Master of science thesis

**Object-relational mapping**

**Hibernate and Struts framework in Java**

Author: Bc. Nguyen Viet Cuong


Prague, January 2008
Declaration

I, Nguyen Viet Cuong, declare that this thesis, submitted in fulfillment of the requirements for the award of Master of Science at Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, is my wholly own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any academic institution. I also agree that the department can use this thesis for any non-commercial purposes in the future.

Nguyen Viet Cuong,
January 2008
Acknowledgments

I would like to express my gratitude toward my supervisor, Ing. Mannová Božena M.Math., without whose assistance and support this thesis would not have been possible. I would also like to thank my parents, my brother for their love, encouragements and faith in me. Many thanks also go to all my friends and all the people that helped me throughout this project.
Abstract

Web development exploded in the mid-1990s. With the emergence of the Java language, distributed Web applications could be built and simultaneously solved a lot of problems with the former client-server model with their new capabilities. Since then, Java has gradually become mature and dominant in the world of enterprise development. However, behind that glory, there are a lot of open source projects and third party communities that continuously contribute to the success of Java. In recent years, Hibernate and Struts stand firmly among those in the web application development area. There was a question whether open source tools could replace entirely the classical way of using heavy Enterprise Java Bean (EJB) containers. Within this project, we are going to find our own answer and judgement toward this doubt. Our process will begin with the idea of the object-relational mapping model and the tools currently available for working with Java. We shall focus on Hibernate, its advantages and disadvantages. Along with that, we shall get acquainted with Struts - an open-source Web Application Development Framework and analyze the usage of Hibernate in Struts projects. To really understand how it is possible building a web application using these open source tools, as an important part of this project, a Content Management System (CMS) as a web application using Hibernate technologies and Struts framework will be designed and implemented.
## Contents

Declaration 2

Acknowledgements 3

Abstract 4

1 Introduction 11

2 Object - Relational mapping 13
   2.1 The problem ................................................. 13
   2.2 What is ORM ................................................. 13
   2.3 The paradigm mismatch ...................................... 14
   2.4 Why ORM .................................................... 15
   2.5 Current tools, comparisons ................................ 16
      2.5.1 Criteria when choosing a ORM tool .................. 16
         2.5.1.1 Technical criteria ............................... 16
         2.5.1.2 Other important criteria ....................... 19
         2.5.1.3 Remarks .......................................... 20
      2.5.2 Current tools for Java ............................... 20
         2.5.2.1 Table of ORM tools comparison ................. 20
         2.5.2.2 Comments ........................................ 22

3 Hibernate 25
   3.1 Introduction ............................................... 25
   3.2 Hibernate 3 project features ............................. 26
   3.3 Architecture .............................................. 27
      3.3.1 From a simple view .................................. 28
      3.3.2 Instance states ...................................... 31
   3.4 Mapping concepts and methods ............................ 31
      3.4.1 Mapping declaration ................................ 32
      3.4.2 Mapping File Structure ............................... 32
         3.4.2.1 Doctype ........................................ 32
         3.4.2.2 Hibernate-mapping ................................ 32
         3.4.2.3 Class ........................................... 33
         3.4.2.4 Id .............................................. 36
         3.4.2.5 Discriminator .................................... 39
         3.4.2.6 Other elements ................................. 40
CONTENTS

3.4.3 Hibernate Types ............................................ 42
   3.4.3.1 Entities and values .................................. 42
   3.4.3.2 Basic value types ................................... 42
3.4.4 Mapping a class more than once .............................. 43
3.4.5 Metadata alternatives to mapping files ....................... 44
   3.4.5.1 5.5.1. Using XDoclet markup ......................... 44
   3.4.5.2 Using JDK 5.0 Annotations ............................ 44
3.5 Hibernate Query Language ..................................... 44
3.6 Conclusion ...................................................... 46

4 Apache Struts 2 49
   4.1 Introduction ................................................. 49
   4.1.1 History of Struts 2 ....................................... 49
4.2 Struts 2 features ............................................... 50
   4.2.1 The MVC Architecture .................................... 50
      4.2.1.1 Model ................................................. 50
      4.2.1.2 View ................................................ 50
      4.2.1.3 Controller .......................................... 51
   4.2.2 How this model applied in Struts 2 ....................... 51
   4.2.3 What Struts 2 could bring ................................ 51
   4.2.4 Looking into Struts 2 architecture ....................... 52
   4.2.5 What is so good about Struts 2 .......................... 55
4.3 Struts 1.x VS Struts 2.x ...................................... 58
4.4 Usage of Hibernate in Struts projects .......................... 60
4.5 Conclusion ...................................................... 61

5 Design and Implementation of a CMS system 63
   5.1 The requirements ........................................... 63
   5.2 What do we have in hands ................................... 64
   5.3 The architecture ............................................ 65
      5.3.1 The layered architecture .............................. 66
         5.3.1.1 The presentation layer ............................. 66
         5.3.1.2 The Persistence Layer .............................. 66
         5.3.1.3 The Business Layer ................................ 67
         5.3.1.4 The Domain Model Layer ........................... 68
   5.4 The real work ............................................... 68
      5.4.1 Setting up .............................................. 68
      5.4.2 Creating the work directory ............................ 69
      5.4.3 Database design ....................................... 72
      5.4.4 Application architecture ................................ 74
         5.4.4.1 Domain object layer ................................ 74
         5.4.4.2 Persistence Layer Configuration ..................... 76
         5.4.4.3 Business Layer Configuration ....................... 77
         5.4.4.4 Business Service Objects ......................... 79
         5.4.4.5 UI Layer Configuration .............................. 86
      5.4.5 Design and Implementation ............................. 87
         5.4.5.1 Login box .......................................... 87
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4.5.2</td>
<td>Administration interface</td>
<td>88</td>
</tr>
<tr>
<td>5.4.5.3</td>
<td>Article Manager</td>
<td>90</td>
</tr>
<tr>
<td>5.4.5.4</td>
<td>User Manager</td>
<td>95</td>
</tr>
<tr>
<td>5.4.5.5</td>
<td>Category Manager</td>
<td>97</td>
</tr>
<tr>
<td>5.4.5.6</td>
<td>Site Manager</td>
<td>98</td>
</tr>
<tr>
<td>5.4.5.7</td>
<td>File Uploading</td>
<td>99</td>
</tr>
<tr>
<td>5.4.5.8</td>
<td>Advertisement module</td>
<td>101</td>
</tr>
<tr>
<td>5.4.5.9</td>
<td>Image processing</td>
<td>102</td>
</tr>
<tr>
<td>5.4.5.10</td>
<td>Web interface</td>
<td>103</td>
</tr>
<tr>
<td>5.4.6</td>
<td>Conclusion</td>
<td>108</td>
</tr>
</tbody>
</table>

6 Conclusions 111

A Directory Structure 115

B Application Deployment 121
List of Figures

3.1 High-level view of Hibernate architecture ......................... 28
3.2 View of the runtime architecture ................................... 29
3.3 The “full” architecture with Connections and Transactions Management ........................................... 30

4.1 Struts 2 high level overview of request processing .................. 53
4.2 Struts 2 Architecture .................................................. 54

5.1 Framework architecture with Struts, Spring, and Hibernate .......... 65
5.2 Ant build successful .................................................... 72
5.3 Objects arranged without IoC. Object A creates objects B and C. .......... 78
5.4 Objects arranged with IoC .................................................. 79
5.5 Login box for CMS administration interface ........................ 88
5.6 CMS Administrator Interface ........................................... 90
5.7 Article Manager section ................................................ 91
5.8 Edit Article form ......................................................... 92
5.9 User Manager .......................................................... 96
5.10 Edit User page .......................................................... 96
5.11 Category Manager ...................................................... 97
5.12 Category Editor ........................................................ 98
5.13 Site Manager .......................................................... 98
5.14 Our simple Advertisement module ................................... 102
5.15 Web Layout ............................................................ 105
5.16 Final Web Layout .......................................................... 108

6.1 Employer search statistics on Monster.com - Resumes posted in a month 2007 . 112
6.2 Hibernate job trend at indeed.com from year 2005 to 2007 ............ 113
Chapter 1

Introduction

In today’s IT world, Java is a leading technology in the world of enterprise development. As management demands more from technology, complexity in infrastructure seems to grow exponentially, leaving many unable to keep up with the demand of such a fast-moving world. These complexity can be seen in the Java 2 Enterprise Edition (J2EE) specifications. However, the question whether J2EE is the best approach to enterprise development still needs to be judged. Is Java 2 Enterprise Edition the only option for enterprise web applications? Is it really the best way to do it?

While there were still questions needed to be answered, on the other side of the road, over time we started to witness the developers’ trend of making use of reusable Open Source tools that dramatically reduced the time taken to develop these applications. Developers started questioning which technologies were actually helping them and which did not satisfy their needs. In fact, some of the Open Source tools have already become favorites and many developers prefer using those tools instead of the heavy weight J2EE option. A lot of open source tools appeared in all layers and aspects within the process of building a web application.

Looking closer into the process, when working with the data persistence in particular, we can witness Hibernate - a popular object-relational mapping tool that has became a common choice among alot of developers. On the other hand, from upper layers, Struts, an open-source framework based on the MVC design paradigm for building web application running on Java platform has also become widely-used. These tools have helped reduce the complexity of enterprise Java development significantly. Furthermore, these technologies enable plain old Java objects (POJOs) to be deployed in light-weight containers versus heavy-handed remote objects that require heavy EJB containers.

To really see how this trend has become obvious, and whether Open Source tools nowadays really a better alternative to traditional heavy weight J2EE for building enterprise application, we will take the approach to our journey by first analyzing the object-relational mapping model in Java, some current significant tools in the market will be introduced. One of the best candidates among them - Hibernate will be focused in chapter 3. Coming up to higher layer, in chapter 4 we will introduce Struts 2 web application development framework, Struts 2 is the newest version of legendary Struts framework with the integration of WebWork from OpenSymphony. The usage of Hibernate in Struts applications is also analyzed. After that, applying the knowledge upon real world application, a CMS system as a web application will be built based on Struts and Hibernate technologies. The whole process of design, implementation and the experience along with it will be described in chapter 5. Finally we will sum
up our gained experience as well as some overview of current trends in the web application development in our conclusion chapter.
Chapter 2

Object - Relational mapping

“Unfortunately we need to deal with the object relational impedance mismatch” - Scott W. Ambler.

2.1 The problem

In object-oriented programming, everything looks like an object. In relational database, everything looks like a table. Certainly we see that an object and a table are two different concepts. However, as a programmer there are plenty of situations where we have to work with both object-oriented programming and relational databases, we surely have noticed that these are two different paradigms. The relational model deals with relations, tuples and sets - it is very mathematical by nature. The object-orientated paradigm however deals with objects, their attributes and associations to each other. As soon as we want to make objects persistent using a relational database, one issue will appear: There is a gap between these two paradigms, the so called object-relational gap. A object-relational mapper (ORM ) is a tool to bridge that gap.

In this chapter we are not going to describe the theory behind object relational mapping but more practically our purpose is to have an overview over the tools that currently available for Java in the market and perhaps giving a guideline and experience on picking up an appropriate ORM tool.

2.2 What is ORM

First thing first, what is ORM? Some definitions can be found but generally we can agree upon the following idea:

Object-Relational mapping is a programming technique for converting data between incompatible type systems in databases and object-oriented programming languages. In relational database we have data represented in the form of table with columns and rows, however in most of object-oriented programming languages, data often appears in the form of some objects. A good mapping tool is the key to compensate this gap. There are both free and commercial packages available that perform object-relational mapping, although some programmers may prefer to create their own ORM tools.

Difficulties occur when we have to deal with the real systems implementing object and relational models. These systems have implementations that are deficient or inconsistent with
the theoretical approaches. Relational databases have been deficient for multiple decades in correctly implementing the core concepts of relational theory. On the other hand, object modeling is not standardized, so each programming environment implements its own variation. Because of these deficiencies object-relational mapping is more complicated than it needs to be.

Fortunately, object modeling and relational modeling have such different concerns that they are actually extremely compatible. Relational theory is concerned with knowledge and object techniques are concerned with (primarily) behavior. Mapping between the two models requires deciding how the two worlds can refer to each other. In the next section we will describe the difference between these two worlds and what are the problems when these two need to be integrated.

### 2.3 The paradigm mismatch

The tabular representation of data in a relational system is still fundamentally different than the networks of objects used in object-oriented Java applications. We often see the object-relational impedance mismatch in development of applications, and we frequently see that the importance and cost of this mismatch is underestimated. Nevertheless, the good news is we now have a range of tools and solutions available to deal with this problem.

The object-relational mismatch comes into play when the data store is an SQL-based relational database management system. For instance, a network of objects can’t be saved to a database table, it must be disassembled and persisted to columns of portable SQL datatypes. The paradigm mismatch can be seen as the following problems:

- **The problem of granularity:** Granularity refers to the relative size of the types we’re working with. For example, what if we want to add a new datatype to our database to store an instance of a Java class representing a Customer (which has name, address, ...) in a single column?

- **The problem of subtypes:** In Java, we implement type inheritance using superclasses and subclasses. SQL should probably include standard support for supertables and subtables?

- **The problem of identity:** Although the problem of object identity may not be obvious at first, we could encounter it often in a more complicated system, such as when we need to check whether two objects are identical. It is easy in the object oriented world, but how do we do that effectively in the relational database world?

- **Problems relating to associations:** In our domain model, associations represent the relationships between entities. Object-oriented languages represent associations using object references; but in the relational world, an association is represented as a foreign key column, with copies of key values (and a constraint to guarantee integrity). There are substantial differences between the two representations. How do we resolve this?

- **The problem of data navigation:** There is a fundamental difference in the way to access data in Java and in a relational database. In Java, when we access a user’s billing information for instance, we could call something like `User.getBillingDetails().getAccountNumber()`. This is the most natural way to access object-oriented data, and it’s often described as
walking the object network. You navigate from one object to another, following pointers between instances. Unfortunately, this isn’t an efficient way to retrieve data from an SQL database.

We now have quite a list of object-relational mismatch problems, and it will be costly (in time and effort) to find solutions. This cost is often underestimated, and I think this is a major reason for many failed software projects. Statistics shows that the main purpose of up to 30 percent of the Java application code written is to handle the tedious SQL/JDBC and manual bridging of the object-relational paradigm mismatch. Despite all this effort, the end result still doesn’t feel quite right. The next section will show us how an ORM tool could bring positive benefits into these problems.

2.4 Why ORM

What do we get from using an ORM tool? What are the reason for using it? Clearly a supposed advantage of ORM is that it relieves developers from messy SQL. Hibernate for example is an ORM tool that promises to relieve the developer from more than 90 percent of common data persistence related programming tasks. Let’s look closer at some of the benefits of ORM.

Productivity

Writing persistence-related code in a Java application is often boring. Using a ORM tool helps eliminate much of this boring work and lets us concentrate on the business problem. ORM used together with the appropriate tools, will significantly reduce development time, hence increase overall productivity.

Maintainability

Using object/relational persistence substantially reduces lines of code. Fewer lines of code make the system more understandable, because it emphasizes business logic rather than plumbing. Most important, a system with less code is easier to refactor. This makes the system easier to maintain and more coherent to debug, allowing more elegant use of object orientation on the Java side, and insulating each model from minor changes to the other.

Performance

You may surprise with this performance claim and may suppose that hand-coded persistence can always be at least as fast, and can often be faster, than automated persistence. This is true in the same sense that it’s true that assembly code can always be at least as fast as Java code. However, this implication will be true only if the effort required to implement at-least-as-fast hand-coded persistence is similar to the amount of effort involved in utilizing an automated solution. The really interesting question is what happens when we consider time and budget constraints? In a project with time constraints, hand-coded persistence usually allows you to make some optimizations.

Furthermore, automated persistence using ORM improves developer productivity so much that we can spend more time hand-optimizing the few remaining bottlenecks. Finally, the people who implemented your ORM software probably had much more time to investigate
performance optimizations than we have. Did you realize that updating only the changed columns of a table can be significantly faster for some databases but potentially slower for others? In your handcrafted solution, how easy is it to experiment with the impact of these various strategies?

**Vendor independence**

An ORM abstracts our application away from the underlying SQL database and SQL dialect. If the tool supports a number of different databases (and most do), this confers a certain level of portability on your application. We shouldn’t necessarily expect write-once/run-anywhere, because the capabilities of databases differ, and achieving full portability would require sacrificing some of the strength of the more powerful platforms. Nevertheless, it’s usually much easier to develop a cross-platform application using ORM. Even if we don’t require cross-platform operation, an ORM can still help mitigate some of the risks associated with vendor lock-in. In addition, database independence helps in development scenarios where developers use a lightweight local database but deploy for production on a different database.

### 2.5 Current tools, comparisons

Now that we know a ORM tool could bring a lot benefits to our application, but how could we pick the right one to maximize these benefits? The fact is there are many object-relational mapping tools are available to help. Hence, choosing the right one is important. However, this task is not easy, the choice that we are about to make may effect the success of a whole system. The good news is, no matter it is an easy task or a difficult one, we could still atleast pick up one based on some criteria that we could look at before making our decisions. The following part will give us a more comprehensive view on that.

#### 2.5.1 Criteria when choosing a ORM tool

##### 2.5.1.1 Technical criteria

Objects relational mapping tools in the market are various. As we all know, when choosing an object-relational mapping tool to suit the needs of our application, several criteria should be kept in mind, and certainly it will depend on on a lot factors, from features to flexibilities, from performance to design and even financial aspects (commercial - non commercial ) etc. In terms of functionality and technical features , the following criteria specific to object-relational mapping should be considered:

1. **Basic features**
   - Be able to use inheritance, create hierarchies between entities, and use polymorphism.
   - Can handle any type of relations (1-1, 1-n, n-n).
   - Support for transactions.
   - Aggregates (equivalent to SQL’s SUM, AVG, MIN, MAX, COUNT).
   - Support for grouping (SQL’s GROUP BY).
2. Extended features

- Supported databases. A big advantage of mapping tools is that they provide an abstraction of the underlying database engine. Most of them allow switching easily between RDBMSs (Relational Database Management Systems).
- Query language (OQL - Object Query Language, OPath). We very frequently have to execute dynamic queries. It’s the case at least with searches based on filters. It is important to be able to use a powerful query language.

3. Flexibility

- Customization of queries. We often need to go beyond what is possible with the provided query language. In these cases, we need to be able to provide custom SQL queries. A good example is HQL, which is a strong point of Hibernate, allows for this. We could also wish a dynamic mapping to be possible from developer provided SQL queries.
- Support any type of SQL joins (inner join, outer join)
- Concurrency management (support for optimistic and pessimistic approaches)
- Support for the data types specific to the database management system (identity columns, sequences, GUIDs, autoincrements)
- Be able to map a single object to data coming from multiple tables (joins, views). Most of the tools handle a direct mapping of a class to one table. We often need more.
- Be able to dispatch the data from a single table to multiple objects.

