Model Base generation of a 3 tier application

Student	Yassin Asri
Mentor (UvA)	Mark van den Brand
Mentor (Mattic)	Jeanot Bijpost
Company	Mattic Software, Lectoraat Software Kwaliteit Hogeshool van Amsterdam
Date	June-2004
Master Thesis Software Engineering	Universiteit van Amsterdam UvA

Foreword

'Time flies when you're having fun', is a remarkable proverb. Looking back on this past year, I can certainly state that time did play some tricks on me. However boring software engineering may seem to outsiders, I truly had a lot of fun in dealing with the challenges we were confronted with. And, as the saying goes; time was flying fast indeed.

I would sincerely like to thank all of my teachers for doing their best. I would also like to compliment Mr Bijpost, for he was an excellent mentor during my internship period.

Last but not least, I want to thank my family and my friends for their unlimited support.

Yassin Asri,
June 2004

Summary

Creating a data model before starting the actual project offers many benefits. The data model serves as view of the system. There are numerous tools for creating a data model (Fully Communication Oriented-Information Modelling FCO-IM, Unified Modelling Language UML). The research in which I was engaged, deals with the ability of generating a complete system, with a 3 tier architecture conform J2EE by taking the data model as input.

Before building the generator, I had to do research about code generation concepts. I also had to look at different already existing generators in order to find out what these are capable of. Last but not least I had to do a lot of experimenting with J2EE.

The vision of J2EE to database is very simple. J2EE tries to get on top of a database to reduce the effects of the impedance mismatch (the difference between OO and RDBMS). Every table is represented by an Entity Bean and each instance of the bean represents a row in that table. Relationships and primary keys are also taking in consideration.

The generator should consider many aspects that are important in software construction. The generator should be easy to use and the generated files should be provided with different features such as comments and exception handling to satisfy the needs of different users. The generator generates the user interface layer (HTML/JSP and Swing), the business layer (EJB) and the database.

Using a data model to generate the three tiers will bring us many advantages. It can save a lot of time and it can increase productivity and quality. The generated files in different layers can work together and they are combinable in one architecture.

Table of contents

9 APPENDIX 1.....26

1 Goal and Context

1.1 Introduction

Object Oriented, often cryptically referred to as 'OO', and Relational Database Management Systems (RDBMS) concepts are nowadays essential to create reliable and efficient information systems. Modern systems require a combination of both relational database and object expertise. Therefore it is very important to connect these two technologies with each other to assure a good integration of the system to be built.

There are numerous ways to make a connection between OO and RDBMS. Object to Relational Mapping Concepts [1] provides different ways how to map a relational database into OO-classes. The Java 2 Enterprise Edition (J2EE) approach of the relational database is represented with Entity Beans of Enterprise Java Beans (EJB). Each entity bean represents a table in the database and each instance of the bean corresponds to a row in that table.

In today's world, software architecture is gaining more and more attention. Systems should not be built out of the blue, but according to a pre-defined architecture. An architectural view, or Data Model, is often used to represent a system. A suitable architecture combines OO and RDMS. The layered architecture consists of multiple tiers exchanging data mutually where OO and RDMS both play their unique role.

Using a code generator to generate classes brings many advantages in a software development project. This can increase reliability by invoking highly tested code and increase productivity by reducing valuable development time.

1.2 Background

Mattic Software BV is a small company specialised in designing, developing and implementing information systems. Mattic also supports the development of two case-tools: Infagon, better known as FCO-IM, for the development of databases and Cathedron for the development of information systems.

Mattic wants to upgrade Cathedron with the ability to generate a 3 tier application conform J2EE. By starting with the conceptual model of Infagon, a second goal can be achieved: getting Infagon and Cathedron closer to each other. This survey may become a steppingstone for further integrating these two tools. The research will produce a tool that the students of the University of Amsterdam (UvA) and the Hogeschool van Amsterdam (HvA) can experiment with.

1.3 The assignment

It is usually common to generate a database schema according to an information model (created in UML/FCO-IM/EAR). We also know that it is possible to generate an Objected Oriented Model based on that model.

