intelliun
empowering your business

Demonstrating the Productivity of The Virtual Enterprise for Building Applications and Web Services

Prepared by
Intelliun Corporation

CONTENTS

Introduction

This article demonstrates the power and productivity of The Virtual Enterprise (VE) through the process of developing a simple HR application to manage employees, and search for an employee by employee number. By following the steps as documented, the reader can understand how VE provides a significant improvement in developer productivity relative to standard application development approaches. VE provides an integrated platform for capturing the complete application business logic that can be deployed on a wide verity of J2EE implementations without requiring expertise in J2EE. These advantages translate into significant cost benefit and time-to-application advantages for development organizations.

In order to compare VE to traditional development platforms we present the following application requirements for a simple HR System for maintaining Employee Information.

The application consists of two use cases:

1. Manage employee information
2. Find employee information by employee number

For each employee, the application captures core employee information, namely employee number, name, social security number (SSN), hire date, title, address (street, city, state, and zip), and contact information (phone, email, mobile, and mail drop).

As far as non-functional requirements, the application must support access via web-browser, Web services, and handheld devices (PDA). In addition, it must be platform independent, database independent, and scalable.

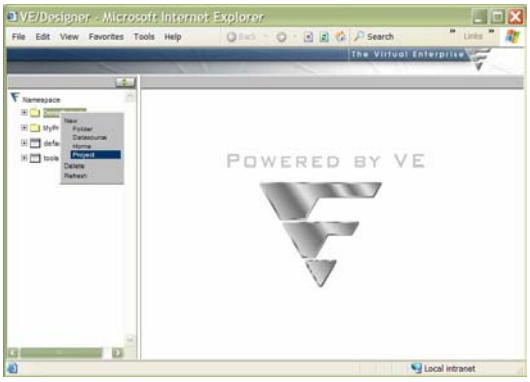
The stated efforts below did not account for any analysis or design activities. The simple models were scribbled on a piece of paper before starting the exercise and were used as the foundation for the VE implementation.

Building a Sample Application Using VE

Required Skills:
VE/Designer

Cumulative Effort:
10 seconds

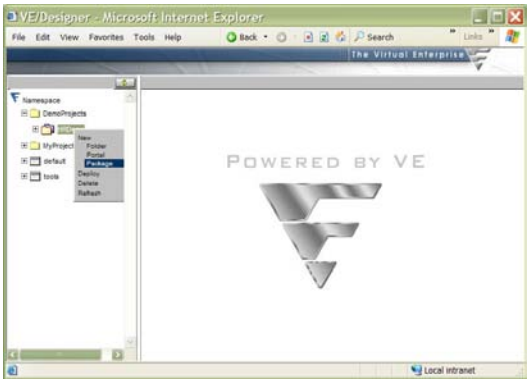
Step 1:
After starting VE/Designer, create a new project by right-clicking on any folder then selecting **New Project**. Name the project *HRDemo*. An icon for the project is inserted under the selected folder. With this step you have created your project workspace.



Required Skills:
VE/Designer

Cumulative Effort:
20 seconds

Step 2:
A typical application will be broken into logical groups called packages analogous to Java packages. For this sample application we will create a package named *hr* by right clicking on the project and selecting **New Package**. An icon is inserted into the namespace under the project *HRDemo* and you will name it *hr*. With this step you have created a package within the project.

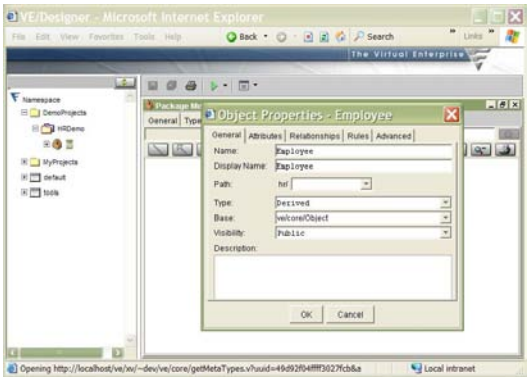


Required Skills:
VE/Designer

UML Class Diagram

Cumulative Effort:
1:20 minutes

Step 3:
VE/Designer allows you to build the object model visually. To create an object for the employee information, drag the **Object Tool** in the tool bar and drop it on the canvas. A dialog box will appear in which you can enter the meta object name *Employee* and then enter the attributes *number*, *name*, *SSN*, *hireDate* and *title*, then press OK. The object appears on the canvas.



