WAVE

A Java based Warehouse Visualisation Environment

Laurent Somers

Supervisor: Mark O'Brien

Faculty of Information Technology,

University of Akureyri

Submitted April 2006, in partial fulfilment of

the conditions of the award of the degree B.Sc.

I hereby declare that this dissertation is all
my own work, except as indicated in the text:

Signature _____

Date 18/04/2006

Abstract

This report details the work performed as a final year project at the University of Akureyri in the spring of 2005. The project involves extending a previous warehouse management application to fully support an underlying database as well as provide the necessary foundation for a 3D view as well as a 2D view. The project leverages Open Source components to implement a clean architecture, using Hibernate 3.0 for persistence and data abstraction. Other technologies are made use of such as XML.

The application basically performs functions which temporal databases do natively, providing storage information about products as well as a complete history of storage locations.

Table of Contents

Motivation for the work

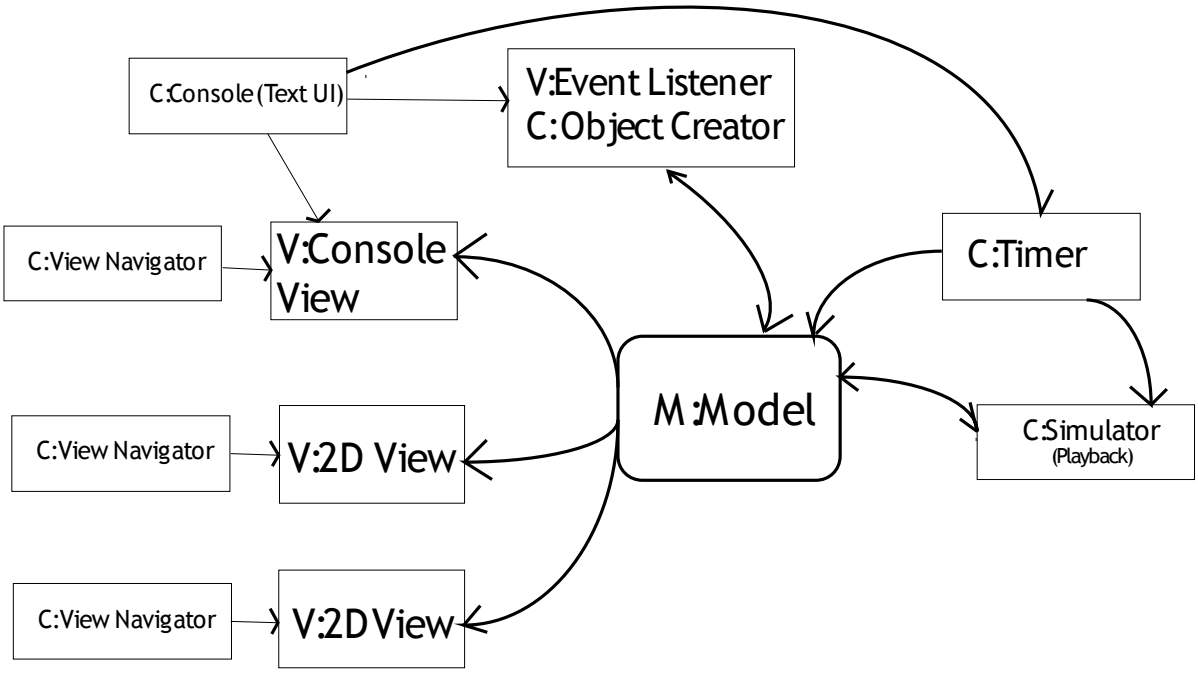
WAVE 2 was a project aimed at creating a RDBMS-enabled, extensible architecture for the visualisation of product traffic within a warehouse. This software would provide, a simple, efficient framework.

The primary goals were:

- To implement the concepts of lightweight framework
- To create an extensible architecture
- To implement an RDBMS-independent solution
- To maximise the simplicity of the architecture

Description of the work

WAVE 2 is a framework which uses the Model-View-Controller pattern as its main theme.