The assignment is to connect these two concepts by developing an algorithm that is capable of:

- Generating the database schema.
- Approaching the database by implementing middleware conform J2EE architecture.
- Generating a simple front end (in both HTML/JavaScript and Java-Swing) to access the database via the middleware.

The generator will be provided with a conceptual model as input and should generate an application with 3-tier architecture working together seamlessly (Interoperability).

The end result is a description of the used methods, techniques, algorithms and a prototype that is capable of doing what is described earlier with an example of a given database.

The essential part of the assignment is that we can convince that the relational model and the OO model are close to each other and that they are easy to combine in one architecture.
We can also prove that it is possible to generate an application from a Datamodel.

1.4 Motivation

Currently, J2EE technology is one of the most used technologies for building information systems. J2EE has approved its efficiency to build reliable systems globally and there is a widespread need for people that can build J2EE based systems.

In my view, this assignment deals with 3 points that are fundamental in better understanding J2EE.

1. Code generation principles:
   In today's world we have to seek the best and most efficient ways to write and to use source code. Writing a generator has to be seen from many perspectives. It is not only a program that generates line of codes according to some input, but it has to be something that satisfies different kind of users. The principles of the generation here are very important.

2. J2EE technology:
   Before contacting Mattic, I wanted to do something with J2EE due to its increasing popularity. My knowledge of Java is quite good and using it with the J2EE framework will certainly increase my skill and my techniques.

3. Data modeling:
   Before generating the source code, we have to understand the model. This includes relationships, columns, tables etc. This will increase my vision and knowledge about modeling and databases.

1.5 Action plan

The assignment was planed for a period of 3 months, starting in April and ending at the end of June. This period was divided as follow:

Period 1 or Investigation period: This took about 5 weeks and it was used as a period for the necessary literature study. My knowledge of J2EE at start beginning was very limited and constrained to some general concepts. I used the J2EE Sun tutorial to learn more and to get familiar with it and to experiment with the J2EE 1.4 SDK (the server provided by Sun Microsystems for developing J2EE applications). I also searched the internet for articles or examples concerning the view and the vision of J2EE on relational databases and data models.

Beside this, Code generation in action [2] was used as main literature to learn about code generation in general. I also looked at some different example of EJB code generator available on the internet.

To enrich my knowledge of data modeling, I used De UML toolkit [3] book as an important source.

Period 2 or Building period: After learning more about data modeling, J2EE (EJB) and code generation, the next step was to build the generator. This period was planed for 4 weeks.

Period 3 or final period: This period was divided between writing this document and trying to add more features to the generator such as customizing the generated code and producing a better user interface. This period took about 4 weeks.

2 J2EE Concepts

2.1 Introduction

Java 2 Enterprise Edition (J2EE) is a platform for developing Enterprise applications and portals. It is one of the most commonly used technologies to build n-tiers applications. J2EE uses Java, an objected oriented programming language. J2EE provides an architectural framework where components can be placed and communicate according to protocols. J2EE describes the architecture for designing, developing and deploying component-based, enterprise-wide applications.

The J2EE application consists of 3 or 4 tiers:

- Client-layer: the components are run on the client machine.
- Web-layer: the components are run on the J2EE server
- Business-layer: the components are run on the J2EE server
- Database-layer: run on a database server.