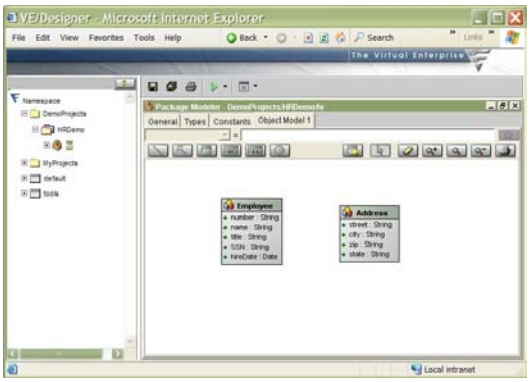
Required Skills:
VE/Designer

UML Class Diagram

Cumulative Effort:
2:00 minutes

Step 4:

To capture the employee address information, repeat Step 3, however, name the meta object *Address* and assign it *street*, *city*, *state* and *zip* as attributes. The object appears on the canvas.



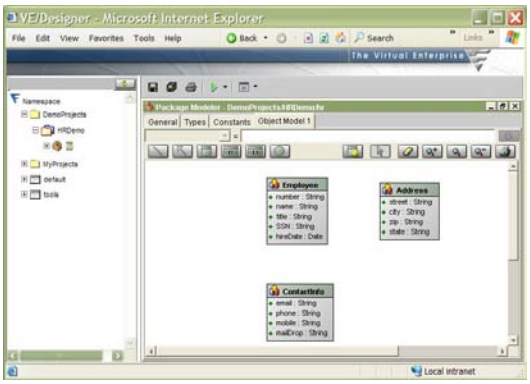
Required Skills:
VE/Designer

UML Class Diagram

Cumulative Effort:
2:50 minutes

Step 5:

Repeat Step 3 again to create the employee contact information, while assigning the meta object the name *ContactInfo* and adding attributes *phone*, *email*, *mobile* and *mailDrop*. The object appears on the canvas.



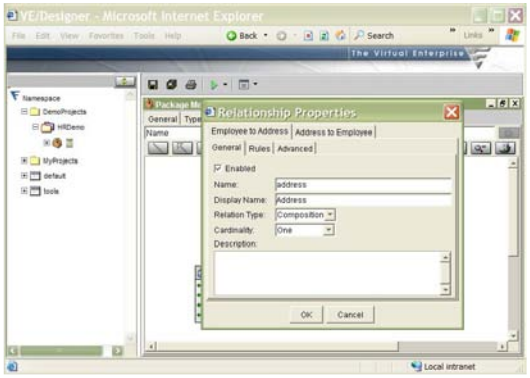
Required Skills:
VE/Designer

UML Class Diagram

Cumulative Effort:
3:10 minutes

Step 6:

To model the relationships between the objects on the canvas select the **Relationship** tool in the tool bar. Select the first object in the relationship clicking and dragging the line to the second object. In this manner you will create a relationship named *address* between *Employee* and *Address*. At this point you will also select the relationship type and cardinality.



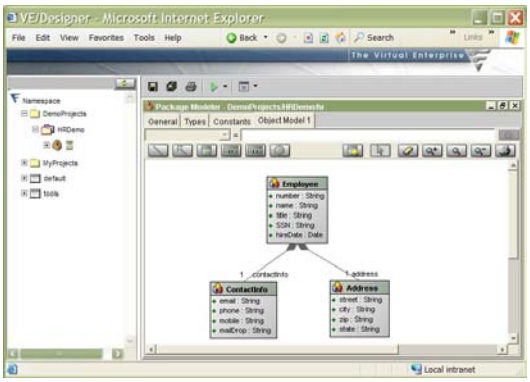
Required Skills:
VE/Designer

UML Class Diagram

Cumulative Effort:
3:40 minutes

Step 7:

Repeat Step 6 to create another relationship between the *Employee* and *ContactInfo* objects. This relationship will be named *contactInfo*.