Related work

One producer of visualisation software was found on the Internet, Aldata Solution produces G.O.L.D, a bundle of modules[1] revolving around the traceability of goods moving through a logistics network.[2] In common with WAVE 2 is the Java-based client interface. Details about the software's internal design are unknown.

Aldata identifies some key benefits of its G.O.L.D Stock system, such as[3]:

- Productivity gains
- Optimisation of storage facilities
- Improved organisation and optimisation of order fulfilment
- Increased reliability from reception through dispatch (up to customer reception)
- Just-in-Time flows for outbound operations

- Better traceability and visibility throughout the logistics network of all stock movements
- Better quality information for optimum control, management and decision support
- Superior customer service levels

Design

Building tools

For the software development, a number of tools were used. These were:

- Borland JBuilder
- CVSNT/Subversion
- JUnit

JBuilder

JBuilder was chosen primarily for historical and familiarity reasons; JBuilder had been used for the previous version. At that time, JBuilder had been chosen over Eclipse because of its maturity, its CVS and JUnit support and its visual tool for building user interfaces.

CVSNT

CVSNT was originally a port of CVS 1.10 and then 1.11 to Windows. It evolved into a de facto fork as new features were added. It was subsequently ported back to the original Unix platform. The CVSNT client and server currently support Windows, Mac OS X 10.3+, Linux Red Hat v4, Solaris 8/9, and HPUX 11i. The advantages of using a CVS for a single developer project is not as great as for a group, yet it is valuable, as changes can be traced over time.

CVSNT was used for WAVE 1 which proved invaluable due to that being a group project. Setting it up did prove time consuming at the time because of lack of documentation, yet it proved very useful to synchronise the source code with other team members and task delegation. Subversion was evaluated, but didn't offer any measurable advantages over CVSNT for this project.

JUnit

JUnit is the de facto standard regression unit testing framework for the Java language. Experience with the framework in the original WAVE and other projects demonstrated the advantages of using regression testing on software development projects.

WAVE 2 components

For the software itself, WAVE 2 relied especially on the Hibernate framework. MySQL was used as the target test database. Although not yet adopted, Piccolo has been taken under serious consideration because of very time-consuming issues with the Java 2D libraries.

The OO-RDBMS impedance mismatch

Impedance mismatch is a term coined to the problem of mapping an Object-Oriented (OO) software model manipulating the data to the underlying legacy Relational Databases (RDBMS) storing the data.[4] As software becomes more complex, the different nature of the two makes the manipulation increasingly cumbersome.

The quest for a better system has led developers down two different paths, towards Object Oriented Database Management Systems (OODBMS) on the one hand, and Object Relational Database Management Systems (ORDBMS). OODBMS can be said to be true OO systems, abandoning the old SQL standard, whereas ORDBMS extend SQL with OO features.[5]

A third alternative to solve the problem is to attack the problem in the intermediate layer.

Various solutions attempt to bridge this gap.

The beauty of Hibernate is that it enables the developer to focus on the true problems of software design and not on re-implementing solutions to the issue of basic storage. An issue which would not exist if it were not for the existence of impedance mismatch.

Consider the case where a product name has been misspelled and is used in various records across the database. If the misspelled word was a key, all tables where the key might occur would have to be scanned and the word replaced. The Hibernate approach provides a level of database integrity by isolating the database from erroneous user input. Correcting the misspelled word in such a case is a simple one-record edit.

The trade-off is that slightly more memory and database storage space is required, since a key is introduced which bears no relation to the data. This is easily compensated by the simplicity it introduces into the development.

Hibernate

Hibernate is a framework which uses the tried and true Data Mapper/Identity Map/Unit of Work patterns. The Data Mapper pattern is "a layer of mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself"[1]

A big difference between the OOPL and DB approaches is that the former uses pointers or references to link objects, whereas databases use explicitly defined primary keys to achieve the same thing. To support the database approach, Hibernate includes a suite of ID generators. The effect of this is that primary keys may not have any business meaning. This is less an issue than it appears at first, since using keys with meaning other than that of linkage introduces problems of its own.

