



# Java Application Servers Report

A report by  
TechMetrix Research



---

# Java Application Servers Report



BEA

WebLogic



GemStone

GemStone/J



IBM

VisualAge – WebSphere



Inprise

JBuilder – Inprise App. Server



Oracle

JDeveloper – Oracle App. Server



Symantec

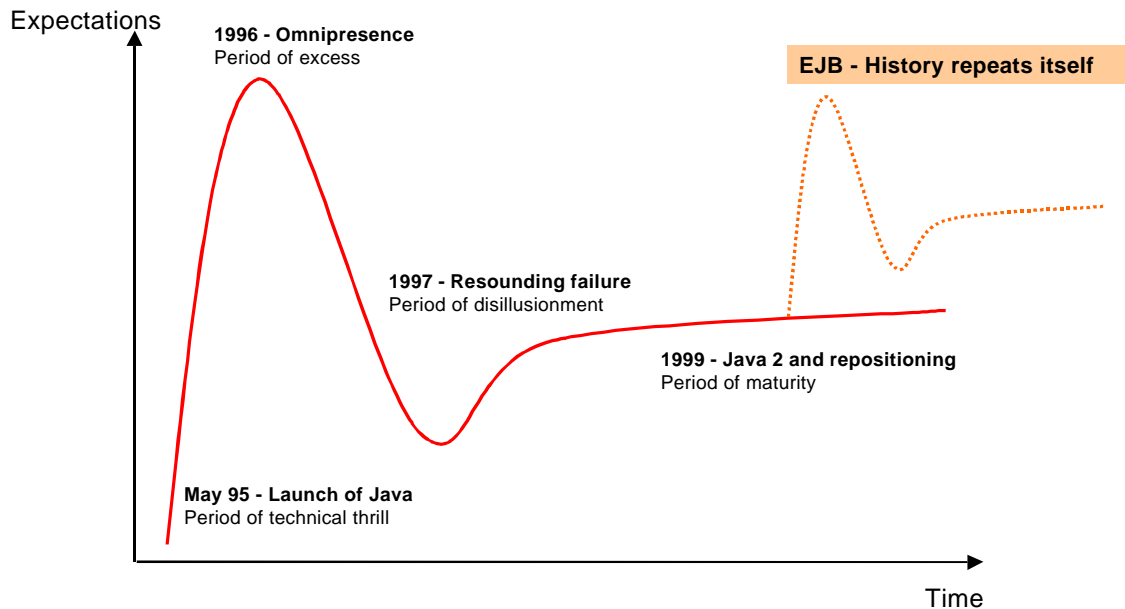
VisualCafé

A report by  
TechMetrix Research  
(R&D Department of Groupe SQLI)



# FOREWORD

Anyone who has not yet heard of Java, its possibilities or its potential has an unflinching passion for absolute isolation. On the other hand, seeing a Java application being developed that completely satisfies its users is a privilege that only a handful of computer specialists have shared. How can this be explained? Very simply by the clever exploitation of the techie nature of our community by another community, that of marketing professionals. Americans have given a name to this phenomenon. They call it "hype". The hype curve, which we present below for Java, is classic and occurs systematically whenever a new technology is born. Let's follow it to acquire a better understanding of the current positioning of Java:



In the beginning, the first foundations were laid. This is the period of technical thrill. The launch of Java at SunWorld in May 1995 was the peak of this period. Throughout 1996, editors such as Netscape, Oracle and IBM rallied with Sun. These promoters proclaimed far and wide that Java would be the language of the Internet due to its multi-platform character. The press grabbed hold of this phenomenon. The media storm was raging. At that time, while actively monitoring the topic, TECHMETRIX RESEARCH recommended that its clients beware of applets and prioritize HTML-HTTP technology.

Following these excessive expectations, disappointment was not far behind. The limitations of applets became evident. However, large projects ended in resounding failure (Corel, etc.). Numerous clients turned their backs on Java. Nevertheless, since 1997, we have been validating Java in our study of Intranet HTML version 3 development tools with NetDynamics as leader of the group leader. At the end of 1998, for version 5, out of the 11 tools in the study, seven allow for development using Java. Numerous editors (SilverStream, IBM, Progress Aptivity, etc.) have since repositioned themselves on this HTML-Java market.

---

Now we believe that the technical element of Java has reached a certain maturity and this new report identifies areas in which Java has been adapted to production environments. It is amusing to note that, while certain "media" detractors had reproached us for putting the brakes on the movement, we must now convince our clients who got their fingers burnt through bad experience of the potential of Java. This is one of the reasons that pushed us to invest nearly 200 days of R&D time on this topic. Nevertheless, we must not forget that history repeats itself. This report highlights the truth behind the misleading marketing statements by editors in order to avoid a new wave of "hype".

Finally, the role of our analysis firm is to update one of the weaknesses of the community of marketing professionals (outdoing one's competition) for the benefit of the community of computer specialists.

Jean-Christophe Cimetiere - CEO of TechMetrix Research (jcc@techmetrix.com)

---

**How to contact us:**

---

**In the US:**

TechMetrix Research  
6 New England Executive Park, Suite 400  
Burlington, MA 01803

Tel.: +1 (781) 270-7486  
Fax: +1 (781) 270-7489

<http://www.techmetrix.com>  
[info@techmetrix.com](mailto:info@techmetrix.com)

**In Europe (French headquarters):**

TechMetrix/Groupe SQLI  
Département R&D  
55/57 Rue Saint Roch  
75001 PARIS

Tel.: +33 1 44 55 40 00  
Fax: +33 1 44 55 40 01

<http://www.sqli.fr>  
<http://online.sqli.fr>  
[R&D@sqli.fr](mailto:R&D@sqli.fr)



---

# TABLE OF CONTENTS

<b>1. INTRODUCTION.....</b>	<b>1</b>
1.1. HISTORY OF JAVA.....	1
1.2. KEY JAVA APIS.....	9
<b>2. JAVA AND THE CLIENT INTERFACE .....</b>	<b>15</b>
2.1. CLIENT-SERVER JAVA.....	15
2.1.1. AWT and Swing .....	15
2.1.2. JavaBean components .....	22
2.2. JAVA APPLETS .....	26
2.2.1. Support of Browsers.....	26
2.2.2. Communication with the server.....	27
2.3. HTML INTERFACE .....	33
2.3.1. Servlets.....	33
2.3.2. JSP .....	37
<b>3. JAVA AND SERVER PROCESSING .....</b>	<b>39</b>
3.1. APPLICATION SERVER .....	39
3.1.1. Java runtime environments: JVM, JIT, Hotspot .....	39
3.1.2. Access to data sources and processing .....	40
3.2. OBJECT SERVER (EJB).....	42
3.2.1. Introduction of EJBs .....	42
3.2.2. General introduction to the architecture .....	43
3.2.3. Architecture component details.....	46
<b>4. THE TRUE ROLE OF JAVA IN INFORMATION SYSTEMS.....</b>	<b>53</b>
4.1. JAVA AS A COMPETITOR TO THREE-TIER CLIENT-SERVER TECHNOLOGY .....	54
4.2. JAVA AND HTML INTRANET APPLICATIONS .....	56
4.3. A SCENARIO USING JAVA .....	58
4.3.1. Introduction .....	59
4.3.2. Application server domain .....	65
4.3.3. Object Server Domain .....	69
4.3.4. Scenario supplements .....	75
<b>5. POSITIONING OF THE MAIN EDITORS.....</b>	<b>77</b>
5.1. SYNTHESIS .....	77
5.2. PRODUCT SHEETS.....	83
5.2.1. BEA: WebLogic Application Server 4.0.2 .....	83
5.2.2. GemStone Systems: GemStone/J 3.0 .....	87
5.2.3. IBM: VisualAge for Java Enterprise Edition 2.0, WebSphere Advanced Edition 2.0.2 .....	91
5.2.4. Inprise: Inprise Application Server – JBuilder 2 for Application server.....	95
5.2.5. Oracle: JDeveloper 1.1 – Oracle Application Server 4.07 .....	99
5.2.6. Symantec: VisualCafé 3 Database Edition.....	103



---

<b>6.</b>	<b>EVALUATION OF THE MARKET LEADERS.....</b>	<b>107</b>
6.1.	EVALUATION OF WEBLOGIC APPLICATION SERVER 4.0.2 .....	107
6.1.1.	Introduction .....	107
6.1.2.	Application server.....	108
6.1.3.	Object server.....	118
6.2.	EVALUATION OF GEMSTONE/J 3.0 .....	129
6.2.1.	Introduction .....	129
6.2.2.	Application server.....	130
6.2.3.	Object server.....	137
6.3.	EVALUATION OF VISUALAGE FOR JAVA 2.0 - WEBSPHERE .....	145
6.3.1.	Presentation .....	145
6.3.2.	Application server.....	153
6.3.3.	Object server.....	160
6.4.	EVALUATION OF THE INPRISE JBUILDER 2 FOR APPLICATION SERVER .....	167
6.4.1.	Introduction .....	167
6.4.2.	Application Server .....	174
6.4.3.	Object Server .....	180
6.5.	EVALUATION OF JDEVELOPER 1.1 – OAS 4.07 .....	183
6.5.1.	Introduction .....	183
6.5.2.	Application server.....	190
6.5.3.	Object server.....	195
6.6.	EVALUATION OF SYMANTEC VISUALCAFÉ 3 .....	197
6.6.1.	Introduction .....	197
6.6.2.	Application server.....	205
6.6.3.	Object server.....	208



---

## TABLE OF DIAGRAMS

The Java Platform .....	4
Implementation of the Application.java example .....	8
A portion of the Swing class hierarchy .....	17
Delegation model used by JavaBeans .....	23
Communication between a Java applet and server objects over RMI .....	27
Interactions between a server, a client and the RMI Registry .....	29
Communication between a Java applet and server objects on IIOP .....	30
A Corba ORB must be present at each end of the communication chain.....	31
Exchanges between a Corba server and a client .....	32
Example of POST in an HTML form .....	35
EJB architecture.....	44
Interaction between Entity and Session Beans .....	50
Generic architecture of a Java-based IS .....	54
Positioning of market offerings.....	57
Thirteen steps grouped into three areas: presentation, application server and object server.....	58
JavaBean development process.....	61
Display of information in tables.....	63
Print management.....	64
Openness to distributed objects.....	65
Database access.....	66
API richness.....	67
Workload management .....	68
Modeling using business objects .....	69
EJB import .....	70
Persistence .....	71
Average per domain of the six environments evaluated .....	77
Positioning of tools tested in all three domains.....	79
Positioning of the application server and object server domains.....	81
BEA WebLogic Architecture .....	84
GemStone/J 3.0 Architecture.....	88
IBM VisualAge for Java Enterprise architecture .....	92
Inprise Architecture .....	96
Oracle Architecture .....	100
Symantec Architecture .....	104



---

# 1. Introduction

---

## 1.1. History of Java

---

It all started back in 1991 at Sun Microsystems Corporation, one of the key players in the UNIX workstation market. Patrick Naughton, an engineer at Sun, proposed a project to his friend and CEO Scott McNealy for the development of a small, simple, portable computer environment that could control all types of electronic components. Scott McNealy not only gave him carte blanche, but also gave him the support of James Gosling, one of the most distinguished system architects in the company.

The language must use the principle of a virtual machine to make it compact and portable. It must also be based on the current object-oriented language, C++, to increase its chances of appealing to the international community of developers. The project went ahead under the name "green" and the language was based on an old model of USCD Pascal, which makes it possible to generate interpretive code.

In 1992, as soon as the first development results surfaced, Sun attempted to sell products that used this technology. The first green product was an ultra-intelligent remote control with the code name "\*\*7", which, in the end, was of interest to no one. The objective of the project was then changed in 1993. The language was renamed "Oak" and was oriented towards the object model that more closely matched the culture of in-house developers than the Pascal procedural model.

The Web took flight in the summer of 1993. This tool developed at CERN completely changed the use of the Internet, the network of networks, formerly dedicated to universities and researchers. Given that Sun's project was not profitable in its initial form, it was quickly redirected towards this new means of communication. The portability and compact nature of Oak made it the perfect candidate for use on the Internet, which was well known for its slow speed and heterogeneity on connected machines. It took nearly two years to adapt Oak, and in January 1995, after long brainstorming sessions and coffee breaks, Sun renamed Oak Java. On May 23, 1995, during the SunWorld Exposition, the technological achievements of Java language were presented to the public and brought quick success. Netscape almost immediately supported this technology and found in it an excellent method for animating all of the Web's static pages by using "applets".

In 1996, Netscape 2.0 was launched and opened the door to Java development. Oracle and IBM supported the Java language. Faced with this surge in popularity, Microsoft adopted it hoping to control it and minimize its impact. Sun presented its deliverable architecture, Pico Java, added components called Beans and started a "100% JAVA" certification program validating the portability of applications.

In 1997, version 1.1 of the Java Development Kit (JDK) corrected some early problems. The OMG (Object Management Group) recognized Java that year. Sun was then recognized by the ISO (International Standard Organization) as the only supplier of JAVA specifications.

At the end of 1998, version 1.2 of the Java Development Kit (JDK) was nearly ready. It included new APIs and some real improvements to the human-computer interface. The Java language owes much of its popularity to the success of the Internet. The first developments in Java were oriented towards Web applications, which made it possible to increase the potential of HTML pages by using applets. Applets are actually software components that can be incorporated into the code of HTML pages. A ticker is a concrete example of an applet. It is used to refresh the data on an HTML page and display real-time data to the user.

However, it turns out that applets require significant bandwidth and, above all, cause portability problems with various Internet browsers. An applet developed to function on the virtual machine of Internet Explorer 3 will not function on a Windows 3.11 station, for example. Applet development is therefore only of interest for very specific Internet applications. However, Java must also be seen as a programming language used to develop truly autonomous applications or one that can be used in client-server environments.

### ➤ **Java, the language**

Java has all the characteristics of a programming language. The following are its primary characteristics:

#### *Java is an object-oriented language*

Java is a true object-oriented language. It uses the principles of abstraction, encapsulation and inheritance. It is largely based on the notion of classes and requires the developer to structure data, something that is not necessarily true in the case of C++ development.

#### *Java and multithreading*

Java language makes it very easy to develop concurrent processes, which execute simultaneously within an application. This is called multithreading. This multithread mechanism provides rich functionality to applications. A user may, for example, want to fill in a form while printing a document. Multithreading also lets Java function on multi-processor machines without too many problems adapting. Other languages provide the multithreading mechanism, but they use non-standard external libraries to accomplish this. Java integrates this capability directly into the language, which provides greater portability with Java programs.

#### *Java and dynamic linking (changing classes dynamically)*

Links are not edited in Java. Link editing is the step after compilation to link external libraries. In Java, the existence of libraries is verified during compilation, and code is loaded from these libraries when the program is executed. This decreases the size of executables and makes it possible to optimize the loading of libraries. A packaging mechanism lets the compiler and the virtual machine function. This mechanism requires the developer to structure all files in a precise tree. Any reference to a library requires that it be accessible at the time of compilation.

However, the most significant contribution of dynamic linking is that it allows the creation of applications that can be extended dynamically by simply adding new classes, without requiring possession of the source code.

### *Java manages its runtime memory*

Java has a garbage collector, which is a mechanism that clears memory of all objects that are no longer being used. In C and C++ programs, developers must handle the destruction of created objects themselves. This method of managing memory in these programs requires many lines of code and a lot of fine-tuning. The pointer concept no longer exists in Java.

### *Java is secure*

All Java applications run in a secure environment. The virtual machine performs very strict verification of Java code before it is executed. Code cannot bypass the protection mechanisms imposed by the language. A class cannot for example access a field of another class that has been declared private. The code also cannot try to define pointers to directly access memory. Applications must not cause a stack to overflow. Higher-level verifications are performed by browsers. An applet cannot, for example, access machine resources, even though version 1.1 of the JDK allows the removal of certain security barriers by providing access to the machine using signed applets.

### *Java, a simple language*

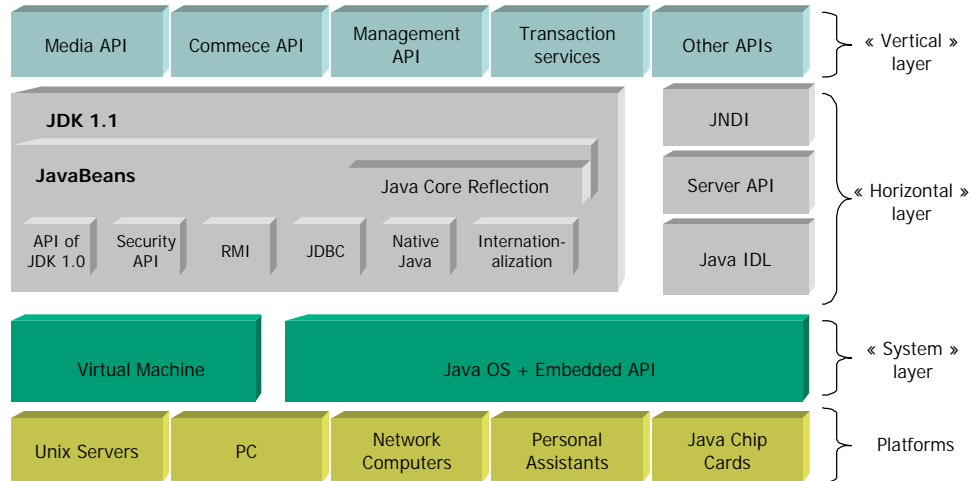
The syntax of Java language is simplified and based on C++. The absence of pointers and memory management through the garbage collector, as well as the requirement that all development must use objects, makes application code much easier to read. However, developers must nevertheless understand and integrate the hierarchy of the object model of Java APIs.

What about portability? Portability is the most argued topic when discussing Java. In reality, this characteristic is not related to the language itself, but rather to the runtime environment (the virtual machine) of Java programs. There are currently virtual machines for Unix, Windows and Macintosh platforms. A version for Linux has also been announced. Since it concerns portability, we can say that Java has taken a step forward compared with what existed before. Nevertheless, the only guarantee for portability in an application is effective testing because problems can still occur between the different platforms.

## ➤ **From language to the Java “platform”**

The Sun product distinguishes between two types of platforms. The first is the Java Base Platform, which improves as new APIs are published and integrated by Sun. The second is the Java Embedded Platform. This is a specialized embedded platform for terminals or peripherals with a minimal graphical interface (in fact, no graphical interface) and is executed using less than 0.5 MB.

The Java Base Platform is a complex construction of Java libraries and classes. The key element of this platform is the runtime environment of Java applications, the virtual machine. This is the "system" layer, which communicates with the operating system and provides application portability. The Java virtual machine is actually a runtime environment that interprets resource files (bytecode) or p-code in its own environment.



*The Java Platform*

The "horizontal" layer is composed of libraries that can be used in all traditional applications, as opposed to "vertical" libraries, which are for more specific applications. One of the most confusing characteristics of the Java language is that it evolves very quickly. A new API ("Application Programming Interface") is published practically every month. However, if we look more closely, we can see that all these developments are foreseen and that each new API integrates into the existing set.

#### *Java and Java APIs*

A distinction between the Java language and its APIs must be made. The language is the tool used to write applications or applets. The source code of a Java application can call APIs provided as standard by Sun, third party APIs or APIs developed by the developer.

To compile a Java program, the program source and the "bytecode" (compiled source) of APIs must be available. On execution, there is no difference between the code written by the programmer and the API code. Standard APIs from Sun and the language are available for free on the Internet as a batch of downloadable files known as the JDK (Java Development Toolkit). Therefore, when we talk about a version of Java (for example, 1.2), we are referring to the JDK version, or a version of the APIs.

#### *The JDK provided by Sun on the Internet includes:*

A Java compiler, an interpreter for a specific platform and a number of tools (debugger, a tool for generating documentation from the source, etc.).

Note: The version of an API can be checked using the following command:

```
| java -version
```

Java APIs are designed as "packages" (a set of classes). Standard libraries are stored in a resource file (src.jar).



Package examples:

- java.applet: contains base class of Java applets
- java.awt: contains graphics package from JDK 1.1 (graphical classes, events, etc.)
- java.beans: contains reusable components
- java.io: manages input/output flow
- java.lang: defines base types in Java language
- java.math: contains mathematic libraries
- java.net: manages networks
- java.rmi: manages communication protocol between Java objects
- java.sql: contains objects for accessing relational databases
- java.text: manages data formats (dates, etc.)
- java.util.zip: contains classes for compressing/decompressing data etc.

#### *The JDK: A brief history of the versions*

At this time, the most recent version of JDK is version 1.2. It is important to note that there have been several versions of Java language, but the two main ones are:

A pre-launch version, which was version 0.8 of JDK and a final version, version 1.0.2 of JDK. There are some important differences between the pre-launch version 0.8 and the final version 1.0.2.

The language is now stabilized. All programs written in 1.0.2 can be compiled and executed in future versions. Naturally, the APIs are richer in newer versions and upward compatibility has been provided since version 1.0.2. However, there are differences between version 1.0.2 and subsequent 1.1.x versions that restrict this upward compatibility. The primary difference between version 1.0.2 and 1.1.x is in the "AWT" package, which includes classes for programming graphical interfaces. Note that in the latest version of JDK 1.2, Sun has integrated a new API, the JFC, which significantly improves human-computer interface (HCI) possibilities and makes the AWT package obsolete.

The other essential contribution of the 1.1.x versions as compared with version 1.0.2 is a new model for managing user events. This new management method is radically different from the preceding method and the two programming models cannot be "mixed" in a given program. A choice must be made between one or the other. The java.rmi package, which makes it possible to implement inter-object communications systems in Java, must also be mentioned. Furthermore, between versions 1.0.2 and 1.1.1, method names have been changed. However, the former names are still supported.