4. Assistance, ease of use

- GUI to set up the mapping. Such a graphical tool presents the relational data model and lets you specify the objects to be created or at least the links between the objects and the tables.
- Generation of the classes. This can speed up the development, even if in a lot of cases we prefer to map the database to hand-coded classes or to classes generated from UML for example.
- Generation of the database schema. Some tools work only with a database they generated. This can be a big constraint, especially if you have to work with a legacy database of course. Otherwise, it all depends on whether you are an expert in database modeling, or if you prefer not to have to deal with the database schema.

5. Optimizations, performance, design

- Global performance (good implementation of the object-relational mapping concept)
- Lazy loading (the loading of some data is deferred until it’s needed)
  - For the data through relations.
CHAPTER 2. OBJECT - RELATIONAL MAPPING

• For some columns. When we want to display just a list of names, we don’t need all the columns of a table to be loaded. We may need the blob fields only at certain point, under certain conditions, and so it’s better to load them only at that time.

▶ Cache dynamically generated queries, so that they don’t get rebuilt at each call.
▶ Cache some data to avoid too many calls to the data source.
▶ Optimized queries (update only the modified columns; detect situations where the number of executed queries can be reduced).
▶ Handle circular references without duplication of objects.
▶ Handle cascade updates. Deleting a master record should delete the linked details if wished so.
▶ Bulk updates or deletions. When we want to update or delete thousands of records at a time, it’s not possible to load all the objects in memory, while this can be easily and quickly done with a SQL query (DELETE FROM Customer WHERE Balance < 0). Support from the tool is welcome to handle such massive operations without having to deal with SQL. A tool like Hibernate is not very good on this point for example.

6. Evolution, compatibility

▶ Maintainability (what happens if the database schema changes? If I need to add a new collection?).
▶ Possibility to move to a new mapping tool (what would it imply? At what cost?).
▶ Serialization. Serialization can be used to persist data outside of the database. Serialization can be done into a binary format, or in XML.
▶ Distributed objects (remoting, web services, requires support for serialization).

7. Additional features

Sometimes, an additional feature can make a huge impact on the choice of a tool. Some of the following could be one of those:

▶ Freedom in the design of the classes (no base class for the entities; no mandatory interface; no specific class for collections). Support POJO (Plain Old Java Object).

▶ Less constraints as possible on the database schema (eg. support for composite data keys)

▶ State information on data. It can be useful to know by looking at an object if the entity has been added, modified, deleted.

▶ External mapping file or not? Attributes (annotations) in code or not?

• Advantages of external files: mapping entirely externalized, no intrusion in the classes and these files can be generated.
• **Disadvantages of external files:** We have to deal with one or multiple additional files, with syntax to learn if no GUI is provided. To understand the links between the code and the database requires some effort;

• **Advantages of attributes/annotations:** everything at hand at code-level, the mapping is obvious since directly present on classes and class members, can be used to generate external mapping files using reflection if needed. If the mapping tool isn’t used anymore, they are ignored.

• **Disadvantages of attributes/annotations:** the code is "polluted”; the code depends on a specific mapping framework.

▶ Advanced compatibility with the development platform.

▶ Support for disconnected mode (fill objects from a database, close the connection, the session, create/update/delete some objects, and apply this modifications back to the database later).

▶ Interceptors and delegation mechanisms to be able to react when the handling of the persistence happens (eg. to be able to log what happens).

▶ Support for stored procedures. The advantages of stored procedures compared to dynamic SQL queries make for a hot debate, but it is better to have the choice.

▶ Filtering objects in memory (without having new queries executed on the database).

▶ Be able to defer the updates on the database, and apply them at a given time using a specific method call, instead of having them systematically applied immediately.

### 2.5.1.2 Other important criteria

Beside those more technical criteria as we see above, there are also alot of other criteria that could influence our final choice and should be considered in our checklist:

▶ Price (free or commercial, what is the price?).

▶ Performance.

▶ Resource consumption (memory).

▶ Scalability.

▶ Ease of use, time to be up and running.

▶ Flexibility.

▶ Documentation. Using a mapping tool is not always a snap. Check the quality of the documentation and the provided samples.

▶ Maturity. Many tools are relatively new, and are still not mature. Some are still at the beta level, or even alpha.

▶ Frequency of the updates, bug fixes, evolutions.
CHAPTER 2. OBJECT - RELATIONAL MAPPING

- Support, forums, community. This would help a lot in debugging, or finding the answer to our questions.

- Vendor’s reputation and stability. This is a big problem today. There are a lot of tools, but natural selection is constantly at work. Which products will still be there tomorrow? Some have already fallen. Time will probably continue to eliminate some actors in this very competitive market.

- Source code provided or not.

- Support for multiple platforms (Windows and Linux?).

2.5.1.3 Remarks

As we can see, the criteria are numerous. Hence, we should perhaps start by defining which of them are critical for us (those must-have) and which are less important (those nice-to-have).

After that, we have to make our own decision. The fact is developers have different opinions and different approach on any same problem. As with any tool, the experience is not hesitate to test extensively. We could download evaluation versions, do more than simply taking a look at the provided demos, use the tools on a prototype in our own application domain. Then we could evaluate, compare, use, criticize and comment on each tool that we have experienced. This is not a simple task, however, this is the place where the role of an open source community comes into play.

2.5.2 Current tools for Java

In this section we will have a look at some major ORM tools that currently available for Java. As we see from those criteria above, the choice of using which ORM tool depends pretty much on how our application requires and the experience of the developers. The table bellow is a quick comparison on the features of some major tools currently available.

2.5.2.1 Table of ORM tools comparison

Table of features across different tools is as follow:\(^1\)

---

\(^1\)The symbol “?” in the table means the feature is unknown or unspecified.
### 2.5. CURRENT TOOLS, COMPARISONS


2.5.2.2 Comments

- It is always good to enjoy something "free" as in "free beer" - anyone can use for free. The sense is when we get a free beer we can freely enjoy it, however the recipes are somehow kept secretly from us. We can see there are a lot of wonderful tools available for free in the sense of "free beer" like Hibernate, Apache Cayenne, DataBind. Besides, there are also tools that are quite good and the price can be considered cheap like Enterprise Objects Framework or FireStorm. Commercial tools are available with various prices.

- "Free" as in "free speech", this means that it is not only free but we also know all the words that contribute to the speech. In other words, the recipes are available publicly and these are called fully open-source tools. A good thing is open-source ORM tools are numerous, we can see Castor, Apache Cayenne, Hibernate, Data Bind, Expresso, Hibernate, Hydrate among those. In my opinion, some of the open source projects are highly mature and complete, some of them could really compete to those commercial and even are better in some aspects. This is one of the reasons why tools like Hibernate are so popular and widely used.

Talking about other features, we could see the following points:

- A GUI for designing mapping is a nice-to-have feature however an experienced developer often doesn't find it much different in comparison with mapping using XML or other configuration files.

- There is also some tools that require some special superclass or interface in order to persist an object (SimpleORM or Expresso for instance), but others can just persist any arbitrary classes.

- In term of RDBMS support, almost all the tools can support various types or provided with an adapter for this purpose. For example, Cayenne ships with adapters for Oracle, Sybase, MySQL, PostgreSQL, HSQLDB, Firebird, OpenBase, DB2, SQLServer, and a generic JDBC one. Hibernate: JDBC, special for Oracle, DB2, Sybase, MS SQL Server, PostgreSQL, MySQL, HypersonicSQL, Mckoi SQL, SAP DB, Interbase, Progress, Pointbase.

- In term of support to other features, we can see that most of the tools support grouping (GROUP BY clause), EJB support, mapping supports aggregate functions (count(), avg(), etc.), relationships between objects, includes full support of lazy resolution of all queries. It’s a pity that some tools like Lido, SimpleORM doesn’t support aggregate functions.

- A pretty handy feature is the ability to generate mapping as well as the Java Objects themselves, so we don’t duplicate information in the Java Objects and the related mapping information. For example in Abra, mapping, SQL, and classes are all generated from XML.

- Aggregate Mappings is needed when a single field maps to multiple fields in database and support composite primary keys, however not all the tools have this ability. In some tools they probably think that primary keys should be unique DB ids not related to any business specification.
Like any other tiny thing in object oriented programming, the support of inheritance and polymorphic queries are vital. In Hibernate for instance we can see all three mapping strategies: table-per-hierarchy, table-per-concrete-class, table-per-subclass.

The ability to support logging all the executed SQL statements in a SQL file is a handy feature when we need to debug and find error from a piece of code. Most of the tools currently available in the market have this feature. Hibernate, Hydrate, Lychee, Cayenne have a very good logging mechanism. We can log anywhere, via log4j, we can even email the logs.

In terms of multiservers and support of clustering and simultaneous access by other applications without loss of transaction integrity, tools like Hibernate, Cayenne and all commercial ones have done it quite well, however some tools don’t support this feature at all. Another good feature is the support for massive updates (with batch and transaction size tuning) which could be seen in Cayenne and some commercial tools.

The comparison table gives us a comprehensive view on what features are supported across different tools. From the table, we can see that Hibernate has a lot of good qualities. In this project, we will focus on Hibernate, not only because it is free as in “free beer” and open source as a “free speech” but also because of its luxurious features and widely used community.
Chapter 3

Hibernate

“Just because it is possible to push twigs along the ground with one’s nose does not necessarily mean that that is the best way to collect firewood” - Anthony Berglas.

3.1 Introduction

Thinking about the quote from Anthony Berglas, we could clearly see that using our noses to collect firewood is definitely not an efficient way. However, when looking further into the programming world, we could also see that this was not only the problem happening with collecting firewood but we also encountered the same problem when working with data persistence in Java. Luckily, a solution called object-relational mapping (ORM) now has wide acceptance and Hibernate is an open source implementation of it.

Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities and can significantly reduce development time otherwise spent with manual data handling in SQL and JDBC.

Hibernate’s goal is to relieve the developer from 95 percent of common data persistence related programming tasks. Hibernate may not be the best solution for data-centric applications that only use stored-procedures to implement the business logic in the database, it is most useful with object-oriented domain models and business logic in the Java-based middle-tier. However, Hibernate can certainly help you to remove or encapsulate vendor-specific SQL code and will help with the common task of result set translation from a tabular representation to a graph of objects.

A practical way to get acquainted with a new tool is perhaps to write a demo code or some classical “Hello world” applications. If you wish to see to the implementation section, you can jump straightly to chapter 5 - “Design and Implementation” to see how Hibernate was used in our implementation. The sections below however will show you the architecture and features of Hibernate. This will give us a sense of what Hibernate is capable of and can also act as a reference to the usage and features of this tool.
3.2 Hibernate 3 project features

Hibernate 3 can be downloaded and freely used under LGPL open source license from www.hibernate.org. The LGPL open source license allows the use of Hibernate and NHibernate freely in both open source and commercial projects. Hibernate project comes with several packages and extensions:

- **Hibernate Core** Hibernate for Java, native APIs and XML mapping metadata.
- **Hibernate Annotations** Map classes with JDK 5.0 annotations.
- **Hibernate EntityManager** Standard Java Persistence API for Java SE and Java EE.
- **Hibernate Shards** Horizontal data partitioning framework.
- **Hibernate Validator** Data integrity annotations and validation API.
- **Hibernate Search** Hibernate integration with Lucene for indexing and querying data.
- **Hibernate Tools** Development tools for Eclipse and Ant.
- **NHibernate** The NHibernate service for the .NET framework.
- **JBoss Seam Framework** for JSF, Ajax, and EJB 3.0/Java EE 5.0 applications.

The Hibernate Core is also known as Hibernate 3.2.x, or Hibernate. It’s the base service for persistence, with its native API and its mapping metadata stored in XML files. It has a query language called HQL (almost the same as SQL), as well as programmatic query interfaces for Criteria and Example queries. There are hundreds of options and features available for everything, as Hibernate Core is really the foundation and the platform all other modules are built on.

We can use Hibernate Core on its own, independent from any framework or any particular runtime environment with all JDKs. It works in every Java EE/J2EE application server, in Swing applications, in a simple servlet container, and so on. As long as we can configure a data source for Hibernate, it works. Our application code (in persistence layer) will use Hibernate APIs and queries, and our mapping metadata is written in native Hibernate XML files.

Hibernate Annotations is a new way to define application metadata became available with JDK 5.0: type-safe annotations embedded directly in the Java source code. Many Hibernate users are starting to get acquainted with this concept, as the XDoclet software supports Javadoc metadata attributes and a preprocessor at compile time (which, for Hibernate, generates XML mapping files).

Hibernate EntityManager is a small wrapper around Hibernate Core that provides JPA compatibility. Hibernate Annotations should be considered in combination with Hibernate EntityManager.

In our project, we will focus on Hibernate Core, the most important, widely-used and crucial part of the Hibernate project.
3.3 Architecture

In general, looking from a simple logic, we can see that to use Hibernate, it is required to create Java classes that represents the table in the database and then map the instance variable in the class with the columns in the database. Then Hibernate can be used to perform operations on the database like select, insert, update and delete the records in the table. Hibernate automatically creates the query to perform these operations.

Hibernate architecture has three main components:

1. **Connection Management Hibernate**: Connection management service provide efficient management of the database connections. Database connection is the most expensive part of interacting with the database as it requires a lot of resources of open and close the database connection.

2. **Transaction management**: Transaction management service provide the ability to the user to execute more than one database statements at a time.

3. **Object relational mapping**: Object relational mapping is technique of mapping the data representation from an object model to a relational data model. This part of the hibernate is used to select, insert, update and delete the records form the underlying table. When we pass an object to a Session.save() method, Hibernate reads the state of the variables of that object and executes the necessary query.

Hibernate provides a lot of flexibility in use. There is a “Lite” architecture when we only uses the object relational mapping component. While in “Full Cream” architecture all the three component Object Relational mapping, Connection Management and Transaction Management) are used.
3.3.1 From a simple view

Hibernate uses the database and configuration data to provide persistence services (and persistent objects) to the application. Objects will be persisted to database using data from configuration files, these files include XML mapping files and Hibernate properties files. Figure 3.1 demonstrates that view from high-level.

Looking closer with more detailed view of the runtime architecture, we can see the two alternatives. The “lite” architecture has the application provide its own JDBC connections and manage its own transactions. Therefore, connections and transactions management from Hibernate is not needed. This approach uses a minimal subset of Hibernate’s APIs (Figure 3.2).
On the other hand, using another alternative, we can witness the “full Hibernate” architecture in which the application is taken away from the underlying JDBC and JTA APIs and lets Hibernate take care of the details. This approach utilizes fully connections and transactions management from Hibernate (Figure 3.3).
For better understanding the definitions of the objects in the diagrams and how they were presented in Hibernate Core, we can break them down into the following items:

- **SessionFactory (org.hibernate.SessionFactory)**: A threadsafe cache of compiled mappings for a single database. A factory for Session and a client of ConnectionProvider. Might hold an optional (second-level) cache of data that is reusable between transactions, at a process- or cluster-level.

- **Session (org.hibernate.Session)**: A single-threaded, short-lived object representing a conversation between the application and the persistent store. It wraps a JDBC connection.

- **Persistent objects and collections**: Short-lived, single threaded objects containing persistent state and business function. These might be ordinary JavaBeans/POJOs, the only special thing about them is that they are currently associated with (exactly one) Session. As soon as the Session is closed, they will be detached and free to use in any application layer (e.g. directly as data transfer objects to and from presentation).

- **Transient and detached objects and collections**: Instances of persistent classes that are not currently associated with a Session. They may have been instantiated by the application and not (yet) persisted or they may have been instantiated by a closed Session.
3.4. MAPPING CONCEPTS AND METHODS

- **Transaction**: (Optional) A single-threaded, short-lived object used by the application to specify atomic units of work. Abstracts application from underlying JDBC, JTA or CORBA transaction. A Session might span several Transactions in some cases. However, transaction demarcation, either using the underlying API or Transaction, is never optional!

- **ConnectionProvider**: (Optional) A factory for (and pool of) JDBC connections. It abstracts application from underlying Datasource or DriverManager. It is not exposed to application, but can be extended or implemented by the developer.

- **TransactionFactory**: (Optional) A factory for Transaction instances. Not exposed to the application, but can be extended or implemented by the developer.

- **Extension Interfaces**: Hibernate offers many optional extension interfaces you can implement to customize the behavior of your persistence layer.

In the “lite” architecture, the application bypasses the Transaction/TransactionFactory and/or ConnectionProvider APIs to talk to JTA or JDBC directly.

### 3.3.2 Instance states

An instance of a persistent classes may be in one of three different states, which are defined with respect to a persistence context. The Hibernate Session object is the persistence context:

- **transient**: The instance is not, and has never been associated with any persistence context. It has no persistent identity (primary key value).

- **persistent**: The instance is currently associated with a persistence context. It has a persistent identity (primary key value) and, perhaps, a corresponding row in the database. For a particular persistence context, Hibernate guarantees that persistent identity is equivalent to Java identity (in-memory location of the object).

- **detached**: The instance was once associated with a persistence context, but that context was closed, or the instance was serialized to another process. It has a persistent identity and, perhaps, a corresponding row in the database. For detached instances, Hibernate makes no guarantees about the relationship between persistent identity and Java identity.

### 3.4 Mapping concepts and methods

When working with Hibernate Core, we can imagine that each class needs information on how it is stored into database, it’s crucial for Hibernate to map objects using data from the configuration file. A mapping XML file is the place to do that.

In this section we will present the fundamental mapping options, explaining how classes and properties are mapped to tables and columns. We will show how to handle database identity and primary keys, and how various other metadata settings can be used to customize how Hibernate loads and stores objects. All mapping examples are done in Hibernate’s native XML format.
3.4.1 Mapping declaration

Object/relational mappings are usually defined in an XML document. The mapping document is designed to be readable and hand-editable. The mapping language is Java-centric, meaning that mappings are constructed around persistent class declarations, not table declarations.

Even though many Hibernate users choose to write the XML by hand, a number of tools exist to generate the mapping document, including XDoclet, Middlegen and AndroMDA.

