing examples without any change. Class StringParser is used to analyze the restriction condition, judge whether this condition is true or not. Meanwhile another class we named as MethodsSet is necessary, all of the methods are put in this class.

3.3.4 Comments in RestrictionL

Obviously, it is necessary to design a method to add comments in RestrictionL especially when the software or system is very big otherwise it will be very hard to understand and maintain the whole project.

Since our solution about RestrictionL is a language based on XML, and we use the Java Xpath to get the information from the file, therefore we decide to use the comments format in XML for our RestrictionL. The format is “<!......> ”.

Example:

```xml
<Restriction>
  <! Below the function is for adaptive computing and provide the information.>
  <function>
    <rule>
      <if>inputType() = Array AND elementType() = Integer</if>
      <restrict>sort1</restrict>
    </rule>
  </function>
</Restriction>
```

3.3.5 Importing information from other RestrictionL files

It is easy to handle a small software or system by one RestrictionL file, but when it comes to a large scale it will be very hard and complicated to handle and maintain it within one RestrictionL file. Therefore, it is necessary to design a method to solve this problem.

What can be imported? RestrictionL files, function performance profile files, data files and etc.

The format is:

```
"<restrict>SearchFile="filename.xx",Parameters=x;x;x;...</restrict>"
```

As shown above, when the system read this line, it will search a file and what exactly information are necessary is stored in the parameter list “Parameters”. 
Example:
One system is doing some calculation work. The word needed be done within a short time, so when it read the RestrictionL find out that the requirement is “speed” and the data type is “Integer”. And then the system will execute the line “<restrict>
SearchFile="profile.xx",Parameters=speed;Integer</restrict>”. The system will search file “profile.xx” and find the information about parameter “speed and Integer”.

3.4 Methodology
This section is about how to use our solution to implement a feature model which has adaptive computing. Below we will provide the working procedure (workflow).

3.4.1 Working procedure
Here is a particular procedure. Generally there are three processes to go through in the whole process of an implementation for adaptive computing, Feature Model Design, Translate Feature Model to RestrictionL and Implementation in Code.
First step: Feature Model design
In this step, someone will list all the features the product/software will have and the relationship between these features. This step the designer will clear what kind of adaptive computing this product/software will have. Finally, after this designer finished the designing work, a feature Model can be used.

Second step: Feature Model to RestrictionL
In this step, this designer will translate the Feature Model to RestrictionL. As the same time, designer will give the structure of the program. The designer also should specify methods, classes. Like an architect provides a blueprint of a building, every detail should be included in this step.

The last step: Implementation
For this step, the implementation work is for a professional programmer to fill the whole structure with code. Like a builder build a building according to the blueprint designed by the architect. This task is for programmer to finish.

3.4.2 Example for Methodology
In this example, a Compress & Decompress Software will be designed. Below are some features of this software:
✧ Different Compression and Decompression algorithms to different types of file.
✧ Three modes for compression and decompression: Speed, Middle and Size.
✧ Performance will be improved for each different computer.
✧ One or several files will be used to store the performance information for each individual computer. Which means for different computer the best configuration
of performance will be different.

- Optimize the computers with Multi-Processor/Core

**Step one: Feature model design**

In this step, someone will list all the features the product/software will have and the relationship between these features. This step the designer will clear what kind of adaptive computing this product/software will have. Finally, after this designer finished the designing work, a feature Model can be used. According to the basic features required, a designer will design a Feature model.

**Features:**

- **Compression:** Media Compression1, Media Compression1, Text Compression1, Text Compression2, General Compression1, General Compression2, Hybrid Compression, Multi-Processor compression.
- **Decompression:** Media Decompress1, Media Decompress2, Text Decompression1, Text Decompression2, General Decompression1, General Decompression2, Multi-Processor Decompression.
- **File types:** Audio file, Image file, Text file, Video file.
- **Compression & Decompression Mode:** Speed, Middle and Size. (These three will have three Restriction Features for each, which will lead to one or several performance profile file to help do configuration)

![Feature model for compression and decompression system](image)

Figure 0.19 Feature model for compression and decompression system
Step two: Feature model to RestrictionL