A particular cause for consternation was the hard-coded user and password information Hibernate makes use of by default. For JSP solutions, where the source is kept on the server and not transferred to the client computer this is not an issue, but for client-side executable Java, this is a serious matter. This would mean that the database could be compromised by scanning the downloaded java files. A desirable approach is to use the login user name and password to the system to establish a connection to the database. The users must thus all be registered as users in the database. If the database connection fails, the user does not have access to the system either, and thus terminates. There is thus no 'secret key' which could reveal a backdoor to the system.

1 ***

The database works in this way as a coarse access control layer to the underlying data. Further access control must be done by business logic which in this case is in the client tier. With client-controlled business logic there is of course a possibility of this being used as a point of attack since the client could be hacked more easily than server-based logic.

Another advantage of Hibernate is its implicit support for transitive persistence[2], or what databases term referential integrity. In the example of a product order, if the product order is deleted, the product items belonging to that order should be deleted to, as their existence depends on the product order.

A key concern when Hibernate was selected for the project was its maturity, reliability and likelihood of future support for the framework. Google references to the project, the project's web site (http://www.hibernate.org/), the project's user forums, the project's roadmap and development timeline and latest version releases were examined. On the basis of those results it is clear that the product is stable for development as well as production projects, well supported and widely used.

XML-formatted database metadata

Instead of defining the database as a series of SQL statements mixed in between the general code portions, Hibernate enables the database to be defined in a separate XML file. The benefits are considerable: All the persistent data definitions are all in one place, and can be modified without recompiling the code. They are also written in fairly well structured XML.

```
<class name="wave.Node" table="NODES">
      <id name="id" column="NODE_ID" type="long" unsaved-value="null" >
            <generator class="native" />
      </id>
      <property name="name"  column="NAME"  type="string" />
      <property name="shape" column="SHAPE" type="integer" />
      <property name="colour"  column="COLOUR"  type="integer" />
      <property name="temperature"  column="TEMPERATURE"  type="integer" />
      <property name="x"  column="X"  type="integer" />
      <property name="y"  column="Y"  type="integer" />
      <property name="z"  column="Z"  type="integer" />
      <set name="connectedNodes" table="CONNECTIONS">
            <key column="NODE_ID" />
            <many-to-many column="END_NODE" class="wave.Node" />
      </set>
</class>
```

The advantages of modularity and extensible frameworks

Considerable emphasis in the design of WAVE 2 was good modularisation.

Looking at successful open source software projects such as Linux, Apache, Eclipse, it is interesting to note that these are all highly modular projects. It is worth considering whether one factor attributing to their success is not their modular nature:

Code modularity is the key to many successful open source projects (Linux, Apache, the re-vamped Mozilla, to name a few). Modularity matters because of the organizational nature of development teams.[6]

2 http://web1.theserverside.com/articles/article.tss?l=RailsHibernate

Extensible frameworks

It is evident that extensible architectures are growing into a flourishing ecosystem of service providers and third-party plugins, both Open Source and commercial. Open Source examples abound, such as the Mozilla Firefox browser and the numerous available extensions, and Mambo, for which many components are available, both free and commercial.[3]

MVC pattern

A key revelation came from the natural segregation which is found in the Model-View-Controller (MVC) pattern. The MVC design pattern was actually created by Xerox PARC for Smalltalk-80 in the 1980s.[7] It has become increasingly popular especially among PHP, Coldfusion and Java developers, although as a design principle it can apply to other platforms and languages as well.

