

Comparison of Development Efforts between MDD and Traditional Development

A productivity analysis using VE
and Eclipse

Prepared by Intelliun Corporation

CONTENTS

- ✓ Executive summary
- ✓ Project and technology overview
- ✓ Results matrix
- ✓ Development task details
and comparisons
- ✓ Conclusions

Executive summary

This whitepaper compares the development of a small but realistic business application using a traditional IDE (Eclipse) and The Virtual Enterprise (VE) platform.

Using VE:

- ✓ Required less development time overall
- ✓ Required less customization of default behavior
- ✓ Allowed early validation of design decisions.
- ✓ Minimized tool setup and configuration time.
- ✓ Reduced repetition of model data across the implementation, and captured the data once at the business-model level.

Introduction

About the Project

The test project involved a small invoice management application. Although relatively simple, the project represents a realistic example of the kind of internal business application often produced by IT departments. The application had the following web pages:

- ✓ Invoice list
- ✓ Sample invoice with item detail
- ✓ Item detail page

The implementation of both projects concentrated on basic application functions. Although much of the default error handling and default user interface look and feel for each environment was left in place, the end result was intended to be an application that was ready to be deployed into a production environment.

About the Technology

VE overview and key development features

The Virtual Enterprise (VE) is a sophisticated model-driven development environment that enables the construction of complex business applications. VE presents a simple, highly integrated interface with a full set of built-in tools. VE has two main components: VE/Designer and VE/Server.

VE/Designer contains a rich set of visual modeling tools and a powerful user interface customization mechanism. It is built on top of the NetBeans™ platform from Sun and can be used on a wide variety of operating systems and platforms.

VE/Server is the runtime environment that directly executes the models constructed with VE/Designer. It is implemented in Java, and deploys as a servlet or EJB into any J2EE compliant application server.

Eclipse overview and key development features

Eclipse is an industry-standard Open Source Integrated Development Environment. It consists of a complex array of powerful frameworks and modules that can be assembled to form a myriad of customized environments. Eclipse ships several standard configurations, but for this exercise we chose the "Eclipse IDE for Java Developers" package, with the addition of the following tools:

- The TOAD database schema management tool
- The MyEclipse JPA toolkit for persistence object generation
- The MyEclipse JSF Web application toolkit for the development of the user interface.

Other Infrastructure

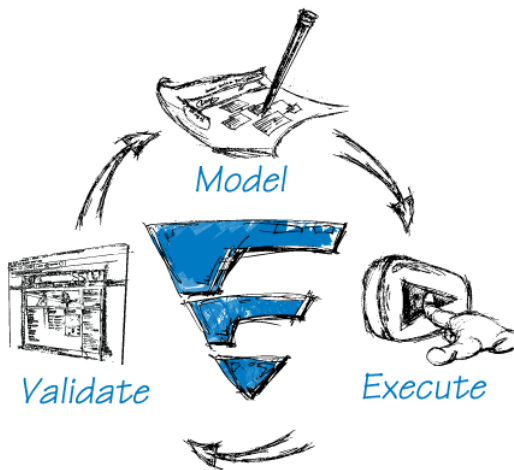
Both sample applications use the MySQL 5.0 database and the Tomcat 5.5 application server.

About the Approach

The development effort consisted of an initial analysis phase (the same for both approaches) followed by an implementation phase. However, the tools and best-practice techniques available in each environment differ to such an extent that it isn't possible to make an exact one-to-one comparison. Instead, we will present the details of each effort, and then compare the overall work required.

VE Approach Details

The VE approach emphasizes a series of low friction iterations. Since the model is directly interpreted, changes can be instantly validated by re-running the model and observing its behavior.



A developer generally operates in a model/execute/validate cycle that can last anywhere from a few seconds to many hours.

Development using VE focuses on iteratively refining the high-level business model rather than on low-level coding. This difference in focus is central to the effective use of VE, and is reflected in the nature of the steps in the development process:

- Capture the initial model
- Persistently capture realistic test data
- Refine the business model
- Capture use cases as Portals and Navigations
- Finalize the GUI

VE's support for change management and fast iterations encourages an iterative approach, although in this case the model was simple enough that only one pass was required.

Eclipse Approach Details

The Eclipse project contains many tools at all levels of sophistication and maturity. There is no one single approved development methodology. The individual tools generally integrate at the code level, and low-level technology issues tend to define the steps in the development process. (Of course, developers with enough discipline can always operate in a model-driven manner regardless of tooling support.) For this project, the following steps were taken:

- Assemble the tools and create the project
- Create the database schema
- Generate and customize the persistence layer
- Create the JSF beans
- Wire the beans
- Embed application code into the beans
- Design the web pages
- Design page navigations
- Implement event handling

Different tool choices might produce a different sequence of steps. With Eclipse, there's generally a tradeoff between setup and configuration time for the tools, the level of expertise required to use the tool effectively, and the time saving effect of the tool. In this case we tried to take a middle-road approach, using tools that were relatively easy to set up but that saved substantial time over pure hand-coding.