#### *The actual process of defining APIs*

The publication of Java APIs is a process called "Sun's Open Development Process" by Sun. The idea is to make it "seem as if" these APIs followed an official standardization process (an ISO process, for example), except that Sun calls all the shots. This process includes the following steps:

1. Initialization. A small group of experts was formed within Sun and with its partners. These experts produced a first batch (only on paper) of specifications for a new API.
2. This small group of experts contacted universities and industrial experts in the technologies covered by the APIs. These discussions resulted in several successive versions.
3. The specifications issued during the preceding step are distributed to licensed Java enterprises (companies that have implemented a virtual machine) for suggestions and approval. A new version is created.
4. This version is published on the Internet so that anyone can read it and provide comments. This results in two or three new versions.
5. Up to this point, the specifications were only on paper. When the version published on the Internet is considered definitive, Sun implements (usually for Windows and its own OS) the Java classes of this API. The Internet community can then use these classes and test them to provide their comments and report bugs. After several versions (0.x) of the API, Sun considers it to be mature and either publishes it as a standard extension or integrates it directly into a version of JDK.

*A couple of comments about this process*

The specifications of these APIs are public (their documentation is readily available on the Internet) and any enterprise may implement them as long as they obtain a Java license, which means that they must respect the specifications of the API and ensure its development. In this way, Sun has implemented the Java virtual machine for Windows, while IBM implemented the Java virtual machine on its machines and operating systems. Another example is that the JDBC drivers (a Java API) were implemented by ORACLE following the public specifications of this API.

This process reinforces Sun's position and the proposed API is then adopted by consensus by the Internet community even before its definitive version is finalized. However, with this process, Java developers can accurately foresee future developments of APIs.

Everything is done over the Internet, via e-mail and newsgroups. The whole process usually takes less than one year. This is what Sun calls "Internet speed".

➤ **Principle of implementing a Java application**

There are two categories of Java programs:

- Standalone applications, which are entirely separate applications and have a graphical interface, if necessary.
- Applets, which are applications hosted on a server and loaded onto a client (Web browser) using the HTTP protocol. The browser controls their execution. The applet class of the java.applet package is the foundation for all these applications, which always use the same graphical interface.

*An example of a standalone Java application:*

Below are the contents of the Application.java file that demonstrates our example:

```
Class Person {  
    //A variable of the object instance  
    private String name;  
  
    //A manufacturer to instantiate the object  
    //constructs a Person object named theName  
    public Person (String theName) {  
        name = theName.toUpperCase();    //Formats a string in upper case  
    }  
  
    //A method of the Person class to display its characteristics  
    public void introduceYou() {  
system.out.println("My name is" + name);  
    }  
}  
  
Class Test {  
    public static void main(String args[]){  
        Person quidam; // Declaration of a Person object  
        quidam = new Person("Dupond"); //Instantiation of a Person object  
        quidam.IntroduceYou(); //Call to the IntroduceYou method of the  
        Person class  
    }  
}
```

Comments in the source are preceded by the // characters.

The Application.java file contains two classes: The Person class and the Test class. The Person class defines the data and methods of the Person object. The Test class does not define any variables. It only specifies a single process, the public main method. The instructions in this method instantiate a Person object and send the introduceYou message to the object.

The Application.java file must be compiled using the Java compiler:

```
| javac Application.java
```

Two bytecode files are created when compiled: Person.class and test.class

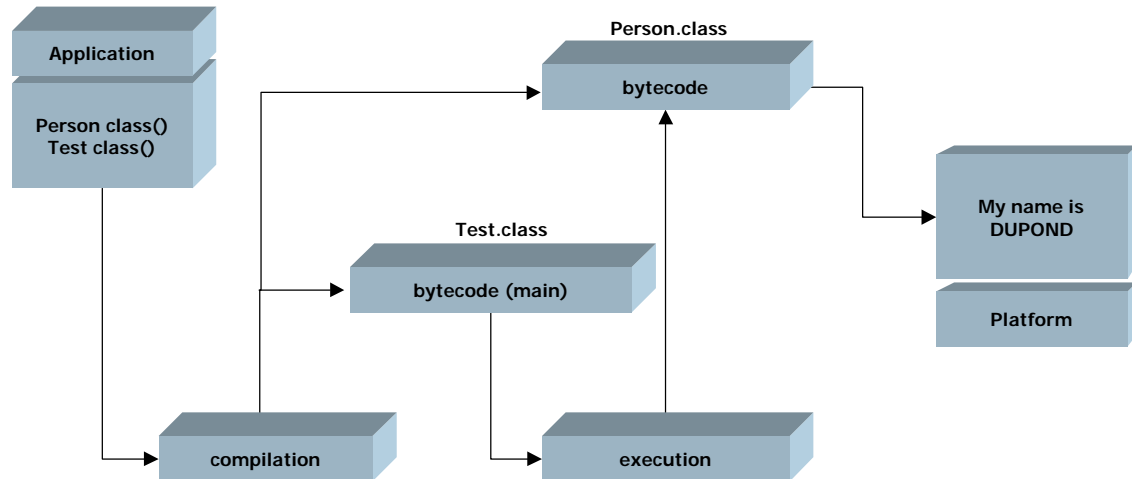
The Test.class file is interpreted by the Java virtual machine:

```
| java Test
```

The following is the result of the standard output:

```
| My name is Dupond
```

The virtual machine interpreter starts executing the "main" method of the Test class. It must have access to all the classes required for execution. In this case, this includes the Person class and the classes obtained by default on execution, such as the String class.



Implementation of the Application.java example

### What about performance?

Given that Java is an interpretive language, its performance is obviously not the same as with a compiled language. However, JIT (Just In Time) compilers can be used to accelerate the execution of code. These JITs act like memory buffers that store classes that have already been interpreted. Whenever a class that is already in memory is executed, the code is not interpreted again, but read directly from memory (machine code), which is much faster. Another option is to use native Java compilers (specific to a given platform), which generate executable files.

### A Java applet example:

Below are the contents of the myFirstApplet.java file that demonstrates our example:

```

import java.applet.* ;           //Java package required to support
applets
import java.awt.*                // Java package required to support graphic
objects

public class myFirstApplet extends Applet {

    public void paint(Graphics g){
        g.drawString("My first applet",25,50) ;
    }
}

```

To activate this applet, the myFirstApplet.java file must first be compiled and the following code must be inserted into an HTML page:

```

<APPLET code = "myFirstApplet.class" width = 150 height = 100>
</APPLET>

```

The code parameter identifies the compiled file of the Java class. The width and height parameters define the area reserved for executing the applet. The HTML page can be viewed with a browser or by using a specialized tool such as appletviewer (supplied with the JDK).

---

## 1.2. Key Java APIs

---

The Java platform has matured substantially and provides APIs in various fields. The main APIs are described below:

➤ **SDK 2 (Software Development Kit)**

*Base classes*

Base classes include the elements required for programming with Java. The following are the packages that group these classes:

- `java.applet`: contains base class of Java applets
- `java.awt`: contains graphics package from jdk1.1 (graphic classes, events, etc.)
- `java.io`: manages input/output flow
- `java.lang`: defines base types in Java language
- `java.math`: contains mathematic libraries
- `java.net`: manages networks and TCP/IP sockets
- `java.text`: manages text, numbers, dates, etc.
- `java.util`: contains various utility classes (zip compression management, random number generator, internationalization classes, etc.)

*Java Foundation Classes (Swing, Pluggable Look&feel, Accessibility, Drag & Drop)*

JFCs extend the `java.awt` API by providing a palette of graphical components that are richer and completely independent of the windows manager in the operating system, as well as being AWT compatible. The new components are named "Swing components" and their look is interchangeable on execution. There are actually 3 "looks" (Windows, Motif and metal). The `javax.Accessibility` package defines an interface that combines graphical components and assisted technologies (i.e. for people with visual difficulties). Drag/drop support has been improved to function with non-Java applications.

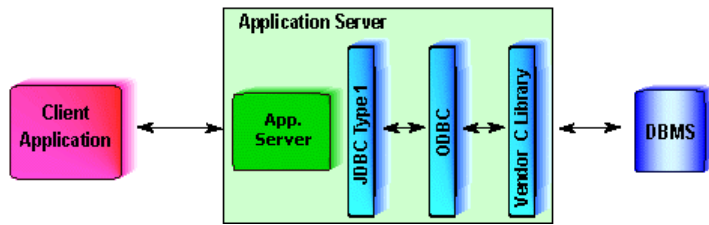
*Security (java.security)*

The security API makes it easy for developers to integrate security functions into Java programs (cryptography, authentication) using a single interface.

*JDBC (java.sql)*

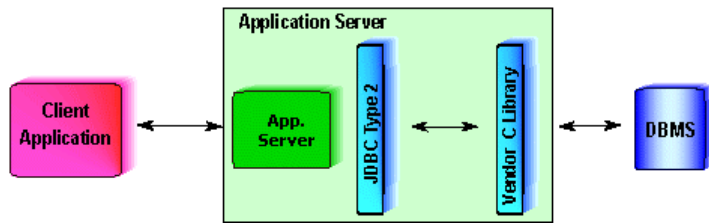
JDBC provides a unique interface for accessing relational databases. To make their data sources accessible from the Java environment, SGBD editors supply JDBC compatible drivers. There are four types of JDBC drivers:

- Type 1: bridge between JDBC and ODBC interfaces
- Type 2: driver that converts JDBC calls to calls native to DBMS clients
- Type 3: driver that converts JDBC calls to a format that is DBMS-independent and 100% Java. This format is then interpreted by an intermediate JDBC server to be transmitted to a DBMS-specific format.
- Type 4: driver that converts JDBC calls to native DBMS calls



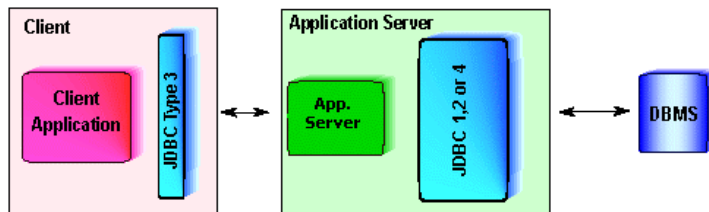
**Type 1**

A type 1 JDBC driver makes it possible to access a database via ODBC. This is an interesting solution if the database does not provide Java-specific drivers and only provides ODBC access. From the point of view of performance, this is not the best solution because the ODBC layer is usually quite slow. In addition to the JDBC driver, ODBC and the native access layer (SQL Net, for example) must be installed.



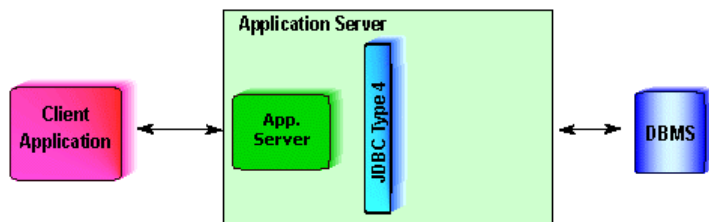
**Type 2**

A type 2 driver communicates directly with the native access layer of the database. This is currently the best solution to implement if a good type 4 driver is not available. This architecture provides good performance. In addition to the JDBC driver, the native access layer must be installed.



**Type 3**

A type 3 JDBC driver lets a client query the database using the two-tier model method. Important: The client cannot access the database directly. It must communicate with WebLogic, which relays and adds significant functionality such as the connection pool. This model is very interesting in comparison with the classic two-tier model because it does not require the use of native drivers on each client and provides advanced functionality. On the server side, the database is accessed using one of the three other models described.



**Type 4**

A type 4 JDBC driver is entirely written in Java and implements the communication protocol specific to the database. It can communicate directly with the database and does not require the presence of other software layers. This is potentially the most interesting solution in terms of ease of use and performance. Currently, the driver for Oracle is very average and a type 2 JDBC architecture is used instead. For Sybase and SQL Server, type 4 drivers are satisfactory (it is preferable to use the JConnect driver for Sybase).

*JavaBeans (java.beans)*

This API supplies classes for creating reusable and portable JavaBean components. Specifically, it makes it possible to include these components in visual development environments and provides introspection mechanisms (modification of component properties during the development phase). JavaBeans are based on the event delegation model.

*RMI (Remote Method Invocation) (java.rmi.\*)*

The RMI API provides mechanisms for invoking object methods on a remote virtual machine.

*Java 2D*

Java2D includes a set of classes for manipulating graphics, images and text in coherent models.

*Java printing API (java.awt.print)*

New in JDK 1.2, this API provides developers with a method of specifying the types of documents to be printed, define page layouts for documents, and print and control print jobs by connecting to printers defined in the operating system.

Below is an example of code that demonstrates the Java-printing API:

```
import java.awt.*;
import java.awt.event.*;

public class PrintingTest extends Frame implements ActionListener {

    PrintCanvas canvas;

    public PrintingTest() {
        super("Printing Test");
        canvas = new PrintCanvas();
        add("Center", canvas);

        Button b = new Button("Print");
        b.setActionCommand("print");
        b.addActionListener(this);
        add("South", b);

        pack();
    }

    public void actionPerformed(ActionEvent e) {
        String cmd = e.getActionCommand();
        if (cmd.equals("print")) {
            PrintJob pjob = getToolkit().getPrintJob(this,
                "Printing Test", null);

            if (pjob != null) {
                Graphics pg = pjob.getGraphics();

                if (pg != null) {
                    canvas.printAll(pg);
                    pg.dispose(); // flush page
                }
                pjob.end();
            }
        }
    }

    public static void main(String args[]) {
        PrintingTest test = new PrintingTest();
        test.show();
    }
}

class PrintCanvas extends Canvas {

    public Dimension getPreferredSize() {
        return new Dimension(100, 100);
    }

    public void paint(Graphics g) {
        Rectangle r = getBounds();
    }
}
```

```
        g.setColor(Color.yellow);  
        g.fillRect(0, 0, r.width, r.height);  
  
        g.setColor(Color.blue);  
        g.drawLine(0, 0, r.width, r.height);  
  
        g.setColor(Color.red);  
        g.drawLine(0, r.height, r.width, 0);  
    }  
}
```

## ➤ Extensions

### *InfoBus*

The Infobus provides dynamic data exchange between JavaBean components by defining the interfaces and a protocol between the JavaBeans that are to communicate. All JavaBeans that implement these interfaces can exchange structured data (values, tables, records) over the infobus on the same virtual machine. Contrary to the classic event model, an object that uses data and that is defined on the infobus does not need to know the object that will produce the information it requires. The event semantic is however weakened.

### *Java3D*

The Java3D API includes high-level classes for creating graphic 3D objects and their renderings.

### *Java Media Framework*

The Java Media Framework API includes a set of classes for managing media type contents using a single interface. It also includes methods for capturing media, as well as an interchangeable code architecture.

### *Java Cryptography (JCE)*

The JCE API is an extension of the security services provided as a standard in JDK 1.2. The JCE API cannot be exported outside North America.

The Java Serial Port, Java Management and Java Advanced Imaging API are being developed.

## ➤ Enterprise APIs

### *EJB (Enterprise JavaBeans)*

The EJB specification defines a set of APIs that facilitate the creation, management and activation of components at the server level. EJBs provide lag, transaction and naming services and help the developer focus on the industry logic of his or her components.



### *JNDI*

The JNDI API is an interface that provides a unique access interface to directory services available on the market. NDS from Novell, NIS from Sun Solaris and LDAP standard directories are accessible via JNDI. This lets Java developers use a single interface for accessing directories on the market and their services.

### *Java servlet*

This API includes a standard and multi-platform method for extending the functionality of Web servers through a servlet engine. Numerous compatible servlet engines are available on the market (Jrun by Allaire-Live Software, ServletExec by New Atlanta Communications, etc.). Servlets are an alternative to CGI interfaces.

### *Java IDL (org.omg.CORBA)*

This API includes mechanisms that ensure interoperability between Corba applications. JDK 1.2 includes an IDLtoJava compiler and a light IIOP compatible ORB.

### *JSP (Java Server Pages)*

JSP technology makes it possible to embed Java code into HTML pages (server-side scripting technique). The HTML code is called dynamically when the JSP page is called. This technique makes it possible to separate the presentation from the industry logic in HTML pages.

### *JTA (Java Transaction API)*

JTA is a high-level API used for the specification of transaction management for resource managers and transactional applications in distributed Java environments.

### *JTS (Java Transaction Service)*

JTS specifies the implementation of a transaction manager using the JTA API at a high level and Java OTS mapping from OMG at a low level.

### *JMS (Java Message Service)*

The JMS specification provides developers with a single interface for accessing asynchronous communication middleware (MOM: Message-Oriented Middleware). The framework provider supplied by Sun lets editors of this type of middleware make their solutions accessible through the JMS API.

### *JavaMail*

The JavaMail API includes a set of abstract classes that model messaging systems. By supplying their JavaMail implementations, editors make it possible to develop Java messaging applications that are independent of the messaging system used.

➤ **API for embedded computing**

*PersonalJava*

PersonalJava is an environment for network applications on home, office or mobile machines. It is composed of a virtual machine and an API adapted for machines with limited computing resources (advanced mobile telephones, digital assistants, etc.). Personal Java includes Embedded Java and additional functionality (graphic display, network connections).

*EmbeddedJava*

EmbeddedJava is an environment for applications on machines with very limited computing resources (routers, beepers, mobile telephones, printers, instrumentation equipment, etc.).

*JavaCard*

The JavaCard specification is used to run Java programs on chip cards or other units with limited memory. The coexistence of multiple Java programs on a single chip card is the basis for multi-application cards.

*JavaTV*

The JavaTV API includes a development and deployment environment for interactive Java television applications. The various modules of the API are used to manage audio/video flow, conditional access, channel management, display management, etc.

---

## 2. Java and the Client Interface

---

### 2.1. Client-Server Java

---

#### 2.1.1. AWT and Swing

---

➤ **Introduction**

To design its graphical interface, the Java developer has the standard API provided by Sun (the AWT -- Abstract Windowing Toolkit). This API includes the set of graphical components in all operating systems that support the Java platform. Numerous criticisms have been made regarding this API. The Java developer must redevelop the graphic controls in all user interfaces worthy of the name. In response to the criticisms and to develop the standard APIs in the Java platform, Javasoft has created JFCs (Java Foundation Classes) based on the IFCs (Internet Foundations Classes) from Netscape. With the Swing API, human-computer interfaces will be able to rival those on proprietary systems that host the Java platform.

The JFC product includes the following four APIs:

- The Swing API: set of "light" graphic components.
- The 2D Java API: API used to manage all types of 2D drawings, such as images, colors, fonts, etc.
- The Java Accessibility API: a set of technologies that provide access to Java applications through systems to help people with disabilities.
- The Drag & Drop API: technology that transfers data graphically between Java applications and native applications or within the same Java application.

➤ **Versions of the Swing API**

Currently, there are two versions of Swing API that can be used.

- The JDK 1.2 version, which is completely incorporated into JDK 1.2 (the release version has been available since December 1998).
- The JFC 1.1 version, which is used with the JDK 1.1.x versions (versions of the JDK higher than 1.1.2; version 1.1.7 is currently available.)

In practice, it is simpler to use JDK 1.2 version, which is fully integrated because using JFC 1.1 version requires loading the specific components of Swing and adding this API into JDK.

**Comment:**

Between versions 1.1.x and the beta versions of JDK 1.2, the names of the Swing packages have changed. Developers may encounter problems with some code samples. Therefore, they must verify the syntax of import statements in the source code header.

In the case of JFC 1.1 versions and the JDK 1.2 release version, the following syntax must be used:

```
| import javax.swing.* ;
```

In the case of the beta versions of JDK 1.2, the following syntax must be used:

```
| import com.sun.java.swing.* ;
```

To avoid problems in converting packages, Sun provides the "PackageRenamer.class" class on its site. It handles all these import statement problems based on the JDK version.

➤ **Evolution of Swing classes in comparison with the AWT**

As mentioned earlier, the AWT APIs are not adequate for developing rich graphical interfaces. The AWT makes it possible to manage a set of components, restricted to the common denominator, out of the components on the various platforms. This consequently restricts the group of reusable components. Javasoft has opted for a completely different approach by developing its Swing components based on lightweight components. This model of lightweight components was introduced with JDK 1.1 version and consists of drawing the components directly with low-level display elements instead of calling windowing systems like Windows or Motif. Therefore, Swing classes are implemented without any native methods. This approach provides, above all else, better display performance, but it also provides the very original possibility of simply and dynamically changing the "look & feel" of an application. The "look & feel" is the appearance of components (their colors and shapes) and the behavior of these components (how they react to user events).

This "look & feel" mechanism is very powerful and makes it possible to manipulate graphical user interfaces in a new way. The developer may of course select a native look for his or her application (a button would have the Windows appearance on a Windows platform and a Motif appearance on a Unix platform). The programmer may also select a "cross platform" look (neutral). His application would then have the same appearance on all target platforms (this could be reassuring for a user who is progressing on heterogeneous systems). But it is also possible to develop his own "look & feel" and personalize the aspect of all the applications of a company.

The "lightweight" component model adds other characteristics to the Swing API. The graphical components may therefore be transparent. This characteristic makes it possible to have round components by defining transparent areas, for example. The components of the AWT API are unavoidably opaque and rectangular.