RestrictionL is as following shows:

```xml
<Restriction>
    <Compression>
        <rule>
            <if>Mode() = Speed AND SystemInfo() = Single Processor</if>
            <restrict>SearchFile="PerformanceProfile.dat",Parameters=getType()</restrict>
        </rule>
        <rule>
            <if>Mode() = Speed AND SystemInfo() = Multi-Processor</if>
            <restrict>SearchFile="PerformanceProfile2.dat",Parameters=getType()</restrict>
        </rule>
        <rule>
            <if>Mode() = Middle AND SystemInfo() = Single Processor</if>
            <restrict>SearchFile="PerformanceProfile.dat",Parameters=getType()</restrict>
        </rule>
        <rule>
            <if>Mode() = Middle AND SystemInfo() = Multi-Processor</if>
            <restrict>SearchFile="PerformanceProfile2.dat",Parameters=getType()</restrict>
        </rule>
        <rule>
            <if>Mode() = Size AND SystemInfo() = Single Processor</if>
            <restrict>SearchFile="PerformanceProfile.dat",Parameters=getType()</restrict>
        </rule>
        <rule>
            <if>Mode() = Size AND SystemInfo() = Multi-Processor</if>
            <restrict>SearchFile="PerformanceProfile2.dat",Parameters=getType()</restrict>
        </rule>
    </Compression>
</Restriction>
```
<Compression>
<Decompression>

<rule>
  <if>Mode() = Speed AND SystemInfo() = Single Processor</if>
  <restrict>SearchFile="PerformanceProfile.dat",Parameters=getType()</restrict>
</rule>

<rule>
  <if>Mode() = Speed AND SystemInfo() = Multit-Processor</if>
  <restrict>SearchFile="PerformanceProfile2.dat",Parameters=getType()</restrict>
</rule>

<rule>
  <if>Mode() = Middle AND SystemInfo() = Single Processor</if>
  <restrict>SearchFile="PerformanceProfile.dat",Parameters=getType()</restrict>
</rule>

<rule>
  <if>Mode() = Middle AND SystemInfo() = Multit-Processor</if>
  <restrict>SearchFile="PerformanceProfile2.dat",Parameters=getType()</restrict>
</rule>

<rule>
  <if>Mode() = Size AND SystemInfo() = Single Processor</if>
  <restrict>SearchFile="PerformanceProfile.dat",Parameters=getType()</restrict>
</rule>

<rule>
  <if>Mode() = Size AND SystemInfo() = Multit-Processor</if>
  <restrict>SearchFile="PerformanceProfile2.dat",Parameters=getType()</restrict>
</rule>

</Decompression>
</Restriction>
4. Evaluation

In this Chapter, we would like to evaluate our solution to see whether it can solve the expected problem or not. In our evaluation, we evaluate our solution both in theory way and in practice.

4.1 Evaluation for Feature Model with Restriction Features

As we set the designing goal, it is easy to figure out that the theory should achieve all the pre-request.

Based on the implementation from section 3.3, we can give an evaluation:

System current state: Yes, it can check the current state of the system; the information about the media type is the one of the system information.

Resource condition: Yes, the band-width is a resource condition. As we’ve provided in the implementation.

Components combination information: Yes, we have provided this information in our restriction features.

Components restriction information: Yes, the restriction feature is designed for that purpose.

Parameter value: Yes, from the design of our Feature model this is can be provided.

Priority for combination: This function works in theory, but in the example we were not able to include it, so this one needs future work.

Sequence for combination: This function works in theory, but in the example we were not able to include it, so this one needs future work.

Then, we can fill the evaluation table as:

<table>
<thead>
<tr>
<th>Information type</th>
<th>Restriction Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>System current state</td>
<td>YES</td>
</tr>
<tr>
<td>Resource condition</td>
<td>YES</td>
</tr>
<tr>
<td>Components combination information</td>
<td>YES</td>
</tr>
<tr>
<td>Components restriction information</td>
<td>YES</td>
</tr>
<tr>
<td>Parameter value</td>
<td>YES</td>
</tr>
<tr>
<td>Priority for combination</td>
<td>(Not included in Ex.)</td>
</tr>
<tr>
<td>Sequence for combination</td>
<td>(Not included in Ex.)</td>
</tr>
</tbody>
</table>

*Table 4.14: Evaluation form for theory*

For what defined at the beginning of this project, the first step is to find the solution which works in theory. The solution for adaptive computing in feature model we found after researching is “restriction feature”. As shown in 3.1.1, simply speaking, the solution “restriction feature” is a way considers the restriction relationship between features as a feature. These restriction features contains very important information which will be the key for the system to make the evaluation for adaptive computing.

As shown in 3.1.1, this put our theory in a video player. In this software model,
adaptive computing was used in “Band-width and Decoder”, “Media format and Decoder” and “Decoder and Visual effect”. Form the evaluation form we can figure out that the in the case we studied achieve almost all the items. Only the “Priority for combination” we cannot test it within this example.

These features will be connected in certain way and their relationship should be clarified. For this video player, the information it will want to know are “Which decoder should be chosen in certain band-width condition?” , “Which decoder should be chosen for certain media format?” and “What effects can add when certain one decoder is in using?”

The requirement in this example is clear, and the “restriction feature” provides what required.

Hence, as this model shows, this solution has the prospective function. Although this solution possibly has certain limitations which were not found in this model, this model proves that it works in some models.