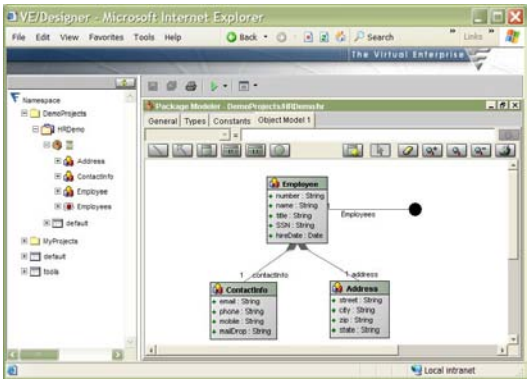
Required Skills:
VE/Designer

UML Class Diagram

Cumulative Effort:
4:20 minutes

Step 8:

The final step in the object model is to create a root object (Singleton) named *Employees* to identify the entry point into the object model and determine the objects that are persisted to the database in the application. Select the **Root** tool in the toolbar and select a spot on the canvas to place it. Add a relationship between the root and the *Employee* object. You now have a complete object model that has all objects, attributes, relationships, and the persisted data requirements determined.



Required Skills:
VE/Designer

UML Activity Diagram

Cumulative Effort:
5:40 minutes

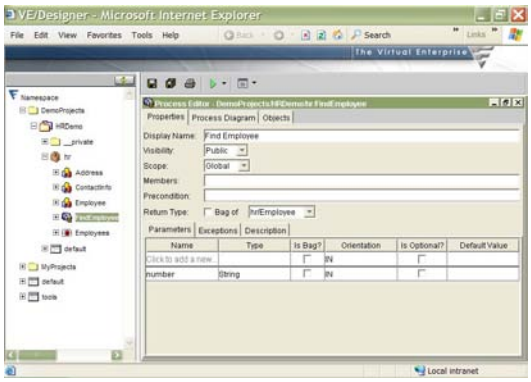
Step 9:

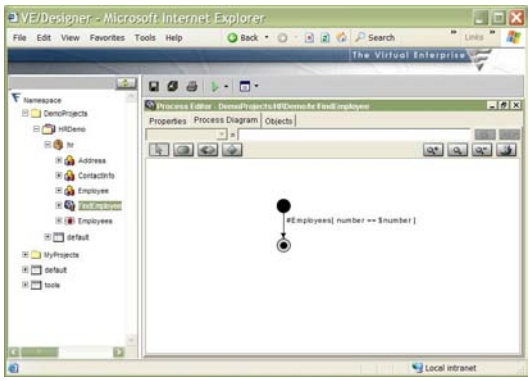
Create a new process by right-clicking on the package icon and selecting **New Process**. Name the process *FindEmployee*. An icon for the process is inserted under the package. Under the properties tab, add a *number* parameter and specify the return type as *Employee*. Switch to the Process Diagram tab and add the desired business logic. In this case, this is a very simple process with no actual activities. Instead, all we need is to find an employee object in the root *Employees* with a *number* equal to the *number* parameter, then return to the caller. More specifically, the formula for the return value is:

```
#Employees[ number == $number ]
```

The resulting process model is shown on the canvas below right.

At this point all information to complete the application itself, including both user interfaces as well as application logic has been entered and the application can be run in memory by simply clicking on the green play button—no code-generation nor compile time.



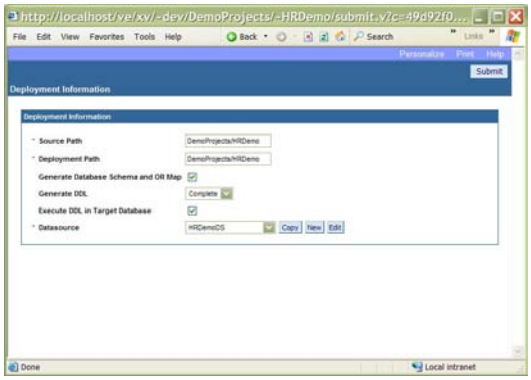


Required Skills:
VE/Designer

Cumulative Effort:
6:40 minutes

Step 10:

To create and deploy the application to a database, right click on the project icon and select **Deploy**. VE will provide you with a set of pages to create a new database instance and a datasource. Name the datasource *HRDemoDS,* then submit the form. VE will analyze the object model and create the appropriate DDL (database schema), object-relational mapping and optionally update the database automatically.