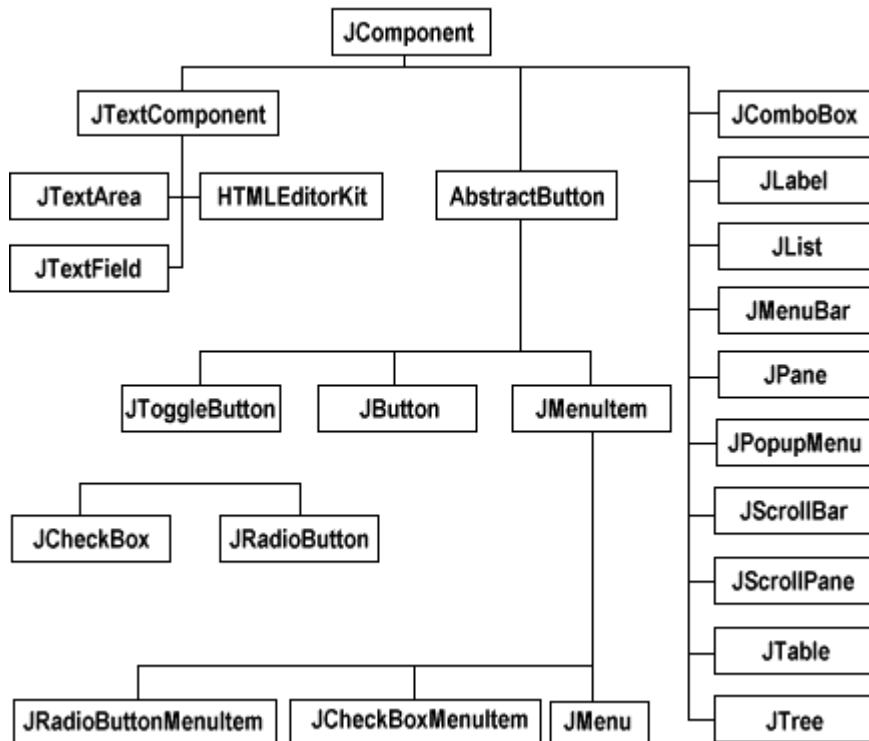
All of the Swing components are "lightweight", except for the high-level components (JWindow, JFrame, JDialog and JApplet), which serve as containers for base components.

**Comment:**

Whenever possible, mixing "lightweight" Swing components and "heavy" AWT components in the same application should be avoided because user events are not perceived in the same manner.

➤ **Hierarchy of Swing classes (extract)**

All classes of visual Swing components are prefixed with the letter J.



*A portion of the Swing class hierarchy*

➤ **Swing and the Web**

Swing components can be used in applets. For the moment, there are no Web browsers that are compatible with Swing components, and the only runtime environment for these applets is the appletviewer supplied in JKD 1.2. Sun provides the Plug-in technology to resolve this problem. After the Plug-in has been installed, applets are executed naturally, without intervention, on environment variables because the Plug-in works independently of the browser's virtual machine.

If a user loads a "Plug-in compatible" Web page containing an applet and the Plug-in is not installed on the user's workstation, a dialog box is displayed asking the user to download the Plug-in from Sun's site. For this to occur, the HTML page must be Plug-in compatible. This means that it respects a certain syntax in the HTML code.

As far as syntax is concerned, the only difference between an HTML page and a "Plug-in optimized" HTML page is the difference in the <APPLET> and </APPLET > tags. The developer has two choices:

- He may use the Java Plug-in converter tool (available on the Sun site) to generate the code of his applet.
- He may develop his HTML page himself, knowing that a distinction must be made in the code for Netscape and Internet Explorer browsers.

The <OBJECT> and </OBJECT> tags must be used for Internet Explorer.

Example:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-0805F499D93"
width="120" height="160" align="left"
codebase="http://java.sun.com/products/plugin/1.1/
jinstall-11-win32.cab
#Version=1,1,0,0">
<PARAM NAME="code" VALUE="PlugInApplet.class">
<PARAM NAME="codebase" VALUE="../assets/applets">
<PARAM NAME="type" VALUE="application/x-java-applet;version=1.1">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
No JDK 1.1 support found!
</OBJECT>
```

Whereas for a Netscape browser, the <EMBED> and </EMBED> tags must be used.

Example:

```
<EMBED type="application/x-java-applet;version=1.1"
width="120" height="160" align="left"
code="PlugInApplet.class"
codebase="../assets/applets" model="models/Dukie"
pluginspage="http://java.sun.com/products/plugin/1.1/
plugin-install.html">
<NOEMBED>
No JDK 1.1 support found!
</NOEMBED>
</EMBED>
```

But, be careful! To make sure that the HTML page that contains the Plug-in optimized applet can be read by all browsers without problems, a little trick must be used because Netscape Navigator ignores the <OBJECT> tag, but Internet Explorer does not ignore the <EMBED> tag.

The solution is the use the <COMMENT> and </COMMENT> tags which are considered as comments by Internet Explorer. The code for the Netscape browser is inserted between these two tags.

## ➤ An example of a Swing application

The example of the LookAndFeel.java source illustrates the “look & feel” principle.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LookAndFeel extends JPanel {

    static JFrame frame;

    static String metal= "look Metal";
    static String metalClassName =
"javax.swing.plaf.metal.MetalLookAndFeel";

    static String motif = "look Motif";
    static String motifClassName =
        "com.sun.java.swing.plaf.motif.MotifLookAndFeel";

    static String windows = "look Windows";
    static String windowsClassName =
        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel";

    JRadioButton metalButton, motifButton, windowsButton;

    // Manufacturer of the LookAndFeel class
    public LookAndFeel() {
        // Creation of buttons
        JButton button = new JButton("Button");
        button.setMnemonic('h'); //Exclusively for memorizing the aspect
of the button
        metalButton = new JRadioButton(metal);
        metalButton.setMnemonic('o');
        metalButton.setActionCommand(metalClassName);

        motifButton = new JRadioButton(motif);
        motifButton.setMnemonic('m');
        motifButton.setActionCommand(motifClassName);

        windowsButton = new JRadioButton(windows);
        windowsButton.setMnemonic('w');
        windowsButton.setActionCommand(windowsClassName);

        // The radio buttons are grouped to make them mutually exclusive
        ButtonGroup group = new ButtonGroup();
        group.add(metalButton);
        group.add(motifButton);
        group.add(windowsButton);

        // A Radio Buttons event Listener is added to the buttons
        RadioListener myListener = new RadioListener();
        metalButton.addActionListener(myListener);
        motifButton.addActionListener(myListener);
        windowsButton.addActionListener(myListener);

        // The buttons are added to the Frame
        add(button);
        add(metalButton);
        add(motifButton);
        add(windowsButton);
    }
}
```

```
// RadioListener code
class RadioListener implements ActionListener {
public void actionPerformed(ActionEvent e) {
    String lnfName = e.getActionCommand();

        try {
            UIManager.setLookAndFeel(lnfName);
            SwingUtilities.updateComponentTreeUI(frame);
            frame.pack();
        }
    catch (Exception exc) {
        JRadioButton button = (JRadioButton)e.getSource();
        button.setEnabled(false);
        updateState();
        System.err.println("Cannot load the Look&Feel : " +
lnfName);
    }
}

// This class handles the change in status of buttons
public void updateState() {
    String lnfName = UIManager.getLookAndFeel().getClass().getName();
    if (lnfName.indexOf(metal) >= 0) {
        metalButton.setSelected(true);
    } else if (lnfName.indexOf(windows) >= 0) {
        windowsButton.setSelected(true);
    } else if (lnfName.indexOf(motif) >= 0) {
        motifButton.setSelected(true);
    } else {
        System.err.println("The following look & Feel is unknown : " +
lnfName);
    }
}

public static void main(String s[]) {

    LookAndFeel panel = new LookAndFeel();

    frame = new JFrame("Example of Look and Feel");
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {System.exit(0);}
    });
    frame.getContentPane().add("Center", panel);
    frame.pack();
    frame.setVisible(true);
    panel.updateState();
}
}
```

To compile the class using JDK:

```
| javac LookAndFeel.java
```

To execute the program:

```
| java LookAndFeel
```



Upon execution, the program allows for the Look & Feel to be changed dynamically.

The “cross platform” or metal look:



The Windows look:



The Motif look:



A package for a MacOS Look is in the process of being validated.

### ➤ The key Swing components

Below is an overview of the primary Swing components, by type.

*High-level containers:*

- JApplet
- JDialog
- JFrame

*Intermediate or generic containers:*

- JPanel
- JScrollPane
- JsplitPane
- JTabbedPane

*Specific containers:*

- JInternalFrame
- JToolBar

*Base controls:*

- JButton
- JComboBox
- JList
- JSlider

*Menus:*

- JMenuBar
- JMenu
- JMenuItem
- JCheckBoxMenuItem
- JRadioButtonMenuItem
- JSeparator

*Controls to edit user information:*

- JLabel
- JProgressBar
- JToolTip

*Controls for changing formatted information:*

- JColorChooser
- JFileChooser
- JTable
- JTree

## 2.1.2. JavaBean components

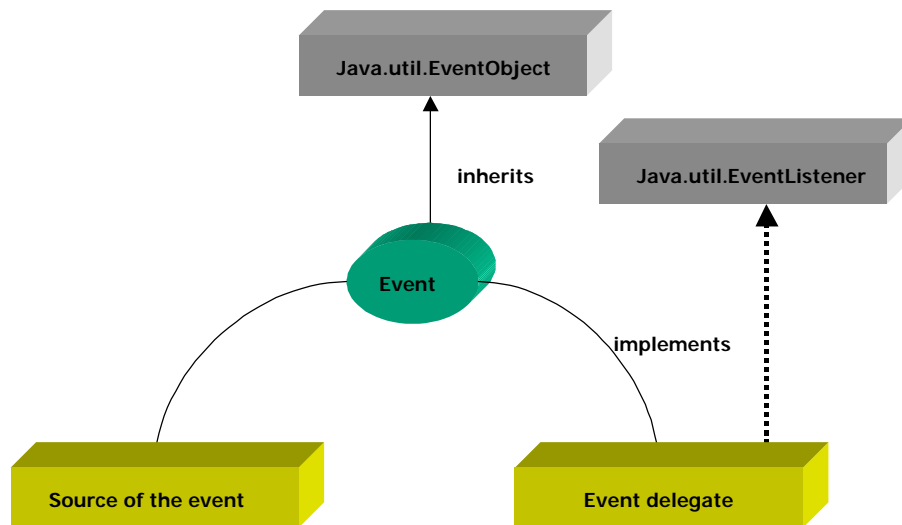
---

Through the success of quick development tools, a new concept was born: software components. They make it possible to work at a higher abstraction level because they can be manipulated using the development tools. Instead of aligning lines of code, the developer uses a predefined component, configures its behavior using an interface and integrates it into his project. It is also possible to specify interaction between various components. Microsoft provides the ActiveX component model. Sun responded by providing the JavaBean model, which was integrated into JDK 1.1. Numerous editors have integrated the JavaBean model into their development shops to increase their productivity. The characteristics of JavaBeans are described below:

### ➤ Event-driven model used by JavaBeans

Interaction between JavaBeans involves a communication model. A new event model, known as the delegation model, has been in use since JDK 1.1 and lets graphical and non-graphical JavaBeans communicate much more effectively than before.

A JavaBean 'A' affected by the activities of a JavaBean 'B' connects to certain events issued by an EventListener type of interface. We say that the management of B's events are delegated to A. When JavaBean 'B' issues an EventObject type of event that affects 'A', it passes it as a parameter to the event receiver (method) of 'A'.



*Delegation model used by JavaBeans*

### ➤ Properties and configuration

Properties are the attributes of a Bean, and access methods (`getNAME_PROPERTY()` and `setNAME_PROPERTY(value)`) are associated to each of these properties. Access to the properties of Beans via these methods can be assigned during development in the IDE or at the time of execution. This process is called configuration or customization.

Various types of properties are available:

#### *Indexed*

This parameter is used when the values of a property are to be indexed in a table.

#### *Bound*

It is often useful to inform an application or component that the value of a property has changed. Bound-type properties are able to send a `PropertyChangeEvent` type of event when a value changes. Components associated with this type of event will receive the event and can start executing a section of code.

For example: The "Traffic light" component has a "color"-bound type property. When it changes, it sends a "color-of-light-changed" event. This event is received by the "automobile" component, which is associated with it and initiates a stop action.

### *Constrained*

A JavaBean with constrained-type properties lets other JavaBeans veto changes to these properties. Components with this right to veto implement the "VetoableChangeListener" interface and throw a PropertyVetoException type of exception.

For example, the Bean Car has a constrained property named "price". The Bean "dealer" awaits changes to the price property of Bean "cars". When the price of the Bean car changes and the Bean dealer finds it too expensive, it throws a PropertyVetoException type of exception.

### *Customization using PropertyEditors*

To facilitate editing certain complex properties at development time and at runtime, the developer of Bean may use a special editor for a property (PropertyEditor). When the Bean is placed on the IDE palette and the property in question is changed, the property editor is displayed to help the developer assign the value.

For example: Imagine a URL-type property editor. It could first propose the possible protocols (HTTP, FILE, LDAP, etc.), assist in validating the URL syntax, and then suggest using the TEST button to validate the existence of the URL. When the URL is completed and validated, the editor is closed and the value of the property is updated.

## ➤ **Introspection**

Introspection is a process for analyzing methods and public members of a class. When the developer wants to provide explicit information from a portion of all methods and properties of the "MyBean" JavaBean, it writes a "MyBeanBeanInfo" class. When the JavaBean is used in an IDE, its BeanInfo class would be identified and a list of the Bean's properties would be displayed to the developer. If no BeanInfo class is found, the Bean would be probed using the reflection mechanisms supplied with the java.lang.reflect API. It must be noted that a substantial amount of information can be provided using this mechanism.

For example: A developer creates a complex Bean that digitally simulates a mechanical event. To hide its complexity, the developer creates a BeanInfo class that describes the input and output properties of the Bean, which would be the only variables seen by other developers using this component.

➤ **Persistence and Packaging**

A JavaBean should implement the `java.io.Serializable` interface. This makes it possible to save the state of the JavaBean on any medium (socket, file, database, etc.) and later retrieve it.

The JAR format described in the JavaBean specification is used to compress several files into an archive in a structured manner in order to decrease the download time on networks, particularly for applets.

## 2.2. Java Applets

---

---

### 2.2.1. Support of Browsers

---

There are two types of client workstation architectures for Java interfaces. The first type is the client-server architecture where the application used is stand-alone. To make the interface work, a Java virtual machine is required on the client workstation. Its role is to interpret the Java code. How the application works is, in this case, strictly related to the pre-installed JVM.

















The other type of application is the multi-level application in which Java applets are loaded onto HTML pages. The client is a Web browser, and the Java applets are downloaded when the first page is requested. Then, depending on the circumstances, the application continues to function as multi-level, or as bi-level if the applet communicates directly with the database or server. In either of these situations, the applet will use the Java virtual machine integrated with the browser and will therefore depend on the version provided with it.

It is possible to execute Java applets with a different Java virtual machine than the one provided with the browser. However, for this to work, a client-side "Java Plug-in" must be installed, as well as an additional Java virtual machine. This is not the best solution for non-expert users.

For browsers to support Java, two things must be understood. The first involves the platform on which the browser is hosted. If a 16-bit operation system is used, one cannot expect conclusive results. Java applets operating under Windows 3.x, for example, cannot fully exploit the potential of the JDK.

Secondly, the various versions of browsers do not implement the same versions of JDK. Here, a minimum of JDK version 1.1 is required to enable communication between client workstations and servers. In fact, the RMI API, one of the standard communication protocols between Java and server-side Java objects, has only been in JDK since version 1.1. In short, communication to sockets must be coded by the developer. This makes the development phase much more complex. Otherwise, the Java applets would be limited to display functions.

To recap, the following table presents browser support for Java when communication with an application server is required:

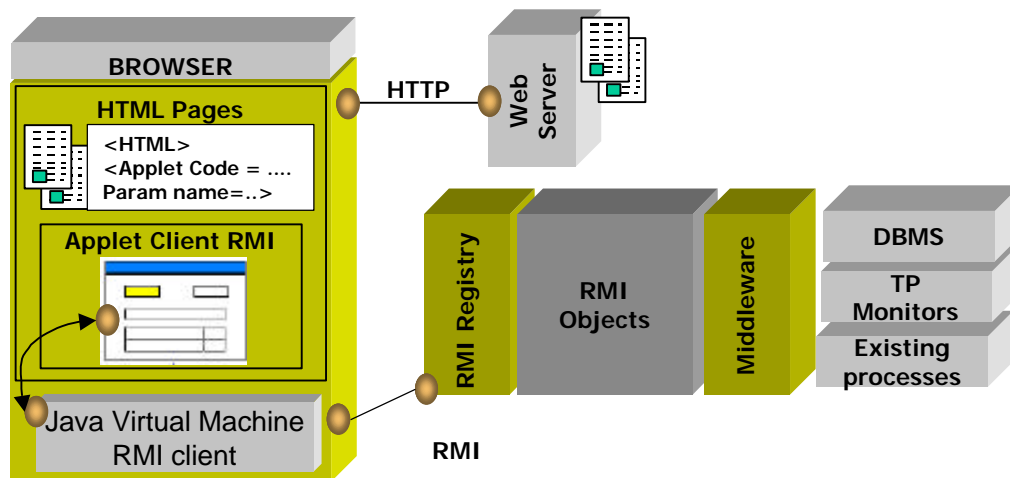
Browser - Architecture	Internet Explorer 3	Internet Explorer 4 and 5	Netscape 3	Netscape 4 and 4.5
Java RMI – 32-bit	 JDK 1.0		 JDK 1.0	
Java IIOB – 32-bit (Corba)	 JDK 1.0	 ORB Deployment	 JDK 1.0	
Java RMI – 16-bit	 JDK 1.0	 Incorrect RMI implementation	 JDK 1.0	 Incorrect AWT implementation
Java IIOB – 32-bit (Corba)	 JDK 1.0	 ?	 JDK 1.0	 Incorrect AWT implementation

### 2.2.2. Communication with the server

If the Java applet is initially integrated onto the HTML page, and then downloaded using the HTTP protocol, communication with the server is then established using various protocols. The key communication protocols are Corba/IIOB and RMI. While there are other solutions, such as direct communication via sockets, or proprietary products promoted by the editors themselves, we will simply focus on the two most widely known communication protocols, IIOB and RMI.

### Java RMI

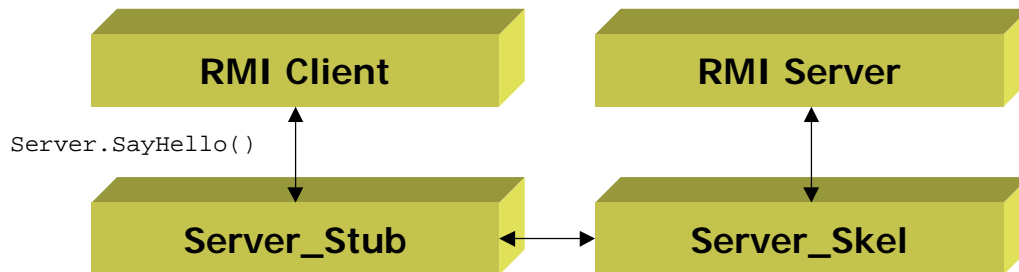
➤ **Architecture**



*Communication between a Java applet and server objects over RMI*

## ➤ Operating principles

RMI (Remote Method Invocation) provides a communication infrastructure that lets applications or Java applets communicate with each other. This middleware was created in the JavaSoft Laboratories with Sun's encouragement. Since it is fully reliant on Java, RMI provides a relatively simple implementation. The architecture is based on the principle of stubs and skeletons (already used in RPC communications). A stub loaded onto the client plays the role of server proxy. In this way, calls to remote methods are accomplished along the stub and make the server location transparent to the developer. A call to a remote method would use the same syntax as a call to a local method. The counterpart of the stub on the server side is the skeleton. Running on servers, skeletons behave as clients to the server.



Version 1.1 of JDK (Java Developer Kit), which introduces RMI, provides a mechanism that can generate these two classes from the server class programmed by the developer.

Concurrently, with this stub-skeleton mechanism, RMI uses the Java interface concept. For each RMI server, the developer must supply two classes:

The server interface, which will have a descriptive role (attributes, methods of my class, etc.).

```

public interface Hello extends java.rmi.Remote {
    String sayHello() throws java.rmi.RemoteException;
}
  
```

Implementation of the server (the mechanics, like database processing, etc.).

```

public class HelloImpl
    extends UnicastRemoteObject implements Hello
{
    private String name;
    ...
    public String sayHello() throws RemoteException {
        return "Peek-a-boo";
    }
    ...
}
  
```



The interface makes it possible to establish a service contract between the client and the server (equivalent to IDL for Corba). For the client to be in possession of the contract, RMI uses the RMIRegistry naming service. This registry meets the requirements for connecting to RMI servers that have registered themselves by returning their interface to the client. Calls are then made through this interface:

```

...
try {
Hello server = (Hello) Naming.lookup("//" + getCodeBase().getHost() +
"/HelloServer");
message = server.sayHello();
}
...