Results summary

The high-level findings and observations made during the development of both sample applications include:

- ✓ Fewer tools to configure means faster and easier project setup with VE.
- ✓ With VE, the impact from new or changing requirements is much smaller because the IDE handles the bulk of the change “ripple effect”.
- ✓ VE leverages its knowledge of the model to eliminate time-consuming development tasks like generating the persistence layer, creating JSF beans, and wiring beans. There is only a single, high-level realization of the model.
- ✓ Embedding the business logic code—typically the most time consuming and error prone development task—required substantially less effort with VE.
- ✓ Web page design and navigation are dramatically simplified with VE.

Results matrix

TASK	INTELLIUN VE (Hrs)		
	Init	Debug	Total
Capture the initial model	0:10	0:05	0:15
Persistently capture realistic test data	0:10	0:00	0:10
Refine the business model	0:20	0:10	0:30
Capture use cases as Portals and Navigations	0:10	0:00	0:10
Finalize the GUI	0:15	0:00	0:15
TOTALS	1:05	0:15	1:20

TASK	ECLIPSE (Hrs)		
	Init	Debug	Total
Create project	0:15	0:00	0:15
Create database schema	0:20	0:00	0:20

Generate and customize persistence layer	0:35	0:20	0:55
Create JSF beans	0:15	0:15	0:30
Design web page	0:40	0:00	0:40
Embed code into beans	0:30	0:30	0:20
Wire beans	0:20	0:10	0:30
Design page navigation	0:25	0:45	0:70
Implement event handling	0:15	0:20	0:35
TOTALS	3:35	2:20	5:55

Development task details for VE

Capture the initial object model

The VE/Designer installation includes a copy of Tomcat and an in-memory object store. The main effort in starting a new project is coming up with a good name and the entire process takes only a few moments.

For both the VE and Eclipse efforts, an initial first-pass object model was developed. The model was captured using VE/Designer's built-in diagramming tool. VE allows even very early phase models to be executed and validated.

Capturing the model and running through a few initial rounds of refinement took about fifteen minutes.

Persistently capture realistic test data

Although VE's in-memory object database is helpful during early model validation, we find it useful to persistently capture realistic test data very early in the development process.

VE helps with this process by generating a complete Object/Relational mapping based on the model, then automatically updating the mapping when the model changes. Updates leave unchanged parts of the model intact, so there's generally no need to re-enter data. In addition, VE dynamically generates the user interface needed for basic data manipulation. As soon as there's a model, there's a GUI.

Entering the initial test data took less than ten minutes.

Refine the business model

The bulk of the development effort in a VE project consists of iterative refinement of the business model. In this case the model is relatively simple and was largely captured during the initial analysis phase. We skipped complex package structure and model reuse considerations and concentrated on a small, simple model in a single package.

VE/Designer captures details of the model several ways:

- The relationships between classes can be specified in detail. For example, a bidirectional composition relationship is mapped to the user interface and database layers differently than a unidirectional association.

- VE makes use of the exact attribute types given. For example, Date fields are presented in the user interface via a calendar widget, and constrained-value String attributes are presented as combo-boxes.
- Attribute can be further constrained via inline formulas (a default value of “today()” for a date field.)
- Workflows can be represented by a sequenced series of actions, as represented by either a diagram or by a traditional textual function definition. VE handles the mapping of the sequence of actions into a series of HTML pages. The complex manual coding associated with maintaining page state is eliminated.

Model refinement took about half an hour. In a more complex case, this stage could last several weeks and involve hundreds of classes spread across dozens of packages. Even in the complex case VE ensures that the model can be executed and verified at all times, either by the developer or by end-user stakeholders.

Capture use cases with Portals and Navigations

VE will dynamically manage many common user interactions based on model constraints. For example, a CRUD (Create/Read/Update/Delete) editing system is built-in and automatically available as soon as the model is defined.

Application specific interactions, often recorded via use cases, can be captured using VE/Designer’s Portal and Navigation tools. By default, use case actors are captured as Portals and presented as login-protected top-level application entry points. The individual use cases for an actor are captured as Navigations, and presented as dynamically generated tabs, panels and buttons in the application’s menu system.

Since most of the features in this application were implemented using VE’s built-in mappings, relatively little effort was required and the simple use cases were captured in about ten minutes.

Finalize the user interface

Although VE will automatically generate a user interface based on the application model, it also includes an extensive set of user interface personalization mechanisms. Developers can use a simple WYSIWYG interface to customize the presentation of the model or drop down and code in a template language. Customizations can optionally be registered into the development framework and made available for reuse by other developers.

For large projects a substantial amount of effort can go into the design of the final look and feel of the application. Special colors, logos and even complete custom HTML

templates can be specified. However, for the sample application the defaults were used and this step took about fifteen minutes.

Development task details for Eclipse

Create the project

This task consisted of defining a new project and configuring the various tools needed for modeling, persistence, web services, and web page authoring. With Eclipse, each tool must be configured. Most tools feature a “wizard” style setup process to simplify and streamline the configuration steps. Project setup took about fifteen minutes.