3.4.2 Mapping File Structure

Let's look at an example of a Hibernate mapping XML configuration file which was used in our application’s implementation:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="user.business">
  <class name="User" table="Users">
    <id name="id" column="id" type="int">
      <generator class="increment" />
    </id>
    <property name="userName" column="userName" type="string" />
    <property name="password" column="password" type="string" />
    <property name="email" column="email" type="string" />
    <property name="group" column="userGroup" type="string" />
    <property name="isOnline" column="isOnline" type="int" />
  </class>
</hibernate-mapping>
```

As a XML file, the content of the mapping document looks quite readable, we will now discuss its elements and attributes.

3.4.2.1 Doctype

All XML mappings should declare the doctype shown. The actual DTD may be found at the URL above, in the directory hibernate-x.x.x/src/org/hibernate or in hibernate3.jar. Hibernate will always look for the DTD in its classpath first. If you experience lookups of the DTD using an Internet connection, check your DTD declaration against the contents of your classpath.

3.4.2.2 Hibernate-mapping

This element has several optional attributes. The schema and catalog attributes specify that tables referred to in this mapping belong to the named schema and/or catalog. If specified, tablenames will be qualified by the given schema and catalog names. If missing, tablenames will be unqualified. The default-cascade attribute specifies what cascade style should be assumed for properties and collections which do not specify a cascade attribute. The auto-import attribute lets us use unqualified class names in the query language, by default.
3.4. MAPPING CONCEPTS AND METHODS

<hibernate-mapping
    schema="schemaName" (1)
    catalog="catalogName" (2)
    default-cascade="cascade_style" (3)
    default-access="field|property|ClassName" (4)
    default-lazy="true|false" (5)
    auto-import="true|false" (6)
    package="package.name" (7)
/>


2. *catalog* (optional): The name of a database catalog.


4. *default-access* (optional - defaults to property): The strategy Hibernate should use for accessing all properties. Can be a custom implementation of PropertyAccessor.

5. *default-lazy* (optional - defaults to true): The default value for unspecified lazy attributes of class and collection mappings.

6. *auto-import* (optional - defaults to true): Specifies whether we can use unqualified class names (of classes in this mapping) in the query language.


If we have two persistent classes with the same (unqualified) name, we should set auto-import="false". Hibernate will throw an exception if we attempt to assign two classes to the same "imported" name.

One experience is that the hibernate-mapping element allows us to nest several persistent <class> mappings, as shown above. It is however good practice to map only a single persistent class in one mapping file and name it after the persistent superclass, for example *Cat.hbm.xml*, *Dog.hbm.xml*, or if using inheritance, *Animal.hbm.xml*.

### 3.4.2.3 Class

You may declare a persistent class using the class element:
1. **name** (optional): The fully qualified Java class name of the persistent class (or interface). If this attribute is missing, it is assumed that the mapping is for a non-POJO entity.

2. **table** (optional - defaults to the unqualified class name): The name of its database table.

3. **discriminator-value** (optional - defaults to the class name): A value that distinguishes individual subclasses, used for polymorphic behaviour. Acceptable values include null and not null.

4. **mutable** (optional, defaults to true): Specifies that instances of the class are (not) mutable.

5. **schema** (optional): Override the schema name specified by the root `<hibernate-mapping>` element.

6. **catalog** (optional): Override the catalog name specified by the root `<hibernate-mapping>` element.

7. **proxy** (optional): Specifies an interface to use for lazy initializing proxies. You may specify the name of the class itself.

8. **dynamic-update** (optional, defaults to false): Specifies that UPDATE SQL should be generated at runtime and contain only those columns whose values have changed.
9. *dynamic-insert* (optional, defaults to false): Specifies that INSERT SQL should be generated at runtime and contain only the columns whose values are not null.

10. *select-before-update* (optional, defaults to false): Specifies that Hibernate should never perform an SQL UPDATE unless it is certain that an object is actually modified. In certain cases (actually, only when a transient object has been associated with a new session using update()), this means that Hibernate will perform an extra SQL SELECT to determine if an UPDATE is actually required.

11. *polymorphism* (optional, defaults to implicit): Determines whether implicit or explicit query polymorphism is used.

12. *where* (optional) specify an arbitrary SQL WHERE condition to be used when retrieving objects of this class.


14. *batch-size* (optional, defaults to 1) specify a “batch size” for fetching instances of this class by identifier.

15. *optimistic-lock* (optional, defaults to version): Determines the optimistic locking strategy.

16. *lazy* (optional): Lazy fetching may be completely disabled by setting lazy="false".

17. *entity-name* (optional, defaults to the class name): a new feature in Hibernate3, this allows a class to be mapped multiple times (to different tables, potentially), and allows entity mappings that are represented by Maps or XML at the Java level. In these cases, you should provide an explicit arbitrary name for the entity.


19. *rowid* (optional): Hibernate can use so called ROWIDs on databases which support. For example on Oracle, Hibernate can use the rowid extra column for fast updates if you set this option to rowid. A ROWID is an implementation detail and represents the physical location of a stored tuple.

20. *subselect* (optional): Maps an immutable and read-only entity to a database subselect. Useful if you want to have a view instead of a base table.


It is perfectly acceptable for the named persistent class to be an interface. We would then declare implementing classes of that interface using the `<subclass>` element. We may persist any static inner class.

Immutable classes, `mutable="false"`, may not be updated or deleted by the application. This allows Hibernate to make some minor performance optimizations.

Implicit polymorphism means that instances of the class will be returned by a query that names any superclass or implemented interface or the class and that instances of any subclass of the class will be returned by a query that names the class itself. Explicit polymorphism
means that class instances will be returned only by queries that explicitly name that class and that queries that name the class will return only instances of subclasses mapped inside this \(<\text{class}>\) declaration as a \(<\text{subclass}>\) or \(<\text{joined-subclass}>\). For most purposes the default, \(\text{polymorphism} = \text{"implicit"}\), is appropriate.

Note that the \(\text{dynamic-update}\) and \(\text{dynamic-insert}\) settings are not inherited by subclasses and so may also be specified on the \(<\text{subclass}>\) or \(<\text{joined-subclass}>\) elements. These settings may increase performance in some cases, but might actually decrease performance in others.

Use of \(\text{select-before-update}\) will usually decrease performance. It is very useful to prevent a database update trigger being called unnecessarily if we reattach a graph of detached instances to a Session.

If we enable \(\text{dynamic-update}\), we will have a choice of optimistic locking strategies:

- \(\text{version}\) check the version/timestamp columns
- \(\text{all}\) check all columns
- \(\text{dirty}\) check the changed columns, allowing some concurrent updates
- \(\text{none}\) do not use optimistic locking

### 3.4.2.4 Id

Mapped classes must declare the primary key column of the database table. Most classes will also have a JavaBeans-style property holding the unique identifier of an instance. The \(<\text{id}>\) element defines the mapping from that property to the primary key column.

```xml
<id
    name="propertyName" (1)
    type="typename" (2)
    column="column_name" (3)
    unsaved-value="null|any|none|undefined|id_value" (4)
    access="field|property|ClassName"> (5)
</id>

<generator class="generatorClass"/>
```

1. \(\text{name}\) (optional): The name of the identifier property.
2. \(\text{type}\) (optional): A name that indicates the Hibernate type.
3. \(\text{column}\) (optional - defaults to the property name): The name of the primary key column.
4. \(\text{unsaved-value}\) (optional - defaults to a "sensible" value): An identifier property value that indicates that an instance is newly instantiated (unsaved), distinguishing it from detached instances that were saved or loaded in a previous session.
5. \(\text{access}\) (optional - defaults to property): The strategy Hibernate should use for accessing the property value.
If the name attribute is missing, it is assumed that the class has no identifier property. The unsaved-value attribute is almost never needed in Hibernate3. There is an alternative \texttt{<composite-id>} declaration to allow access to legacy data with composite keys.

**Generator**

The optional \texttt{<generator>} is a child element of \texttt{<id>} element. It names a Java class used to generate unique identifiers for instances of the persistent class. If any parameters are required to configure or initialize the generator instance, they are passed using the \texttt{<param>} element.

```
<id name="id" type="long" column="cat_id">
  <generator class="org.hibernate.id.TableHiLoGenerator">
    <param name="table">uid_table</param>
    <param name="column">next_hi_value_column</param>
  </generator>
</id>
```

All generators implement the interface \texttt{org.hibernate.id.IdentifierGenerator}. This is a very simple interface; some applications may choose to provide their own specialized implementations. However, Hibernate provides a range of built-in implementations. There are shortcut names for the built-in generators:

- **increment**: generates identifiers of type long, short or int that are unique only when no other process is inserting data into the same table. Do not use in a cluster.
- **identity**: supports identity columns in DB2, MySQL, MS SQL Server, Sybase and HypersonicSQL. The returned identifier is of type long, short or int.
- **sequence**: uses a sequence in DB2, PostgreSQL, Oracle, SAP DB, McKoi or a generator in Interbase. The returned identifier is of type long, short or int
- **hilo**: uses a hi/lo algorithm to efficiently generate identifiers of type long, short or int, given a table and column (by default \texttt{hibernate_unique_key} and \texttt{next_hi} respectively) as a source of hi values. The hi/lo algorithm generates identifiers that are unique only for a particular database.
- **seglo**: uses a hi/lo algorithm to efficiently generate identifiers of type long, short or int, given a named database sequence.
- **uuid**: uses a 128-bit UUID algorithm to generate identifiers of type string, unique within a network (the IP address is used). The UUID is encoded as a string of hexadecimal digits of length 32.
- **guid**: uses a database-generated GUID string on MS SQL Server and MySQL.
- **native**: picks identity, sequence or hilo depending upon the capabilities of the underlying database.
- **assigned**: lets the application to assign an identifier to the object before \texttt{save()} is called. This is the default strategy if no \texttt{<generator>} element is specified.
CHAPTER 3. HIBERNATE

- **select:** retrieves a primary key assigned by a database trigger by selecting the row by some unique key and retrieving the primary key value.

- **foreign:** uses the identifier of another associated object. Usually used in conjunction with a `<one-to-one>` primary key association.

- **sequence-identity:** a specialized sequence generation strategy which utilizes a database sequence for the actual value generation, but combines this with JDBC3 getGeneratedKeys to actually return the generated identifier value as part of the insert statement execution. This strategy is only known to be supported on Oracle 10g drivers targetted for JDK 1.4. Note comments on these insert statements are disabled due to a bug in the Oracle drivers.

### Hi/lo algorithm

The hilo and seqhilo generators provide two alternate implementations of the hi/lo algorithm, a favorite approach to identifier generation. The first implementation requires a "special" database table to hold the next available "hi" value. The second uses an Oracle-style sequence.

```xml
<id name="id" type="long" column="cat_id">
  <generator class="hilo">
    <param name="table">hi_value</param>
    <param name="column">next_value</param>
    <param name="max_lo">100</param>
  </generator>
</id>

<id name="id" type="long" column="cat_id">
  <generator class="seqhilo">
    <param name="sequence">hi_value</param>
    <param name="max_lo">100</param>
  </generator>
</id>
```

Unfortunately, we can’t use hilo when supplying our own Connection to Hibernate. When Hibernate is using an application server datasource to obtain connections enlisted with JTA, we must properly configure the hibernate.transaction.manager lookup class.

### UUID algorithm

The UUID contains: IP address, startup time of the JVM (accurate to a quarter second), system time and a counter value (unique within the JVM). It’s not possible to obtain a MAC address or memory address from Java code, so this is the best we can do without using JNI.

### Identity columns and sequences

For databases which support identity columns (DB2, MySQL, Sybase, MS SQL), we may use identity key generation. For databases that support sequences (DB2, Oracle, PostgreSQL,
Interbase, McKoi, SAP DB) we may use sequence style key generation. Both these strategies require two SQL queries to insert a new object.

```xml
<id name="id" type="long" column="person_id">
  <generator class="sequence">
    <param name="sequence">person_id_sequence</param>
  </generator>
</id>

<id name="id" type="long" column="person_id" unsaved-value="0">
  <generator class="identity"/>
</id>
```

For cross-platform development, the native strategy will choose from the identity, sequence and hilo strategies, dependent upon the capabilities of the underlying database.

**Assigned identifiers**

If we want the application to assign identifiers (as opposed to having Hibernate generate them), we may use the assigned generator. This special generator will use the identifier value already assigned to the object's identifier property. This generator is used when the primary key is a natural key instead of a surrogate key. This is the default behavior if we do no specify a `<generator>` element.

Choosing the assigned generator makes Hibernate use `unsaved-value="undefined"`, forcing Hibernate to go to the database to determine if an instance is transient or detached, unless there is a version or timestamp property, or we define `Interceptor.isUnsaved()`.

### 3.4.2.5 Discriminator

The `<discriminator>` element is required for polymorphic persistence using the table-per-class-hierarchy mapping strategy and declares a discriminator column of the table. The discriminator column contains marker values that tell the persistence layer what subclass to instantiate for a particular row. A restricted set of types may be used: string, character, integer, byte, short, boolean, yes_no, true_false.

```xml
discriminator
column="discriminator_column" (1)
type="discriminator_type" (2)
force="true|false" (3)
insert="true|false" (4)
formula="arbitrary sql expression" (5)
</discriminator>
```

1. column (optional - defaults to class) the name of the discriminator column.

2. type (optional - defaults to string) a name that indicates the Hibernate type.
3. **force** (optional - defaults to false) "force" Hibernate to specify allowed discriminator values even when retrieving all instances of the root class.

4. **insert** (optional - defaults to true) set this to false if our discriminator column is also part of a mapped composite identifier. (Tells Hibernate to not include the column in SQL INSERTs.)

5. **formula** (optional) an arbitrary SQL expression that is executed when a type has to be evaluated. Allows content-based discrimination.

Actual values of the discriminator column are specified by the `discriminator-value` attribute of the `<class>` and `<subclass>` elements.

The force attribute is (only) useful if the table contains rows with "extra" discriminator values that are not mapped to a persistent class.

### 3.4.2.6 Other elements

There are also other elements in the mapping file that we are not going to discuss in detail here but just giving some brief description:

**Timestamp (optional)**

The optional `<timestamp>` element indicates that the table contains timestamped data. This is intended as an alternative to versioning. Timestamps are by nature a less safe implementation of optimistic locking. However, sometimes the application might use the timestamps in other ways.

**Many-to-one**

An ordinary association to another persistent class is declared using a many-to-one element. The relational model is a many-to-one association: a foreign key in one table is referencing the primary key column(s) of the target table.

**One-to-one**

A one-to-one association to another persistent class is declared using a one-to-one element.

**Property**

The `<property>` element declares a persistent, JavaBean style property of the class.

**Natural-id**

A natural key is a property or combination of properties that is unique and non-null. If it is also immutable, even better. Map the properties of the natural key inside the `<natural-id>` element. Hibernate will generate the necessary unique key and nullability constraints, and your mapping will be more self-documenting.
Component, dynamic-component

The <component> element maps properties of a child object to columns of the table of a parent class. Components may, in turn, declare their own properties, components or collections.

Properties

The <properties> element allows the definition of a named, logical grouping of properties of a class. The most important use of the construct is that it allows a combination of properties to be the target of a property-ref. It is also a convenient way to define a multi-column unique constraint.

Subclass

Polymorphic persistence requires the declaration of each subclass of the root persistent class. For the table-per-class-hierarchy mapping strategy, the <subclass> declaration is used.

Joined-subclass

Alternatively, each subclass may be mapped to its own table (table-per-subclass mapping strategy). Inherited state is retrieved by joining with the table of the superclass. We use the <joined-subclass> element.

Join

Using the <join> element, it is possible to map properties of one class to several tables.

Key

It appears anywhere the parent mapping element defines a join to a new table, and defines the foreign key in the joined table, that references the primary key of the original table.

Column and formula elements

Any mapping element which accepts a column attribute will alternatively accept a <column> sub-element. Likewise, <formula> is an alternative to the formula attribute.

Import

Suppose our application has two persistent classes with the same name, and we don’t want to specify the fully qualified (package) name in Hibernate queries. Classes may be "imported" explicitly, rather than relying upon auto-import="true". We may even import classes and interfaces that are not explicitly mapped.

Any

The <any> mapping element defines a polymorphic association to classes from multiple tables. This type of mapping always requires more than one column. The first column holds the type of the associated entity. The remaining columns hold the identifier.
CHAPTER 3. HIBERNATE

For more information on these elements, we could always look at Hibernate reference documentation that comes with the download or is accessible from Hibernate website. Hibernate provides users with a good amount of tutorial and documentations, the community forum is also very useful and popular.

3.4.3 Hibernate Types

3.4.3.1 Entities and values

To understand the behaviour of various Java objects with respect to the persistence service, we need to classify them into two groups:

- An entity exists independently of any other objects holding references to the entity. Contrast this with the usual Java model where an unreferenced object is garbage collected. Entities must be explicitly saved and deleted. Entities support circular and shared references. They may also be versioned. An entity’s persistent state consists of references to other entities and instances of value types. Values are primitives, collections (not what’s inside a collection), components and certain immutable objects.

- Unlike entities, values (in particular collections and components) are persisted and deleted by reachability. Since value objects (and primitives) are persisted and deleted along with their containing entity they may not be independently versioned. Values have no independent identity, so they cannot be shared by two entities or collections.

The bridge between both systems is provided by Hibernate: for entities we use `<class>`, `<subclass>` and so on. For value types we use `<property>`, `<component>`, etc, usually with a type attribute. The value of this attribute is the name of a Hibernate mapping type.

3.4.3.2 Basic value types

In Hibernate mapping, the built-in basic mapping types may be categorized into:

- integer, long, short, float, double, character, byte, boolean, yes_no, true_false
  Type mappings from Java primitives or wrapper classes to appropriate (vendor-specific) SQL column types. boolean, yes_no and true_false are all alternative encodings for a Java boolean or java.lang.Boolean.

- string
  A type mapping from java.lang.String to VARCHAR (or Oracle VARCHAR2).

- date, time, timestamp
  Type mappings from java.util.Date and its subclasses to SQL types DATE, TIME and TIMESTAMP (or equivalent).

- calendar, calendar_date
  Type mappings from java.util.Calendar to SQL types TIMESTAMP and DATE (or equivalent).
3.4. MAPPING CONCEPTS AND METHODS

- **big_decimal, big_integer**
  Type mappings from java.math.BigDecimal and java.math.BigInteger to NUMERIC (or Oracle NUMBER).

- **locale, timezone, currency**
  Type mappings from java.util.Locale, java.util.TimeZone and java.util.Currency to VARCHAR (or Oracle VARCHAR2). Instances of Locale and Currency are mapped to their ISO codes. Instances of TimeZone are mapped to their ID.

- **class**
  A type mapping from java.lang.Class to VARCHAR (or Oracle VARCHAR2). A Class is mapped to its fully qualified name.

- **binary**
  Maps byte arrays to an appropriate SQL binary type.

- **text**
  Maps long Java strings to a SQL CLOB or TEXT type.

- **serializable**
  Maps serializable Java types to an appropriate SQL binary type.

- **clob, blob**
  Type mappings for the JDBC classes java.sql.Clob and java.sql.Blob. These types may be inconvenient for some applications, since the blob or clob object may not be reused outside of a transaction.

- **imm_date, imm_time, imm_timestamp, imm_calendar, imm_calendar_date, imm_serializable, imm_binary**
  Type mappings for what are usually considered mutable Java types, where Hibernate makes certain optimizations appropriate only for immutable Java types, and the application treats the object as immutable. For example, we should not call Date.setTime() for an instance mapped as imm_timestamp. To change the value of the property, and have that change made persistent, the application must assign a new (nonidentical) object to the property.

Unique identifiers of entities and collections may be of any basic type except binary, blob and clob. The basic value types have corresponding Type constants defined on org.hibernate.Hibernate. For example, Hibernate.STRING represents the string type.

In our implementation of a SMS system in chapter 5, we will use mostly only string, timestamp, integer, long, boolean types.

### 3.4.4 Mapping a class more than once

It is possible to provide more than one mapping for a particular persistent class. In this case we must specify an entity name to disambiguate between instances of the two mapped entities. (By default, the entity name is the same as the class name.) Hibernate lets us
specify the entity name when working with persistent objects, when writing queries, or when mapping associations to the named entity.