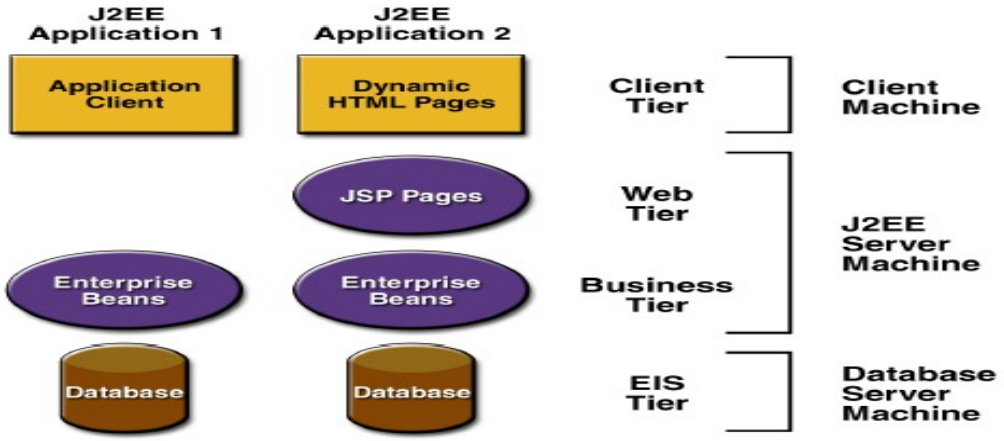

Figure 1: the J2EE architecture, Source: the J2EE tutorial [1]

J2EE is made of 13 different technologies. The main ones are:

- Enterprise JavaBeans (EJB) architecture: an EJB is a server side component (located in the business layer). There are 3 kinds of beans: session, message driven and entity beans. The architecture defines an API that allows developers to create, deploy and manage the components.
- Java Server Pages (JSP): JSP is based on Java language. JSP provides a method for creating dynamic web sites. JSP is comparable with technologies as ASP, PHP and Perl.
- Java Servlets: Java Servlets are Java classes where HTML is embedded. Java Servlets are Comparable to a CGI.

The Entity Beans are very important for this assignment. An Entity Bean represents a business entity that exists in the storage. Each entity bean has an underlying table in a

database and each instance of the bean represents a row in that table. The Entity Bean is accessed by the local and the remote interfaces. The remote interface contains the business methods of the application whereas the home interface contains the life cycle methods such as remove and create.

Portability: Applications written in J2EE are in theory portable across any J2EE server. J2EE uses Java. J2EE applications will run on any hardware and operating system with a J2EE compliant application server.

"Vendors justify the addition of APIs to their implementation on the grounds it enables them to add value and differentiate themselves against competitors" [5].
To retain portability the application must not be specified to any vendor extensions. Switching from a server to another does not always happen smoothly. Sometimes some configuration work must be done to ensure that the program works.

Buildability: Some services such as transactions do not have to be built. They are ready to use. The developers can concentrate more on building the business logics of the system.

Scalability: The scalability is the measure of the capability of an application. The infrastructure and layer abstraction reduces performance but increases scalability. J2EE applications are built with the proper design patterns. The multi-tiered architecture makes it easy to expand the numbers of servers.

Minimizing traffic between layers is the objective when creating a scalable solution. It is often the reason for a scalable system to have a bad performance. The standard J2EE pattern is not enough to guarantee a scalable and a perform application in the same time. The use of other design patterns or techniques such as database caching is required.

Usability & Continuity: J2EE provides Java technology, Java Server Pages and Java Servlets to suit different users. J2EE is a standard that is supported by different software providers. There is a wide range of people and organizations that adopt the J2EE technology.

J2EE finds Microsoft's .NET as its main competitor. There is a big debate about which one is better and which one to use. Both of them target the market for enterprise applications and web services. J2EE is a Java centric and platform independent and is basically a series of standards, while .NET is Windows centric and uses Microsoft products.

J2EE provide solutions from multiple vendors with different tools and product. This adds a varied functionality but it can also be a drawback. Different vendors means that the applications may not run with every server which introduces a problem of portability. Microsoft's .NET provides a solution of tools and services from Microsoft.

Maintainability: The n-tier architecture is easy to maintain because the layers are separated and work independent of each other.

Performance: The performance of an application depends on many factors: hardware configuration, server configuration and the code. Within the J2EE Server the components can be distributed on different machines. This can increase the performance of the system.

J2EE applications are Java programs that utilize a database. Good coding techniques such as the use of design patterns and the efficient use of database are necessarily to ensure a good performance.

Coding and designing J2EE applications has its benefits, but it also has its downsides. One of the more inconvenient aspects is that we have to do a lot of configuration work to get an application working and it is more complicated than other type of applications. J2EE applications are scalable, but we can have the same results with other technologies such as COM+ and there are also some technologies for the transaction management. By using the Java Data Objects the object relational persistence can also be guaranteed.