### 4.2 Evaluation for RestrictionL

Based on the Implementation on the implementation in section 3.3 the Sort & Save System.

**Restriction Feature:** Yes, the features about Sort and Datatype will be connected together under the rule of Restriction Feature.

**“AND” and ”OR”:** Yes, the demo code has shown it can work well.

**Comparison(“==”,“<”,“>”,“>=”,“<=”,“!” and “!”):** Yes, they all work well in the demo code.

**Dynamic function call:** Yes, in the demo code, the system will call a method dynamic, which means the call is decided at runtime.

**Notation:** Yes, we just use the rule of XML to do notation in our RestrictionL.

**Importing file:** Yes, the RestrictionL has the function of importing one or several files.

The evaluation of the RestrictionL:

<table>
<thead>
<tr>
<th>RestrictionL function</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction Feature</td>
<td>YES</td>
</tr>
<tr>
<td>&quot;AND&quot; Relation</td>
<td>YES</td>
</tr>
<tr>
<td>&quot;OR&quot; Relation</td>
<td>YES</td>
</tr>
<tr>
<td>&quot;==&quot; Comparison</td>
<td>YES</td>
</tr>
<tr>
<td>&quot;&gt;=&quot; Comparison</td>
<td>YES</td>
</tr>
<tr>
<td>&quot;&lt;=&quot; Comparison</td>
<td>YES</td>
</tr>
<tr>
<td>&quot;&gt;&quot; Comparison</td>
<td>YES</td>
</tr>
<tr>
<td>&quot;&lt;&quot; Comparison</td>
<td>YES</td>
</tr>
<tr>
<td>&quot;!=&quot; Comparison</td>
<td>YES</td>
</tr>
<tr>
<td>Dynamic function call</td>
<td>YES</td>
</tr>
<tr>
<td>Notation</td>
<td>YES</td>
</tr>
<tr>
<td>Importing file</td>
<td>YES</td>
</tr>
</tbody>
</table>

*table 15*
Based on the program we had designed, we can get fill the evaluation form above. As shown, this RestrictionL solution can achieve all the requirements that we set at first.

Efficiency & Robustness: These are two very important aspects to evaluation about this solution, but it is impossible to give an evaluation without a real practice in code level to these two aspects. These two aspects needs further work on it.

4.3 Generality

We have designed a RestrictionL for adaptive computing, in order to verify our RestrictionL can work successfully, we’ve applied it on two small programs: sort & save system and Compression & Decompression system. As expected, the RestrictionL works well on this application. The goal is to design a RestrictionL for adaptive computing, that means this RestrictionL should be a universal language, it is not only can be used on the sort and save example, but also can be used on other applications which have adaptive computing. Therefore, this RestrictionL is a universal solution toward this topic.

Here, shows another example which has adaptive computing. Which is a video player as we mentioned before, this player can both play local media and online media. When it plays local media, according to the media format (RMVB, WMV, etc) it should choose different method (different method using different decoder) to play the media. When it tries to play online media, according to different bandwidth, it should also choose different method to play.

In order to implement this player by using RestrictionL, it is necessary to separate the work into two parts: one part is for designer, his work are design feature model for this program, possible restrictions and implement all the methods that are needed in RestrictionL, the other part is for the RestrictionL writer whose work is just to using our RestrictionL to write a document.

First, it is discussion about the work done by designer. Designer should design the feature model of this program at first; the feature model is as figure 1 shows. The possible restriction features are in the dashed frame. Then the designer should implement all the methods which will be used, and put them in a class, suppose this class name is MethodsSet. This class may look like this:

```java
public class MethodsSet {
    public localPlay(){
        Use Java reflection and call the method in Class RestrictionL to parse RestrictionL file.
    }
    public localPlay1(){
        Use decoder 1......
    }
}
```
public void localPlay2()
{
    Use decoder 2……
}

public void localPlay3()
{
    Use decoder 3……
}

public void onlinePlay1()
{
    Use decoder 4……
}

public void onlinePlay2()
{
    Use decoder 5……
}

public void onlinePlay3()
{
    Use decoder 6……
}

public String judgeLocalMediaType()
{
    .......
}

public String judgeBandwidth()
{
    .......
}
The next step for the designer to do is determine the workflow of the system, the workflow of the video player is like this:

```
<Restriction>
  <localPlay>
    <rule>
      <if>judgeLocalMediaType() = RMVB OR judgeLocalMediaType() = WMV</if>
      <restrict>localPlay1</restrict>
    </rule>
    <rule>
      <if>judgeLocalMediaType() = MPEG</if>
      <restrict>localPlay2</restrict>
    </rule>
    <rule>
      <if>judgeLocalMediaType() = AVI</if>
      <restrict>localPlay3</restrict>
    </rule>
  </localPlay>
  <onlinePlay>
    <rule>
      <if>judgeBandwidth() LESS= 2</if>
      <restrict>onlinePlay1</restrict>
    </rule>
    <rule>
      <if>judgeBandwidth() MORE 2 AND judgeBandwidth() LESS= 5</if>
      <restrict>onlinePlay2</restrict>
    </rule>
    <rule>
      <if>judgeBandwidth() MORE 5</if>
      <restrict>onlinePlay3</restrict>
    </rule>
  </onlinePlay>
</Restriction>
```