```xml
<class name="Contract" table="Contracts"
     entity-name="CurrentContract">
  ...
  <set name="history" inverse="true"
       order-by="effectiveEnd asc">
    <key column="currentContractId"/>
    <one-to-many entity-name="HistoricalContract"/>
  </set>
</class>

<class name="Contract" table="ContractHistory"
     entity-name="HistoricalContract">
  ...
  <many-to-one name="currentContract"
               column="currentContractId"
               entity-name="CurrentContract"/>
</class>
```

3.4.5 Metadata alternatives to mapping files

XML isn’t for everyone, and so there are some alternative ways to define O/R mapping metadata in Hibernate.

3.4.5.1 Using XDoclet markup

Many Hibernate users prefer to embed mapping information directly in sourcecode using XDoclet @hibernate tags. We will not cover this approach in this document, since strictly it is considered part of XDoclet.

3.4.5.2 Using JDK 5.0 Annotations

JDK 5.0 introduced XDoclet-style annotations at the language level, type-safe and checked at compile time. This mechanism is more powerful than XDoclet annotations and better supported by tools and IDEs. IntelliJ IDEA, for example, supports auto-completion and syntax highlighting of JDK 5.0 annotations.

3.5 Hibernate Query Language

Hibernate Query Language or HQL for short is extremely powerful query language. HQL is much like SQL and are case-insensitive, except for the names of the Java Classes and properties. Hibernate Query Language is used to execute queries against database. Hibernate automatically generates the SQL query and execute it against underlying database if HQL is used in the application. HQL is based on the relational object models and makes the SQL object oriented. Hibernate Query Language uses Classes and properties instead of tables and
columns. Hibernate Query Language is extremely powerful and it supports Polymorphism, Associations.

There are other options that can be used while using Hibernate. These are Query By Criteria (QBC) and Query By Example (QBE) using Criteria API and the Native SQL queries. In the implementation of the DAO objects in our application, we will see more how HQL were used.

What are the reasons to use HQL?

- **Full support for relational operations:** HQL allows representing SQL queries in the form of objects. Hibernate Query Language uses Classes and properties instead of tables and columns.

- **Return result as Object:** The HQL queries return the query result(s) in the form of object(s), which is easy to use. This eliminates the need of creating the object and populate the data from result set.

- **Polymorphic Queries:** HQL fully supports polymorphic queries. Polymorphic queries results the query results along with all the child objects if any.

- **Easy to Learn:** Hibernate Queries are easy to learn and it can be easily implemented in the applications.

- **Support for Advance features:** HQL contains many advance features such as pagination, fetch join with dynamic profiling, Inner/outer/full joins, Cartesian products. It also supports Projection, Aggregation (max, avg) and grouping, Ordering, Sub queries and SQL function calls.

- **Database independent:** Queries written in HQL are database independent.

In this section we will not go into details about HQL but just introduce it as a good feature of Hibernate.

**Understanding HQL Syntax**

Let’s look at a HQL example as follow:

```
select cat.name from DomesticCat cat
where cat.name like 'fri%'
--------------------------------------
select cat.weight + sum(kitten.weight)
from Cat cat
join cat.kittens kitten
group by cat.id, cat.weight
```

We could see that any Hibernate Query Language may consist of following elements:

- Clauses
- Aggregate functions
- Subqueries
CHAPTER 3. HIBERNATE

Clauses in the HQL are:

- `from`
- `select`
- `where`
- `order by`
- `group by`

Aggregate functions are:

- `avg(...)`, `sum(...)`, `min(...)`, `max(...)`
- `count(*)`
- `count(...)`, `count(distinct ...)`, `count(all...)`

Subqueries

Subqueries are queries within another query. Hibernate supports Subqueries if the underlying database supports it.

3.6 Conclusion

We have seen Hibernate and its various features giving an application with benefits in productivity, maintainability and performance. However, it is not easy to understand Hibernate fast, it requires Java developers to have a sufficient level of familiarity with and appreciation of relational modeling and SQL in order to work with ORM. To use Hibernate effectively, we must be able to view and interpret the SQL statements, resolve its issues and understand the implications for performance.

To sum up, the following points illustrate my opinions about Hibernate:

What’s so good about Hibernate?

- It is non-intrusive, almost completely transparent, persistence even for Plain Old Java Objects (POJOs).
- There is no need for pre- or post-processing of source- or byte-code. This means a simpler tool chain and quicker edit-compile-deploy cycle.
- There are alot of tools come along with it, alot of frameworks supporting it.
- Hibernate has a good documentations, there are also lots of books and tutorials dealing with Hibernate.
- In terms of performance, its codes are very robust, almost bug-free, the development team are experts and have alot of experience in the field.
3.6. **CONCLUSION**

- It has a very good query language (HQL), with the option of falling back to SQL when needed.
- There is a huge community and forum which can help developers to exchange knowledge and support.
- An open-source project!

**What’s not so good about Hibernate?**

- For all practical purposes, Hibernate is owned by JBoss, and with given JBoss’s revenue model, it might come as no surprise that support is relegated to a forum and responses from the development team tend to be just a little bit terse, at times.
- Debugging is a bit painful, nevertheless improvements will be made in the next-coming versions.
- Developers need to have a deep knowledge of database design in order to use Hibernate efficiently.

I have to admit that Hibernate, like other sophisticated frameworks, has a steep, but short learning curve, but once you got it - it is very worthy. It becomes a breeze to work with, and it feels like a pleasure.

Nevertheless, I would highly recommend Hibernate for anyone who tends to work with object persistence in Java, it is a really powerful, high performance object-relational persistence and query service.
Chapter 4

Apache Struts 2

“Apache Struts 2 is an elegant, extensible framework for creating enterprise-ready Java web applications. The framework is designed to streamline the full development cycle, from building, to deploying, to maintaining applications over time” - The Apache Software Foundation.

4.1 Introduction

Apache Struts is an open-source framework that is used for developing Java web application. Originally developed by the programmer and author Craig R. McClanahan, this was later taken over by the Apache Software Foundation in 2002. Struts have provided an excellent framework for developing application easily by organizing JSP and Servlet based on HTML formats and Java code. Strut1 with all standard Java technologies and packages of Jakarta assists to create an extensible development environment. However, with the growing demand of web application, Strut 1 does not stand firm and needs to be changed with demand. This leads to the creation of Strut2, which is more developer friendly with features like Ajax, rapid development and extensibility.

4.1.1 History of Struts 2

Since arriving on the scene in 2000, Apache Struts has enjoyed a very successful run, by most any standard, helping to build many, if not most, of the Java-based web applications deployed today. Its history tells of how Struts provided a solid framework to organize the mess of JSP and Servlets to make developing applications, which used mostly server-generated HTML with a touch Javascript for client-side validation, easier to develop and maintain. As time moved forward, and customer demands of web applications grew and grew, Struts 1 pretty much stayed the same, leaving more and more plumbing to the web developer.

At JavaOne 2005, several of the Struts developers sat down with a few Struts users to discuss the future of Struts and came up with the Struts Ti proposal, which described a framework that brought together a lot of good things that were developing in the web framework community. The problem is that the Struts 1 code base didn’t lend itself to drastic improvements, and its feature set was rather limited, particularly lacking in features such as Ajax, rapid development, and extensibility. At the same time, they also met developers from other frameworks such as the Spring WebFlow project and started discussing around an
idea of the possibility of combining efforts into one framework. However, there were several obstacles and questions about project ownership, brand, and identity that seems to block the progress. Fortunately, developers from Struts and WebWork did not lose the momentum and suggested the idea of a merger.

At that time, Struts developers were also working on Apache Shale, a web framework based off JSF, as a Struts subproject, along side Struts Action 1 (now called Struts 1) and Struts Action 2 (the graduated WebWork 2 code). Unfortunately, these subprojects were confusion to the developer and user community, accustomed to the name “Struts” referring to a single framework. After an attempt to unify the Struts Action 2 and Shale subprojects into a single Struts 2 framework, the Shale developers felt it would be better if they were their own top level project where they are today. Struts Action 2 was soon after renamed to simply Struts 2.

Today, the Apache Struts project has two major versions of its framework, but it is one action-based framework project. WebWork continues to deliver patch releases, and will certainly continue to do so until Struts 2 goes on, but all new development takes place on the Struts 2 code. Since the initial appearance, the Struts 2 code has added major features including a plugin framework, new APIs, and better Ajax tags and will still keep on improving.

4.2 Struts 2 features

4.2.1 The MVC Architecture

Inherited from Struts 1, Struts 2 is also an open source framework used for developing J2EE web applications using Model View Controller design pattern. The main aim of the MVC architecture is to separate the business logic and application data from the presentation data to the user.

Here are the reasons why we should use the MVC design pattern.

- **They are reusable**: When the problems recurs, there is no need to invent a new solution, we just have to follow the pattern and adapt it as necessary.

- **They are expressive**: By using the MVC design pattern our application becomes more readable and understandable.

4.2.1.1 Model

The model object knows about all the data that need to be displayed. It is model who is aware about all the operations that can be applied to transform that object. It only represents the data of an application. The model represents enterprise data and the business rules that govern access to and updates of this data. Model is not aware about the presentation data and how that data will be displayed to the browser.

4.2.1.2 View

The view represents the presentation of the application. The view object refers to the model. It uses the query methods of the model to obtain the contents and renders it. The view is not dependent on the application logic. It remains same if there is any modification in the business logic. In other words, we can say that it is the responsibility of the of the view’s to maintain the consistency in its presentation when the model changes.
4.2. STRUTS 2 FEATURES

4.2.1.3 Controller

Whenever the user sends a request for something then it always go through the controller. The controller is responsible for intercepting the requests from view and passes it to the model for the appropriate action. After the action has been taken on the data, the controller is responsible for directing the appropriate view to the user. In GUIs, the views and the controllers often work very closely together.

4.2.2 How this model applied in Struts 2

The Struts 2 Controller Components

Whenever an user request for something, the request is handled by FilterDispatcher. When the FilterDispatcher receives the request, it intercepts the URL and based on the Struts Configuration files, it gives the handling of the request to the Action class. Action class is a part of the controller and is responsible for communicating with the model layer.

The Struts 2 View Components

The view components are responsible for presenting information to the users and accepting the input from them. They are responsible for displaying the information provided by the model components. Mostly we use the Java Server Pages (JSP) for the view presentation. To extend the capability of the view we can use the Custom tags, java script etc.

The Struts 2 Model Components

The model components provides a model of the business logic behind a Struts program. It provides interfaces to databases or back-ends systems. Model components are generally a Java class. There is not any such defined format for a Model component, so it is possible for us to reuse Java code which are written for other projects.

4.2.3 What Struts 2 could bring

Struts 2 contains the combined features of Struts Ti and WebWork 2 projects that advocates higher level application by using the architecture of WebWork2 with features including a plugin framework, a new API, AJAX tags etc. Hence, Apache Strut 2 is a dynamic, extensible framework for a complete application development from building, deploying and maintaining.

The Strut 2 framework is designed for the compilation of the entire development cycle including of building, developing and maintaining the whole application. It is very extensible as each class of the framework is based on an Interface and all the base classes are given an extra application and even you can add your own. The basic platform requirements are Servlet API 2.4, JSP API 2.0 and Java 5.

Let’s look at some of the general features of the current Apache Struts 2 framework:

► Architecture – First the web browser request a resource for which the Filter Dispatcher decides the suitable action. Then the Interceptors use the required functions and after that the Action method executes all the functions like storing and retrieving data from a database. Then the result can be seen on the output of the browser in HTML, PDF, images or any other.
Tags - Tags in Strut 2 allow creating dynamic web applications with less number of coding. Not only these tags contain output data but also provide style sheet driven markup that in turn helps in creating pages with less code. Here the tags also support validation and localization of coding that in turn offer more utilization. The less number of codes also makes it easy to read and maintain.

MVC – The Model View Controller in Strut 2 framework acts as a coordinator between application’s model and web view. Its Controller and View components can come together with other technology to develop the model. The framework has its library and markup tags to present the data dynamically.

Configuration – Provides a deployment descriptor to initialize resources in XML format. The initialization takes place simply by scanning all the classes using Java packages or you can use an application configuration file to control the entire configuration. Its general-purpose defaults allow using struts directly Out of the box. Configuration files are re-loadable that allows changes without restarting a web container.

Apart from that, there are also allot of other features that make web development more easy:

- All framework classes are based on interfaces and core interfaces are independent from HTTP.
- Check boxes do not require any kind of special application for false values.
- Any class can be used as an action class and one can input properties by using any JavaBean directly to the action class.
- Strut 2 actions are Spring friendly and so easy to Spring integration.
- AJAX theme enables to make the application more dynamic.
- Portal and servlet deployment are easy due to automatic portlet support without altering code.
- The request handling in every action makes it easy to customize, when required.

4.2.4 Looking into Struts 2 architecture

As Struts and Webwork has joined together to develop the Struts 2 Framework. Struts 2 Framework is very extensible and elegant for the development of enterprise web application of any size. In this section we are going to discuss the architecture of Struts 2 Framework.

First, let’s take a look at a request Lifecycle in Struts 2 applications:

1. User Sends request: User sends a request to the server for some resource.
2. FilterDispatcher determines the appropriate action: The FilterDispatcher looks at the request and then determines the appropriate Action.
3. Interceptors are applied: Interceptors are configured for applying the common functionalities such as workflow, validation, file upload etc. are automatically applied to the request.
4. **Execution of Action**: Then the action method is executed to perform the database related operations like storing or retrieving data from the database.

5. **Output rendering**: Then the Result renders the output.

6. **Return of Request**: Then the request returns through the interceptors in the reverse order. The returning request allows us to perform the clean-up or additional processing.

7. **Display the result to user**: Finally the control is returned to the servlet container, which sends the output to the user browser.

For a more detail picture\(^1\) of the whole process, the following diagram depicts the architecture of Struts 2 Framework and also shows the initial request goes to the servlet container such as tomcat, which is then passed through standard filer chain.

\(^1\)Diagrams and some explanation used in this section were taken from *Struts 2 tutorial* at [www.roseindia.net](http://www.roseindia.net).
As we can see from the diagram, a request will be sent from an user to server, this request then pass to FilterDispatcher to determine the appropriate action with the help of ConfigurationManager and configuration files, after that, interceptors are applied for functionalities such as workflow, validation. Then the action method is executed to perform tasks such as database related operations like storing or retrieving data from the database. The result renders the output, the request returns through the interceptors in the reverse order. The returning request allows us to perform the clean-up or additional processing. Finally the control is returned to the servlet container, which sends the output to the user browser.

Looking closer at the process, we can see that the filter chain includes:

- **Action ContextCleanUp** filter: The ActionContextCleanUp filter is optional and it is useful when integration has to be done with other technologies like SiteMash Plugin.

- **FilterDispatcher**: Next the FilterDispatcher is called, which in turn uses the ActionMapper to determine whether to invoke an Action or not. If the action is required to be invoked, the FilterDispatcher delegates the control to the ActionProxy.
4.2. STRUTS 2 FEATURES

ActionProxy: The ActionProxy takes help from Configuration Files manager, which is initialized from the struts.xml. Then the ActionProxy creates an ActionInvocation, which implements the command pattern. The ActionInvocation process invokes the Interceptors (if configured) and then invokes the action. The the ActionInvocation looks for proper result. Then the result is executed, which involves the rendering of JSP or templates.

Now some of us might wonder, how do we get this filters to work? The answer is: it is configured in web.xml file. Following is a simple example how we configured this filter in our application:

```
<filter>
  <filter-name>struts2</filter-name>
  <filter-class>
    org.apache.struts2.dispatcher.FilterDispatcher
  </filter-class>
</filter>

<filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

4.2.5 What is so good about Struts 2

The new version Struts 2.0 is a combination of the Struts action framework and Webwork. According to the Struts 2 release announcement, some key concepts and features are:

- **Simplified Design** - Programming the abstract classes instead of interfaces is one of design problem of struts1 framework that has been resolved in the struts 2 framework. Most of the Struts 2 classes are based on interfaces and most of its core interfaces are HTTP independent. Struts 2 Action classes are framework independent and are simplified to look as simple POJOs. Framework components are tried to keep loosely coupled.

- **Simplified Actions** - Actions are simple POJOs. Any java class with execute() method can be used as an Action class. Even we don’t need to implement interfaces always. Inversion of Control is introduced while developing the action classes. This make the actions to be neutral to the underlying framework.

- **No more ActionForms** - ActionForms feature is no more known to the struts2 framework. Simple JavaBean flavored actions are used to put properties directly. No need to use all String properties.

- **Simplified testability** - Struts 2 Actions are HTTP independent and framework neutral. This enables to test struts applications very easily without resorting to mock objects.
Intelligent Defaults - Most configuration elements have a default value which can be set according to the need. Even there are xml-based default configuration files that can be overridden according to the need.

Improved results - Unlike ActionForwards, Struts 2 Results provide flexibility to create multiple type of outputs and in actual it helps to prepare the response.

Better Tag features - Struts 2 tags enables to add style sheet-driven markup capabilities, so that we can create consistent pages with less code. Struts 2 tags are more capable and result oriented. Struts 2 tag markup can be altered by changing an underlying stylesheet. Individual tag markup can be changed by editing a FreeMarker template. Both JSP and FreeMarker tags are fully supported.