2.2 Impedance mismatch

The Impedance mismatch is the difference between relational and Object technologies [6]. OO and RDBMS are two different paradigms. Where OO is based on software engineering principles, RDBMS on the other hand is based on mathematical principles. These two technologies are different which means that they will not work together seamlessly. With OO we traverse object via their relationships whereas with the relational we join the data row of a table. J2EE tries to solve this problem by mapping the relational schema [1]. A table will represent a class; a column and a foreign key will become an attribute.

Concept	Implementation in J2EE	Observations
Relationship	EJBs are Java classes that support relationships. An EJB can be related to another EJB. It supports different kinds of relationships: one to one, many to one and many to many.	Java does not support pointers. A foreign key is represented by the association of the target class. The EJB spec does not specify how to deal with relationships. The developer has to develop his own Java classes to handle them.
Persistency	Persistency means that the data exists in a storage mechanism even after a system is shut down. Entity Beans are persistence because their state is saved in a database.	
Shared Access	Entity Bean can be shared by multiple clients. The EJB container is responsible for transactions in case that a client wants to change data.	If the number of the clients increases, a problem of performance may arise.
Uniqueness	Like in a table with a primary key, an Entity Bean has an internal unique object identifier called also primary key. This object permits to locate a particular entity bean.	The structure of the classes may become complicated as the number of classes with composite primary key increase.
Concurrency	J2EE applications support many clients sharing a number of db connections.	
Data Representation	J2EE is based on Java. The data is represented by objects, methods, inheritance.	J2EE uses only Java. A good knowledge of Java is necessary.

3 The generator

3.1 Research on existing generators

EJBGen: http://www.beust.com/cedric/ejbgen/index.html
EJBGen is a command line EJB 2.0 generator. The idea behind it is to concentrate on building the bean class and annotate it with Javadoc tags (@tags). The generator will then generate the rest of the classes (the home, remote, local and localhome interfaces).

EJBGen provides a number of features which are very interesting:

- CMP 2.0 Entity beans, including relationships (one-one, one-many, many-many, unidirectional and bidirectional)
- Stateful Session Beans
- Stateless Session Beans
- Message Driven Beans
- Local interfaces
- Value objects
- Compound Primary Keys
- Home methods
- Isolation levels
- Support for ant
- Inheritance of tags and attributes

EJBGen is not a model based generator. It does not provide for an automated process to generate the beans since we have first to create the bean class itself. Furthermore, we have to specify which methods and relationships we want to implement in the Javadoc tags. If we want to generate a large number of beans, this would result in a lot of handwork.

Percolator: http://www.backsource.org/source/java/percolator/
Percolator is a tool that lets you reverse engineer a database and auto generate EJB classes representing that database. In order to use Percolator we will have to create an XML description for each "component" in our information domain. This means an XML file for every table. If there are more tables, several XML files have to be written. Unfortunately Percolator does not support relationships between tables and does not generate the user interface and the database layers.

XDoclet: http://xdoclet.sourceforge.net/xdoclet/index.html
XDoclet is one of the most popular tools for generating EJBs. It is an open source code generation engine and it is based on Javadocs tags. There is a wide support for XDoclet (2 books: XDoclet in action and Java Open Source Programming). There is also a good documentation available on the site.

Although XDoclet provides many interesting features and has a lot of support, it is not a model based generator. You can not use a model as input, instead we have to specify every bean in a descriptor file.

3.2 Observations on existing generators

The generators that have been used in my research do not use a data model or something similar to generate the source code, instead they require to write a file to describe separate beans themselves. This results in a lot of hand work. A data model is an ideal view and description of the system, so why not use it as input for the generator?

The second conclusion is that these generators do only generate the EJBs, while it is possible to generate basic user interfaces in both HTML/JSP and Swing. Generating SQL statements for creating the database is also possible from a data model.