```

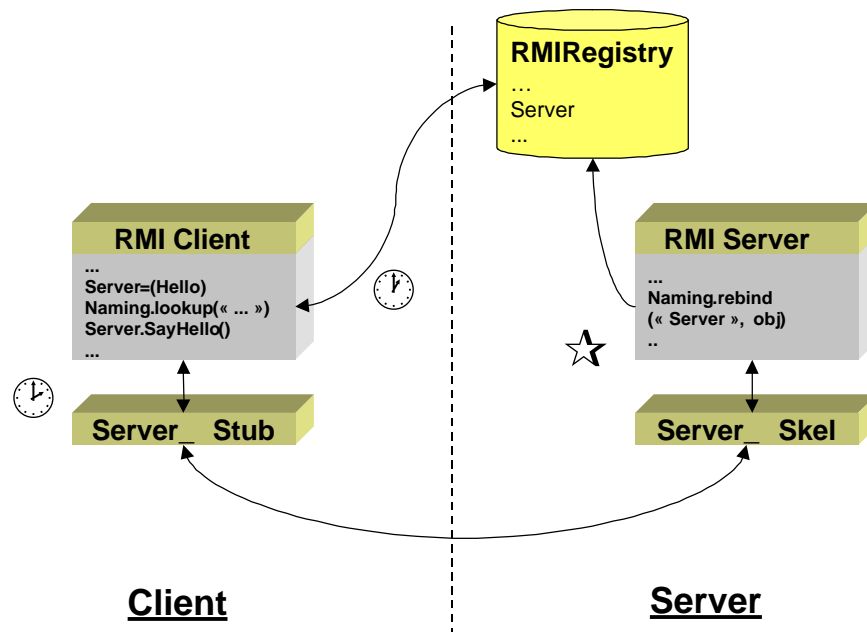
Since the stub is loaded in a fully transparent manner, communication is completed without requiring specific programming. The only point left to take care of is the registry of the service in the RMIRegistry when it is launched:

```

public static void main(String args[])
{
// A Security manager is indispensable
System.setSecurityManager(new RMISecurityManager());
try {
HelloImpl obj = new HelloImpl("HelloServer");
Naming.rebind("HelloServer", obj);
System.out.println("server registered");
}
catch (Exception e) {
}
}

```

Note: The RMIRegistry is an application that runs in memory, similar to any other executable.

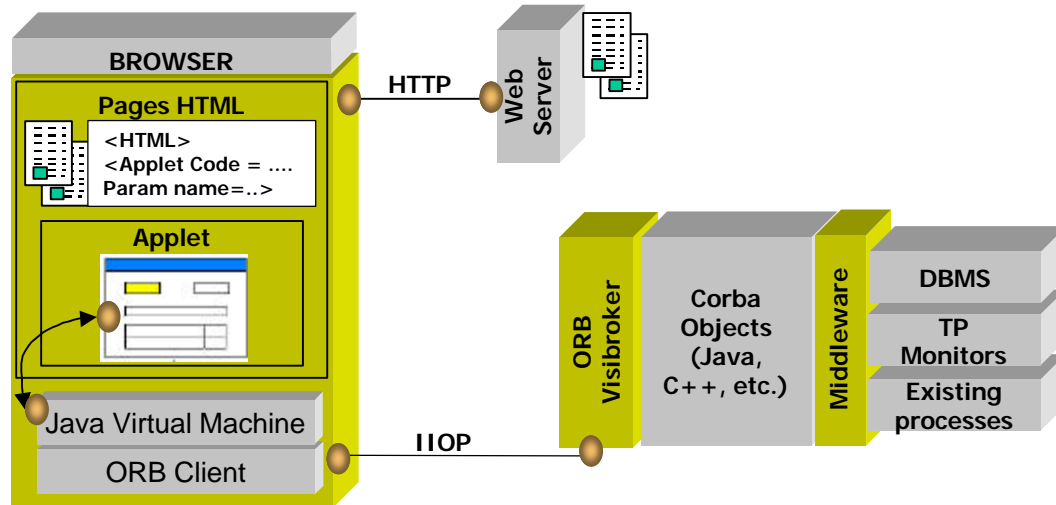


*Interactions between a server, a client and the RMI Registry*

Currently, the RMI is based on the RRL (Remote Reference Layer) to interface with transport layers. This layer could be replaced by IIOp in future versions of JDK 1.2 to provide interoperability with Corba.

## Java - IIOp (Corba)

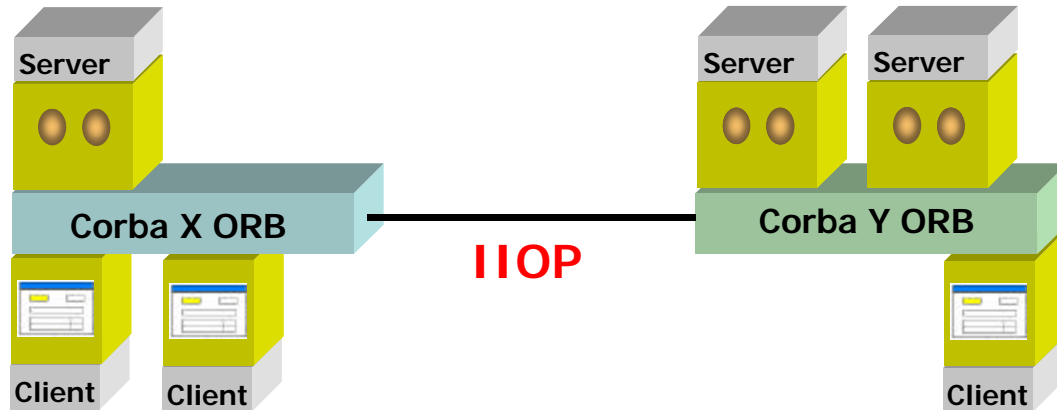
### ➤ Architecture



*Communication between a Java applet and server objects on IIOp*

### ➤ Operating principles

CORBA (Common Object Request Broker Architecture) is a specification defined by the OMG in an effort to standardize distributed object environments. While Corba exists only on paper, some editors (Iona, BEA, Visigenic - Inprise, etc.) have implemented their Corba ORBs. Since the specifications are rather vague (and even impossible to implement for some of the 15 Corba services), each of the ORBs functions internally in a proprietary manner. The role of the IIOp protocol (Internet Inter ORB Protocol) is to allow communication between the various ORBs.



## 🟡 Distributed Object Corba

*A Corba ORB must be present at each end of the communication chain*

As shown in the diagram above, each of the machines involved in a Corba application must execute an ORB. For servers, installation is accomplished as it is with any other standard software. For clients, traditional installation may be selected or the client ORB may be downloaded through an applet each time the application is started.

The communication architecture between the objects is based on the principle of "stubs and skeletons" as it is with RMI and DCOM (proxy and stub in DCOM). A stub loaded onto the client plays the role of server proxy. In this way, calls to remote methods are accomplished along the stub and make the server location transparent to the developer. A call to a remote method would use the same syntax as a call to a local method. The counterpart of the stub on the server side is the skeleton. Running on servers, skeletons behave as clients to the server.

To inform a client of the public methods and attributes of a server, Corba uses IDL (Interface Definition Language). Use of this declarative language lets clients and servers written in different languages (Java, C, SmallTalk, etc.) cooperate. As soon as the IDL object is declared, there are compilers (for example, `idl2java` for Visibroker) that automatically generate the interface according to (or based on) the language used. All that's left is to develop the implementation of the object.

An example of an IDL file:

```
interface Automobile
{
    long color();
    long price();
};
```

During implementation, a Corba server must be registered for it to be accessible. An IOR (Implementation Object Reference) will be attributed to the object. This IOR will be a unique identifier that will let it communicate with the server once it has been retrieved:

```

public class Server {
public static void main(String[] args) {
// Initializes the ORB.
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
// Initializes the BOA.
org.omg.CORBA.BOA boa = orb.BOA_init();
Automobile bargain_of_the_week = (Automobile)
    new_example_Automobile("bargain_of_the_week");
// Exports the object created and waits for requests
boa.obj_is_ready(bargain_of_the_week );
boa.impl_is_ready();
}
}

```

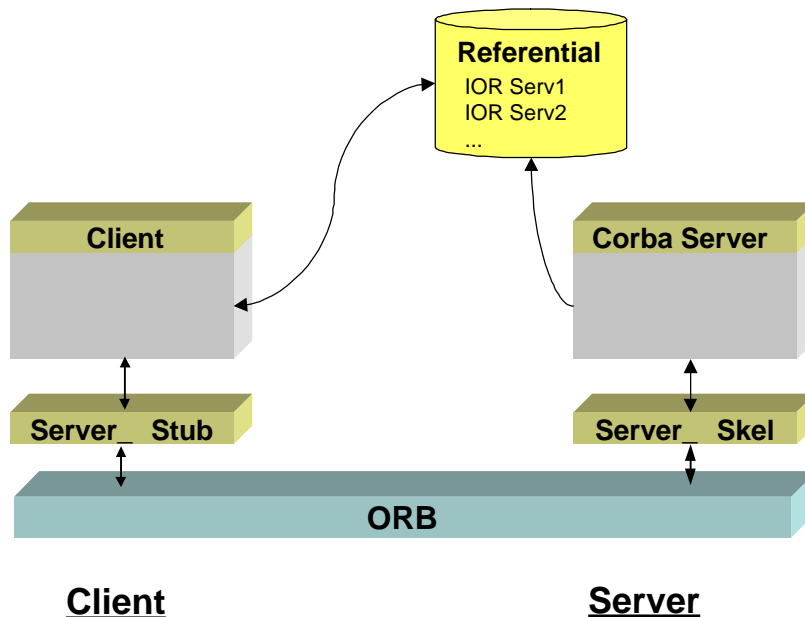
The client will then use the public methods of the object after connecting through IOR as if they belonged to a local object:

```

public class Client {
public static void main(String[] args)
{
// Initializes the ORB.
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
Automobile auto = AutomobileHelper.bind(orb,"bargain_of_the_week");
int price = auto.price();
System.out.println("The price is: " + price);
}
}

```

The diagram below illustrates the exchanges between a Corba server and a client:



*Exchanges between a Corba server and a client*

Note: Most editors offer a choice between several IOR retrieval mechanisms (bootstrapping, retrieval on an HTTP URL, etc.).

## 2.3. HTML Interface

---

### 2.3.1. Servlets

---

➤ **Definition**

A servlet is the counterpart of an applet on the server side. In other words, a Java class that runs on an HTTP server. The server instantiates the servlet, either when the server is started itself or when the servlet is first invoked. When it has been instantiated, the servlet remains in memory and is used by other clients that create the need.

➤ **Comparison between CGI and Servlet**

The CGI interface was the first solution for dynamic requests on Web sites. This type of interface always reaches its limitations. In fact, for each client request, the HTTP server must load, execute and unload the appropriate CGI module from memory. This type of architecture is becoming particularly memory-intensive when the site is nothing more than a transactional application. Furthermore, for this type of site where there are a large number of requirements for database access and user context management are important, only the CGI interface is difficult to implement.

Contrary to the CGI module, servlets are persistent, platform independent (because they are written in Java) and include a number of possibilities regarding security, data access and context management.

- **Platform independence:** This advantage is due to the Java language and its operational architecture. All a Java program needs to run is a virtual program. On the contrary, a CGI module is platform-specific, because it is generated for a given operating system.
- **Performance:** Unlike a CGI module, a servlet is only loaded into memory once, and all servlets use the same virtual machine (JVM). Obviously, servlets must be developed in the most optimized manner possible to maximize performance (for example, creating database connection(s) on initialization of the servlet).
- **Upgradability:** Due to the fact that a servlet must be written in Java (and this may be its only weak point when compared to CGI modules, which can be written in many different languages), it benefits from all the advantages of the language in terms of inheritance, polymorphism, platform-independence, etc.
- **Available Services:** Servlet servers add their own features such as connection pool management, user-session management, applicative security, etc.

➤ **What do we need to run a servlet?**

A servlet requires a Java virtual machine (JVM), and its development must respect the Java servlet API defined for interfacing with HTTP servers. For this purpose, development is done using JDK 1.1 and the Java Servlet Development Kit (JSDK 2.0), available from the Sun site. A description of the packages is also available on the Internet.

Given that servlets are instantiated by the HTTP server, a Web server that is capable of performing such a task must be used (even though many editors now provide native methods for supporting servlets). There are several options:

- Use a Web server written in Java directly, such as Java Web Server (formerly Jeeve's) or Jigsaw from W3C
- Use Plug-ins (on the server side) available in JSDK for Netscape, IIS and Apache
- Use more elaborate products provided by some editors such as: JRUN from LiveSoftware-Allaire who provides a set of classes for the following servers: Microsoft Internet Information Server 3.0 and 4.0, Microsoft Personal Web Server for Windows 95 and O'Reilly WebSite Professional. IBM WebSphere (the section for servlets was known as ServletExpress).

➤ **Servlet API**

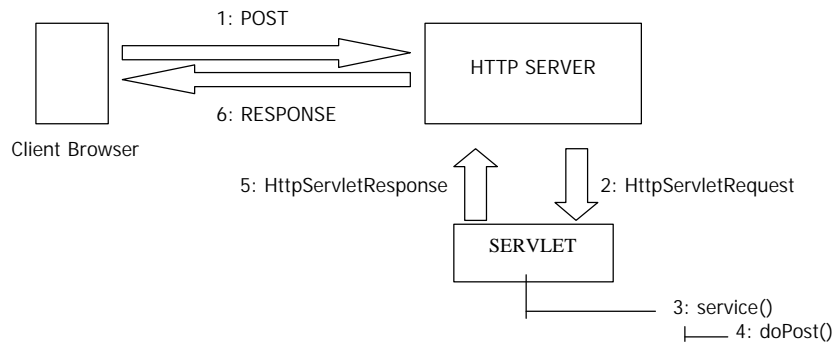
*Introduction*

A servlet is a Java class that derives from the `GenericServlet` base class or from the `HttpServlet` class when used with the HTTP protocol. In the latter case, implementation of the servlet may overload a certain number of methods:

- The "init()" method that is invoked when the servlet is loaded into memory. It is normally used to initialize global variables that may be used by all client or database connections.
- The "destroy()" method that is invoked when the servlet is removed from memory. This usually includes freeing the connections used or various resources made available to the servlet.
- The "service()" method that is called each time a client makes a request. This method is literally the heart of the servlet.

The default method invokes the appropriate `do` method, based on the request made, from the HTTP server. For a `GET` method, the `service()` method invokes the "do get()" method; for a post, it invokes the "doPost()" method. These methods may be overloaded to retrieve information from the request, process it and provide a response to the client.

Comment: If the servlet is referenced directly in the URL of a page, the method will be a `GET`. If it is in the `ACTION` property of a form, it will depend on the `METHOD=GET/POST` property.



*Example of POST in an HTML form*

1. The form is submitted to the HTTP server via a POST method
2. The HTTP server triggers the `service()` method of the servlet that is already in memory by sending it the two `HttpServletRequest` and `HttpServletResponse` objects
3. The `service()` method triggers the `doPost()` method of the servlet
4. The `doPost()` method is executed (overloaded from the servlet)
5. The `service()` method from the servlet ends possibly with return parameters
6. The HTTP server turns the response to the client browser

#### *Communication with the HTTP server*

When the server calls the `service()` method from a servlet, it passes the following two objects: `HttpServletRequest` and `HttpServletResponse`. By using these two objects, communication between the client and the server can be established.

The Request object has a certain number of methods to decode the potential parameters of the request:

- `getParameterNames()` retrieves the names of the parameters
- `getParameter()` retrieves the value of the parameter specified
- `getParameterValues()` retrieves the various values from a single parameter

For example:

The following URL is requested:

<http://www.sqli.fr/servlet/MyServlet?prenom=Alain&nom=DUPOND&qualite=ingenieur&qualite=sqlien>

getParameterNames retrieves a table that contains the names of the two parameters (first name, last name).

getParameter(name) retrieves the value of the name parameter (DUPOND).

getParameterValues(capacity) retrieves a table that contains the two values of this parameter (engineer, sqlink).

The Response object has a certain number of methods used to return the response to the client. The method of the Response getwriter() object is used to obtain a PrintWriter object. It has print() and println() methods used to write to the HTML code on the standard output device.

### ➤ Invoking a servlet

There are various ways to invoke a servlet:

- Type the URL of the class that represents the servlet directly into the address field of the browser by providing its possible parameters. For example:  
<http://www.sqli.fr/servlet/MyServlet?prenom=Alain&nom=DUPOND> ,
- Use the ACTION property from the <FORM> tag in an HTML form to trigger the servlet
- Invoke the servlet directly on an HTML page. This option is called Server Side Includes (SSI) and uses the <SERVLET> tag. At the location of the <SERVLET> tag, the Web server will invoke the corresponding Java class and will replace the tag with the result of the servlet.

### ➤ Testing a servlet

When finished, a servlet can be tested quickly using a program supplied with JSDK. This program is named "servletrunner" and can be found in the \bin directory of the JSDK installation directory.

It operates as follows (for the examples, the servlet to be tested is named TestServlet.class):

- Configure the servlet.properties file by adding a line to specify the name and class of the servlet as well as its possible initialization parameters. For example: lines from the servlet.properties file:

```
| servlet.TestServlet.code=TestServlet  
| servlet.survey.initArgs= \  
|     nbconnexions=0
```



- Start the servletrunner module in the "command prompt" window. Be careful to properly initialize all parameters when you start the module (to obtain a list of parameters, use the servletrunner -help command). For example: To start the module on port 8080:  
servletrunner \*p 8080
- Requesting that the servlet run directly within the browser, in the URL or address field, using the http://localhost:<port number>/servlet/<Name of servlet> command. For example: http://localhost:8080/servlet/TestServlet
- Put into production or debug.

### ➤ Deployment

Deployment of a servlet depends on the target, but is nevertheless quite simple. Here are two examples of deployment on an NT platform:

- With Netscape Enterprise Server: Regarding Programs / Java administration parameters, the Java interpreter must be activated and the directory in which the servlets are located must be indicated. Netscape will automatically create a "servlet alias" that points to this directory.
- With IBM WebSphere Application Server: If you want the servlet to be loaded when WebSphere is started, you must declare it using the administration applet, and then configure it to indicate how it is loaded (on request or upon launch). The default directory for the servlets is \WebSphere\Appserver\servlets. An alias that points to this directory must be created at the HTTP server level.

## 2.3.2. JSP

---

JSPs, or Java Server Pages, were introduced by Sun in 1998. The operating system is very similar to the ASPs from Microsoft (Active Server Pages), and it seems the primary difference is that VBScript has been replaced in the operating mode by Java language. JSPs make it possible to rely directly on an HTML page design template and to insert dynamic information in them from business processes.

The design logic of the application changes in regard to the development mode normally seen in traditional systems. We no longer start with applicative logic when generating HTML interface tags as output when required, but instead start from the interface, which is then filled with data from applicative processing.

### ➤ Advantages

The primary advantage to this solution is obviously ease in the development phase. In fact, a developer without much experience in object concepts could use a sequential approach in his developing by starting with his HTML interface. From a static page created elsewhere, it

would suffice to selectively execute Java class methods when the page needs to be filled with dynamic data.

One of the other advantages is the "static" approach that makes it possible to design a prototype of the HTML application and use the pages created to directly activate the application. With this solution, it becomes easy to modify the ergonomics and the image map of an application by directly updating the HTML code in JSP pages.

An example of JSP code:

```
<%@ page import = "num.myBean" %>
<jsp:useBean id="calculate" class="num.myBean" scope="request" />
<jsp:setProperty name="calculate" property="*" />

<html>
<head><title>Conversion to euros</title></head>
<body bgcolor="white">
<font size=4>

<% if (calculate.accesBean()) { %>
    You have retrieved a value of <%= calculate.obatinPriceEuro() %>
euros.<p>
...

```

### ➤ Disadvantage

This operating principle is contrary to the principles of object design and modeling. Using the interface from which you sequentially code the execution of Java code, you must respect certain rules of development to save a true object model. The temptation to include too much code in the HTML file is strong. This reduces the possibility of reusing components and makes maintenance more difficult.

### ➤ Operating principle

In fact, the operation of JSPs is based on the translation engine that generates the HTML code and Java code embedded as Java code in the form of a servlet. The virtual machine will then execute the service based on the traditional principle.

To invoke the servlet, a specific engine is required. Today, there are many solutions, most of which work on the majority of HTTP servers on the market (Internet Information Server from Microsoft, Enterprise Server from Netscape, Apache, WebSite from O'Reilly, etc.). Among the JSP engines, there are both independent products, such as JRun from LiveSoftware-Allaire, or products that are integrated into application servers, such as the JSP engine in WebSphere from IBM.

---

## 3. Java and Server Processing

---

### 3.1. Application server

---

#### 3.1.1. Java runtime environments: JVM, JIT, Hotspot

---

Initially designed in hopes of creating a universal programming language, Java was originally based on a runtime environment known as the Java virtual machine. This portability came with a loss of performance. To respond to the numerous criticisms of this weakness, Sun and the editors involved initiated changes. The first was to include JIT technology in the JDK and the second was to implement a new virtual machine, Hotspot. For their part, editors of development tools fine-tuned specific optimization techniques.

#### The Java virtual machine

The Java virtual machine was the first runtime environment for Java programs. After writing a source program with a .JAVA extension, the developer calls a compiler and produces a pseudofile compiled with a .CLASS extension. Regardless of the platform used for compiling, this file with the Java "byte code" will be interpreted by any virtual machine, which will transform this byte code into code that can be understood by the machine.

While this mechanism provides portability in compiled programs, it slows down the execution of applications by passing through numerous software layers. This slowdown resulted in much criticism and many process optimization initiatives were implemented to improve performance.

#### JIT (Just In time Compiler) technology

By converting the Java byte code into native code at runtime, JITs considerably improve the speed of Java programs. The JIT compiler has been available since JDK 1.1.6 as an upgrade. It is now part of the Java 2 platform and is called by the java.exe interpreter by using an option ("java -Djava.compiler=NONE" to deactivate JIT compilation).

By not calling the traditional Java interpreter to execute Java programs, JITs use native access libraries through the JNI (Java Native Interface) interface; this results in increased speed.