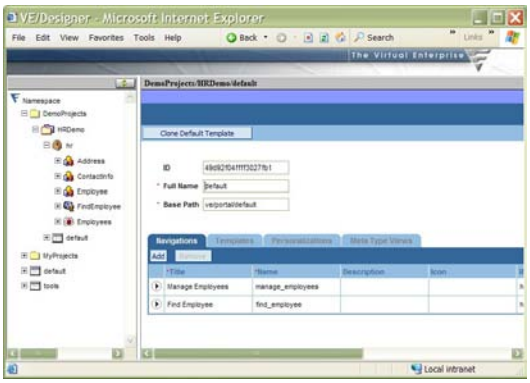


Required Skills:
VE/Designer

Cumulative Effort:
7:50 minutes

Step 11:

To complete the final step of the application, select the *default* portal under the project, then add two navigations, *Manage Employees* and *Find Employee*. Specify the root object *Employees* under the *Manage Employees* navigation and set its type to **Manage**. For the *Find Employee* navigation, specify the *FindEmployee* process and set its type to **Execute**.

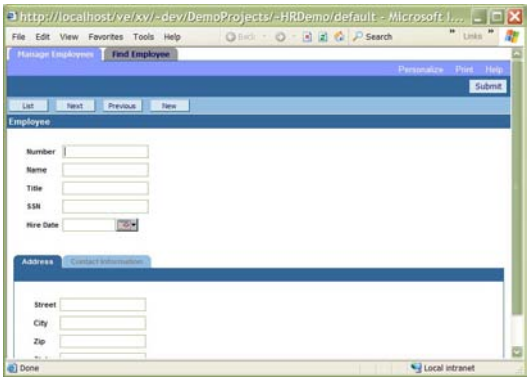


Required Skills:
VE/Designer

Cumulative Effort:
8:00 minutes

Step 12:

At this point the application is complete. Right-click on the *default* portal and select **Execute**. The portal will launch, and the first navigation will be executed (in this case it's the *Manage Employees*). All interactions with the application are live and running on the originating J2EE server. Any changes to the state of the system will be persisted on the created database.



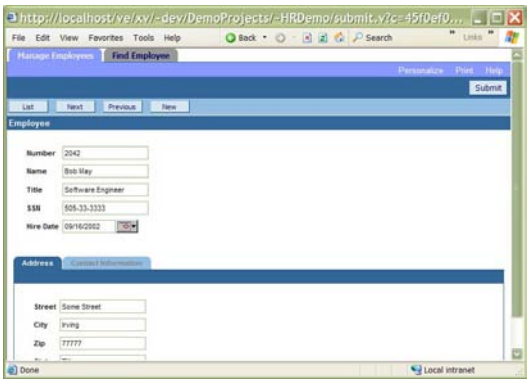
We're done!

This application is developed and deployed in under ten minutes from start to finish using The Virtual Enterprise. Admittedly, this is a simple example, however, it covers a set of common functionality found in real life applications. More complicated applications can similarly be developed by purely capturing the business logic, namely objects, processes, and rules, and without writing a single line of Java code nor learning the 10 to 20 different standards and specifications involved in building n-tier J2EE applications and Web services.

The following section provides an overview of the completed application.

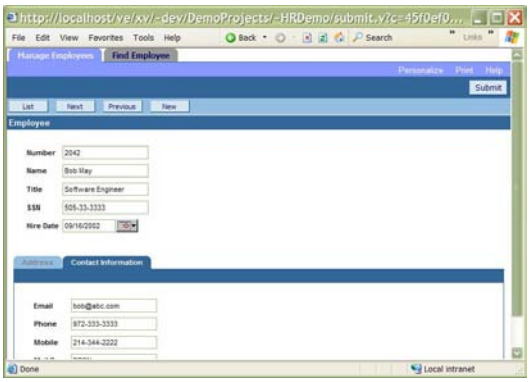
Executing the Application

This screen shows the execution of the Manage Employees use case of the application using test data for the entry of an employee. This is a detailed view of the employee record, including address and contact information.