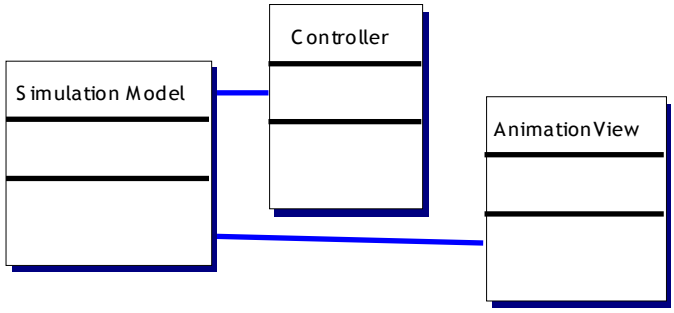


Illustration 1: Global selection – local stateless view

The MVC pattern divides the work between the:

- View – the passive observer of the model (output)
- Controller – the unit(s) exerting change upon the model
- Model – the core data, logic and business intelligence

The nature of the three elements also means that the

Applying MVC also brings to the equation a certain level of expectation about how things work together from the perspective of the developer. As an example, a model can have multiple views.

An issue here is for example how selection should work when there are multiple views on-screen. The MVC perspective is the default one; therefore a selection operation by the user forwards a screen coordinate or object selection operation to the model, which changes the model. The controller then notifies all views of this change. This yields a solution where selecting (with visual feedback such as highlighting) an object in one view reproduces it in all others. Some may argue that this may not be the expected behaviour; that a selection should be local to the view. In some cases, that may be more desirable. The other point of view can also be argued.

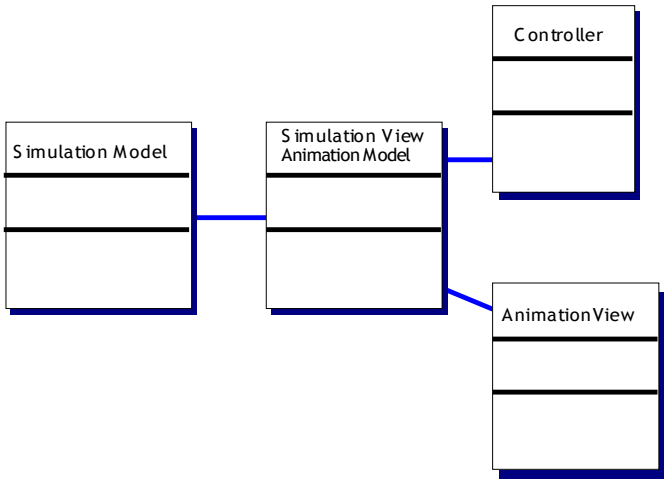


Illustration 2: Local selection – local stateful view

In reality, when analysing the problem, it

3 http://forum.mamboserver.com/forumdisplay.php?f=32

becomes clear that this is not a judgement call but a question of how the MVC principle is applied holistically. If local view selections are allowed, then states are actually being admitted into the view. A new, view local instance of the MVC pattern would thus be implemented. Instead of stateless views and controllers and a stateful model, certain view-related elements of the warehouse model state would be subjected to a reduced scope.

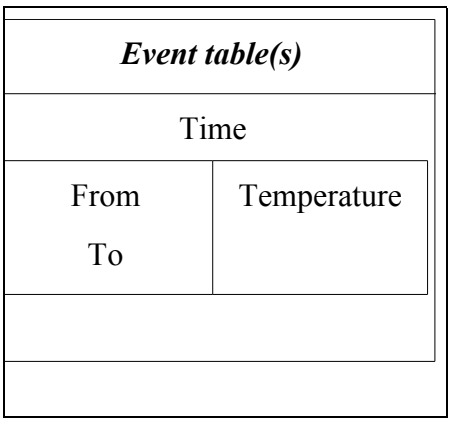
Implementation

Core architecture changes during implementation

The aim with WAVE 2 was to have a highly abstract core architecture. The manipulated objects, such as pallets, and crates on pallets, were to be manipulatable in an completely abstract, generic and equivalent manner. Objects could be containers for other objects without any limitations detail-wise; pallets could contain crates, crates could contain (yet smaller) boxes, which could contain individual pieces, if need be. Taking the principle in the other direction, a warehouse could be part of a warehouse complex.