The JIT uses a special method invoker. When a method is called for the first time by the JIT, it is compiled in native code, kept in memory and its related ACC\_MACHINE\_COMPILED Boolean is set to 1. Future calls to this method will point directly to the compiled code.

## Hotspot

The second generation Java virtual machine is Hotspot, which works with JDK 1.2. Sun completely revised the internal mechanisms of the virtual machine to improve program performance while respecting the specifications of the traditional virtual machine to ensure compatibility with previous developments. Hotspot improvements include:

- detection and optimization in native code of portions of code that are crucial to programs
- speed of thread synchronization is improved
- accuracy and reliability of the garbage collector is improved

The main difference with the JITs is the optimization technique used. Rather than wasting time by compiling portions of a program that are not commonly used, Hotspot retrieves the information based on how the code is used to optimize only the "critical" portions of the code.

## Native Java applications

Some editors took initiatives to propose Java source compilation directly in native code for the development environment platform. For example, IBM provides a compiler called HPJC in VisualAge for Java Enterprise Edition and Symantec provides the ability of creating Win 32 applications in VisualCafé Professional and Database Edition.

### 3.1.2. Access to data sources and processing

---

Continuing to affirm its role on the server, Java language will become the unifying platform between mixed systems. To achieve this, Java is equipped with mechanisms for accessing source data and processing. Two categories of APIs for accessing data and processing are available to developers:

## Standard APIs from Sun

In an attempt to simplify the development of enterprise Java applications, Sun fine-tuned various APIs for accessing data and processing. These APIs define an interface rather than a complete implementation. They were designed to encourage editors to develop their implementations by providing them with the API specifications. For example, the JDBC API cannot be used to access databases, but does provide the unique knowledge required to use the API to access RDBMSs with Java. Sun developed numerous specifications. There is however a time lag between the date the APIs were made available by Sun and their implementation by editors.

- IIOP for accessing Corba objects
- JDBC for accessing relational databases
- RMI for accessing Java objects
- JMS for accessing asynchronous middleware
- JTA for accessing transactional middleware
- JNDI for accessing directories

## Proprietary APIs for accessing specific systems

Some editors did not wait for Sun's specifications for their systems and implemented specific Java APIs for accessing their data and processing. While these APIs have a longer learning curve, they have the advantage of working properly and being useful. There are however numerous cases in which editors are required to create their own implementations since Sun APIs were not planned. Here are some examples of proprietary API implementations:

- IBM CICS Gateway for Java
- IBM MQSeries Client for Java
- Domino Agent Runner (development of Notes agents in Java)
- SAP Access Builder (VisualAge Enterprise Edition)
- BEA Jolt (Java classes for accessing Tuxedo)
- Encina Access Builder (VisualAge Enterprise Edition), etc.

## 3.2. Object server (EJB)

---

The object server, which is often integrated into the application server, manages extended object functions. Its functions far surpass what is included in the basic JDK. Specifically, an enterprise object server must provide:

- A rich and flexible object model
- A high-quality development environment and documentation.
- Persistence for enterprise objects. This can be accomplished through object-relational mapping or by using a native persistence technique (object database, for example). Optimization techniques for placing the object diagram in memory must be used.
- Powerful management services for the object diagram at runtime (management of referential integrity for relationships between objects, management of the life cycle, etc.)
- Flexible, robust and scalable support. A dynamic model that does not generate stubs/skeletons is preferable. Robust management of the life cycle of distributed objects must be provided.
- A true object transactional monitor. Efficient strategies for managing concurrent data update conflicts must be used.
- There are numerous object server technologies and they are all different, but they all have the following functions as objectives. The EJB specification is an attempt to standardize these technologies. It is interesting that the model for future Corba 3 components, currently being developed at OMB, is directly inspired by the EJB specification. It is anticipated that the EJB will be the mapping of this Corba model for Java (Corba being programming language independent, its semantics must be "mapped" for each language). In this section, this EJB specification is described in detail.

### 3.2.1. Introduction of EJBs

---

In its specification version 1.0 of EJBs (Enterprise JavaBeans), Sun Microsystems describes an architecture model of server-side components. This specification is a reference document primarily for editors who want to implement and market EJB servers.

EJBs are not a product, but an architectural standard for server-side Java components. The final developer gains an advantage and he is responsible for developing the business application. He can therefore focus on writing business logic for his application, without having to worry about implementing basic services such as database access, security or transaction management.

The Sun specification describes a minimal, but essential, set of services to take into consideration. This specification does not require anything of editors for their implementations, but does recommend a set of answers to key services such as transaction management or naming.

JavaBeans and Enterprise JavaBeans should not be confused. The two terms are similar because they both describe a model of Java components.

JavaBeans primarily identify a model of components that can be configured visually and assembled in a tool using a Bean palette. This model also refers to event and customization management concepts. A JavaBean can also be an object model on the server, but without basic services (transaction, security, etc.).

The EJB model is something entirely different. It is a completely separate architecture and a set of services in which Java components can be used in a standard and portable manner. In this model, event management is not the issue. Customization is possible; however it does not intervene during the development phase, rather at the application deployment stage. The deployment process is described in the specification as a standard process. This is accomplished using a very specific "deployment descriptor" file.

It would be wrong to think that JavaBeans intervene in client-side development and that EJBs are used in server-side development. In fact, it is entirely possible to create server-side applications using JavaBean models. All that is required is to develop an architecture based on non-graphic Beans. The difference is that, in this case, the developer must completely rewrite the implementation of all required services. This is always more tedious than reusing an existing architecture.

### 3.2.2. General introduction to the architecture

---

#### ➤ The various building blocks

In its specification, Sun describes all the software building blocks involved in the definition of its architecture.

##### *The EJB server*

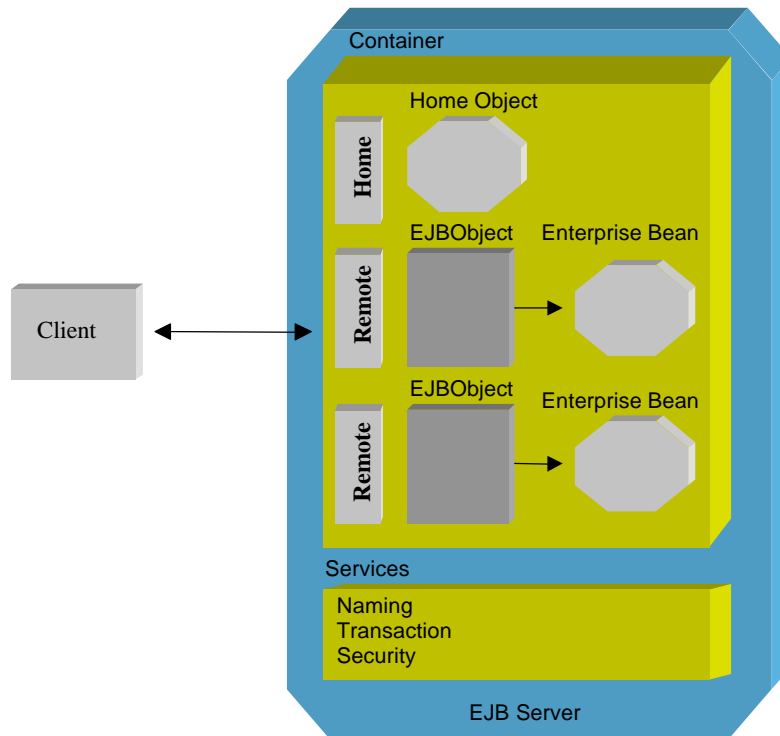
This is the highest level component in this architecture. Its primary function is to provide access to system services. The EJB server also provides characteristics that are specific to each editor, such as optimized access to a database, support for Corba or SSL services. This server must above all provide a naming service for all components accessible via the JNDI (Java Naming and Directory Interface) API. This API provides unified access to enterprise directories (LDAP for example). EJB servers are object servers. The server is the highest layer and includes all the components in the architecture. It manages and contains another EJB entity, the containers.

### Containers

Containers are entities that manage and contain the EJB components. For the moment, there is no specification to describe the interface between the EJB server and its containers. Currently, the EJB server editors provide containers, but in time, EJB containers will be interchangeable on various EJB servers and could be provided by third-party editors.

#### ➤ General diagram of components

An EJB component is a standard Java class that must implement interfaces specific to the EJB API (javax.ejb package).



*EJB architecture*

#### ➤ The Home interface

This interface of the EJB API lists all of the methods for managing EJB components. The Home Interface defines methods for locating, creating and deleting components more precisely.



### *The Home object class*

This Java class is in fact an implementation of the Home Interface. The editor and container provider must make tools for generating the Home Object class from the Home Interface available.

### *The Remote interface*

Business methods from the EJB component are declared in this specific interface.

### *The EJBObject class*

The EJBObject implements the Remote Interface. This is the object the client must access to invoke business methods of the EJB class instance. In fact, a client program would have a reference to an instance of an EJBObject and when it invokes a method, the EJBObject receives a request and delegates it to an instance of the EJB component. The container editor must also provide tools for generating the EJBObject class.

To summarize, a client is an application that uses a Home Object instance to locate, create or delete an instance of an EJB class. The client however uses an instance of the EJBObject class to invoke business methods of the EJB component.

### *Communication between objects*

The RMI (Remote Method Invocation) mechanism is a communication protocol allowing invocation between Java objects that reside on different virtual machines. A client application can be written in Java using RMI and access Home and EJBObject objects, but it can also be written in another language and use the Corba/IIOP protocol to communicate with objects on the server.

An EJB must be portable and able to run on different EJB servers.

EJBs and Corbas complement each other. EJBs orient themselves toward Corba/IIOP to provide a robust transport mechanism. Pure Corba clients could also access EJBs as EJB clients.

The specification is also open to innovations and improvements from editors. We can note, for example, the Object relational mapping mechanism in cases where a Bean uses the persistence linked to a Container.

## ➤ **The various parties involved in the EJB architecture**

The benefits of the EJB architecture can be found in complex applications. In fact, the EJB specification makes it possible to establish a role for each of the parties by assigning the right person to the task.

### *The EJB component developer*

This is a Java developer who is responsible for coding business methods. He can essentially concentrate on the functional aspects of development since all basic services are created for him. He must however have a good understanding of the transaction level he wants to give to his EJB component.

### *The EJB deployer*

This is the person responsible for installation and deployment of the EJB component on the server. He must have a good understanding of the mechanisms of the runtime environment. He must also consider the needs of the developer in terms of transaction and ensure that the EJB class is easily accessible in a name space that conforms to the JNDI API.

### *The EJB container editor*

He provides the software that lets the containers communicate with the servers. For the moment, server editors are also container editors.

The container editor must control the transactional phase, which is not the responsibility of the developer. There must be transparent support for distributed transactions. A developer can actually invoke methods on remote objects that are placed on different servers with different virtual machines. These methods must be placed in the same transactional context.

### *The application developer*

He is the one to truly benefit from this architecture. He is actually the one who writes the client program that calls the business methods from the EJB. This client can be of many different types. It could be an applet, servlet, Java application or any other application that uses the Corba/IIOP protocol to access EJB components.

All low-level services are hidden from the application developer. He can therefore concentrate solely on the functional side of the application and manipulate his data without worrying about how it can be retrieved.

## 3.2.3. Architecture component details

---

### ➤ **The Home interface**

As we have seen, this is a type of EJB component factory. When a client application wants to use a Bean, it creates one through the Home Interface. This interface can display a list of create() methods, which are the entry points for the creation of component instances.

This interface is not directly implemented by a Bean, but by intermediate classes, which are Home Objects.

A Home Object is instantiated on the server and lets the client create the EJB. The Home Object must be easy to locate. For this, a reference to this object must be placed in a naming service (or naming service) accessible by JNDI. In all cases, the location of the name space and JNDI context are supplied to the client when it is initialized. A client applet could, for example, receive the location of the name space as a parameter on the HTML page.

In the Home Interface, create() methods use the same arguments as ejbCreate() methods described in the EJB component.

The Home interface inherits from the `javax.ejb.EJBHome` class.

```
public interface ExampleHome extends EJBHome {
    public myEJBObject create() throws RemoteException;
    public myEJBObject create(String str) throws RemoteException;
}
```

where `myEJBObject` is defined as follows:

```
public interface myEJBObject extends EJBObject{...}
```

The container editor is the one who must generate the Home Object class because he alone can implement the code that creates an instance of an EJB component.

### ➤ The container

A container is not a class, but rather a service contract and it has a set of responsibilities with regard to the Bean.

Examples of services:

- Swapping to storage areas (for component persistence)
- Visibility of the Home Object class in a name space (accessible via JNDI).
- Ensure that business methods execute in the right transactional context
- Implementation of certain security services

EJB support is provided through a close relationship between the container and `EJBObject`. Both are points of entry to the Bean. The container knows, for example, the transactional context of the Bean by reading its deployment descriptor, but it is through the `EJBObject` that business methods are invoked. The container editor supplies both objects and is free to divide the work between these two entities.

There are several possible strategies. Current editors supply their own tool for reading the Home Interface and generating the Home Object class, which then behaves as a container. In this case, the editor uses a container class for each EJB class, but he is free to choose another implementation, such as a container that implements several Home Interfaces. The only requirement is that the container make the Home Object class accessible to the client via JNDI. However that may be, this implementation must be transparent to the developer of the client application.

### ➤ The EJB component

This is the class in which the "business" developer provides the functions of his application. There are two distinct types of EJB components. The developer may create a Session Bean or an Entity Bean. These two types of Beans implement different interfaces and declare their type in the deployment descriptor.

For a Session Bean:

```
| Public class myBean implements javax.ejb.SessionBean ...
```

For an Entity Bean:

```
| Public class myBean implements javax.ejb.EntityBean ...
```

Note that support for the Entity Bean by the container is optional in EJB specification 1.0 and it will become mandatory in version 2.0. Methods of the EJB component will never be invoked directly by the client, but instead indirectly by an EJBObject class, which acts as a proxy. The EJBObject is the network object, for which the stubs and skeletons of the RMI protocol are generated. The Bean is the delegated object. It contains the code for business methods.

### ➤ The remote interface

The developer must create the Remote Interface to describe the business methods of the Bean, which the client must subsequently be able to invoke. Invoking is accomplished through an EJBObject class.

For example:

```
| public interface CountCurrent extends javax.ejb.EJBObject {  
|   public void credit (duplicate amount) throws RemoteException;  
|   public void debit (duplicate amount) throws RemoteException;  
|   public duplicate balance;  
| }
```

All methods declare that they raise the Remote Exception as long as the specification states that stubs and skeletons conform to RMI. However, a different transport protocol, such as Corba IIOP, may be used.

### ➤ The EJBObject class

This is the object that is visible on the network because of its stubs and skeletons that act as proxies for the Bean. There is a customized EJBObject for each class. This class is actually an RMI server object. When the container creates an instance of an EJBObject class, it initializes it with a reference to the Bean instance. There is a one-to-one relationship between an EJBObject instance and a Bean instance. Given that the EJBObject must implement the Remote Interface, the editor of the container generates the source code of the EJBObject on installation of the Bean.

### ➤ Session Beans

The Session Bean is an EJB component for which each instance is created through the Home Interface. A classic example of the use of a Bean session is the personalized shopping cart on E-commerce Web sites.

### *Swapping Session Beans*

Some Session Beans can be swapped on a storage device. This makes it possible to manage pools of instances. Two methods are provided for this mechanism. These are `ejbActivate()` and `ejbPassivate()`. The Bean developer may use these methods to store or reinitialize values. Swapping is accomplished through serialization. Serialization is the principle of writing an object in a stream. If a container does not provide the swapping service, these methods will never be called. A passivated Bean is actually reactivated when a client invokes a business method.

### *State of a Session Bean*

The type of Session Bean must be specified in the deployment descriptor. There are two types: Stateless and Statefull Session Beans.

A Stateless Session Bean must not maintain any client-dependent information between two calls to methods. These Beans are not swapped because they do not maintain their state. A Stateless Session Bean can be shared by multiple clients. A Stateless Session Bean instance pool can be kept to save space in memory.

There must not be any `create()` methods with arguments in the home interface for Stateless Session Beans. This is an error and tools must provide a means of checking that the Home Interfaces of Stateless Session Beans do not contain `create()` methods with arguments.

## ➤ **Entity Beans**

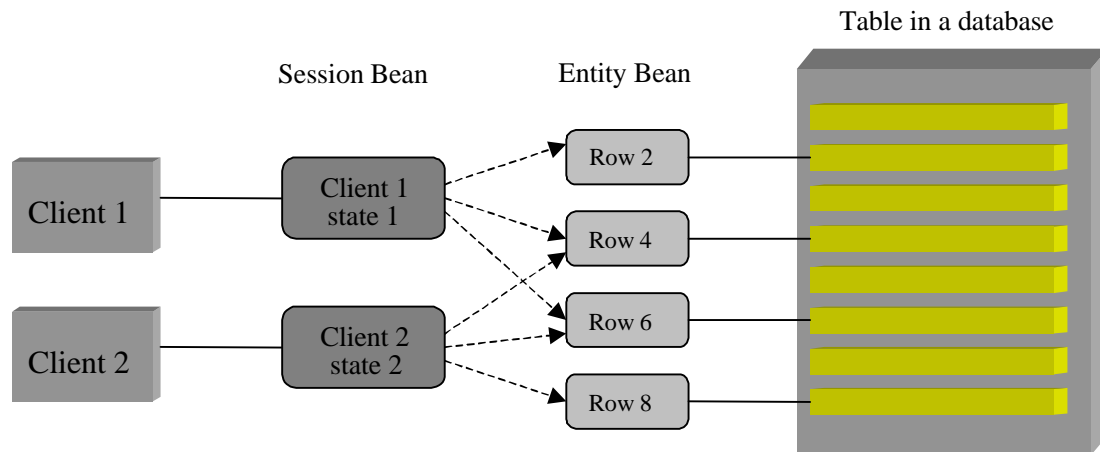
Entity Beans are used to represent basic objects. The most common application for Entity Beans is the representation of data from a database.

For example, a simple Entity Bean could be a row from a table, and each instance of that Bean would represent a specific row. A more complex Bean could be a view of a join between two tables and each instance could represent a client and the client's orders. For this type of Bean, editors are free to provide high-level services such as object-relational mapping.

### *Interaction between Entity and Session Beans*

Entity Beans are used to store rows of information. There is a one-to-one relationship between the Entity Bean and a record in the table.

Entity and Session Beans can work together. A client can call a Session Bean and this Bean then accesses the database via an Entity Bean. It is therefore possible to store information for the client (in the Session Bean) or for the database.



*Interaction between Entity and Session Beans*

### *Finder methods*

For a Session Bean, destroying an instance results in its disappearance from the container and it can no longer be used afterwards since its state is lost. For a client to access an Entity Bean, it must be able to find it. The Home interface must therefore specify search methods.

The finder method returns a unique identifier or primary key to the container. The container then uses this key to instantiate the correct row. If the *find()* method returns multiple keys, they are managed by a collection.

There is a one-to-one relationship between an Entity Bean and its associated EJBObject. The primary key is stored in the instance of the EJBObject. We therefore say it has an identity. For memory resource considerations, the container may decide to instantiate an Entity Bean only when a business method is invoked. A client may obtain the primary key at any time and can use it later to re-establish a reference to the Bean. The primary key may be of any type, as soon as the type can be serialized (to be transferred from the client to the server).

### *Swapping of Entity Beans*

The state of an Entity Bean is always synchronized with the data source. A pool of Entity Beans may be maintained. In order to avoid instantiation operations on objects, which is often costly, the Entity Beans that are no longer used are placed in this pool rather than being destroyed. They are reused more quickly this way. The characteristic of Entity Beans is that they can be persistent. There are two types of persistence. One is controlled by the Bean developer (Bean-Managed Persistence) and the other by the container (Container-Managed Persistence).

### *Bean-managed persistence*

When an Entity Bean represents a data source, it must be able to examine the data in the database. When the container associates an instance of the Entity Bean and an instance of an EJBObject class, the *ejbload()* method is called. With this method, the developer must write the code to extract the data and store it in the Bean. When the container passivates the Bean, it calls the *ejbstore()* method. This means that the data in the database must be re-written. The Bean manages all the synchronization. When the *ejbload()* method has terminated, the Bean may no longer be synchronized with the database.

The invocation of the business method used to create an EJBObject must be declared in the deployment descriptor as being part of the transaction. The container then passes the context

of the transaction to the database, and the rows in the table are blocked by the transaction, based on the transaction isolation level described in the deployment descriptor throughout the context, until it is completed. This is when the `ejbstore()` method is called. Saving the blocked rows between the `ejbload()` method call and the `ejbstore()` method call ensures that the Bean is always synchronized with the database. The client or the deployment descriptor may determine the duration of the transaction.

#### *Container-managed persistence*

If the deployment descriptor declares that the Bean uses persistence management controlled by the container, the `ejbload()` and `ejbstore()` methods are no longer used to access the database. The container loads the data into the Bean and invokes the `ejbload()` method to notify the Bean that it has just received data. The same is true for the `ejbstore()` method. This is the level at which editors can propose Object-Relational mapping mechanism. The specification from Sun describes a minimal set of instructions for editors who want to provide Container Managed Persistence. The deployment descriptor can specify simple mapping of public fields in Bean in the columns of a table. If the editor does not have sophisticated object-relational mapping, the developer will have to manage persistence himself.

#### ➤ **The deployment descriptor**

The deployment of EJB components is described in the Sun specification as a standard and portable process, even if the EJB platforms are not the same.