From this example, we can see our RestrictionL is general or universal; it can be...
applied on other adaptive computing programs. When using our RestrictionL, just follow the above steps we have presented, make sure the RestrictionL files obey our RestrictionL rules. As the examples shown, this RestrictionL is a universal solution for this problem. So far, we can use our RestrictionL to solve all the software and system we have at hand.

### 4.4 Robustness and Efficiency

About Robustness, as far as the implementation we have done. This solution is robust so far, as the code we have provided in the example: Sort & Save System. However it is necessary to do more tests in larger scale software or systems. We have not got that much resource and time to do further tests in large scale software. Therefore, due to this limitation we can not just come to the conclusion that this solution is robust. But, so far, it is robust.

Efficiency, due to the character the systems have to read one or several external files to make configuration and re-configuration, this will influence the efficiency of the whole system. The overhead will grow when it comes to larger scale systems. Therefore the efficiency is not good. But this compromise between performance and function should be made in this solution.
5. Conclusions and Future work

In this chapter, we would like to state our conclusions and discuss the future work that may be faced.

5.1 Conclusions

Feature model and adaptive computing are two separate techniques that have been researched and perfect by the researchers. Feature model is an abstract illustration which is formed by a series of features; each feature connects to other features according to the specific relations between them. And adaptive computing focuses on the methodology and implementation of systems that it adjusts itself to different situations. An adaptive system may change its own behavior to the goals, tasks, interests, and other features of individual users and the environment. Adaptively is important for ubiquitous and pervasive computing.

By combine these two separate techniques, it comes up a new technique: Feature modeling for adaptive computing, which we’ve been working on. In this thesis, our work is based on Feature modeling and Adaptive computing. So first we build a model for feature trees, as shown in 2.3. Then we investigate existing models and draw their feature trees according to the model. We select one of these models which have dynamic adaptations, that is a Video Player which can both play local media and stream media.

After analysis and research on the video player model, we find a method which could be used to express the adaptive information of a feature model. The method we find is to build a new kind of feature—**Restriction feature** for the model which has adaptive computing. It is a kind of feature for features (actually the successor feature of features), the solution for the problem that this thesis should solve is to add a successor feature to the feature which may need some restriction rules in order to achieve adaptive computing at run time. By working at this specific example, we summarize a conclusion, and then generalize it to make it applicable to various entities.

A new RestrictionL is designed for expressing dynamic adaptations to the feature model. The RestrictionL is based on XML format and accompany with some extra syntax which has been presented in section 3.2. Two Java classes are also very important since they are used for parsing and analyzing the RestrictionL documents. As those applications in chapter 3 and 4 shows, the RestrictionL works well and it is generalized.

Thus, the work we done are successful. But it can not detect whether this solution is efficient to all the models. In future, we may examine our solution to find it’s efficient.

5.2 Future work

We finally solve the problem. During our work on it, we found a few further problems,
but due to the limited time we have, we did not solve them. Perhaps we can solve them in future.

- **The further examine of this solution**

What we provide in this solution maybe has disfigurement or limitation which we did not bring into consideration. It's difficult to answer whether this solution will fit all situations. Therefore, further examine in more different models will shows the result.

- **Improvements**

Although the solution can figure out many feature models to approach adaptive computing, there are still two main disfigurements in this solution.

1. There is no evidence testifies that this solution is efficient or not. Therefore, there may be some improvement work to do to make this solution be efficient.

2. It’s an easy solution but may not a sufficient solution for some special models. In the future we may ameliorate this solution to make it be a sufficient solution for all models.

3. Extend the RestrictionL function: it will be necessary to extend the function of our RestrictionL, there might be some very useful function we have not considered in our solution. Once again, this improvement or extension work can be done during further examine work.

4. This RestrictionL solution is based on XML, but other language maybe also be good for this solution which maybe more efficient and sufficient.

Based the consideration of efficiency and sufficiency, there possibility be improvement or complementarily work to do for this solution. In the future, we may spend time on these aspects.
Bibliography

[1] Helsinki Institute for Information Technology [online: web]


[10] ”Xpath” Wikipedia [online: web]


[12] ”The Reflection API” [online: web]
    URL: http://72.5.124.55/docs/books/tutorial/reflect/index.html, 2008.05.03.