3.3 Code generation concepts

Code generation has a lot of advantages in a software engineering project. Choosing, using or developing a good generator can have many benefits for engineers and managers. By reducing the development time we can give more attention to design time. The quality of the code is also high because it has been tested and approved. It also encourages the programmers to work within the architecture.

When developing a generator, many points should be taken in consideration. Not only attention should be paid to the construction of the generator, but also to the generated code. Writing code is a kind of research on its own. We have to seek the best and most efficient way to present it. Software Construction paradigms in these matter are essential. The generator can be used by a variety of developers with different backgrounds. Providing the generated code with standard naming and comments should have a high priority to guarantee a better understanding and use of it.

Before beginning with the development of the generator, it is important that we define the problem we want to solve. For this assignment, the generator will be responsible for generating a 3-tier application conform J2EE (Description with more details will follow later). The problem can be defined by gathering information about what the generator needs to do. The next step is to research which are the best ways how to implement it by making a structure of it, choosing the programming language, using templates or hard codes, define input schemas. The last step would be deploying and documenting it.

Code generation in action [2] page 25-26 is about top ten code-generation rules. There are some very useful advises on how to write a generator. I have followed most of these tips for the generator.

3.4 Goal of the generator

The generator will take as input an XML schema file describing a data model. De data model is usually generated by a data modeling tool such as FCO-IM. The generator should then generate a 3 tier application. This includes a user interface in HTML and Swing components, a business layer consisting of Bean classes, remote, home interfaces and a database layer consisting of SQL statements for creating a database.

As mentioned in the 2nd tip of the top 10 code generation rules [2], it is a must that we first understand the framework before generating the code. The first thing to do is to handwrite a significantly board spectrum of code within the framework and then use that code as the basis of the templates for the generator. This was the case; I began with an example of a movieshop data model. I wrote all the necessary beans and their associated interfaces. The movieshop example was taken as the basis of my generated code.

3.5 Choice of the language and techniques

Choosing the right language for the generator depends on many factors. We have to exploit all the possibilities we have. The generator reads a number of text file(s) and generates a number of text files as well. It is important to observe the format of every file to make some pre-decisions such as using regular expression or not.

Jack Herrington [2] suggests using a different language than the language where the generated code is written. In his case example he used Ruby, a powerful programming language that supports many features such as regular expressions and has a good portable file and directory I/O constructs.

We do prefer a single language for both the generator and the generated code (Java): Combining the generated source code with different code brings some inconveniences. A person needs to learn a new language first if he wants to maintain the generator. This can not always be an easy step and it takes some time.

Some modern text editors can highlight the key words (a string between two quotes is displayed with a different colour). This argument is against who things that it is easy to write a generator in a different language because you can trace the code easily.

Java also provides the following which are very important for the use and the maintenance of the generator:

- There are a large number of Java developers worldwide. Understanding and maintaining the generator will not be a problem.
- With Java, the code will be portable to different operating systems.
- The input schema is an XML file. Java supports XML and there are a wide number of components on the internet to read and to parse an XML file. There is no need to use a complicated regular expression to read the input file.

Templates or regular expressions: We do not prefer to use regular expressions. It is true that they are very powerful, but they are also difficult to maintain. Any change in the structure will result to adequate the expression itself.

By looking at the base code of the Movieshop example we can conclude that it follows a standard format. Using templates is thus ideal to generate the code. The main advantage of templates is that code can be easily added or changed to the file without changing or compiling the source code.

Using templates can have sometimes inconvenient effects. The structure can become more complicated as the number of sub templates arises. This can be solved by making a good and a logic structure to call and to use the different sub templates.

Our generator got the name of JCat (to pronounce as Jee-Cat). See Appendix 1 for a print screen of it and the following figures represent two different views of the generator.