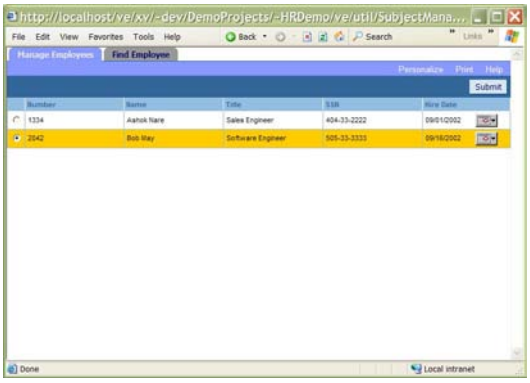


The user can use the **List** button to view a list of all employees, the **Next** and **Previous** buttons to navigate through all employees, and the **New** button to create a new employee.

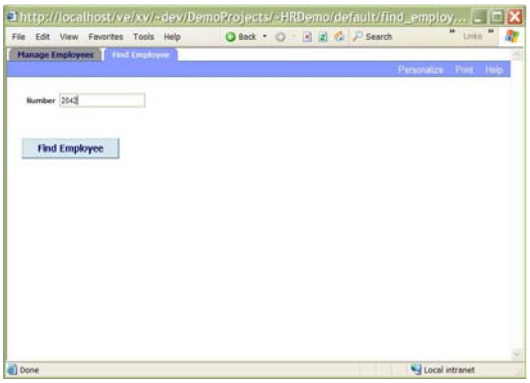
Changes to any employee information are submitted to the server and updated in the database accordingly.

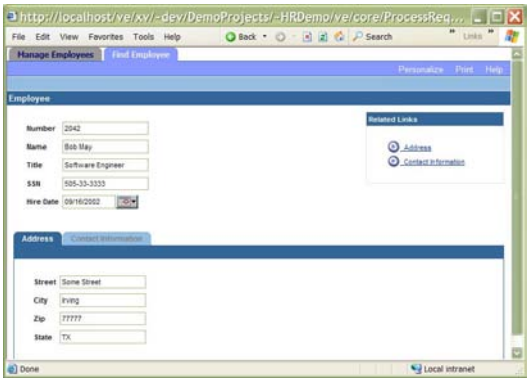


This screen shows the list summary view for employee records as selected by the **List** button in the pages above. Double-clicking on a specific row will bring up the detail view for that record as shown previously.



The **Find Employee** use case is rendered as a separate tab in the portal as shown. Upon clicking on the Find Employee tab, the *FindEmployee* request object is rendered to the user to capture the desired employee number. Upon entering an employee number and clicking on the Find Employee button, the server will execute the *FindEmployee* process and displays the matching employee.



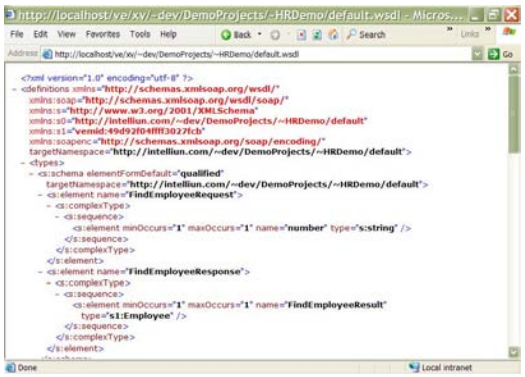


Note that the application shown is completely generated based on the steps described previously according to the default look-and-feel set in VE. VE/Designer also provides a powerful WYSIWYG editor for customizing all aspects of the web pages and rendering a very sophisticated user experience, still without requiring any HTML, CSS, JavaScript, or XSL experience.