As appealing as this may seem at first, the details with such an architecture are complex. Each level (pallet, box) has different features and capabilities. Upon further reflection, it was clear that this approach is unwieldy. Furthermore, there was no straightforward way of achieving the goal with a simple class. The object class was bloating to the point of breaking one of the main principles of good object-oriented coding: Smal, concise classes concerned with performing a specific function.

The solution to this thorny problem lies in taking the plugin principle one step further, and to apply the Concrete Table Inheritance pattern to the object types.



Movement thus becomes one event type, while temperature becomes another. Each event type, and each individual object type (pallet, box) uses the Concrete Table Pattern.

This also yields a more egalitarian framework. Priority may still be dictated, but on the whole, there is no event type that all others revolve around – except time. All events happen in time, and in essence, time is the only guaranteed common attribute to all events.

Movement nevertheless plays a very important part in the framework, so movement events have both a higher priority and require a good access interface for other parts of the framework to interoperate with it.

A byproduct of demoting movement to plugin status is that it yields a more concise class. Instead of a single class taking care of all the different object types, there are now specialised classes for every object type.

- Too many dependencies
- Too much exposure
- Too much state to manage
- Too hard to test

In the latter stage, as unit testing principles were reviewed, it became clear that the concrete table pattern, the plugin and the MVC design all were a better fit for the JUnit tests.

The inert model

In organising the segregation into model, view and controller, the prime aim was to extract as much control logic as possible and introducing it into controller classes. The model becomes a stateful class, whereas the the controllers and views become as stateless as possible. The views should retain only the bare minimum information to produce the view to the user.

This aim is not always that straightforward.

A case in point is the handling of time. Is the linear passage of time as natural a part of the model as the location of a node? Should the controller set the starting times when new time points are selected, or should a controller take care of moving time forward?

A room with many views

By opting for an MVC model, having many views for a single model becomes a natural and straightforward feature. In effect it becomes possible to have different views of the same model, and not only simultaneous, different types of views, but views of different parts of the model, zooming, for example, to one aisle, while having a separate window with a whole view of the storage area.

Hibernate

In the last implementation phase, Hibernate was upgraded from version 2 (H2) to version 3.1 (H3). This caused a few issues. One such was a JNDI javax.naming.NoInitialContextException exception. After scanning the reference manual, and the FAQ files, the solution presented itself in a forum post . Although this used to work with H2, H3 deems the presence of a name within the <session-factory> parameter in hibernate.cfg.xml to be an error. The reason this was not addressed in the FAQ or reference guide is probably that use of hibernate is more common as a web server platform, i.e. In conjunction of

Going from H2 to H3 also required moving the Document Type Definition which the hibernate.cfg.xml file referenced from hibernate-configuration-2.0.dtd to hibernate-configuration-3.0.dtd.

Evaluation

Currently the 2D view is not working correctly. Debugging the Java 2D framework proved very time consuming. The Console unit manages pallets at the moment although there appear to be persistence issues at the moment.

The plugin interface is not yet completed although the necessary groundwork has been completed and its implementation is at this stage quite straightforward.

The project has now reached a stage where the 'hard' issues have been resolved. What remains is to implement certain parts of the architecture in accordance with the architectural principles.

References

[1] http://www.gold-solutions.com/com/File/G.O.L.D.%20Track%20-EN_2005.pdf

[2] http://www.gold-solutions.com/com/File/brochure_stock_2005_en.pdf

[3] http://www.gold-solutions.com/com/File/brochure_stock_2005_en.pdf

[4] Shushman, Dan, Oscillating Between Objects and Relational: The Impedance Mismatch, http://www.odbms.org/download/023.01%20Shusman%20The%20Impedance%20Mismatch%202002.PDF

[5] Chountas

[6] Stone, Mark, 18.8.2004, Going open source: A manager's guide to doing it right, http://business.newsforge.com/article.pl?sid=04/08/18/0715259&tid=111

[7] Kotek, Brian, 30.10.2002, MVC design pattern brings about better organization and code reuse, http://builder.com.com/5100-6386-1049862.html