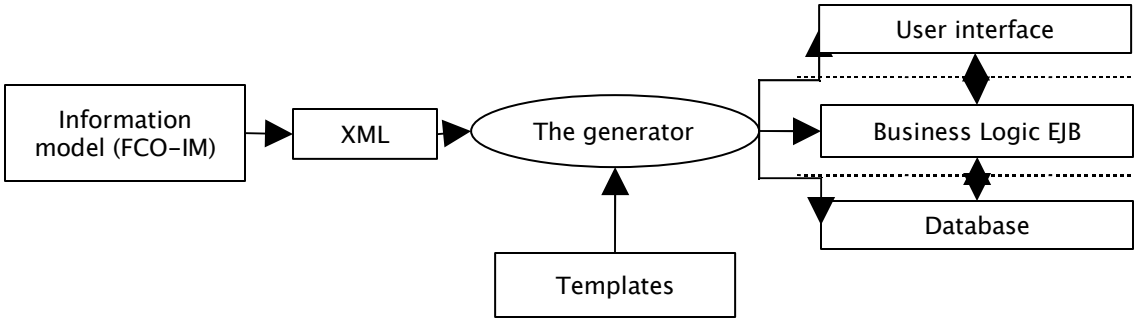

Figure 2: The structure of JCat

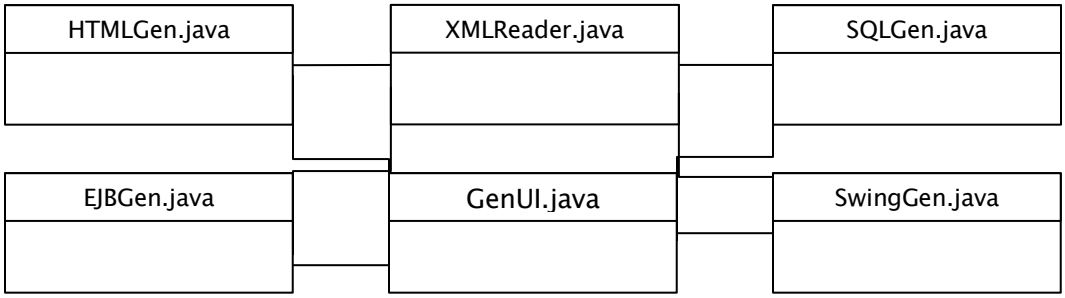

Figure 3: The JCat class diagram

3.6 The generated code

The generated code has to be very easy to understand and to maintain. Code Complete [4] was for me a good reference to deal with different aspects of software construction. The generated java files follow a standard naming convention, they are provided with comments, javaDocs tag and exception handling.

JCat enables users to generate files at choice. They can choose to generate separate files by using the checkboxes provided by the generator.

3.7 Custom code issues

JCat will generate the basic classes for the three tiers. It is obvious that people want to add their own code to the generated one. We took consideration with this fact. JCat will generate basic classes and child classes derived from them. The user can add their code in the body of the derived ones. If the data model is changed and the user wants to re-generate the classes, JCat will only re-generate the base classes; the derived classes will stay intact.

4 The application

This section describes the results and the choices made to implement the different generated code.

4.1 Database layer

The XML schema input represents the data model. Generating the SQL statements for the creating of the database layer is only a matter of reading the input file, parse it, traverse it and then write the output as .sql file. Every table tag becomes a table, a column becomes a column in that table and so on. The foreign keys are also taken in consideration. The data model is a good description of the database. The generated SQL code is a mirror of that data model. No additional things are required.

4.2 Middleware (EJB)

The middleware represented by EJBs is responsible for getting the data from the user interface layer and send it to the database. The business methods of the application are implemented here. There is no need to do this in the client (one of the advantages of the 3 tier architecture). The main charge of this layer is how to deal with the persistency. EJBs of J2EE have a simple vision towards the databases. They try to lean on it as much as possible to stay close to it and to reduce the effects of the impedance mismatch. We tried to follow this vision in our generator.

Tables and columns: Every table tag in the XML schema descriptor will become a single class (a base class). The name of the class corresponds to the name of that table. A column in the table will become an attribute in the corresponding class. The type of the attribute depends on the type of the column. For example a Varchar will become a String, Integer an Int and so on.

Furthermore, the class will be provided with a standard number of business methods plus some other methods depending on the relationship with other tables. The standard classes are: the getters, the setters, ejbCreate, ejbFindByPrimaryKey, ejbRemove, ejbStore, setEntityContext, unsetEntityContext, ejbActivate, ejbPassivate, ejbLoad and ejbPostCreate.