Annotations introduced : Applications in struts 2 can use Java 5 annotations as an alternative to XML and Java properties configuration. Annotations minimize the use of xml.

Stateful Checkboxes - Struts 2 checkboxes do not require special handling for false values.

QuickStart - Many changes can be made on the fly without restarting a web container.

customizing controller - Struts 1 lets to customize the request processor per module, Struts 2 lets to customize the request handling per action, if desired.

Easy Spring integration - Struts 2 Actions are Spring-aware. Just need to add Spring beans!

Easy plugins - Struts 2 extensions can be added by dropping in a JAR. No manual configuration is required!

One cool feature of Struts 2 is AJAX support - The AJAX theme gives interactive applications a significant boost. The framework provides a set of tags to enable us make use of AJAX in our applications. The AJAX features in Struts 2 include:

AJAX Client Side Validation

Remote form submission support (works with the submit tag as well)

An advanced div template that provides dynamic reloading of partial HTML

An advanced template that provides the ability to load and evaluate JavaScript remotely

An AJAX-only tabbed Panel implementation

A rich pub-sub event model

Interactive auto complete tag

To see how AJAX features could be easily integrated into a Struts application, let’s take a look at an example of how we used AJAX in our application, this is the case when we want to edit an article saved in database:
For this purpose, in the jsp file that we want to use AJAX, first we will have to declare Struts 2 taglib and AJAX theme inside element:

```html
<%@ taglib prefix="s" uri="/struts-tags"%>
...
<head>
<s:head theme="ajax"/>
...
</head>
```

In the body of our jsp file, we will create a with id attribute to load its contents using AJAX.

```html
<div id="msg">Press Save to update</div>
```

We also need a form to update values and save them to data store. In the AJAX way, when we press submit form, the content of the form will be submitted and a message will be display inside our without having to reload the whole page. For AJAX to work, we need to have a form with theme="ajax":

```html
<s:form id="edit" action="EditArticleAction" method="POST" theme="ajax">
... //form content here
<s:submit name="submit" formId="edit" value="Save" targets="msg"/>
</s:form>
```

The secret in Struts 2 submit button is that it has attribute targets="msg" to indicate the id of the where we want to reload the content when pressed.

We can see from above that the form action is EditArticleAction, this action will be execute when we press submit. Now let’s take a look at the configuration file of Struts 2 to see what is the result of EditArticleAction:

```xml
<action name="EditArticleAction" class="action.article.EditArticleAction">
  <result>/cms/ajax/editArticleAjax.jsp</result>
</action>
```

We see that EditArticleAction results in editArticleAjax.jsp, this means that the content of with id = “msg” will be loaded with the content inside editArticleAjax.jsp. Therefore, inside editArticleAjax.jsp we just need to place the a test condition and display whether the article has been updated successfully:
Notice that we also set some header parameters to prevent caching at the proxy server. This example demonstrated how simple it is to use AJAX in Struts 2.

4.3 Struts 1.x VS Struts 2.x

The Apache Struts Project offers two major versions of the Struts framework. Struts 1 is recognized as the most popular web application framework for Java. The 1.x framework is mature, well-documented, and widely supported. Struts 1 is suitable for teams who value proven solutions to common problems.

Struts 2 was originally known as WebWork 2. After working independently for several years, the WebWork and Struts communities joined forces to create Struts 2. The 2.x framework is the choice for teams who value elegant solutions to difficult problems.

In the following section, we are going to compare the various features between the two frameworks. Struts 2.x is very simple as compared to struts 1.x, few of its excellent features are:

1. Servlet Dependency:

Actions in Struts1 have dependencies on the servlet API since the HttpServletRequest and HttpServletResponse objects are passed to the execute method when an Action is invoked but in case of Struts 2, Actions are not container dependent because they are made simple POJOs. In struts 2, the servlet contexts are represented as simple Maps which allows actions to be tested in isolation. Struts 2 Actions can access the original request and response, if required. However, other architectural elements reduce or eliminate the need to access the HttpServletRequest or HttpServletResponse directly.

2. Action classes

Programming the abstract classes instead of interfaces is one of design issues of Struts 1 framework that has been resolved in the Struts 2 framework. Struts1 Action classes needs to extend framework dependent abstract base class. But in case of Struts 2 Action class may or may not implement interfaces to enable optional and custom services. In case of Struts 2, Actions are not container dependent because they are made simple POJOs. Struts 2 provides
4.3. **STRUTS 1.X VS STRUTS 2.X**

a base ActionSupport class to implement commonly used interfaces. Therefore, the Action interface is not required. Any POJO object with an execute signature can be used as an Struts 2 Action object.

3. **Validation**

Struts1 and Struts 2 both supports the manual validation via a validate method. Struts1 uses validate method on the ActionForm, or validates through an extension to the Commons Validator. However, Struts 2 supports manual validation via the validate method and the XWork Validation framework. The Xwork Validation Framework supports chaining validation into sub-properties using the validations defined for the properties class type and the validation context.

4. **Threading Model**

In Struts1, Action resources must be thread-safe or synchronized. So Actions are singletons and thread-safe, there should only be one instance of a class to handle all requests for that Action. The singleton strategy places restrictions on what can be done with Struts1 Actions and requires extra care to develop. However in case of Struts 2, Action objects are instantiated for each request, so there are no thread-safety issues. (In practice, servlet containers generate many throw-away objects per request, and one more object does not impose a performance penalty or impact garbage collection.)

5. **Testability**

Testing Struts1 applications are a bit complex. A major hurdle to test Struts1 Actions is that the execute method because it exposes the Servlet API. A third-party extension, Struts TestCase, offers a set of mock object for Struts 1. But the Struts 2 Actions can be tested by instantiating the Action, setting properties and invoking methods. Dependency Injection support also makes testing simpler. Actions in struts2 are simple POJOs and are framework independent, hence testability is quite easy in struts2.

6. **Harvesting Input**

Struts 1 uses an ActionForm object to capture input. And all ActionForms needs to extend a framework dependent base class. JavaBeans cannot be used as ActionForms, so the developers have to create redundant classes to capture input. However Struts 2 uses Action properties (as input properties independent of underlying framework) that eliminates the need for a second input object, hence reduces redundancy. Additionally in Struts 2, Action properties can be accessed from the web page via the taglibs. Struts 2 also supports the ActionForm pattern, as well as POJO form objects and POJO Actions. Even rich object types, including business or domain objects, can be used as input/output objects.

7. **Expression Language**

Struts 1 integrates with JSTL, so it uses the JSTL-EL. The Struts 1 EL has basic object graph traversal, but relatively weak collection and indexed property support. Struts 2 can also use JSTL, however it supports a more powerful and flexible expression language called "Object Graph Notation Language" (OGNL).
8. Binding values into views

In the view section, Struts 1 uses the standard JSP mechanism to bind objects (processed from the model section) into the page context to access. However Struts 2 uses a "ValueStack" technology so that the taglibs can access values without coupling your view to the object type it is rendering. The ValueStack strategy allows the reuse of views across a range of types which may have the same property name but different property types.

9. Type Conversion

Usually, Struts 1 ActionForm properties are all Strings. Struts1 uses Commons-Beanutils for type conversion. These type converters are per-class and not configurable per instance. However Struts 2 uses OGNL for type conversion. The framework includes converters for basic and common object types and primitives.

10. Control Of Action Execution

Struts 1 supports separate Request Processor (lifecycles) for each module, but all the Actions in a module must share the same lifecycle. However Struts 2 supports creating different lifecycles on a per Action basis via Interceptor Stacks. Custom stacks can be created and used with different Actions as needed.

Despite the facts that Struts 2 has many advantages, a large amount of users are still loyal to Struts 1 since there were a huge amount of projects had already been implemented using Struts 1. The Apache Struts Project however still offers two major versions of the Struts framework simultaneously.

4.4 Usage of Hibernate in Struts projects

Persistence is a fundamental piece of an application. Obviously, without persistence all work would be lost. In Struts project, the option of using which tools for the persistence layer is an open choice. However, current trends in web development shows significantly large number of Struts project used Hibernate for their persistence layer.

To employ Hibernate in a Struts application, a best practice is to create a Struts plug-in to give our Hibernate-powered Struts applications a performance boost. A good strategy for this is to create and cache the Hibernate factory to enhance performance. This could be done by two steps:

- First, create a class `HibernatePlugin` that implements the interface `org.apache.struts.action.PlugIn`.

- Second, define a `<plug-in>` tag in the `struts-config.xml` file.
import org.apache.struts.action.PlugIn;
import org.apache.struts.config.ModuleConfig;
...

public class HibernatePlugin implements PlugIn {
    private Configuration config;
    private SessionFactory factory;
    private String path = "/hibernate.cfg.xml";
    ...
    
    config = new Configuration().configure(url);
    factory = config.buildSessionFactory();
    ...
}

This is quite a nice implementation. However, another trend in the run is to use an Inversion of Control (IoC) framework in the middle of Struts and Hibernate and Spring framework is a popular choice for that. Some may question that Spring also does have a web framework component, do we still need Struts? The answer is yes, Spring does provide that, but we don’t have to use it, we just need the IoC container of Spring. Using Spring along with some old Hibernate version provides some benefits over using Hibernate alone, but many of these benefits are included with Hibernate version 3. Using a combination of Spring-Struts-Hibernate brings us great amount of benefits but on the other hand, it is not an easy task. There is also an alternative for Spring with the purpose of an Ioc container that has less extras on it like Hivemind but Spring still seems to be the main stream nowadays used in Struts-Hibernate project. In chapter 5, we will see the benefits when we design our CMS system using Spring, Struts and Hibernate.

4.5 Conclusion

Struts have provided an excellent framework for developing application easily by organizing JSP and Servlet based on HTML formats and Java code. The emergence of Struts 2 not only makes the development work more developer friendly with new features like AJAX but also provides rapid development and extensibility.

Struts 2 contains the combined features of Struts and WebWork 2 projects that advocates higher level application by using the architecture of WebWork 2 with features including a plugin framework, a new API and AJAX. With some unique concepts and constructs of WebWork like its compatibility of working within existing Web APIs in Java rather than trying to replace them completely, Struts 2 promises to specifically take into account the developer’s productivity and code simplicity.

Hence, Apache Struts 2 is claimed to be a dynamic, extensible framework for a complete application development from building, deploying and maintaining. However, since it is still a fresh new frameworks, several major bugs may still can be found, the community is on the run to grow up. Therefore when working with Struts 2, looking for solutions in some
discussion forum is a popular way a developer should do.

Talking about competition, in current trends of Java web application development, it’s hard to predict the future in the field of MVC frameworks since there are also many other competing technologies. In terms of UI component, JSF for instance, is also becoming a standard. JSF can give us a lot of advantages especially when we have to deal with component reusability or support for various devices. Spring on the other hand, has always been widely used. It’s been adopted by more and more developers and we can see Spring on many projects nowadays thanks to its IoC container. However, Struts also has its own space especially with the emergence of Struts 2. If you ever apply for a job as a J2EE developer, having Struts experience is always a good advantage.
Chapter 5

Design and Implementation of a CMS system

“Building non-trivial web applications with Java is no trivial task.” - Mark Eagle.

5.1 The requirements

Now that we have seen enough about Hibernate and Struts, it is the time to get to the real work. Like every good engineer, before we start a project we should prepare our tools and decide what our development process is going to look like. In this first section will we discuss what aspects are expected in our application. We will not going to write a detail specifications or UML diagrams for the requirements but rather discuss about the main features that our application should have and how we can improve the system in the future. Our goal is to build a CMS system as a web application. There will be two main sub-systems in our application:

- **The web interface:** This is the presentation of the web application. This will include the layouts, CSS, image and webpages that are visible to visitors. This component will use libraries and functions provided by the system and data from database to display appropriate content according to requests. The web interface can be customized and designed for different purposes. For our application we will design an interface of a news website.

- **The CMS administration system:** This is the most important part of the system. Here we maintain all the contents and manage all other components of the application. This will include editing, posting, deleting, creating articles, then managing, creating, editing users and all other site-related management.

Users of our system can be broken down into three categories:

- **Normal users:** Can login into CMS system, viewing articles, see other users information but can not edit, modify the content or any action with users.

- **Moderators:** Moderators can login to CMS system, managing articles this includes posting, creating new articles, editing, deleting but can not access user management sections.
Administrators: This is the most powerful user category, administrators can login to CMS system, edit website information such as title, motto, welcome text. Administrators also have all privileges on managing articles as well as users.

Articles from our website also appear in two main categories:

- **News**: News often are short articles with short introduction text and contents.
- **Article**: Articles are fully represented with introduction text, content text, and all other information such as author, visible, etc.

Each article will have name, title, a display image, an introduction text, a body content. The content should have a HTML-tags-enabled feature for the purpose of display and decoration. Each article also belongs to a particular category, each category can be map to a menu item on the web interface. This also enables users to dynamically create their menu items. Categories can be edited, deleted and managed by Moderators and Administrators.

The application should provide functionalities to enable users to create, edit, delete articles. Administrators should be able to edit all site informations such as title, motto, footer text and manipulating all user categories. This includes creating, editing, deleting user accounts. The application should also provide a file upload section where user can upload images or videos to use in an article.

### 5.2 What do we have in hands

Thanks to the Java open source community, there are many tools we could use to get our project done. Let’s look at the preparation phase and what are the tools that we are going to use in our project:

- **Eclipse**: An open source IDE, mostly provided in Java, focused on building an extensible development platform, runtimes and application frameworks for building, deploying and managing software across the entire software lifecycle.

- **Ant**: A pure Java build tool, simpler and easier to use than GNU Make.

- **MySQL**: The world’s most popular open source database.

- **Log4j**: An open source Java logging tool, enable logging at runtime without modifying the application binary.

- **Hibernate**: An open source object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database.

- **Struts**: An open source framework for building Servlet/JSP based web applications based on the Model-View-Controller (MVC) design paradigm.

- **Spring**: An open source framework could be used as a system for assembling components via configuration files with its Inversion of Control (IoC) container.
5.3 THE ARCHITECTURE

All the tools we needed are open source tools, hence it’s easy to download all latest versions of these tools directly from their websites. However, the task of deploying and connecting these frameworks together effectively in our application is not easy. In the next sections we will describe how to make it possible.

5.3 The architecture

There are many things to consider when structuring an architecture for an application. In a layered system with presentation, business logic, data persistence layers, we are faced with decisions about how we are going to construct user interfaces, where the business logic will reside, and how to persist application data. Each of these three layers has their own questions to be answered. What technologies should be implemented across each layer? How can the application be designed so that it is loosely coupled and flexible to change? Does the architecture allow layers to be replaced without affecting other layers? How will the application handle container level services such as transactions? We could think about using a web development framework to get answer to these questions. A good framework however often address only one problem well, but our application will have several layers that might require their own framework. Hence, a combination of several frameworks is the solution to structure our architecture. Again, this is not an easy task.

For our application we will use the strategy for combining frameworks using three popular open source frameworks. For the presentation layer we will use Struts, for our business layer we will use Spring, and for our persistence layer we will use Hibernate. This will also give us the ability to be able to substitute any one of these frameworks in our application and get the same effect. Figure 6.1 shows what this architecture looks like from a high level when the frameworks are combined.

![Figure 5.1: Framework architecture with Struts, Spring, and Hibernate.](image-url)
5.3.1 The layered architecture

Most non-trivial web applications can be divided into at least four layers of responsibility. These layers are the presentation, persistence, business, and domain model layers. Each layer has a distinct responsibility in the application and should not mix functionality with other layers. Each application layer should be isolated from other layers but allow an interface for communication between them. Let’s start by inspecting each of these layers and discuss what these layers should provide and what they should not provide.

5.3.1.1 The presentation layer

At one end of a typical web application is the presentation layer. Many Java developers understand what Struts provides. However, too often, coupled code such as business logic is placed into an org.apache.struts.Action. Here is what Struts is responsible for:

► Managing requests and responses for a user.
► Providing a controller to delegate calls to business logic and other upstream processes.
► Handling exceptions from other tiers that throw exceptions to a Struts Action.
► Assembling a model that can be presented in a view.
► Performing UI validation.

Here are some items that are often coded using Struts but should not be associated with the presentation layer:

► Direct communication with the database, such as JDBC calls.
► Business logic and validation related to your application.
► Transaction management.

Introducing this type of code in the presentation layer leads to type coupling and cumbersome maintenance.

5.3.1.2 The Persistence Layer

At the other end of a typical web application is the persistence layer. This is usually where things get out of control fast. Developers underestimate the challenges in building their own persistence frameworks. A custom, in-house persistence layer not only requires a great amount of development time, but also often lacks functionality and becomes unmanageable. There are several open source object-to-relational mapping (ORM) frameworks that solve much of this problem. In particular, the Hibernate framework allows object-to-relational persistence and query service for Java. Hibernate has a medium learning curve for Java developers who are already familiar with SQL and the JDBC API. Hibernate persistent objects are based on plain-old Java objects and Java collections. Furthermore, using Hibernate does not interfere with your IDE. The following list contains the type of code that you would write inside a persistence framework:
5.3. THE ARCHITECTURE

- Querying relational information into objects. Hibernate does this through an object-oriented query language called HQL, or by using an expressive criteria API. HQL is very similar to SQL except you use objects instead of tables and fields instead of columns. There are some new specific HQL language elements to learn; however, they are easy to understand and well documented. HQL is a natural language to use for querying objects that require a small learning curve.

- Saving, updating, and deleting information stored in a database.

- Advanced object-to-relational mapping frameworks like Hibernate have support for most major SQL databases, and they support parent/child relationships, transactions, inheritance, and polymorphism.

Here are some items that should be avoided in the persistence layer:

- Business logic should be in a higher layer of your application. Only data access operations should be permitted.

- You should not have persistence logic coupled with your presentation logic. Avoid logic in presentation components such as JSPs or servlet-based classes that communicate with data access directly. By isolating persistence logic into its own layer, the application becomes flexible to change without affecting code in other layers. For example, Hibernate could be replaced with another persistence framework or API without modification to the code in any other layer.

5.3.1.3 The Business Layer

The middle component of a typical web application is the business or service layer. This service layer is often the most ignored layer from a coding perspective. It is not uncommon to find this type of code scattered around in the UI layer or in the persistence layer. This is not the correct place because it leads to tightly coupled applications and code that can be hard to maintain over time. Fortunately, several frameworks exist that address these issues. Two of the most popular frameworks in this space are Spring and PicoContainer. These are referred to as microcontainers that have a very small footprint and determine how you wire your objects together. Both of these frameworks work on a simple concept of dependency injection (also known as inversion of control). Our application will focus on Spring’s use of setter injection through bean properties for named configuration parameters. Spring also allows a sophisticated form of constructor injection as an alternative to setter injection as well. The objects are wired together by a simple XML file that contains references to objects such as the transaction management handler, object factories, service objects that contain business logic, and data access objects (DAO).