Web Services Support

Thus far, we have not done anything regarding Web services. However, just like the dynamically generated HTML web pages, VE automatically creates WSDL documents appropriate for each portal, and responds to incoming SOAP messages. VE views Web services as just another interface layer (or access device), and maintains the focus of the functionality on the application logic (i.e. object, processes and rule).

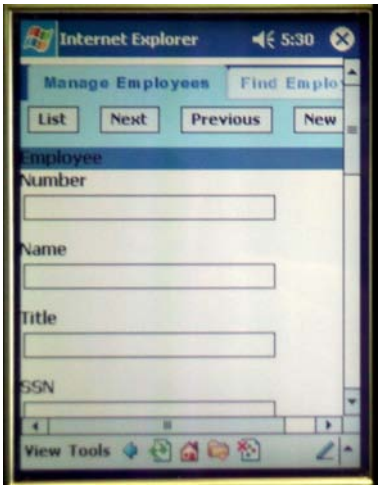
In this example, there is only one portal, *default*. Accordingly, entering the default portal URL with the extension *.wsdl* will dynamically return the appropriate WSDL to match the portal specifications and application logic. Similar to the web-pages personalization, the developer has the ability to alter the generated WSDL (e.g. document vs. rpc style, etc.).

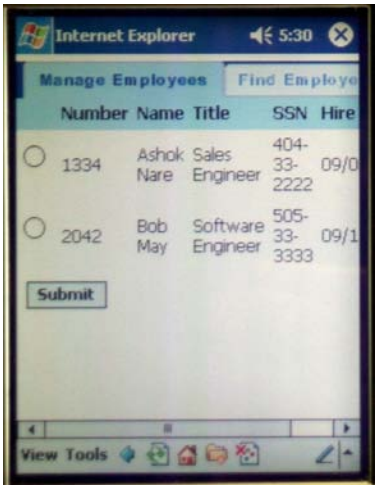


Handheld Device Support

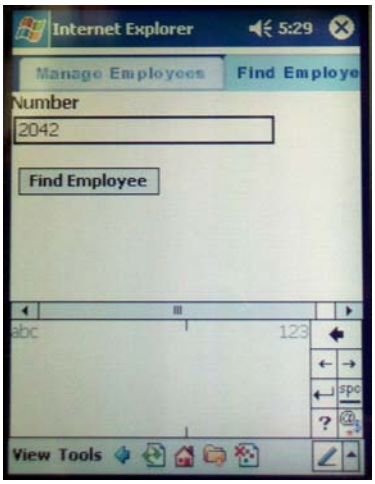
As built and deployed, the application is automatically available for access via handheld devices such as the PocketPC using Pocket Internet Explorer and other similar handheld device browsers. The pictures below show the same application built above without any additional steps accessed from a PocketPC device.

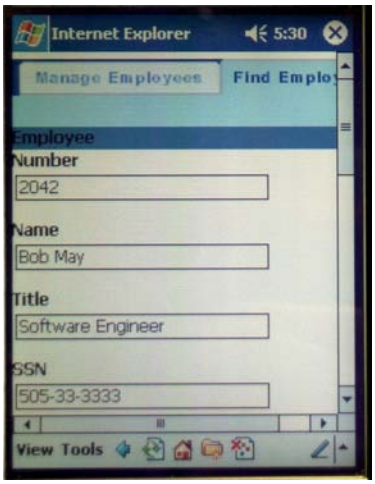
These two pictures of a PocketPC screen show the Manage Employees tab of the application including the employee record and the employee list views. All functionality of the application is available to handheld users, who would use the data entry capabilities of the device to enter information or press buttons as appropriate.





These two pictures of a PocketPC screen show the Find Employee tab of the application including the search form and the result of the search, an employee record.





Note that these pictures show the default application look-and-feel as generated by VE. However, the VE/Designer provides WYSIWYG personalization capabilities for handheld devices, still without requiring any additional skills.

Summary

The Virtual Enterprise (VE) provides an end-to-end platform for business application and Web services development. VE layers on top of any third-party J2EE implementation to deliver robust and scaleable solutions. VE's approach to application development is based on sound software engineering principals keeping the developer focused on the middle tier (the application logic), while automating the top tier (interface) and bottom tier (persistence).