Primary keys:like in a table, an EJB class require a primary key. The primary key is an attribute of the class and corresponds to the primary key in the table. In case of a composite primary key (a primary key composed by more than one column), an apart class will be generated to represent that primary key.

Relationships: A relationship, an important concept of a RDMS and represented by a join, should be also mapped to the classes.

One to one relationship: This is the simplest form of a relationship; each row in a table is related to a single row in another table. The class corresponding to the child table should implement a method findByAttribute where Attribute represents the join column to the parent table.

One to many: This kind of relationship is very common in databases and it occurs when a primary key in a parent table matches multiple foreign keys in the child table. The “parent” class will then associate the “child” class by declaring the home interface and a

list (ArrayList) of it. The “child” class is provided with a method findByForeignKey where Foreignkey represent the primary key in the “parent” table.

Many to many: In a many-to-many relationship, each entity can be related to multiple occurrences of the other entity. For example, a college course has many students and each student may take several courses. In a database, this relationship is represented by a cross-reference table containing the foreign keys [1] page 957. Looking at the example provided for this kind of relationship, I encounter a contradiction: they implement a Session Bean (represents a single client inside the J2EE Server, a session bean is not persistence and not shared) for the cross reference table in place of an Entity Bean. The main task of a session bean is to read data form a database, while in the provided example they used it also to insert the data. The primary key is also not implemented.

Our vision to this is simply. We consider the cross reference table as an Entity Bean class. The many to many relationship will be interpreted as two times one to many relationship.

4.3 User Interface layer

The user interface consists of two types: HTML/JavaScript/JSP and Java Swing.

A brief description of generated Swing classes: The generated application is a Multiple Document Interface (MDI). It consists of a parent screen with a menu for accessing the child classes. For example, if the data model has three tables (e.g. X, Y, and Z) the parent application will have a menu with three items X, Y and Z. Each item has two sub items: “New” which calls a screen for inserting a new record and “View” for viewing all the records in the database. For generating the “New” screens, the application looks if a class is related to other classes. If is this the case, a tab control will be added to the screen to “import” the related records.

A brief description of the HTML/JavaScript/JSP pages: The principle is the same as for the Swing screens. A menu will be generated (using JavaScript) to access different screens. For each table in the data model there will be an insert page called newX.html where X is the name of the table. The HTML page forwards the insert form to the addX.jsp page which is responsible for obtaining the parameters from the HTML page and calling the equivalent bean of the business layer. The “view” screen is called viewX.jsp and it is used to view all the records of a table X.

4.4 Interoperability of the application

The three tiers should work together to ensure the working of the entire application. The user interface communicates with the business layer through the home and remote interfaces of the beans. To inserting a record for example, the create method of the home interface is called from the user interface with the insert parameters. The business layer communicates on its turn with the database using the Java Database Connectivity (JDBC). Figure 4 shows how the layers communicate.

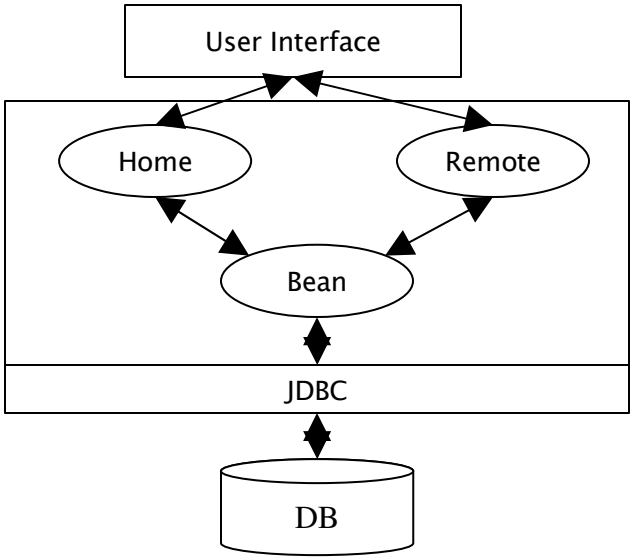

Figure 4: Communication between the layers

5 Conclusions

Having taking all aspects of this assignment in consideration, we can now begin to start drawing conclusions.