The deployment descriptor describes a set of parameters required for deployment. The developer is not expected to know them (for example, the type of drive or address of the machine), but its fields can be populated by the administrator. The deployment descriptor also describes the behavior of the Bean in a transactional environment, as well as security parameters.

Technically, the deployment descriptor resembles a standard Java class. An instance is created, populated with values and then serialized. The serialized class is then placed in a jar file and sent to the deployment zone with Bean classes. The jar file is then imported into the EJB server to install the components.

For generating the deployment descriptor, editors are completely free in that they can provide a configuration in a text file or with a more advanced graphic tool.

#### ➤ **The jar file**

The jar format is a standard format for the deployment of Java classes. In the case of EJBs, this file contains the following elements:

- The classes of all EJB components.
- The interfaces.
- The serialized deployment descriptor.
- A Manifest file.

The Manifest file is a text file that contains instructions for jar. It contains the description of all files in the jar and their characteristics. The deployment descriptor must be declared as an Enterprise JavaBean in the file.





## 4. The True Role of Java in Information Systems

With the arrival of each new technology, the world of professional computing is systematically caught up in the same frenzy known as "hype". In the past, we imagined that artificial intelligence would revolutionize management applications and that the client-server would be the universal solution for solving all problems. A few years later, the dust settles and the true position of each technology is pragmatically identified (in this case, the proverb "you learn by making mistakes" truly applies).

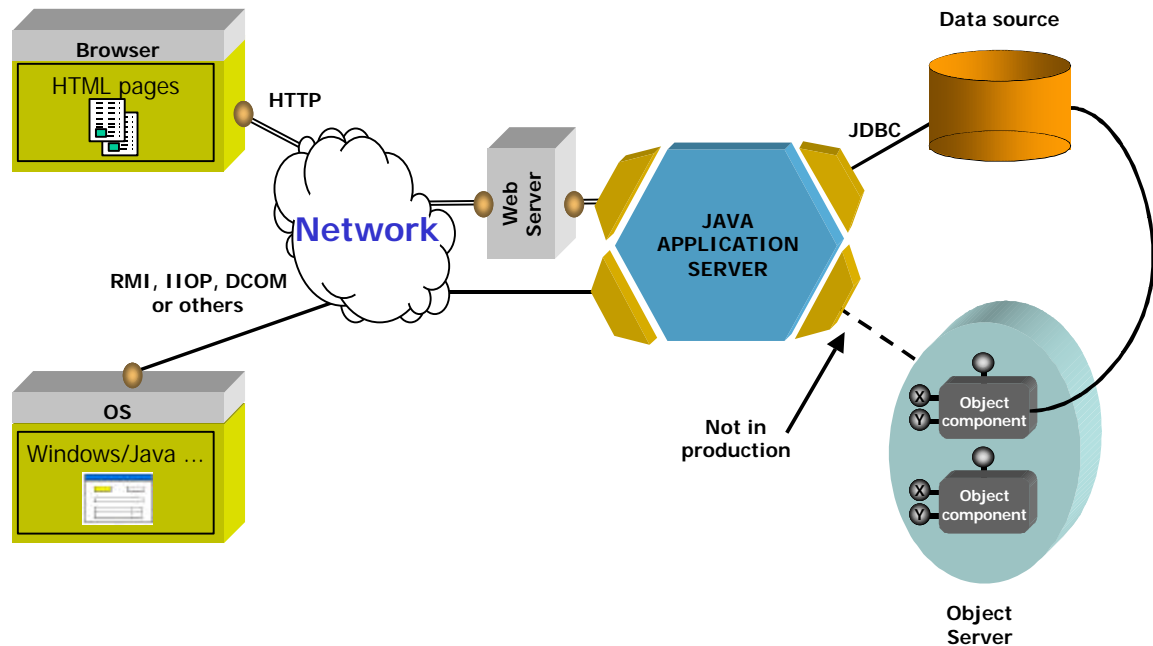
This phenomenon occurred with Java. If we go back a few years, Java started off on the wrong foot with applets. Paradoxically, this use of Java contributed significantly to its distribution, but very nearly killed this "language that was so much more". In fact, the applet route for creating transactional applications on the Internet very quickly suffered from deployment problems (compatibility with browser JVMs), speed concerns and limitations with AWT graphic objects, difficulties in controlling network use for non-experts, etc. However, the reproduction of event-driven interfaces in browsers is a source of confusion for users. They more easily adopt the HTML navigational interface that makes management applications as "easy to navigate" as Internet sites. For all these reasons, we recommend using applets in exceptional cases only. These could include cosmetic applets (pie charts, graphs, etc.) or for push technology in which data needs to be received. Under no circumstances should complete applications be created as applets.

Today, the Java market has matured and the technical nature has been improved with Java 2. We believe the two key areas in which Java is used are the following:

- Java as a competitor to three-tier C/S
- HTML applications using Java on the server side

However, as demonstrated in the report, we believe that EJB server object technologies are not yet mature enough to be adopted for production. Furthermore, this market is filled with takeovers and mergers making durability studies very unreliable.

## 4.1. Java as a competitor to three-tier client-server technology



*Generic architecture of a Java-based IS*

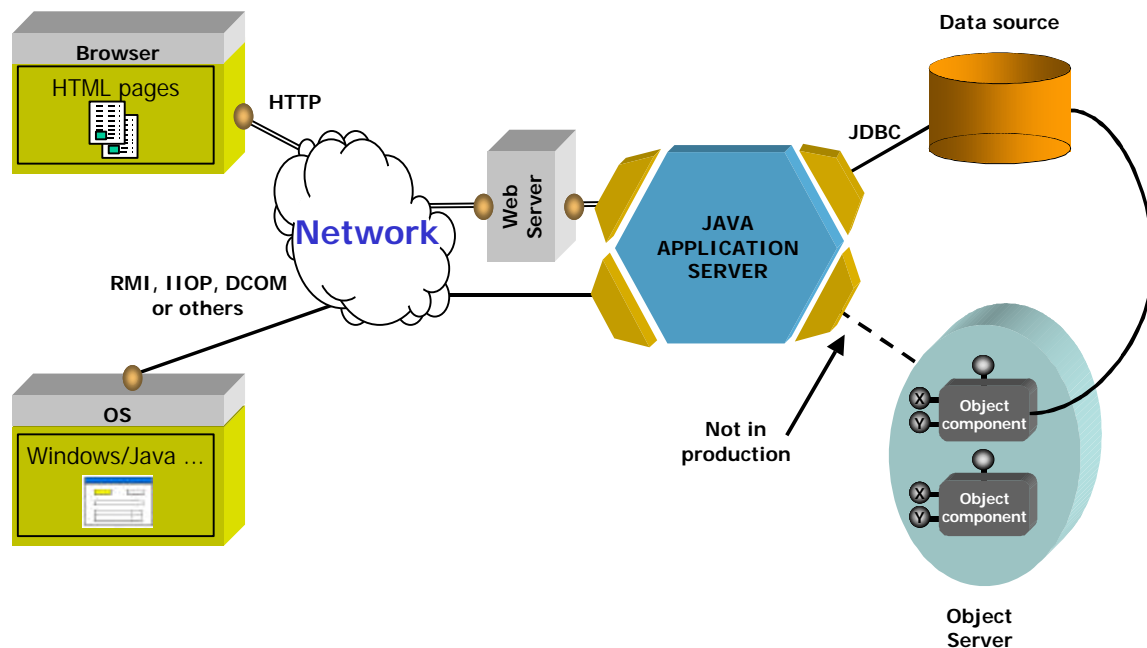
The three-tier client-server concept came into being just before Internet technologies were used to create management applications. Since there is no standardization, clients abruptly dropped this architecture, since they felt that they would be tied for life to an editor while Internet standards provided some autonomy regarding editors. This market has been a niche market for a considerable length of time with players like Forté, Dynasty, Prolifics, etc.

Java has just recently positioned itself as the standard for three-tier client-server applications. This position is founded on several points:

- The JFC/Swing graphical interface brings a level of IHM that is very close to traditional C/S development tools.
- Deployment on client workstations consists of installing the right version of the JVM (no more concern about JVMs imposed by browsers). There is no configuration phase.
- There are now various solutions used to accelerate Java processes (JIT and HotSpot compilers, native compilation, etc.).
- Java provides a certain measure of autonomy regarding suppliers. Important: Much more should be expected in terms of skill portability rather than application portability.

- The fundamental object approach of Java and the availability of communication protocols such as RMI and IIOP match the requirements of three-tier architectures. The application server will be the runtime environment for server processing.
- The movement of the entire computer industry towards Java means that an increasing number of data sources (mainframe, software programs, etc.) will be easily accessible with Java. Java three-tier client-server applications will then be able to hide the complexity of the information system and focus on interfacing with data sources on the application server.
- By adopting this architecture, it will be possible to use the same server processing for event-driven interfaces (JFC/Swing) and navigational interfaces (HTML). This makes it possible to provide solutions for various use requirements for the same application.

## 4.2. Java and HTML intranet applications



The second major application for Java is to leave Java on the server side and produce HTML from servlets, Java Server Pages or via application server engines. The initial "love at first sight" between Java and HTML was delayed, but is now extremely promising and most Web development tools are being created with this architecture in mind.

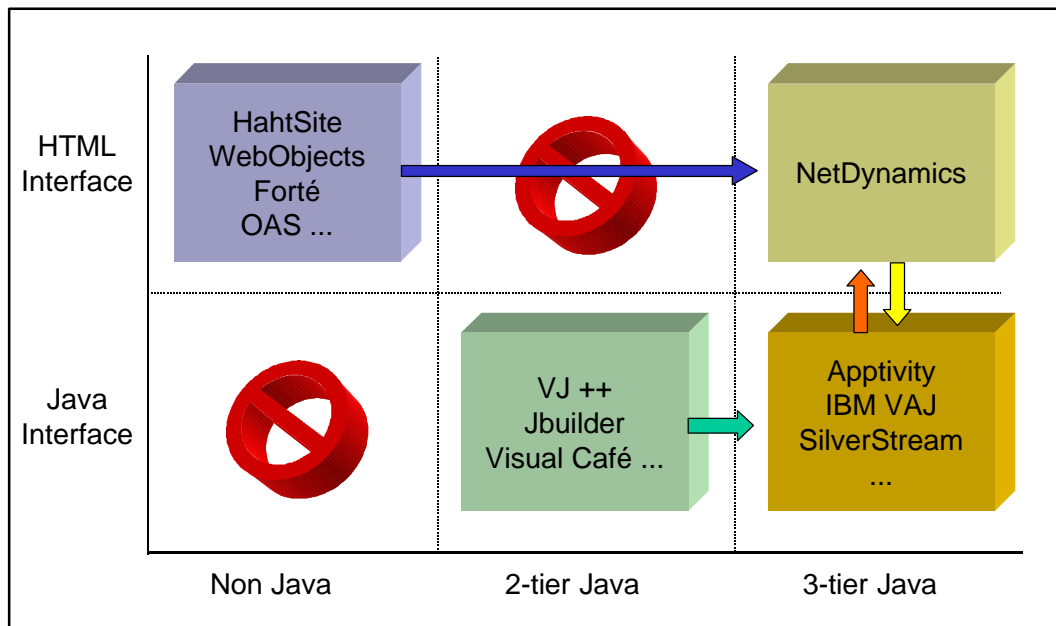
Most of the advantages are mentioned in the preceding chapter. Added to these are the qualities incorporated through the HTML/HTTP pair (no deployment or maintenance constraints, quick adoption by users, limited network costs, etc.).

The pioneer in this market was NetDynamics. Recently, however, numerous editors have extended their products to Java on the server side and HTML on the client side. These editors are from various sources:

- HTML application servers whose technology is not based on Java are increasingly integrating Java as a development language. By doing this, they let the developer decide whether he wants to use the native language or Java. For example, Apple WebObjects makes it possible to develop in Objective C or Java. HahtSite lets developers choose between HahtTalk and Java, etc.
- Three-tier Java application servers with a Java interface have all switched to an HTML interface. Some have even renounced their origin, as is the case with SilverStream. They did not hesitate to announce that Java applets were dead. This editor's product is now oriented toward HTML on the client workstation. IBM with VisualAge for Java and Progress with Aptivity, etc. all took the same path.

- Most editors of two-tier Java development tools with a Java interface are extending their products toward three-tier Java, which is a necessary step before they can consider using HTML interfaces. Some, such as Inprise or Oracle, are already providing solutions as shown in this report.

Only the evolution of NetDynamics is atypical because it has been in this market from the beginning. It extends its possibilities toward managing Java interfaces to provide two types of interfaces for the same processes.

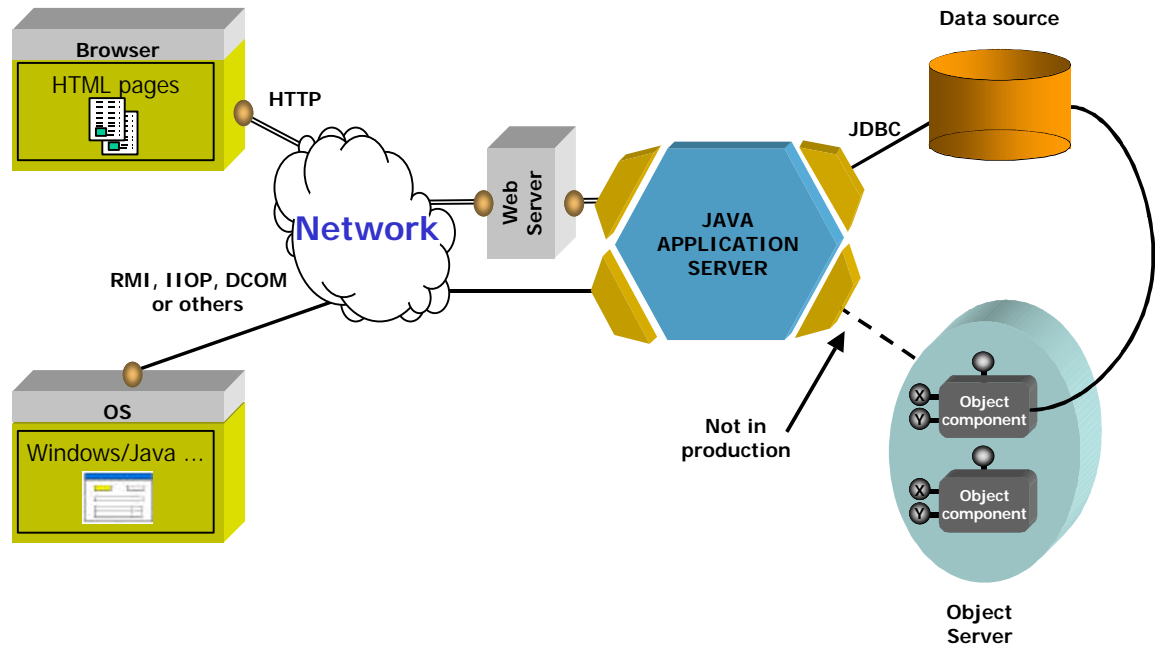


*Positioning of market offerings*

In conclusion, the server-side Java/client-side HTML segment of the market is growing very quickly. This will make it possible to implement applications with a high return on investment for companies. Many players naturally target this market, and the war is raging on. This means that an application server should be chosen with care and an eye should be kept on durability. It should also not be forgotten that non-Javan HTML interfaces that have not made the move toward Java (Allaire ColdFusion, Microsoft ASP, etc.) are competitors to these products. A selection adapted to the context of the company can only be selected with precise criteria.

## 4.3. A scenario using Java

The scenario discussed below has two objectives: to test the various Java tools by using concrete user problems and to highlight the role of Java on each of the levels of an architecture. This scenario is divided into 13 steps. These steps are grouped into three areas that represent the requirements for presentation, application server and object server.



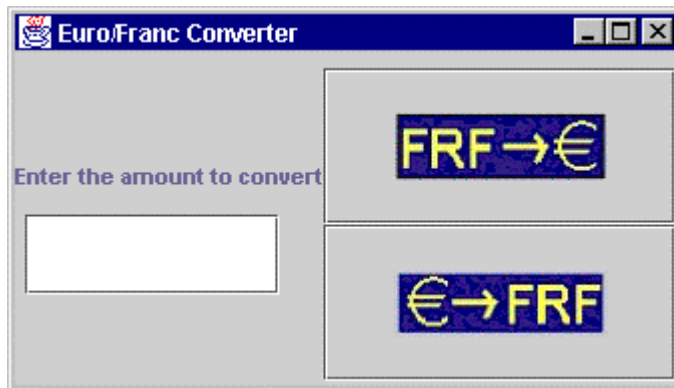
*Thirteen steps grouped into three areas: presentation, application server and object server*

The main thread of our scenario is the creation of functions in a car-order application. Our fictitious salesman needs to manage two interfaces: an event-driven interface for his coworkers and an HTML interface for his e-commerce site. Billing can be done in Francs or Euros. Let's begin...

### 4.3.1. Introduction

---

#### Step 1: Event-driven Interface



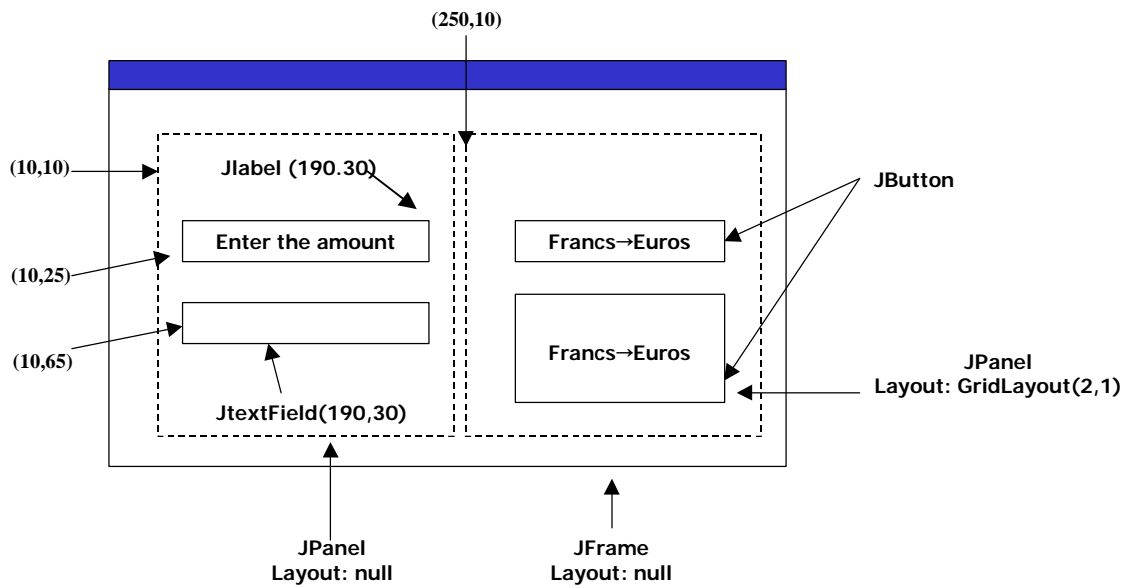
*The Euro/Franc converter*

#### General description

To help the salesman's coworkers, who are responsible for billing, we want to develop a Euro/Franc converter. To limit manipulation errors, the conversion buttons must include images, which represent the conversion direction.

#### Details of the process

This simple scenario will help us evaluate the ability of tools for developing Java interfaces. Today, the library of standard graphic objects has been improved through the support of Swing components. These graphic components make it possible to provide a high-quality interface to the user comparable with those using traditional client-server tools (VB, Delphi, PB, etc.). Java tools with a GUI Builder will let us create our Euro/Franc converter based on the following specifications:



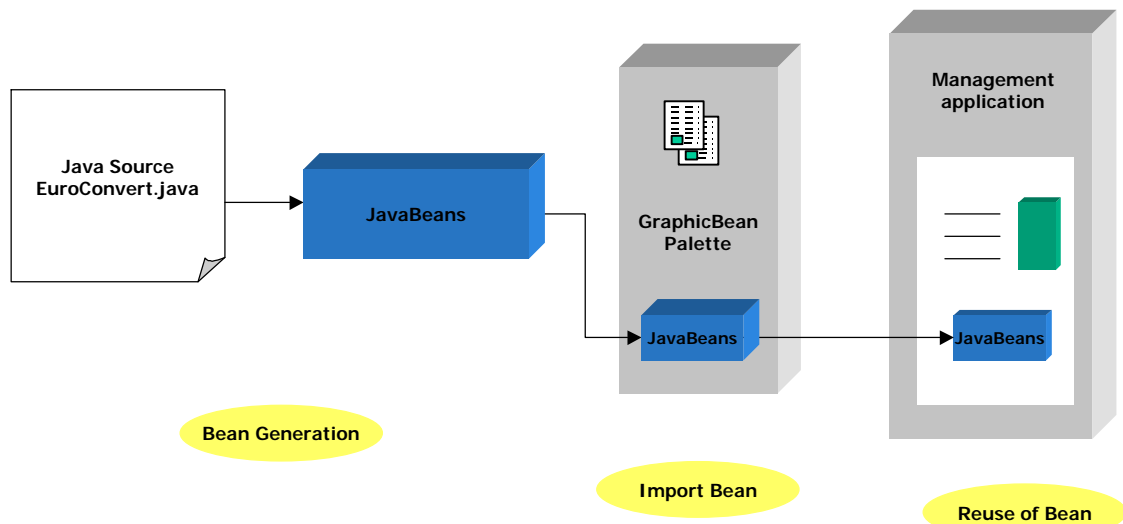
## Key points

The key points that will help us evaluate the potential of the tool in terms of the event-driven interface are:

- Swing components supported
- User-friendliness and flexibility in formatting the graphic components (alignment, etc.)
- Assistance with managing graphic events
- Management of open, close and maximize window events



## Step 2: Creating graphic components



*JavaBean development process*

## General description

Our Euro/Franc converter will be useful in numerous applications. Therefore, we want to create a component that can easily be integrated into any window. We want to be able to modify the Euro/Franc exchange rate should it change.