The way Spring uses these concepts will be discussed later in the next section. The business layer should be responsible for the following:

- Handling application business logic and business validation

- Managing transactions

- Allowing interfaces for interaction with other layers
Managing dependencies between business level objects

Adding flexibility between the presentation and the persistence layer so they do not directly communicate with each other

Exposing a context to the business layer from the presentation layer to obtain business services

Managing implementations from the business logic to the persistence layer

5.3.1.4 The Domain Model Layer

Finally, since we are addressing non-trivial, web-based applications we need a set of objects that can move between the different layers. The domain object layer consists of objects that represent real-world business objects. This layer allows developers to stop building and maintaining unnecessary data transfer objects, or DTOs, to match their domain objects. For example, Hibernate allows us to read database information into an object graph of domain objects, so that we can present it to your UI layer in a disconnected manner. Those objects can be updated and sent back across to the persistence layer and updated within the database. Furthermore, we do not have to transform objects into DTOs, which can get lost in translation as they are moved between different application layers. This model allows Java developers to work with objects naturally in an object-oriented fashion without additional coding.

5.4 The real work

5.4.1 Setting up

After having in mind the architecture of our application, the list of tools that we need, we could start the development work. First, we need to have all the tools ready, assume that we have downloaded the latest production release of Hibernate from the Hibernate website at http://www.hibernate.org and that we unpacked the archive. We also need Apache Ant installed on our development machine. Then we need to have Struts2 and Spring ready. The latest versions of Struts2 and Spring at the time of this document are Struts 2.0.11 available from http://struts.apache.org and Spring Framework version 2.5.0 from http://www.springframework.org.

We also need to have our MySQL database server running on our machine or on some remote machines that we have access to. MySQL is a very popular open source database server, it is always good to have our MySQL database server downloaded and installed on our local machine so that we have more control on the whole system, if you have any problem with installation, there would be a lot of useful information on the Internet that can help you.

An important part is also choosing the application server. For this purpose, we use Resin - a high-performance, scalable Java application server. This choice was due to the performance and the author’s own experience with Resin.

If you haven’t heard a lot about Resin, Resin is the application server of choice for over 6,000 organizations including Fortune 500, innovative startups, government and educational institutions needing reliable and fast Java-PHP solutions. In the latest version, Resin introduced Quercus, the Java implementation of PHP 5 that has been benchmarked at four to six times faster than mod_php. Since Quercus is included with Resin, you can run all your applications - web server, Java, PHP - within one open source software product that has rock
solid stability, is fast and easy to configure. This means that you can run Resin both as a Java web application server and as a PHP web server.

Now that we have all the tools prepared, let’s get to the real work.

### 5.4.2 Creating the work directory

For better of debugging and testing, we will create our application directory directly inside WEBAPPS directory of Resin server. This will enable us to run and test our application directly with Resin since Resin will automatically load all web applications from its WEBAPPS directory on start up.

We also need to create a start up script and configuration for Resin for our application. Since our development process is carried out under Linux, for this purpose we will create our shell script `startresin.sh`:

```bash
#!/bin/sh
JAVA_HOME=/usr/java/jdk1.6
RESIN_HOME=/usr/local/resin3 ROOT_DIRECTORY=/home/www
java=$JAVA_HOME/bin/java
export JAVA_HOME
export RESIN_HOME
export ROOT_DIRECTORY
$java -jar $RESIN_HOME/lib/resin.jar -conf /home/www/conf/resin.conf
```

Environment variable JAVA_HOME and RESIN_HOME points to our home directory of JAVA and RESIN, ROOT_DIRECTORY is where we will place our application directory. For our Resin configuration, `resin.conf` inside `/conf` directory is needed.

Except other parts about the webserver configuration, we will define a host for our application in `resin.conf`:

```xml
<host id="cms.me" root-directory="cms">
  <web-app id="/" root-directory="cms-web/webapps">
  </web-app>
</host>
```

This will enable us to use different hosts for different application on our local machine and whenever we want to access our application locally we could just go to http://cms.me from any of our web browsers.

Now let’s look at our web application’s directory:
CHAPTER 5. DESIGN AND IMPLEMENTATION OF A CMS SYSTEM

#CMS
+src
+logs
- webapps
  + cms
  + jsp
  - WEB-INF
    + classes
    + config
    + lib
    + tld

▶ src: directory is where we place all our Java source codes later

▶ logs: directory for saving logs

▶ cms: directory for all files (mostly jsp) contributing to CMS administration interface.

▶ jsp: directory for all jsp files of the application.

▶ WEB-INF: contains resources pertaining to the Web application including a web.xml file.

▶ classes: all compiled classes from src directory will be placed here.

▶ config: contains configuration files such as webApplicationContext.xml or struts.xml

▶ lib: directory where Resin looks for libraries, we will place our .jar files here.

▶ tld: directory contains Tag Library Descriptor files.

In order to use Ant, under our application directory we also have to write a build.xml file. When running Ant under this directory, Ant will automatically look for build.xml and carry out the build task with data from this file. We will also create build.properties to hold the parameters to apply in build.xml such as classpath directory and library location. The content of build.properties is as follow:

src.dir = src
webinf.dir = webapps/WEB-INF
classes.dir = ${webinf.dir}/classes
lib.dir = ../lib
lib.local.dir = ${webinf.dir}/lib
config.dir = ${webinf.dir}/config

For our build script, we define 3 targets for this build file: clean, copyXmbs and compile. By default when is called, Ant will execute compile, in which clean and copyXmbs are executed as well, this process will clean up old compiled file, copy all .xml and .properties descriptor files to our build destination and recompile all source files.
5.4. THE REAL WORK

Our clean target in build.xml:

```xml
<target name="clean">
  <delete failonerror="no">
    <fileset dir="${classes.dir}"
      <include name="**/**"/>
  </fileset>
</delete>
</target>
```

Our copyXmls target in build.xml:

```xml
<target name="copyXmls" description="copying required xml descriptors to binaries">
  <echo>copying required xml descriptors to binaries</echo>
  <copy todir="${classes.dir}"
    <fileset dir="${src.dir}"
      <include name="**/*.xml"/>
      <include name="**/*.properties"/>
      <include name="**/*.Properties"/>
    </fileset>
  </copy>
</target>
```

Our compile target in build.xml:

```xml
<target name="compile" depends="clean,copyXmls">
  <javac srcdir="${src.dir}" compiler="modern"
    destdir="${classes.dir}"
    classpathref=".classpath"
    source="1.5" target="1.5"
    encoding="iso-8859-1"
    debug="on">
    <include name="**/*.java"/>
    <include name="**/*.xml"/>
  </javac>
</target>
```

These configurations will enable Ant to compile and do the deployment for us just simply by typing 'ant' in our application directory. A successful run of Ant will look like this:
5.4.3 Database design

Once we have MySQL server installed, we will need to create a database for our application, after that several tables need to be created to represent our business objects. Since we are dealing with web articles and a news website, we need to create those basic tables:

- **Articles:** to store articles for the website.
- **Categories (Cats):** to store categories of articles that we have on our website
- **Site:** table to store all information about our site.
- **Users:** table to store all user informations.
- **Files:** table to store information about uploaded files.

Besides those basic tables providing basic functionalities of our web application we may later want to extend the functionalities with comments table, sub categories table, layout table to store information about website layout, components. An advertisement component could be added also if our application goes commercial. For now we will focus only on the basic functionalities.

First we will create a database called `webapps` in MySQL, after that we will create these needed tables for our application. We will also create a properties file with JDBC-related settings for the connection to our database server. We will call this file `jdbc.properties`, later on this file will be used by Spring to inject these properties in creating connection to our database. Content of `jdbc.properties` is as follow:
5.4. THE REAL WORK

# Properties file with JDBC-related settings.
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/webapps
jdbc.username=<username>
jdbc.password=<password>

# Property that determines the Hibernate dialect
hibernate.dialect=org.hibernate.dialect.MySQLDialect

Now, back to to the basic tables for our application, let’s create Articles table using a SQL query:

```sql
CREATE TABLE 'webapps'.'Articles' (
    'articleID' int(10) unsigned NOT NULL auto_increment,
    'name' varchar(50) character set latin1 NOT NULL,
    'title' varchar(300) character set latin1 NOT NULL,
    'photoID' int(11) default '0',
    'introText' text character set latin1,
    'content' text character set latin1,
    'catID' int(11) default '0',
    'subCatID' int(11) default '0',
    'visible' tinyint(1) NOT NULL default '0',
    'authorID' int(11) NOT NULL,
    'type' varchar(3) character set latin1 default 'ART',
    'createDate' datetime NOT NULL default CURRENT_TIMESTAMP,
    'modifyDate' datetime NOT NULL,
    PRIMARY KEY ('articleID')
) ENGINE=MyISAM DEFAULT CHARSET=utf8
```

We can see from here each article has name, title, photoID, introText, content, catID, subCatID, visible, authorID, type, createDate, modifyDate. If Visible == 1 the article will be displayed on the website. Type of article is set to 'ART' by default to indicate that it is an article or can be set to 'NEW' to indicate is a news item. Following is the query to create Users table:

```sql
CREATE TABLE 'webapps'.'Users' (  
    'id' int(11) NOT NULL auto_increment,  
    'userName' varchar(20) NOT NULL,  
    'password' varchar(20) NOT NULL,  
    'email' varchar(30) NOT NULL,  
    'isOnline' tinyint(1) default NULL,  
    'userGroup' varchar(3) default 'GUE',  
    PRIMARY KEY ('id')
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COMMENT='Users table';
```

Users table has the following fields: id, userName, password, email, isOnline, userGroup. UserGroup is set to 'GUE' by default to indicate guest group or can be set to 'ADM', 'MOD','USR' to indicate Administrator, Moderator, User group.
5.4.4 Application architecture

Since we have our directory and the development environment ready, we can now start the design of each layer of our application. Stick to our layered architecture, we will design our domain object layer, then persistence layer, business layer, finally presentation layer.

5.4.4.1 Domain object layer

We will start with creating our domain objects since they will inter-operate with each layer. These objects will allow us to define what should be persisted, what business logic should be provided, and what type of presentation interface should be designed. Next, we will configure the persistence layer and define object-to-relational mappings with Hibernate for our domain objects. Then we will define and configure our business objects. After we have these components we can discuss wiring these layers using Spring. Finally, we will provide a
presentation layer that knows how to communicate with the business service layer and knows how to handle exceptions that arise from other layers.

Since these objects will interoperate across all layers this might be a good place to start coding. Our domain model will contain objects that represent user and article. We will define our business objects as classes implementing Serializable interface:

- `article.business.Article`: contains information for an article.
- `cat.business.Cat`: contains information for a category.
- `user.business.User`: contains information for an user.

Let's look at `Article.java` to see how we could do this:

```java
package article.business;

import java.io.Serializable;
import java.util.Date;

public class Article implements Serializable{

    private Integer articleID;
    private String name;
    private String title;
    private int photoID;
    private String introText;
    private String content;
    private int catID;
    private int subCatID;
    private int visible;
    private String type;
    private Date createDate;
    private Date modifyDate;
    private int authorID;

    public Article() {
    }
    .....//setters and getters methods
}
```

For the purpose of persistence, our class implements Serializable interface. You may have noticed that all attributes of the Article class have JavaBeans style property accessor methods. The class also has a constructor with no parameters. The persistent classes we use in our application will almost always look something like this. The no-argument constructor is a requirement (tools like Hibernate use reflection on this constructor to instantiate objects).

We consider choosing package names for your objects that reflect how your application is layered. In our application, all the business objects will have “business” on their package
name. After that, more specialized domain objects would be located in subpackages under the `user.business` or `article.business` package. The business logic begins with DAO objects are located in the `article.dao`, `cat.dao` or `user.dao` package. This packages contains only abstract classes or interfaces, the implementation of them are located in `article.dao.impl`, `cat.dao.impl` or `user.dao.impl` package. This enables us to have several implementation of a business object, we can implement using Hibernate or change to other implementation just by change to other implementation from `.impl` packages without effecting any other part of business objects. The presentation classes for forms and actions reside in `action` package. Accurate package naming provides a clear separation for the functionality that your classes provide, allows for easier maintenance when troubleshooting, and provides consistency when adding new classes or packages to the application.

5.4.4.2 Persistence Layer Configuration

This is the place where Hibernate comes into play. There are several steps involved in setting up the persistence layer with Hibernate. The first step is to configure our domain business objects to be persisted. Therefore the Article, Cat, File and User objects will need to provide getter and setter methods for all fields that they contain. These objects would contain setter and getter methods for its fields a standard JavaBean format.

Hibernate maps domain objects-to-relational databases in XML files. For this purpose our Article, Cat, File and User objects will need their mapping files to express this. There are tools such as XDoclet to assist with this mapping. However, we will write this manually to get a better view on how Hibernate works. Hibernate will map the domain objects to these files:

- `User.hbm.xml`
- `Cat.hbm.xml`
- `Article.hbm.xml`
- `File.hbm.xml`

Let’s look at `Article.hbm.xml` to see how do we map fields of an article to columns of Articles table:
This looks quite straightforward. We can see that all of the elements used in this mapping file were explained earlier in chapter 3. Similar mappings were done for Article and Cat.

A question may arise about the location of these mapping files in our application. The answer is simple, we will place these files in the same directory with our business objects, this means that User.hbm.xml for example will be in the same directory with User.java (from user.business package). The Hibernate SessionFactory is configured to know which database it is communicating with, the DataSource or connection pool to use, and what persistent objects are available for persistence. Session objects provided by the SessionFactory are the interface used to translate between Java objects and persistence functions such as selecting, saving, updating, and deleting objects. We will discuss configuring the SessionFactory that Hibernate requires to handle Session objects in a later section.

Each time we compile using our build script with Ant tool, these XML files will also be copied to classes directory under WEB-INF. You will find these files for example User.hbm.xml in WEB-INF/classes/user/business/ directory.

5.4.4.3 Business Layer Configuration

This is the place where Spring framework gets into the game. Since we have our domain objects, we need to have business service objects that perform application logic, make calls to the persistence layer, take requests from the UI layer, deal with transactions, and handle
exceptions. To connect all of this together and make this easy to manage we will use the bean management aspect of the Spring framework. Spring uses inversion of control (IoC), or setter dependency injection, to wire up objects that are referenced in an external XML file. Inversion of control is a simple concept that allows objects to accept other objects that are created at a higher level. This way your object is free from having to create objects and reduces object coupling.

Here is an example of an object creating its dependencies without IoC, which leads to tight object coupling:

And here is an example with IoC that allows objects to be created at higher levels and passed into objects so that they can use the implementations directly:
5.4. THE REAL WORK

Objects arranged with IoC. Object A contains setter methods that accept interfaces to objects B and C. This could have also been achieved with constructors in object A that accepts objects B and C.

5.4.4.4 Business Service Objects

As we have understood the IoC concept Spring, we will apply it into our application. The setters we will use in our business objects accept interfaces that allow loosely defined implementations of the objects that will be set, or injected. In our case we will allow our business service object to accept a DAO to handle the persistence of our domain objects. This also gives us the ability to easily switch implementations to a different persistence framework than Hibernate and inform Spring of the new implementation DAO object to use.

Here is the interface for our user service object that is designed for the injection of a DAO object dependency:
package user.service.impl;

import java.sql.SQLException;
import java.util.Collection;
import user.business.User;
import org.springframework.dao.DataAccessException;
import user.dao.UserDAO;
import user.service.UserService;

public class UserServiceImpl implements UserService {
    private UserDAO userDAO; // <---- Spring IoC is used here

    public boolean checkUserLogin(String strUserName, String strPassword)
            throws DataAccessException, SQLException {
        return userDAO.checkUserLogin(strUserName, strPassword);
    }

    public void insertUser(User obj) throws DataAccessException {
        userDAO.insertUser(obj);
    }

    public void setUserDAO(UserDAO userDAO) {
        this.userDAO = userDAO;
    }
}

Some of us will notice that the code above has a setter for a DAO object but no getUser-DAO method because it is not necessary since there is often no need to access the injected UserDAO object from the outside. The DAO object will be used to communicate with our persistence layer. Again using an interface allow us to switch between different implementations.

Now is the time to code our DAO implementation object. As we use Hibernate for our persistence layer, we will implement our DAO interfaces using Hibernate. The following DAO will be implemented:

▶ user.dao.impl.UserDAOHibernate.java
▶ article.dao.impl.ArticleDAOHibernate.java
▶ cat.dao.impl.CatDAOHibernate.java
▶ file.dao.impl.FileDAOHibernate.java

As we can see, we place all our DAO implementation into respective .impl packages. This is again a good convention to follow for a systematic view, flexibility and maintainability of the whole application.
Let’s create our interface for ArticleDAO first before we implement it:
package article.dao;

import java.sql.SQLException;
import java.util.Collection;
import org.springframework.dao.DataAccessException;
import article.business.Article;

public interface ArticleDAO {

    /**
     * Saves article object to the datastore.
     */
    public void insertArticle(Article obj) throws DataAccessException;

    /**
     * Update article object to the datastore.
     */
    public void updateArticle(Article obj) throws DataAccessException;

    /**
     * @param id
     * @return Article if id exists
     * @throws DataAccessException
     */
    public Article getArticle(Integer id) throws DataAccessException;

    /**
     * @param strArticleid
     * @return true if Articleid exists
     * @throws DataAccessException
     * @throws java.sql.SQLException
     */
    public boolean checkValidArticleName(String strArticleid)
            throws DataAccessException, java.sql.SQLException;

    /**
     * @return list of Articles
     * @throws DataAccessException
     * @throws java.sql.SQLException
     */
    public Collection getArticlesList() throws DataAccessException,
            java.sql.SQLException;

    ....
}
Since Spring has built-in support for Hibernate, our DAO Hibernate implementation will extend the HibernateDaoSupport class, which allows us to easily get a reference to a HibernateTemplate, which is a helper class that simplifies coding with a Hibernate Session and handles HibernateExceptions. Let’s look at how we implemented article.dao.impl.ArticleDAOHibernate.java in our application:

```java
package article.dao.impl;

import org.springframework.dao.DataAccessException;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import article.business.Article;
import article.dao.ArticleDAO;

public class ArticleDAOHibernate extends HibernateDaoSupport implements ArticleDAO{

    public void insertArticle(Article obj) throws DataAccessException {
        getHibernateTemplate().save(obj);
    }

    public void updateArticle(Article obj) throws DataAccessException {
        getHibernateTemplate().update(obj);
    }
    ....
}
```

As we can see by calling `getHibernateTemplate()` provided by Spring, we could save a lot of time with implementing codes dealing with Hibernate Session and Exceptions.