Create a database schema

Creating a database schema involves the design of database tables, fields, and constraints, captured in Data Definition Language (DDL) file.

The initial task of entering a schema is relatively simple, and involves manually specifying the appropriate tables, field, constraints and keys. We chose to use TOAD – a free data management utility – to streamline this step.

When the entire model is defined from the outset, this task is generally straightforward. Complexities can be introduced if the model changes. Changes can have negative impacts across all areas of the project. Large projects with more complex object/relational mapping requirements exacerbate change management issues.

This step took about twenty minutes, most of which was spent sketching out the appropriate table structure and foreign key relationships.

Generate and customize the persistence layer

The persistence layer consists of Data Access Objects that interact with the database, and persistence beans that encapsulate the object model.

To generate the boilerplate we selected the MyEclipse JPA plugin from a number of possible options. The plugin generates the base objects, but the persistence beans must be customized with application-specific code. For example when optimizing a query or defining transaction borders. Tools exist that can assist with those tasks but they are generally somewhat complex to configure.

Because the sample application is a small project, we modified the persistence beans manually. Advanced database layer tools take more time to set up and configure, but would be worthwhile on a larger project.

The Eclipse effort required forty lines of code (twenty lines for each JPE bean) and took about fifty five minutes to generate and customize.

Create JSF beans

Creating JSF beans involves creating objects to manage the web page and direct user interactions.

Creating default beans is tedious, even using MyEclipse JSF to assist with the task. We had to re-enter nearly identical information for the persistence entity beans, then lost substantial time correcting mapping inaccuracies.

The generator produced three Java classes: Invoice, Invoices and Item. Generating and customizing the beans took about thirty minutes.

Wire the beans

Wiring the beans creates connections that allow filling web pages with data from the beans and sending user-entered data back to the beans.

Coding the data transfer between beans and web pages can be a long and error prone process. No tool automates it completely. MyEclipse provides only code completion in the web page.

The development effort involved provisioning for every field and action in the JSP pages and method definitions for the actions in the JSF beans. The process took about half an hour, including debugging time.

Embed application code into the beans

Best practice indicates that all business logic must be encoded in the beans. Because each business application is unique, embedding the application code in the beans cannot be automated and Eclipse provides no assistance other than the standard editor tools. The following steps describe the process used to complete this task for the sample application:

1. Add state information to the session, including:
 - ✓ Current object selected
 - ✓ Dirty state of each bean
2. Implement relations between the beans.
3. Enable two-way translation between the JSF beans and persistence beans.
4. Establish error handling for various conditions, including but not limited to the following:

- ✓ Loss of connection
- ✓ Invalid user-entered values
- ✓ Incompatible values in the database
- ✓ User requests that are inconsistent with the current state

Embedding the application code using Eclipse produced two hundred lines of code and six new methods in the JSF beans. It took about an hour.

Design the web pages

This task consisted of creating the JSP templates for the application's web pages. We chose to use the MyEclipse WYSIWYG visual editor from among a number of options. Designing the web page is a potentially graphic-intensive task that involves creating JSP- and JSF-augmented HTML pages.

Six JSP files were created: `index.jsp`, `invoices.jsp`, `invoices.jsp`, `item.jsp`, `ResultInvoice.jsp` and `ResultItem.jsp`. Creating the pages took about forty minutes.

Design page navigation

The navigation structure determines how users will move between the pages in an application.

Because Eclipse does not have a built-in tool for this time-consuming task, we designed the navigation using the MyEclipse JSF plug-in. The plug-in has excellent graphical layout capabilities and generates a JSF XML file with accompanying navigation information.

The web page navigation design resulted in forty lines of code in `face-config.xml`, ten lines of code in the action methods of the JSF beans, and took about thirty five minutes.

Implement event handling

Event handling must be implemented to support user actions that have an immediate impact on the web page. For the sample application, events are used for the provisioning of items.

JSF provides excellent event management support across all HTTP response processing steps. Any JSF bean can be an action listener.

To implement event handling in Eclipse resulted in twenty lines of code for the JSP web page and thirty lines of code in JSF beans and took about thirty five minutes

Conclusions

VE demonstrated clear advantages during the development of the sample application.

- ✓ VE required less development time overall
- ✓ VE required less customization of default behavior
- ✓ VE's support for executing even incomplete models allowed early validation of design decisions.
- ✓ VE contains a full set of closely integrated tools, so tool setup and configuration time was minimized.
- ✓ VE captured high level design once, in the model, and doesn't force the user to re-enter information the system has already been told.
- ✓ Entire steps, such as generating the persistence layer, creating JSF beans, wiring the beans, and event handling, are eliminated from the development process when using VE.

VE's advantage on this small project is multiplied for larger projects. To find out more about VE's support for programming in the large (or small), please visit the Intelliun website at <http://www.intelliun.com>. You'll find product overviews, documentation and walkthroughs of the development process for a larger application.