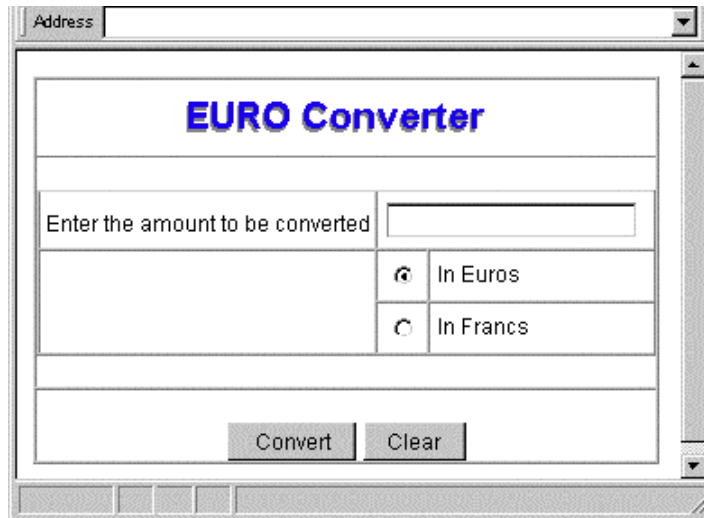
## Details of the process

The tool must generate a JavaBean graphic component based on our Java class. Once the Bean is created, it can be imported into a graphics object palette. This palette must let the Euro/Franc converter be inserted and thus re-used in any window. The tool must offer a method of editing the component's attributes using the windows editor (the exchange rate, in our example).

## Key points

- Assistance in creating the JavaBean
- Integration of new graphic components into a palette
- Bean introspection

## Step 3: HTML interface



### General Description

Currency conversion must be offered as a service on the e-commerce site of our car salesman. Our converter will therefore have an HTML interface and the conversion will be carried out on the server to control, for example, the validity of the exchange rate.

### Details of the process

In this scenario, the tool will be evaluated on its ability to generate an HTML interface. The structure on the following page must implement a highly evolved HTML presentation (with nested tables and cell merges to polish the presentation). The server can carry out the calculation in different ways: either using the servlet technique, server-side scripting or executing a compiled module. The result is displayed on a simple HTML page. The result page includes a link allowing the user to return to the conversion page where the values previously entered by the user are displayed. We will explain the possibilities of context management provided by the tool in detail (server- or client-side management, automatic identifier generation, timeout, etc.).

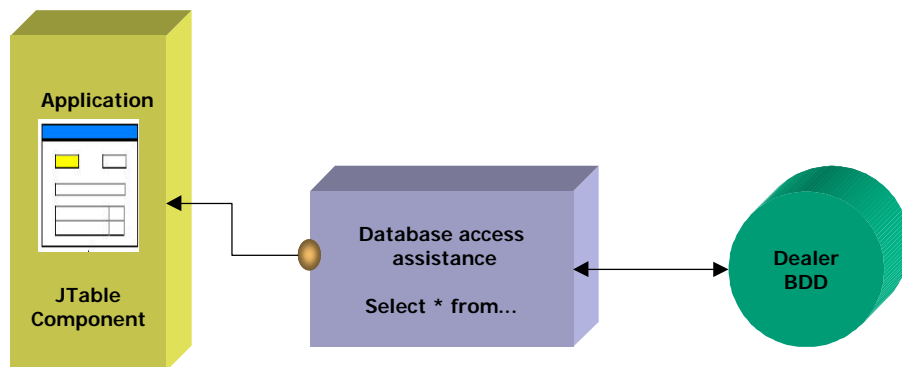
### Details of the HTML presentation:

The interface is based on the nesting of two tables. The main table (3 rows and one column) contains the image "title.gif" in the first row. The second row contains the second table. The third row contains the two form buttons. The second table (3 rows, 3 columns) uses cell merges and contains two radio buttons, one of which is selected by default.

### Key points

- Possible to automatically generate HTML
- HTML code can be easily edited
- User-context management

## Step 4: Displaying information in table form



*Display of information in tables*

### General Description

In this step, we wish to display the list of cars in table form in the event-driven interface used by the salesmen. Each line will include the car model, the price in Euros and the number of cars currently available.

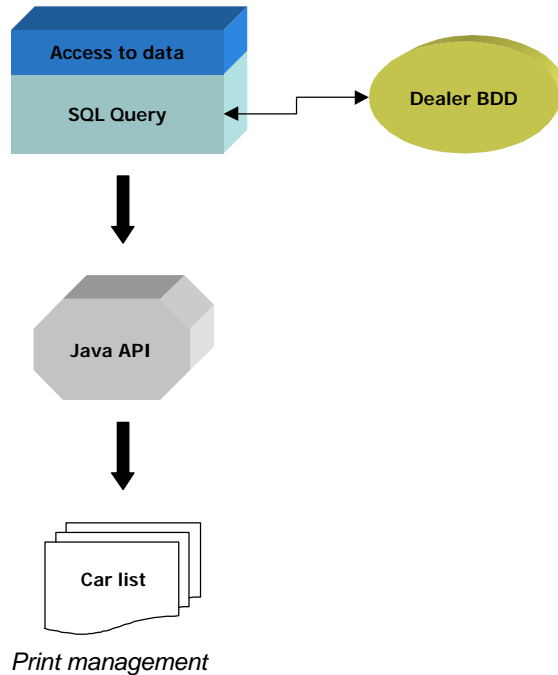
### Details of the process

We study the possibility of graphic access to data sources (databases, mainframes, etc.) and evaluate the ability to automatically generate a graphical interface based on a query with joins. The table can be the JTable (Swing) component or a proprietary graphic component. There are almost endless processing possibilities (selection by rows, moving columns, etc.).

### Key points

- Automatic generation of user interface from a SQL query
- The richness of graphic objects
- Quality of assistants (operating in "both directions", simplicity, etc.)

## Step 5: Print management



### General description

Both the event-driven and HTML applications must make it possible to print out the list of cars. The format used on the e-commerce site is .PDF.

### Details of the process

Print management is an important requirement for all applications. This step will enable us to see whether the tools tested facilitate the implementation of printing functions. The goal is two-fold:

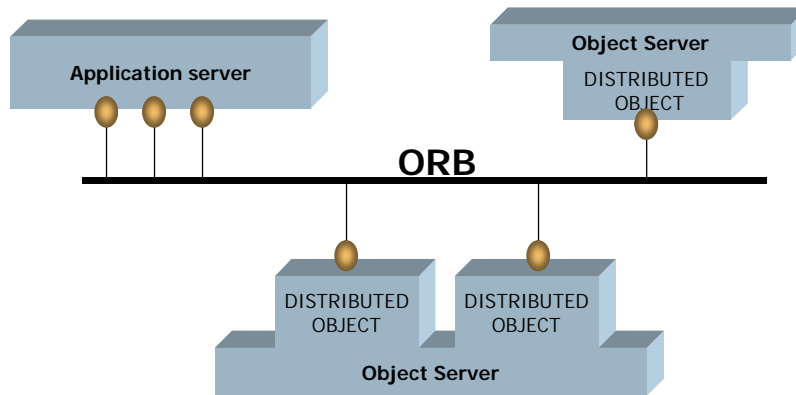
- Print a quality hard copy from a Java interface
- Generate a .PDF document on the server and make it available to the client

### Key points

- Printing from a Java interface
- Generating PDF
- Quality of report writers (conditions, page headers and footers, page breaks, etc.)

### 4.3.2. Application server domain

#### Step 6: Openness to distributed objects



*Openness to distributed objects*

#### General Description

The existing information system already includes an application that allows the bargain of the week to be distributed (in this case, the car that the salesman wants to sell as quickly as possible). This distribution is done through a distributed object. We wish to retrieve this information into our application in order to distribute it to the salesmen and their customers.

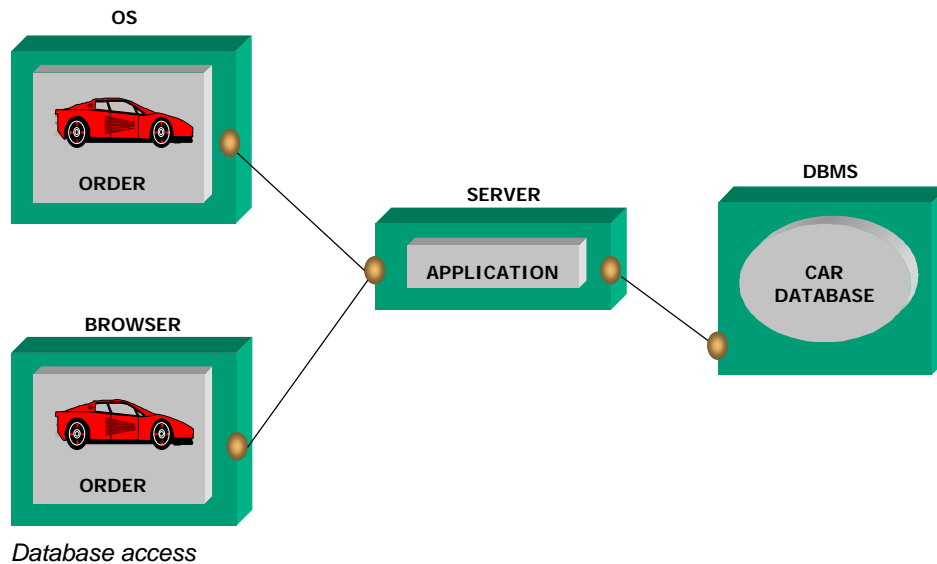
#### Details of the process

This step enables us to evaluate the capabilities of the tool to be the client of a distributed objects system. The distributed objects system is voluntarily omitted because we will also judge the diversity of interfaces provided with such systems (Corba, EJB, DCOM, etc.). We also observe all the capabilities found in the development tool, such as the option of identifying the methods of the remote object (introspection) and the presence of client-creation assistants of this object.

#### Key points

- Diversity of connection possibilities
- Introspection
- Quality of assistants

## Step 7: Database access



## General Description

Both the event-driven application and e-commerce site must make it possible for users to search for the car that corresponds to their chosen criteria. To do this, an index card-style interface is preferred with the navigation options (next, previous, home and end) in the search results. The customer or salesman must be able to place a direct order for a car. In this case, the application will manage stock reduction.

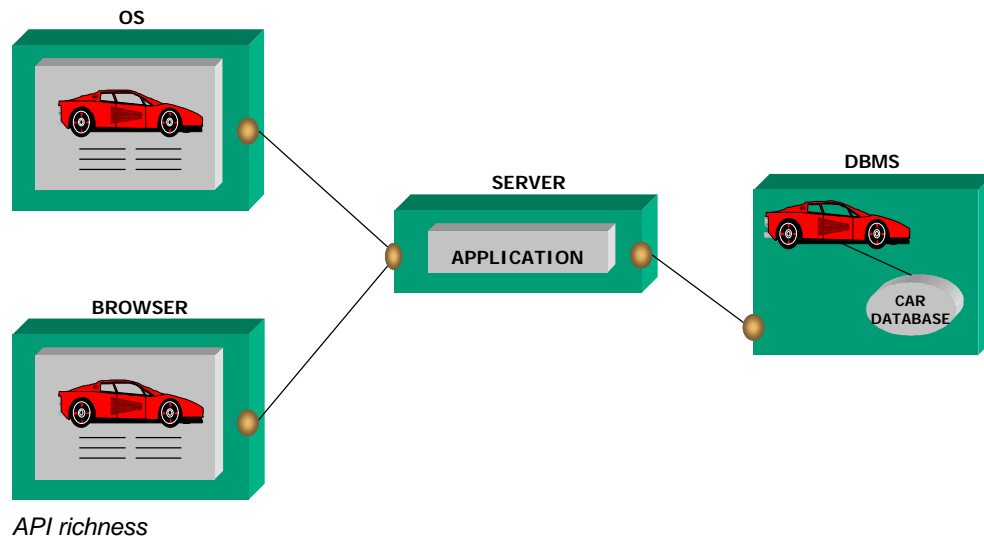
## Details of the process

The first main point is the management of ResultSets (the result of a Select query). In order to effectively provide the desired features, it is essential for the product to allow both forward backward navigation in a ResultSet (scrollable cursor). Having assistants for the event-driven and HTML interfaces is an asset. Coordination of the ResultSet rows with the database entries must be verified. The second point is database transaction management. When an order is placed, the STOCK table is updated and an INSERT is added to the ORDER table. The tool must provide all the mechanisms required for managing concurrent access (lockout, commit/rollback, etc.). Oracle is the benchmark for these tests.

## Key points

- Management of ResultSets
- JDBC support level
- Management of concurrent access

## Step 8: Richness of APIs



### General description

The salesman wants to show pictures of the cars for sale. Furthermore, for the e-commerce site, he wishes to send an order confirmation to the customer by e-mail.

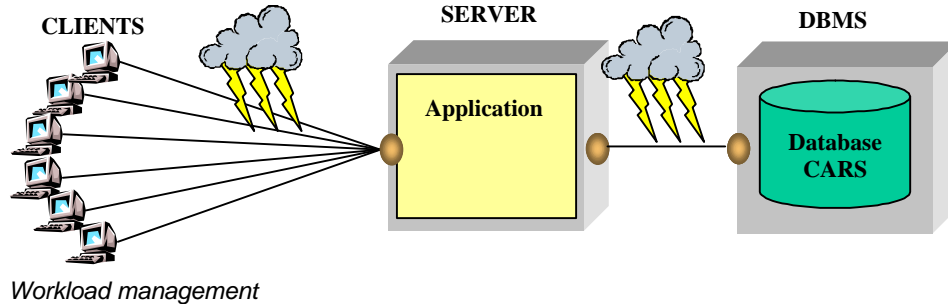
### Details of the process

Multimedia support in development tools is imperative. This step will enable the user to handle images of the cars in stock in BLOB form in a database and display them dynamically. For the event-driven interface, we check whether the tool can display them in a class. For the HTML interface, the images must be generated by the server in a format that can be interpreted by a browser (GIF, JPEG, etc.). Lastly, sending e-mail must be done through SMTP, the Internet's standard message system protocol.

### Key points

- BLOB management
- Image management
- Sending e-mail

## Step 9: Management of load increases



### General description

Convinced of the success of his e-commerce site, the salesman wants to evaluate the abilities of the tool to manage a significant number of users.

### Details of the process

It is important to know in advance how an application will respond when faced with a growing number of users. Using load balance and recovery management in case of error is discussed as well as all factors favoring the support numerous users (database connection pool, etc.). The presence of testing and/or monitoring tools must be validated.

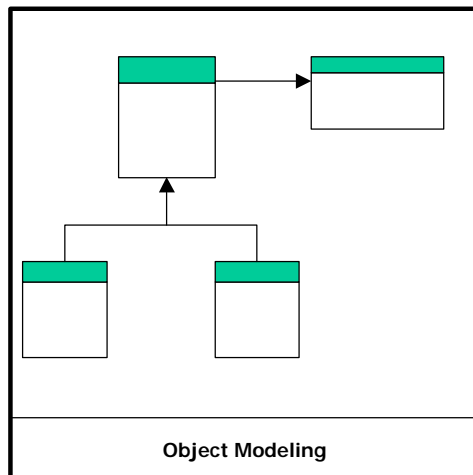
### Key points

- Ability to manage the load
- Load-balance solutions
- Recovery upon error incident
- Administration, testing and monitoring tools



### 4.3.3. Object Server Domain

#### Step 10: Modeling using business objects



*Modeling using business objects*

#### General description

Our salesman is aware that he is in an extremely competitive market. He knows that to keep ahead of the competition, he will constantly need to modify-47 the management rules of his applications (Francs-Euros\*, method of payment, warranty period, etc.) in order to always be in lie with his clients' requests. In this case, it is no longer about waiting for the new version of his application to meet a demand. Each of these building blocks must be able to be modified.

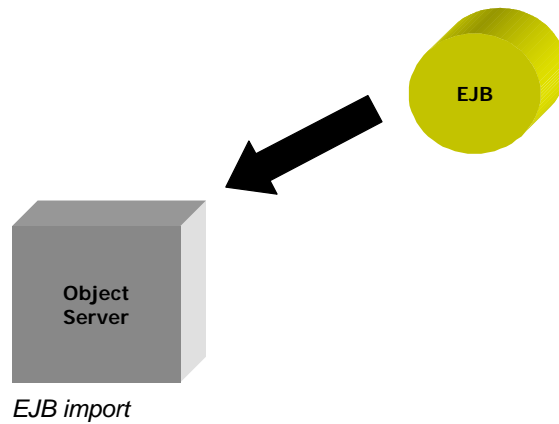
#### Details of the process

Component development adequately answers these needs. This type of development requires the use of techniques that are very different from those used in classic development (relational, Merise, etc.). Certain tools facilitate this approach, while others make it almost impossible. This step illustrates the options offered by the tool to implement modeling using business objects. The business object model we plan to implement is presented in the appendix of this specification. All options of this tool that facilitate the implementation of such a model are under study.

#### Key points

- Presence and quality of an object modeling tool (referential, teamwork, etc.)
- Quality of generated business objects (independence of objects regarding the technical context)
- Quality and power of business object manipulation tools at runtime

## Step 11: Importing EJBs



### General description

The salesman's bank suggests that he offer financing services to his clients. In order to do this, the bank can provide him with EJB components for debt calculation, building financing plans, etc. The salesman wishes to integrate them into his application.

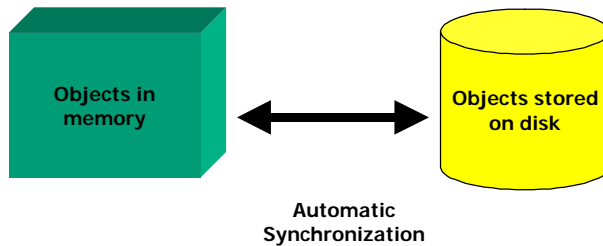
### Details of the process

The goal of the EJB (Enterprise JavaBean) specification is to define a standard for the Java business objects. They are independent of the tools used and must be able to be integrated into any tools that support them. This step illustrates the support offered by the tool to integrate existing EJB components. A "bank" object model was created while respecting the EJB specifications (this creation is not part of the step). This step consists of importing this model into the tool.

### Key points

- Simplicity
- EJB specification support level

## Step 12: Persistence



*Persistence*

### General description

Our salesman is not satisfied with his applications based on object modeling of business processing and entities. In fact, he has come up against a considerable problem: he must manage a relational database and an object layer in memory simultaneously, and, above all, he must synchronize these two subsystems. He realizes that this is almost impossible to do since the potential complexity involved makes any developments too heavy. He has nevertheless heard of a solution that could completely automate his task and even eliminate the requirement of advanced SQL knowledge, since the object layer would become the unique representation of the company's data for new applications. This solution, which automates the storage of objects on disk or their synchronization with a database, is persistence.

### Details of the process

The implemented object model is made persistent. This is accomplished by using the tool's persistence technology. If persistence is done using a relational database, we can base ourselves on the information below; however, if the tool is based on other solutions (object database, etc.), then we use these specific solutions.

*For relational databases:*

One way of representing business objects in the database is as follows: An AUTOMOBILE relational table and a MODEL table are defined. In addition to the attributes described in the object model, the AUTOMOBILE table includes a field entitled "Class", which allows the class of the car to be determined ("1" for a UsedCar or "2" for a NewCar). If this field specifies a used car, the "Status" field will then be filled in. If not, it will remain at NIL. The "Status" field is a numeric code, which means the following:

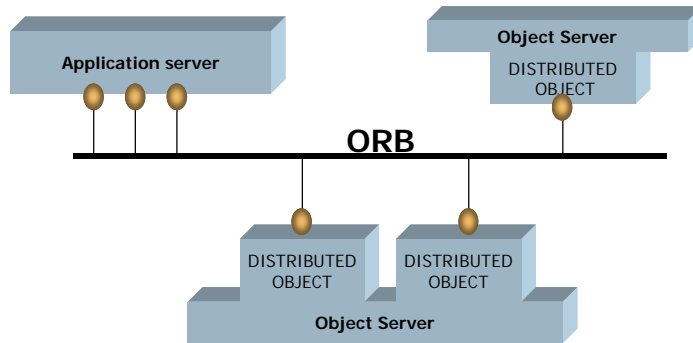
<b>Value</b>	<b>Meaning</b>
1	Very good condition
2	Good condition
3	Poor condition

Mapping with the database, that is, the way in which the object model interacts with the database (creation of objects, entry of changes, etc.), will optimally use the tool's capabilities. We will evaluate the quality of this mapping and its degree of automation.

## Key points

- Simplicity of development
- Completeness of persistence possibilities
- Synchronization strategies with the database and management of concurrent updates
- Optimization techniques implemented for each tool (shadowing, etc.)

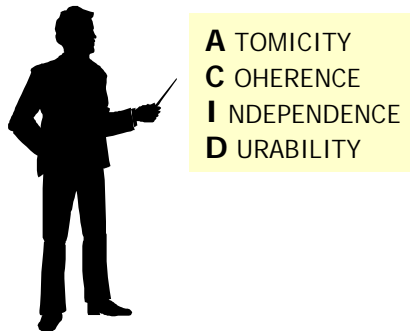
## Step 13: Object distribution and transaction management



### General description

The salesman wishes to make the "bargain of the week" (the car to sell as a priority, see step 6) accessible from any application on his network. While being optimistic about his commercial success, he wants a solid guarantee of service so that the bargain of the week is permanently accessible.