We still have a couple more objects to wire together for our business layer. This includes the HibernateSessionFactory and a TransactionManager object. This is done directly in the Spring configuration file. Spring provides a HibernateTransactionManager, which will bind a Hibernate Session from the factory to a thread to support transactions. Here is the Spring configuration of the HibernateSessionFactory, HibernateTransactionManager and all the business objects definition used in our application:

```xml
<beans>
  <!--============ Start of PERSISTENCE DEFINITIONS ============= -->
  <bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location">
      <value>jdbc.properties</value>
    </property>
  </bean>
  <!--============ End of PERSISTENCE DEFINITIONS ============= -->
</beans>
```

<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName">
    <value>${jdbc.driverClassName}</value>
  </property>
  <property name="url">
    <value>${jdbc.url}</value>
  </property>
  <property name="username">
    <value>${jdbc.username}</value>
  </property>
  <property name="password">
    <value>${jdbc.password}</value>
  </property>
</bean>

<!-- JNDI DataSource for J2EE environments -->
<bean id="dataConnection" class="site.DataConnectionImpl">
  <property name="dataSource">
    <ref local="dataSource"/>
  </property>
</bean>

<!-- Hibernate SessionFactory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="dataSource">
    <ref local="dataSource"/>
  </property>
  <property name="mappingResources">
    <list>
      <value>user/business/User.hbm.xml</value>
      <value>article/business/Article.hbm.xml</value>
      <value>cat/business/Cat.hbm.xml</value>
      <value>file/business/File.hbm.xml</value>
    </list>
  </property>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">${hibernate.dialect}</prop>
      <prop key="hibernate.show_sql">true</prop>
    </props>
  </property>
</bean>
5.4. THE REAL WORK

```xml
<bean id="hibernateTemplate"
    class="org.springframework.orm.hibernate3.HibernateTemplate">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>

<!-- Transaction manager for a single Hibernate SessionFactory -->
<bean id="transactionManager"
    class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory">
        <ref local="sessionFactory" />
    </property>
</bean>

<!---------------- BUSINESS OBJECT DEFINITIONS ------------------->

<!-- Data access object: Hibernate implementation. -->
<bean id="userServiceTarget" class="user.service.impl.UserServiceImpl">
    <property name="userDAO">
        <ref local="userDAO" />
    </property>
</bean>

<bean id="articleDAO" class="article.dao.impl.ArticleDAOHibernate">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>

<bean id="userDAO" class="user.dao.impl.UserDAOHibernate">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>

<bean id="catDAO" class="cat.dao.impl.CatDAOHibernate">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>

<bean id="fileDAO" class="file.dao.impl.FileDAOHibernate">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>
```
CHAPTER 5. DESIGN AND IMPLEMENTATION OF A CMS SYSTEM

Each object can be referenced in the Spring configuration within a `<bean>` tag. In our case the bean `SessionFactory` represents a HibernateSessionFactory and the bean `transactionManager` represents a Hibernate transaction manager. Notice that the `transactionManager` bean has a property element called `sessionFactory`. The HibernateTransactionManager class has a setter and getter for `sessionFactory`, which is used for dependency injection when the Spring container starts. The `sessionFactory` property references the `sessionFactory` bean. These two objects will be wired together when the Spring container initializes. This wiring relieves us from creating singleton objects and factories for referencing and creating these objects, which reduces code maintenance in your application.

The `sessionFactory` bean has three property elements, which translate to setters for dataSource, mappingResources and hibernateProperties. Normally, this configuration would be stored in the hibernate.cfg.xml file if you were using Hibernate outside of Spring. However, Spring provides an easy way to incorporate the Hibernate configuration within the Spring configuration file. This was done in using properties in our `sessionFactory` and `dataSource` beans.

Our data access object beans `catDAO`, `articleDAO`, `userDAO`, `fileDAO` were implemented using Hibernate and these classes extend HibernateDaoSupport, hence we could inject `sessionFactory` directly into these beans using `sessionFactory` property.

5.4.4.5 UI Layer Configuration

The UI Layer for our application uses the Struts framework. Here we will discuss what is related to Struts when layering an application. Let’s begin by examining an Action configuration within the `struts-main.xml` file.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
  <package name="user" namespace="/cms" extends="struts-default">
    <action name="SignIn" class="action.user.LoginAction">
      <result name="input">/cms/login.jsp</result>
      <result type="redirect-action">Menu</result>
    </action>
    <action name="Logout" class="action.user.LogoutAction">
      <result>/cms/login.jsp</result>
    </action>
  </package>
</struts>
```
5.4. THE REAL WORK

For article actions:

```xml
<!-- ARTICLE ACTIONS -->
<action name="EditArticleAction" class="action.article.EditArticleAction">
  <result>/cms/ajax/editArticleAjax.jsp</result>
</action>
<action name="NewArticleAction" class="action.article.NewArticleAction">
  <result>/cms/ajax/newArticleAjax.jsp</result>
</action>
<action name="ArticleListAction" class="action.article.ArticleListAction">
  <result name="input">/cms/articleManager.jsp</result>
  <result>/cms/articleManager.jsp</result>
</action>
<action name="GoArticleAction" class="action.article.GoArticleAction">
  <result name="input">/cms/newArticle.jsp</result>
  <result>/cms/editArticle.jsp</result>
</action>
....
```

Whenever a user request for something, then the request is handled by FilterDispatcher. When the FilterDispatcher receives the request, it intercepts the URL and based on the Struts Configuration files, it gives the handling of the request to the appropriate Action.

Let’s analyze one of these actions. The NewArticleAction is used to persist an article that the user submitted from the UI layer. This is a typical Struts Action. Since we use AJAX to submit data from new article form, the result of this Action will just be the section to display on the page to indicate if our article is successfully saved. Inside newArticleAjax.jsp there will be only a testing condition which read value of parameter saveArticleOK returned from NewArticleAction and display “Article was saved!” or “Error inserting article!” respectively.

The the following section we will get into the insight of the design and implementation process.

5.4.5 Design and Implementation

5.4.5.1 Login box

First we need a login box to provide an interface for user to log in.
This is just a form with two text fields, when data is submitted a login action will handle the login process. The action class implemented in our application for this purpose is `action.user.LoginAction.java`. A default administrator account for first login is username: `admin`, password: `admin`. Login information after successfully validated against data in database will be stored in session.

### 5.4.5.2 Administration interface

For our CMS administration interface, we will design a page with menu on the left and navigation menu on top. The directory structure of our CMS administration interface is as follow:

```
|-- cms
 | |-- ajax
 | | |-- deleteArticleAjax.jsp
 | | |-- deleteCatAjax.jsp
 | | |-- deleteUserAjax.jsp
 | | |-- editArticleAjax.jsp
 | | |-- editCatAjax.jsp
 | | |-- editLayoutAjax.jsp
 | | |-- editUserAjax.jsp
 | | |-- newArticleAjax.jsp
 | | |-- newCatAjax.jsp
 | | |-- newUserAjax.jsp
 | | |-- articleManager.jsp
 | | |-- catManager.jsp
 | | |-- checkLogin.jsp
 | | |-- css
 | | |-- '-- cms.css
```
5.4. THE REAL WORK

| |-- editArticle.jsp
| |-- editCat.jsp
| |-- editCatByID.jsp
| |-- editUser.jsp
| |-- editUserByID.jsp
| |-- img
| | |-- images
| | | |-- backcolor.gif
| | | |-- view_text.gif
| | | |-- view_text_on.gif
| | | |-- left_close.gif
| | | |-- logo.gif
| | |-- popups
| | | |-- about.html
| | | |-- about_license.html
| | | |-- create_table.html
| | | |-- insert_hyperlink.html
| | | |-- insert_image.html
| | | |-- license.html
| | | |-- preview.html
| | | |-- select_color.html
| | | |-- styles
| | | |-- wysiwyg.css
| | |-- index.html
| |-- js
| | |-- wysiwyg-color.js
| | |-- wysiwyg-popup.js
| | |-- wysiwyg-settings.js
| | |-- wysiwyg.js
| |-- layoutManager.jsp
| |-- login.jsp
| |-- main.jsp
| |-- menubar.jsp
| |-- menuset.jsp
| |-- nav.jsp
| |-- newArticle.jsp
| |-- newCat.jsp
| |-- newUser.jsp
| |-- templateBase.jsp
| |-- tools.jsp
| |-- upload-success.jsp
| |-- upload.jsp
| |-- userManager.jsp
|-- fileUpload
| |-- 1200163857790.jpg
| |-- 1200166986643.jpg
| |-- 1200188144906.jpg

Images for CMS interface
Images for openWYSIWYG HTML editor
Html for openWYSIWYG HTML editor
Java scripts for CMS interface
Uploaded files will be saved here
The layout of our interface after spending time with CSS and HTML design is as follow:

![CMS Administrator Interface](image)

Figure 5.6: CMS Administrator Interface

We call our CMS System MYVIKO and create a small logo for it. For the interaction between users and the system, we also needed to write several Java Script functions to make it easy and user-friendly. The detail functions can be seen in the attached source codes along with this documentation.

After finish with the layout design, thinking more about the functionalities, we should have three main sections in our CMS administrator:

- **Article Manager**: where we manage all articles.
- **Category Manager**: where we manage categories.
- **User Manager**: where we manage users.

Beside these sections, we also provide **Site Manager**, **Layout Manager**, **File Upload** sections to directly manage the appearance of our website. For each of this section, we will create an interface in the layout form of a table to conveniently look through all items.

### 5.4.5.3 Article Manager

Let’s start with **Article Manager**:
In Article Manager section, we will be able to chose articles that belongs to different categories, after a category is chosen, all the articles that belong to that category will be listed in the table, we can click on the name to edit a particular article or click on Delete to delete them. This is done with choosing a category from a selection list. In Struts 2, we could do that by:

```xml
<s:select id="catSelect" label="Category"
   headerKey="-1" headerValue="All Categories "
   name="catID"
   list="catList"
   listKey="catID"
   listValue="catName"
   multiple="false"
   value="%(catID)"
   onchange="reloadPage();"
/>
<br/>
```

It looks pretty straight forward that this selection would work, however by default, surprisingly it would not work in Struts 2. It took me a while to figure out why this piece of code did not work. I came up with a solution from a discussion forum, it turned out that for using of this selection in Struts, we need to add `<%@ page deferredSyntaxAllowedAsLiteral="true" %>` to our jsp whenever we use Struts selection tags with `<s:select with list="#{..}"`. A part from that, we also need to write a `reloadPage()` Java script for this selection box:
function reloadPage() {
    var catID = document.getElementById('catSeclect');
    var hrefOnChange='ArticleListAction.action?catID='+
    document.getElementById('catSelect').value;
    window.location.href = hrefOnChange;
}

▶ When click on to edit an article we will just actually execute an action with a parameter:  \textit{GoArticleAction?articleID=<current article ID>}. This action will result in \textit{editArticle.jsp} as we can see from Struts action mapping file.

▶ Deleting an article is implemented in the AJAX way, when we click on Delete, we will execute another action also with \textit{articleID} as its parameter: \textit{DeleteArticleAction?articleID=<current article ID>}. This action will result in the AJAX way, the page will not be reloaded but only the row with the deleted article will disappear.

Now let’s analyze how we could make the Edit Article page, since the process is also quite similar for other CMS pages. First, let’s look at the appearance of \textit{editArticle.jsp}:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{editArticleForm.png}
\caption{Edit Article form}
\end{figure}

This page is designed as a form with several text fields, we also used a WYSIWYG html editor for the ‘content’ text field of Edit Article page. For this purpose, I decided to used
openWYSIWYG scripts. These Java Scripts are placed in /cms/js directory. To use this editor on any text box, theoretically the following codes need to be added to our jsp file:

```
<s:head theme="ajax"/> <!-- AJAX theme from Struts
<script type="text/javascript" src="/cms/js/wysiwyg.js"></script>
<script type="text/javascript" src="/cms/js/wysiwyg-settings.js"></script>

<script type="text/javascript">
  WYSIWYG.attach(‘content’);
</script>
</s:head>
```

However, since we used both AJAX theme from Struts and Java Scripts from openWYSIWYG editor version 1.47, this leads to a complication. The reason is this two versions does not work well together and the problem was the content of of the text field is not updated when we press 'submit' form. To fix this issue we need to call a Java Script function “WYSIWYG.updateTextArea(‘content’);” before we submit. This can be done by adding “onmouseover=WYSIWYG.updateTextArea(‘content’);” to `<s:submit>` tag.

Now that we have the form, the next thing we see is that we need to read article data to be able to edit them. This is where a Struts 2 action comes into play. The jsp file has the role of a Viewer, this Viewer needs data from a Model, hence, as we have mentioned above an action like `GoArticleAction?articleID=<current article ID>` will have to fulfill this job.

```java
package action.article;

import article.business.Article;
import article.dao.impl.ArticleDAOHibernate;
import cat.dao.impl.CatDAOHibernate;
import com.opensymphony.xwork2.ActionSupport;
import java.util.*;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class GoArticleAction extends ActionSupport{
    private Article article;
    private Integer articleID;
    private List catList;

    public String execute()throws Exception{
        if (!articleID.equals(null)){
            ArticleDAOHibernate articleDAO = (ArticleDAOHibernate) (new ClassPathXmlApplicationContext("webApplicationContext.xml").
                getBean("articleDAO").
            );
            article = articleDAO.getArticle(articleID);
        }
    }
```
CatDAOHibernate catDAO = (CatDAOHibernate) (new ClassPathXmlApplicationContext("webApplicationContext.xml").
g.getBean("catDAO"));
catList = (List) catDAO.getCatList();
if (articleID.intValue()==-1) return INPUT;
else return SUCCESS;

// Getters and setters
public Article getArticle(){
    return article;
}

public Integer getArticleID() {
    return articleID;
}

public void setArticleID(Integer articleID) {
    this.articleID = articleID;
}

public List getCatList() {
    return catList;
}

public void setCatList(List catList) {
    this.catList = catList;
}

After that, we also need an action to update our article, EditArticleAction will do this:

package action.article;

import article.business.Article;
...

public class EditArticleAction implements Action, Serializable {

    private static int counter = 0;
    private int ok =0;

    private Integer articleID;
    private String name;
private String title;
....

// Getters and setters
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
....

public String execute() throws Exception {
    ArticleDAOHibernate articleDAO = (ArticleDAOHibernate)
    (new ClassPathXmlApplicationContext("webApplicationContext.xml")
    .getBean("articleDAO");
    createDate = articleDAO.getCreateDate((Integer) articleID);
    modifyDate = new Date();
    Article article = new Article(articleID,name,title,
    photoID,introText,content,catID,subCatID,visible,
    type,createDate,modifyDate,authorID);
    articleDAO.updateArticle(article);
    this.ok = 1;
    return SUCCESS;
}

EditArticleAction works by getting the bean articleDAO from Spring IoC container, after that an article can be updated just by calling articleDAO.updateArticle(article);

5.4.5.4 User Manager

Similar to article manager, our user manager interface looks as follow:
In User Manager section, we could edit user’s data, create new user or delete an user.

- When click on user name to edit an user we will just actually execute an action with a parameter: $GoUserAction?userID=<\text{current user ID}>$. This action will result in $editUser.jsp$ as we can see from Struts action mapping file.

- Like deleting an article, deleting an user is implemented in the AJAX way, when we click on Delete, we will execute an action also with $userID$ as its parameter: $DeleteUserAction?articleID=<\text{current user ID}>$. This action will result in the AJAX way, the page will not be reloaded but only the row with the deleted user will disappear.

Edit User page is designed like this:

In User Manager section, we could edit user’s data, create new user or delete an user.

- When click on user name to edit an user we will just actually execute an action with a parameter: $GoUserAction?userID=<\text{current user ID}>$. This action will result in $editUser.jsp$ as we can see from Struts action mapping file.

- Like deleting an article, deleting an user is implemented in the AJAX way, when we click on Delete, we will execute an action also with $userID$ as its parameter: $DeleteUserAction?articleID=<\text{current user ID}>$. This action will result in the AJAX way, the page will not be reloaded but only the row with the deleted user will disappear.
5.4. THE REAL WORK

For Edit User page, we need to check the group of an user and allow an user to access here only if he belongs to Administrator group. This is done by validating `<s:if test=""#session.userGroup == 'ADM'"">` inside of `editUser.jsp`.

5.4.5.5 Category Manager

Very similar to what we did before with user manager and article manager, category manager works in the same way. For our news website, we added several news categories to make our category manager looks as follow:

![Category Manager](image)

<table>
<thead>
<tr>
<th>Category Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category editor:</td>
</tr>
</tbody>
</table>

Figure 5.11: Category Manager
CHAPTER 5. DESIGN AND IMPLEMENTATION OF A CMS SYSTEM

5.4.5.6 Site Manager

Site Manager is also designed and implemented in the same way. This will enable us to edit columns of 'Site' table in database:

![Site Manager](image)

**Figure 5.13: Site Manager**
5.4. THE REAL WORK

5.4.5.7 File Uploading

When dealing with file upload in latest version at the time of this document Struts 2.0.11, there is a bug that broke the upload process. The interceptor (FileUploadInterceptor) that handles populating the struts action with the values needs the Request object to be an instance of the MultiPartRequestWrapper. This bug is the case the Request has not been wrapped in the MultiPartRequestWrapper. Therefore the FileUploadFilter doesn’t do anything, even though the data is actually there. A fix for it is already there in the next release of Struts 2, we just need to simply download the newer version of the class MultiPartRequestWrapper and this problem is fixed.

To be able to handle the uploaded file, we define our FileUploadAction in Struts configuration:

```xml
<package name="upload" extends="struts-default" namespace="/cms">
  <action name="upload" class="action.FileUploadAction" method="input">
    <result>upload.jsp</result>
  </action>

  <action name="doUpload" class="action.FileUploadAction" method="upload">
    <result name="input">upload.jsp</result>
    <result>upload-success.jsp</result>
  </action>
</package>
```

The interface for file upload is designed in upload.jsp as a form. The form to upload a file in upload.jsp is quite straightforward since we used tag `<s:file>` from Struts:

```xml
<s:form action="doUpload" method="POST" enctype="multipart/form-data">
  <s:file name="upload" size="60"
    cssClass="form_element" label="File"/>
  <s:textfield name="caption" size="60"
    cssClass="form_element" label="Caption"/>
  <s:submit align="left" value="Upload"/>
</s:form>
```

For uploading using Struts, we have to make sure necessary classes for handling and working with files are available in our library, for this purpose we need to add commons-collections-3.1.jar, commons-fileupload-1.1.1.jar, commons-io-1.1.jar to WEF-INF/lib.

In our Action, since Struts2 automatically propagate value of parameters in our form, these codes will read these data in our FileUploadAction.java:
When the action is executed, we also need to copy the uploaded file to `/fileUpload` directory and also add a row with all the file information to database. `upload()` method inside `FileUploadAction.java` will handle this:
public String upload() throws Exception {
    String imageFileName = new Date().getTime() + getExtention(fileName);
    String absolutePath = ServletActionContext.getServletContext().
            getRealPath("/fileUpload") + "/" + imageFileName;
    File imageFile = new File(absolutePath);
    copy(upload, imageFile);
    System.out.println("Cuong:"+ imageFile);

            null,imageFileName,absolutePath,  
            "/fileUpload" + "/" + imageFileName,  
            contentType,caption,"upload from cms");

    FileDAOHibernate fileDAO = (FileDAOHibernate)  
            (new ClassPathXmlApplicationContext(  
            "webApplicationContext.xml").  
            getBean("fileDAO"));

    fileDAO.insertFile(myFile);

    this.absolutePath = absolutePath;
    this.fileID = fileDAO.getFileIDfromFileName(  
            imageFileName).intValue();

    return SUCCESS;
}

Method private static void copy(File src, File dst) were also implemented to copy file to a directory. Besides, we also save the uploaded file with the file name as the current time and keep its extension: imageFileName = new Date().getTime() + getExtention(fileName);

5.4.5.8 Advertisement module

For a news website, there is often an advertisement module, we will create a quick advertisement section and save it on /inc/advertise.jsp. This module will be included on our site just by using: <%@ include file="/jsp/inc/advertise.jsp"%>. This module is easy to implement and include in our CMS system by creating a table holding these advertisement data. However making the site commercial is not in the scope of our project, hence we will keep the advertisement module simple as a jsp file only.
5.4.5.9 Image processing

We will write a Servlet to return an Image according to its *imageID* from database. The *doGet()* method in our Servlet is called by the servlet container to process a GET request. Moreover, *doGet()* returns to the requestor, an image read from a file. The body of *doGet()* is as follow:
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {

    Integer photoID = req.getParameter("photoID");
    ...
    String imgFilePath = fileDAO.getAbsolutePathFromID(photoID);
    String filename = imgFilePath;

    // Get the MIME type of the image
    String mimeType = sc.getMimeType(filename);
    if (mimeType == null) {
        sc.log("Could not get MIME type of "+filename);
        resp.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        return;
    }

    // Set content type
    resp.setContentType(mimeType);

    // Set content size
    java.io.File file = new java.io.File(filename);
    resp.setContentType((int)file.length());

    // Open the file and output streams
    FileInputStream in = new FileInputStream(file);
    OutputStream out = resp.getOutputStream();

    // Copy the contents of the file to the output stream
    byte[] buf = new byte[1024];
    int count = 0;
    while ((count = in.read(buf)) >= 0) {
        out.write(buf, 0, count);
    }
    in.close();
    out.close();
}

With this servlet, everytime we want to display an image, we could just give the servlet link with photoID to the source attribute (<img src=) of the <img> tag for something like <img src="/ImageViewer?photoID=34"/>. This will enable us to display image for each article according to its photoID property in our news website.

5.4.5.10 Web interface

Web interface design can be implemented in various way to suit particular needs for particular website. Since we have our CMS back-end done, we can have several templates for our
website interface. These front-end components just need to call on our service and get data accordingly.

For the purpose of our news website, we will design a simple layout base on a free HTML template. Many of those could be downloaded and use for free from several free template websites. After having a suitable HTML template for a news website, we need to re-edit all the tags and use Struts 2 tags instead of original HTML tags. We also need to break some html pages up into smaller pieces to be able to use Struts <include>. Several jsps will be created and placed under /jsp directory. Our directory structure is as follow:

```
|-- jsp
  |-- images
  |--- inc
  |   |--- advertise.jsp
  |   |--- footer.jsp
  |   |--- header.jsp
  |   |--- menu.jsp
  |--- style
  |   |--- style.css
  |-- index.html
  |-- index.jsp
  |-- viewArticle.jsp
  |--- viewCat.jsp
```

We picked a free web template website, after some re-edit and wording changes, it looks as follow:
A page in our website is broken down into several jsp files:

- **header.jsp**: Headers, title information such as website name, motto like “Simply beautiful” taken from 'Site' table in database.

- **menu.jsp**: Main menu items generated from Categories table in database.

- **advertise.jsp**: Our simple advertisement module, for simplicity we just use jsp tags for this purpose.

- **footer.jsp**: Footer information such as “Copyright (C) 2007 - My Company” also loading from 'Site' table in database.

Having these files, we could simply do an “include” in any page of our website where these information are needed, such as:

```jsp
<%@ include file="/jsp/inc/menu.jsp"%>
<%@ include file="/jsp/inc/advertise.jsp"%>
```
When we click on a menu, an action will be called, the action to perform this is ViewCatAction. ViewCatAction takes a parameter catID, looks into database and returns a list of all article in this category, the View component (for this case is catView.jsp) will then display title, photo, introduction text of each article from the list returns from ViewCatAction. The Struts tag \(<s:iterator value="articleList\>\) is the best way to perform this. Let’s look at this part inside catView.jsp:

```xml
<s:iterator value="articleList">
    <tr><td colspan="2" class="articleTitle">
        <a href="ViewArt.action?articleID=<s:property value="%{articleID}"/>">
            <b><s:property value="%{title}"/></b>
        </a>
    </td></tr>
    <tr><td>
        <img src="/ImageViewer?photoID=<s:property value="%{photoID}"/>" width="120" alt="Pic <s:property value="%{photoID}"/>" class="left" />
    </td><td>
        <p><s:property escape="false" value="%{introText}"/></p>
        <p class="more"><a href="ViewArt.action?articleID=<s:property value="%{articleID}"/>">more</a></p>
    </td>
</tr>
</s:iterator>
```

Now to see how do we get a list of type java.util.List (articleList) from database, we can look at ViewCatAction.java:

```java
package site.action;
import article.dao.impl.ArticleDAOHibernate;
import com.opensymphony.xwork2.ActionSupport;
import java.util.*;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ViewCatAction extends ActionSupport{
    private List articleList;
    private String opt;
    private int catID;

    public String execute()throws Exception{
        ArticleDAOHibernate articleDAO = (ArticleDAOHibernate)(new ClassPathXmlApplicationContext("webApplicationContext.xml").getBean("articleDAO");
        articleList = (List) articleDAO.
    }
```

Well, we can see from these codes that articleList is obtained from our articleDAO bean (obtained with the injection of sessionFactory by Spring) using method getArticlesListWithCatID(Integer catID). Let’s see what is inside this method from our Hibernate implementation of ArticleDAO:

```java
public Collection getArticlesListWithCatID(Integer catID)
    throws DataAccessException, SQLException {
    return getHibernateTemplate().find(
        "from Article a where a.catID = ?", catID);
}
```

Again Spring and Hibernate make everything simple just by using getHibernateTemplate() from Spring’s HibernateDaoSupport and a Hibernate query.

Finally, to have a look at the whole layout of our website, we added some real world news from BBC News website as some of our sample articles, the website lay out now completely looks as follow:
5.4.6 Conclusion

We have designed and implemented our CMS system using many cutting-edge technologies in the field of Java web application development. Our application covers a lot of techniques and architecture. The main concept to get along is how to better separate our application, user interface, persistence logic, and any other application layer you require. By doing this we allow new code components to be added, and make our application more maintainable in the future. The technologies covered in our application are Struts, Spring and Hibernate. The architecture based on these three frameworks here address specific problems well.

Further more, by using this type of architecture we can replace application layers with other technologies. For example, if for some reason we might not want to use Hibernate for persistence. Since we code to interfaces in our DAO objects it should be apparent to us how we might use another technology or framework as a substitute. We could also replace your
UI layer with a different framework than Struts. Switching UI layer implementations should not directly affect our business logic or our persistence layer. Replacing our persistence layer should not affect our UI logic or business service layer. Wiring a web application is not a trivial task but it can be made easier to deal with by decoupling our application layers and wiring it together with suitable frameworks.

In term of possible improvement, our application can be enhanced with the introduction of a cache tool. Since we query everytime for articles of a category, and these contents changes only when we update or add a new article, hence we could cache this data in memory so that we don’t have to query for it everytime. For this purpose, ‘memcached’ could be a good candidate. Memcached server and Java API are available as open-source tools under the BSD License.

Memcached is a high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load. An example with memcached in practice is that it enhanced the speed of LiveJournal.com, a site which was already doing 20 million+ dynamic page views per day for 1 million users with a bunch of webservers and a bunch of database servers. Memcached dropped the database load to almost nothing, yielding faster page load times for users, better resource utilization, and faster access to the databases on a memcache miss.

Final words, building our CMS system as a web application requires strong knowledge about AJAX, Struts, Hibernate, Spring and it was not an easy job. However, after accomplishing, it surely has great qualities and could be used on several real life web projects.
Chapter 6

Conclusions

Open source software is an interesting phenomenon that, in the recent years, has really begun
to show its incredible staying power. With the wide spread usage of Java technologies, Linux
operating system, Java application server, it is clear that open source technology can be
as good as or even better than commercial offerings. In the web application development
flow, with frameworks like Struts, Hibernate, Spring, open source software has created a new
development philosophy, brought in a great alternative to enterprise development. In terms
of our application, with a complete core and separate layer architecture, new platform and
features could be easily implemented and added. Perhaps a mobile platform with WML,
WAP interface to view articles could be added easily to our application if needed. In my
opinion, these frameworks will continue to have huge impact on the Java community in the
future.

When looking closer into these frameworks, it’s hard to predict if one of them will achieve
big success in the future. We could witness around the year 2005, there was a rumour heard
from many developers that Struts was already dead because of such technologies as Tapestry
or JSF, however this was not true. In fact, Struts have been so popular that there’s no doubt
it will be around for a long time. With the emergence of Struts 2, it promises to be a dynamic,
extensible framework for a complete application development from building, deploying and
maintaining. A statistic on Employer Search at *Monster.com* with number of resumés posted
in the last month of 2007 shows significant interests on Struts 2:
Figure 6.1: Employer search statistics on Monster.com - Resumes posted in a month 2007

From my own experience, the more I use Struts 2 I realize that a lot of development and thought has been given into solving real web developer needs. It’s interesting that sometimes when I think of some new feature I would like to have in a web development framework, I frequently find that it’s already there.

In the object persistence area, Hibernate have proved to be a powerful, high performance object-relational persistence and query service. Talking about the future of Hibernate, we may foresee that Hibernate Core will be developed independently from and faster than the EJB 3.0 or Java Persistence specifications. Hence, if someone is interested in a quickly evolving standard, I encourage you to use native Hibernate functionality, and to send feedback to the respective expert group. The current Hibernate job trend statistics shows significant increase of Hibernate experts need in recent years, and this trend will perhaps continue to happen in the future:
It’s likely that a better solution than ORM will exist some day. If that happens, we may have to rethink everything we know about SQL, persistence API standards, and application integration. But we can’t wait, and there is no sign that any of these issues will improve soon. For the time being, ORM is the best solution currently available, and it’s a time-saver for developers facing the object-relational mismatch every day.

In term of integration, Struts and Hibernate are popular and powerful frameworks that can be extended to work more effectively with one other by bridging the gap between the domain model and the MVC view. There are huge number of applications that successfully integrate Struts and Hibernate.

Within the scope of this project, the main tasks were to get acquainted with object-relational mapping, Hibernate, Struts framework, its technologies and introduce an architecture for using these tools in a web application. At this point, I would say all that the tasks were successfully fulfilled.
Appendix A

Directory Structure

The directory structure of our CMS web application is as follows:

CMS-WEB
|-- build.properties
|-- build.xml Ant build scripts for deployment
|-- logs
| |-- access.log
|-- src Directory of our source files
| |-- LICENSE.txt
| |-- NOTICE.txt
| |-- action All action classes come here
| | |-- CmsActionSupport.java
| | |-- EditLayoutAction.java
| | |-- EditSiteAction.java
| | |-- FileUploadAction-validation.xml
| | |-- FileUploadAction.java
| | |-- MenuAction.java
| | |-- article
| | | |-- ArticleListAction.java
| | | |-- DeleteArticleAction.java
| | | |-- EditArticleAction.java
| | | |-- GoArticleAction.java
| | | |-- NewArticleAction.java
| | |-- cat
| | | |-- CatListAction.java
| | | |-- DeleteCatAction.java
| | | |-- EditCatAction.java
| | | |-- GoCatAction.java
| | | |-- NewCatAction.java
| | |-- user
| | | |-- DeleteUserAction.java
| | | |-- EditUserAction.java
| | | |-- GoUserAction.java
| | | |-- LoginAction.java
APPENDIX A. DIRECTORY STRUCTURE

|-- LogoutAction.java
|-- NewUserAction.java
|-- UserListAction.java
|-- article
| | |-- business
| | | |-- Article.hbm.xml
| | | |-- Article.java
| | | |-- dao
| | | |-- ArticleDAO.java
| | | |-- impl
| | | | |-- ArticleDAOHibernate.java
| | |-- cat
| | | |-- business
| | | | |-- Cat.hbm.xml
| | | | |-- Cat.java
| | | | |-- dao
| | | | |-- CatDAO.java
| | | | |-- impl
| | | | | |-- CatDAOHibernate.java
| | | |-- file
| | | | |-- business
| | | | | |-- File.hbm.xml
| | | | | |-- File.java
| | | | | |-- dao
| | | | | |-- FileDAO.java
| | | | | |-- impl
| | | | | | |-- FileDAOHibernate.java
| | |-- jdbc.properties
| | |-- servlet
| | | |-- EditArticle.java
| | | |-- EditCat.java
| | | |-- EditSite.java
| | | |-- EditSubCat.java
| | | |-- EditUser.java
| | | |-- ImageViewer.java
| | |-- struts-main.xml
| | |-- struts.xml
|-- site
| | |-- DataConnection.java
| | |-- DataConnectionImpl.java
| | |-- SiteUtil.java
| | |-- action
| | | |-- ViewArticleAction.java
| | | |-- ViewCatAction.java
| | |-- struts-main.xml
|-- struts.xml
|-- user
| | |-- action
| | | |-- UserAction.java
--- business
| |-- User.hbm.xml
| |-- User.java
| |-- dao
| |-- UserDAO.java
| |-- UserDAOHibernate.java
| |-- UserService.java
| |-- UserServiceImpl.java
| |-- webApplicationContext.xml

--- WEB-INF
| |-- classes
| |-- webApplicationContext.xml

--- config
| |-- jdbc.properties
| |-- userApplicationContext.xml
| |-- webApplicationContext.xml

--- lib
| |-- antlr-2.7.6rc1.jar
| |-- cglib-2.1.3.jar
| |-- commons-collections-3.1.jar
| |-- commons-digester-1.6.jar
| |-- commons-fileupload-1.1.1.jar
| |-- commons-io-1.1.jar
| |-- commons-logging-1.0.4.jar
| |-- dom4j-1.6.1.jar
| |-- ehcache-1.1.jar
| |-- freemarker-2.3.8.jar
| |-- hibernate3.jar
| |-- log4j-1.2.15.jar
| |-- mysql-connector-java-3.1.13-bin.jar
| |-- ognl-2.6.11.jar
| |-- readme
| |-- rsmessaging.jar
| |-- serializer.jar
| |-- servlet-api.jar
| |-- spring.jar
| |-- struts2-core-2.0.11.jar
| |-- xalan.jar
| |-- xwork-2.0.4.jar
| |-- tld
| |-- tmp
| |-- web.xml

--- cms
| | CMS Administration interface jsps
APPENDIX A. DIRECTORY STRUCTURE

|-- ajax
|  |-- deleteArticleAjax.jsp
|  |-- deleteCatAjax.jsp
|  |-- deleteUserAjax.jsp
|  |-- editArticleAjax.jsp
|  |-- editCatAjax.jsp
|  |-- editLayoutAjax.jsp
|  |-- editUserAjax.jsp
|  |-- newArticleAjax.jsp
|  |-- newCatAjax.jsp
|  |-- newUserAjax.jsp
|-- articleManager.jsp
|-- catManager.jsp
|-- checkLogin.jsp
|-- css
| |-- cms.css
| |-- editArticle.jsp
| |-- editCat.jsp
| |-- editCatByID.jsp
| |-- editUser.jsp
| |-- editUserByID.jsp
|-- img
|  |-- images
|  |  |-- backcolor.gif
|  |  |-- view_text.gif
|  |  |-- view_text_on.gif
|  |  |-- left_close.gif
|  |  |-- logo.gif
|  |-- popups
|  |  |-- about.html
|  |  |-- about_license.html
|  |  |-- create_table.html
|  |  |-- insert_hyperlink.html
|  |  |-- insert_image.html
|  |  |-- license.html
|  |  |-- preview.html
|  |  |-- select_color.html
|  |-- styles
|  |-- wysiwyg.css
|  |-- index.html
|-- js
|  |-- wysiwyg-color.js
|  |-- wysiwyg-popup.js
|  |-- wysiwyg-settings.js
|  |-- wysiwyg.js
|-- layoutManager.jsp
|-- login.jsp
Totally 98 directories, 556 files. Contents of several images directories were omitted.
Appendix B

Application Deployment

Our web application was developed and tested on Caucho Resin application server version 3.1 running with MySQL database server version 5.0.38 under Ubuntu Linux. The build tool used was Apache Ant version 1.6.5. The development environment used Java version 1.6.0. Following are several steps needed to deploy our application (these are instructions under Linux operating system, they should be similar on other platform):

2. Download, install and setup JDK 1.6 or later.
3. Download, install and setup Ant.
4. Download, install and setup MySQL database server.
5. Create database and tables needed. This step includes also the configuration editor of jdbc.properties file under /conf directory.
6. Modify Resin configuration for our web application (often in resin.conf). Set up environment variables and directory needed. Also need to configure log4j.properties if needed. This could take some time depends on the experience of user with Resin.
7. Modify build.xml and build.properties scripts with appropriate values of web server environment and libraries.
8. Deploy application using Ant.
9. Run Resin with the modified resin.conf for our application (with root privilege).
10. Browse to our CMS system:

Remember if you config a <host id="cms.me" > for our web application, you also need to point cms.me to 127.0.0.1 in host definition of Linux (/etc/hosts).
Bibliography


[20] Sun Microsystems Inc., Java 2 platform, Enterprise Edition (J2EE) Overview,