It is not logical to start with a project without having a data model. The data model can not only serve as a view for the architecture, but also serve as a base for the programming. Generating classes based on it would therefore be ideal. There is no need for write specific files to describe the classes.

Generating the middleware is not the only possibility. Generating the user interface and the database can also be combined in one generator. Again, the data model can serve as a basis to generate those two layers.

Object Oriented has a better data structures than RDBMS. Writing an own mapping constructs with OO to lead the database is time and cost effective, and it differs by programmer to another (Every programmer can have a different implementation of the mapping). A good system has to be a combination of OO and RDBMS. The frame provided by J2EE is good for creating database applications. Accessing the bean from the user interface occurs via the remote and home interface of it. The bean contains the business logic of the application; the user interface has only to call those methods. Accessing the database occurs via JDBC.

It took me few weeks to write the necessary files for the user interface, business logic and database for the Movieshop example. Generating the same files takes only few seconds. Using a code generator based on a data model as a tool can save a lot of time and can increase the quality of the system. A user of the tool will see on forehand which classes and files will be generated only by looking at the data model.

The generator that has been invoked in my research is a model based. All of the advantages of the Model-based Application Development (MAD) [7] are applicable here.

6 What's next?

JCat was programmed over a period of 3 months and it has not been tested very well yet. A test has to take place to remove possible bugs. The generated files are provided with basic comments, adding more comments would off course aid a better understand of the used code.

The user interface can be also improved or extended by implementing more controls such as radio buttons, check boxes and combo boxes. Adding more pages to deal with the data such as search screens will have a positive effect on the generator as a whole.

I expect this generator will proof to be a useful tool for both students and professionals that want to familiarize themselves with EJB. Undoubtedly, this generator will be an ideal tool for further experimenting with this powerful piece of technology.

7 Evaluation

This research proved to be an excellent way for to learn more about J2EE, data modeling and Code generation concepts. It also was a great occasion to apply my theoretical knowledge of software architecture and software construction acquired during my study Master Software Engineering.

Although the planning of this research didn't always go as I had expected, for example I had to rewrite a few classes for the generator to make it easier to understand, and taking in consideration the occasional bugs that took me days to resolve, I kept a sound believe in this project.

It was something new for me to work with J2EE and to build a generator, but the most important thing for me was how to deal with a research project. This research pushed me to think further than just programming. I really enjoyed it and I think I learnt quite a lot from it. Therefore, I give a 9 for the quality of the research.

I am happy with the results of this research. There is now a model based tool for generating a 3 tier application. I did my best and I had to work very hard on developing the generator including some sleepless nights. For the results I give a 8,5.

I tired to treat every aspect I dealt with during my internship in this report. Some points could be more specific such as chapter 4. The application could be described with more details than I wrote. I tried to keep it simple and brief. The reader of this report would have a clear idea about my research and my results. For this report I give an 8.

8 Literature

[1] The J2EE 1.4 Tutorial, Eric Armstrong et al, March 17 2004, http://java.sun.com/j2ee/1.4/docs/tutorial/doc/
[2] Code Generation in Action, Jack Herrington, ISBN 1930110979
[3] De UML Toolkit, Hans Eric-Eriksson & Magnus Penker, ISBN 9039510156
[4] Code Complete, Steve McConnell, ISBN 1556154844
[5] J2EE Design and Development, Rod Johnson, chapter1: http://www.wrox.com/books/sample-chapters/samplechapter_0764543857.pdf
[6] Understanding Impact of J2EE Applications on Relational Databases, Dennis Leung, http://www.nocoug.org/download/2003-02/j2ee_rdb.pdf
[7] Extremely rapid development of high quality database systems, Jeant Bijpost and Marco de Groot, http://www.mattic.nl

9 Appendix 1

Print screen of the user interface of JCat.