Moreover, the salesman wants to allow several co-workers to change the price and the slogan of the vehicles for sale. In any case, these changes to fundamental information must not disrupt the system and any problems of concurrent changes must be handled.



### Details of the process

Distributed-objects architecture makes possible communication between objects that are found in the different processes, possibly on different machines. Developing such an architecture is complex from a design point of view. It is desirable for a tool to not only offer maximum simplicity, but also power in this area. This step evaluates the way in which distributed-objects architecture can be implemented with the tool. It is the server-side counterpart of step 6 (Openness to distributed systems).

The object server receives an instance of an "Automobile" object, which corresponds to the least expensive car. This instance must be made accessible on the network to other applications through the system of distributed objects offered by the tool, under the public name of "bargain\_of\_the\_week." A small, separate executable file will be the client of this distributed object. This executable file must be able to be launched at any time on a different machine from the one that has the distributed object. This executable file will connect to the distributed object and will query it by calling the "carid", "engine", "price", "slogan" and "Warranty Period" methods.

Not only is it useful to have transactional possibilities in the relational databases at one's disposal, but it is also desirable to have transactional services available within an object tool. The transactions are no longer carried out in a relational database, but rather on the graph of objects. This step also illustrates the use of the tool object transactional system.

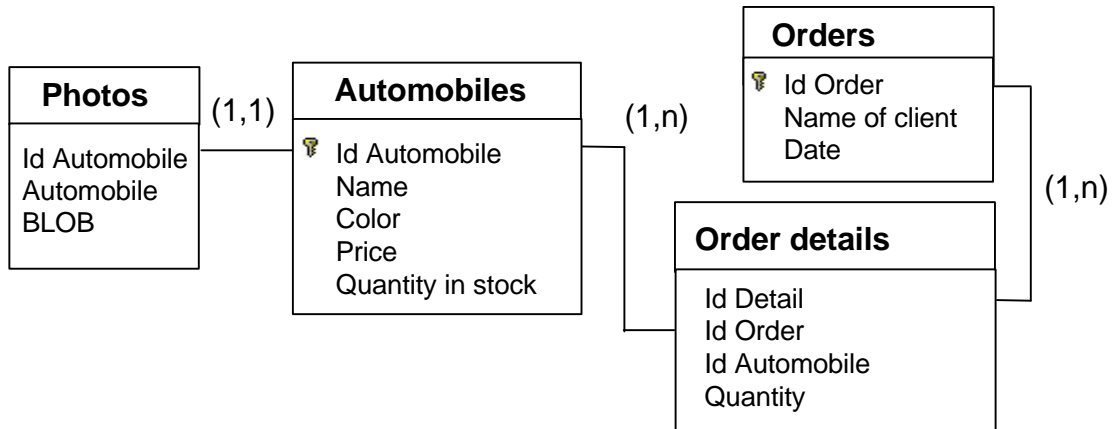
Two processes (for example, two threads), concurrently access the same Automobile object. The first one modifies the slogan and price. At the same time, the second process looks up the price and slogan. We check whether the second process recognizes the values before modification by the first (image before commit). Furthermore, we study that lockout mechanisms are running optimally when processes are simultaneously carrying out modifications on the same object. Rollback on the objects must also be supported. Distributed object transactions are discussed.

## Key points

- Categorization of distributed object technologies provided by the tool
- Power (management of objects' life cycle, re-entry, naming, etc.)
- Administration and monitoring tools
- Standards support
- Transactional system power (ACID features)
- Simplicity in using the transaction monitor
- Optimization techniques implemented

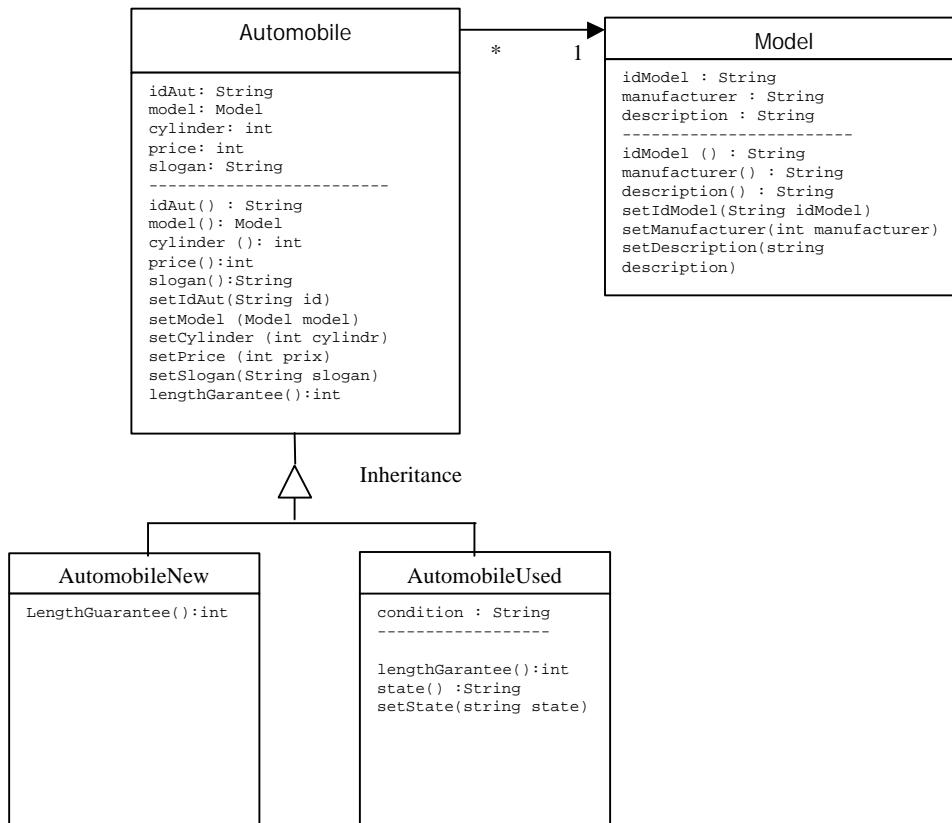
### 4.3.4. Scenario supplements

#### Database diagram



#### Object model of the application

The following is the list of attributes and methods for each class:



We define an Automobile class. This is a base class with two subclasses: AutomobileNew and AutomobileUsed. The object server should be able to instantiate objects belonging to these subclasses. We never instantiate Automobile class objects directly; instead we use one of its subclasses.

➤ **Specification of the business objects model**

The business objects class diagram provides a specification of methods that should be included in the interface of each class. This is a minimal specification. Additional methods or classes may be added if necessary. In addition, the exact location of the implementation of required methods is left open. We will take full advantage of code factorization and modularity abilities of the tool. The details of internal object attributes are provided here as an example. They may change depending on requirements.

Generally, the implementation and use of business objects should be such that as few modifications as possible are required if you want to add new subclasses.

Most methods make it possible to read or modify object attributes.

As we can see in the diagram of business object classes, AutomobileUsed has a specific method that returns the condition of the automobile. Possible values include "very good", "good" and "poor".

The periodWarranty method calculates the proposed warranty period for an automobile. As we can note by looking at the diagram, this is not a class attribute. This is a value calculated at the business object level. Calculation rules depend on whether the automobile is new or used. This method is therefore implemented differently in each of the subclasses. It may be called by an automobile object process without knowing whether they are new or used automobiles. The calculation rules are as follows: For a new car, the proposed warranty period is 36 months if the price is higher than \$16,000. If not, it is 24 months. For used cars, the period is 24 months if the condition is "very good", 12 months if it is "good" and 0 months for "poor".



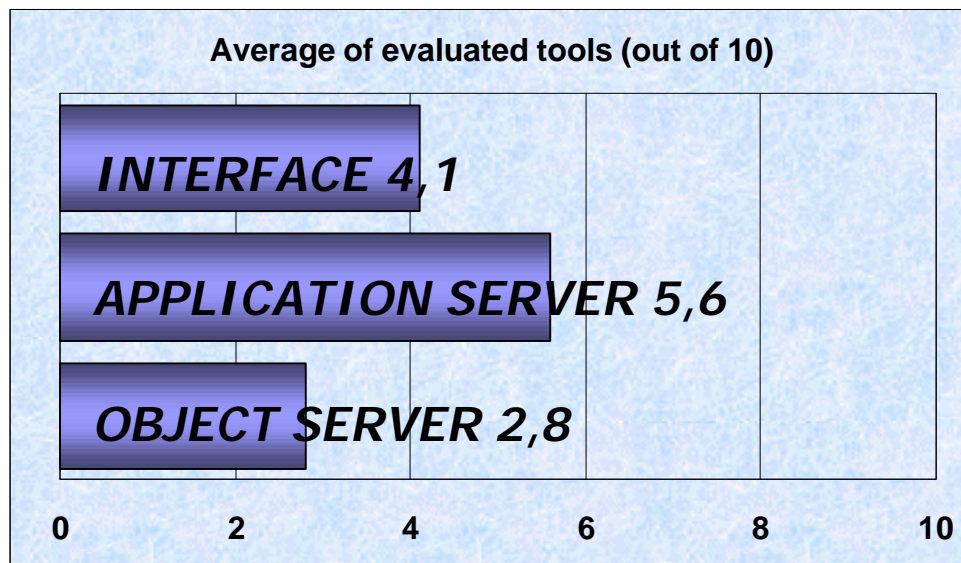
## 5. Positioning of the Main Editors

### 5.1. Synthesis

The evaluations made of the various environments in this report provide information on the positioning of the tools in relation to one another. However, these results also represent a strong indicator of the degree of maturity of the technologies and of the capabilities of the various tools in meeting the real needs of business as best as possible.

Based on the steps described in the evaluation scenario (paragraph 4.3), we get a rather precise vision of the state of the market as well as the reality surrounding the emerging technologies. These steps are representative of the concrete needs of design, development and deployment teams in the development of intranet transactional applications.

The diagram below summarizes for each domain scenario the average performance of the six offers evaluated in this Java report in the form of a rating between 0 and 10:



*Average per domain of the six environments evaluated*

The first observation involves the low results obtained. With a general average of 4.1 out of 10 for all domains, it is understood that the choice of a Java architecture for the development of transactional applications will be dependent on a close analysis of the functional and technical needs required and a judicious choice regarding the production environment.

Even if the positioning of certain offers in a particular domain can partly justify the low overall results, it is even more indicative of the lack of completeness of each product and the almost systematic need for third party offers to cover the production cycle of business applications in their entirety.

If we focus more specifically on the potential of the tools tested for assisting business application development with a Java event-driven interface, the general statement of fact remains similar. In fact, by not taking into account step 3 (HTML interface generation, features specific to other development environments) and by forgetting step 11 (import of EJBs, technology still too immature to be integrated into a business context), the general average only increases to 4.3 out of 10, which is still largely insufficient. By analyzing more precisely the results obtained in each domain, the following conclusions are drawn:

➤ **Interface:**

With 4.1 out of 10, the general average for the interface generation segment remains low. Nevertheless, the distribution between each step provides very disparate information. As for the Java graphical interface design assistance and the representation possibilities of values stemming from data sources, the tools tested proposing a development platform showed a certain degree of maturity. Things deteriorate when proposing an HTML interface, for its environments are not particularly designed for this type of architecture, at least in regards to the development platform where wizards are quite limited. But the most problematic part remains the generation of print jobs, for the tested environments propose only very brief reporting solutions, during both the design phase and the actual generation of the report.

This need, which appeared as one of the major problems of HTML interface architectures, is also present in a Java context. It has more to do with technical deficiencies due to the newness of the intranet and Java development tools rather than difficulties inherent with architecture types. In other words, the editors will have to work harder at this level since the printing needs within a business context still remain unavoidable today.

➤ **Application Server:**

This is the domain where the products propose the most features as a whole. Whether it be for access to existing distributed objects in the business' information system, for the quality and completeness of the access to relational databases, or the possibilities offered in terms of workload support, today's application servers are sufficiently complete to meet most needs.

The only weakness of the offers evaluated involves the completeness of the APIs, which often translates into a lack of access to existing technologies. If the completeness of JDK Java allows all of the environments to propose temporary solutions, most of the time it is still necessary to go through extensive and costly developments. Elements of the highest level masking the technical complexity would not be superfluous. Therefore, the generation of server images from, for example, blob fields, or access to a message server will be particularly painful tasks which will translate into heavy programming.

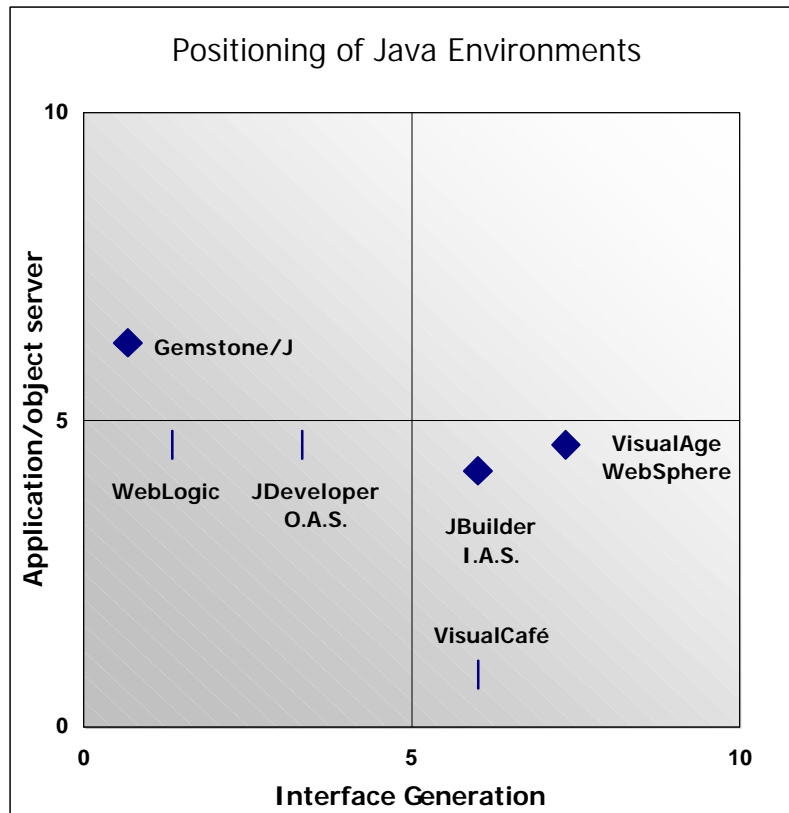
➤ **Object Server:**

Indisputably, the environments evaluated in this Java report present, with the exception of GemStone/J, deficiencies that are too significant to be assimilated into real object servers. The exaggerated marketing that today accompanies Enterprise JavaBeans does not reflect the reality of the terrain and the immaturity of this technology for a true business context. With specifications that are still very concise and proprietary components such as the "deployment descriptor", the EJBs are still a far cry from the standard distributed object model everyone wants. In other words, environments that want to be EJB object servers must immediately base themselves on another element models in order to be assimilated to object servers.

The specific offers, based for the most part on proprietary communication protocols between objects still have a good few years to go. And within this context, GemStone/J, Forté and other WebObjects, with mature offers based on object models that are directly applicable to concrete cases, present undeniable assets in the field.

The only positive point from a general point of view involves the distribution of objects and the OTM support. For this last item, there is a certain complexity in the implementation, directly produced by the reduced capacity of the tools in order to ensure the persistence of the data contained in the objects. For the import of EJBs, the specificity of certain component parts, such as the “deployment descriptor”, considerably reduces standardization and requires a specific adaptation for each case. Finally, none of the evaluated tools integrates object model graphical tools adapted to their own application server. In this area, we will have to count on third party offers.

In conclusion, we can see that each of the tested environments presents one or more shocking deficiencies. This is confirmed in the synthesis diagram below, in which no offer succeeded in entering the magic square:



*Positioning of tools tested in all three domains*

In fact, we can already separate the various evaluated offers into very distinct categories. The positioning of the offers within the above diagram provides a few elements of information at this level.

➤ **Integrated Runtime Environments:**

First of all, we have the integrated environments, or at least the assimilated ones, since none of them succeeded in appearing in the magic square in our evaluation. Some such offers are **IBM, Inprise** and **Oracle**. As a common point, these three tools were the least impressive in regards to the object server segment, which confirms the general conclusions provided on this issue by all the environments.

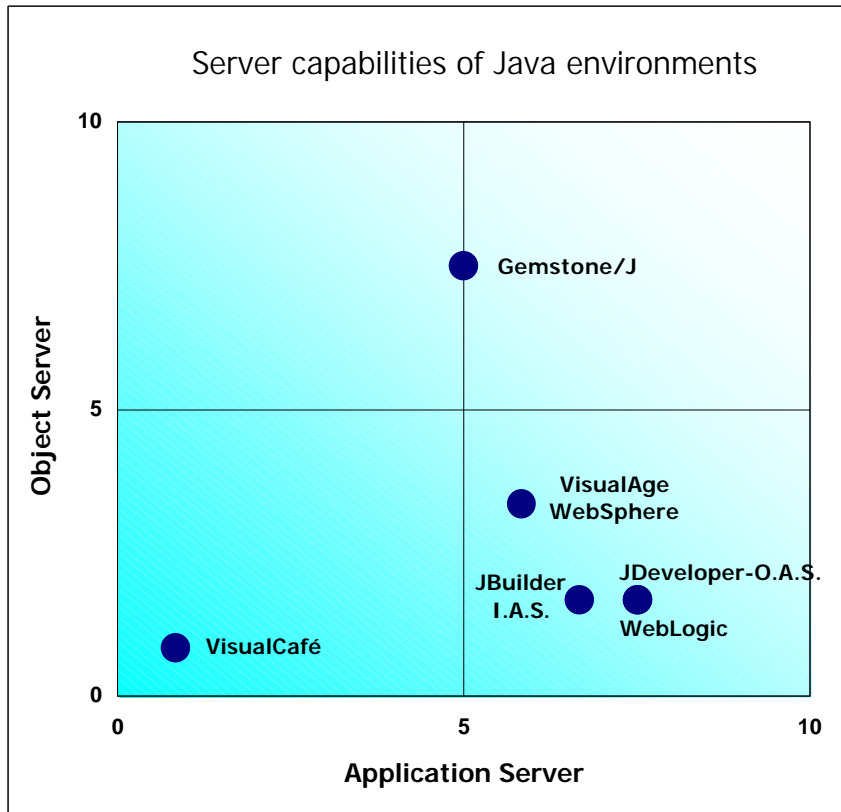
IBM presents the most assets in terms of completeness, due to three components that are more or less integrated and which fulfill each of the needs. VisualAge is the application development platform, which allows among other things to cover the interface generation segment. WebSphere, for its part, is used as an application server. If the EJB object is conditionally committed to WebSphere, currently it is the persistence builder that offers the most possibilities in terms of object capacity. With a model tool and support for the persistence and object transactions, the only thing remaining is integration with WebSphere to truly present a homogenous offer.

Inprise presents characteristics that are similar to those of IBM. The major difference involves the recommended object model since Inprise Application Server is based on ORB Visibroker and Corba 2 while the offer is based on EJBs in WebSphere. If not, JBuilder represents Inprise application server's integrated development platform and is, as the name suggests the house application server. If IAS is complete for the administration and deployment of distributed Corba applications, it's Web technologies do not integrate well with servers and IDEs. The object server capabilities are still concise enough, even though the existing building blocks can technically satisfy this need. We should also point out the presence of a very complete transaction management environment based on Corba called ITS. What is especially lacking in the Inprise offer are wizard tools and top-end modules to reduce the cost of object model developments, and a basic capacity: object persistence. In the future, the EJB support should soon be in IAS.

Finally, Oracle JDeveloper and Oracle Application Server provide a complete offer in terms of coverage but there are still too many deficiencies. JDeveloper 1.1, for example, does not reach the degree of maturity of the other development platforms. We will have to wait for version 2.0. However, we can appreciate the effort put in by the editor to adapt the technology produced by Borland-Inprise technologies to the Oracle Application Server. The latter has no reason to envy its competitors in terms of support features that are specific to application servers. Supported by various cartridges, Oracle Application Server can, depending on the context, be opened to different languages, thereby extending the scope of its technical possibilities. We have also noted the presence of numerous modules that are specific to the Oracle editor that make it possible to correct certain limits. For the concept of object servers, the proposed solution is based on a Corba 2 ORB. Operational, this offer still lacks assistance services. This architectural building block should eventually be integrated into Oracle 8i. It still remains to be seen how OAS and Oracle 8i will cooperate with one another and if the distribution of application processing and distributed object services between the two modules will be clear and complementary.

➤ **The Servers:**

Two environments tested in this report are primarily dedicated to the execution of applications, in the form of business objects or rather application services. The two products within this category, GemStone/J by **GemStone** and WebLogic Application Server by **BEA**, are characterized by the absence of a development platform however, they do not cover the same domains, as is illustrated in the following diagram:



*Positioning of the application server and object server domains*

GemStone/J is a real object server. Thanks notably to a proprietary implementation of RMI based on IIOP, GemStone/J is based on a technology that is certainly proprietary but has now been well tested, thereby producing an environment capable of meeting the type of specific needs that we should expect from an object server, such as persistence, distribution and object modeling. Only the import of EJBs is missing a few final touches, but this is not really with the editor's purview. However, it would be a bit premature to define GemStone/J as an object server exclusively, since the tool is equally an application server. It still lacks a development platform to facilitate the implementation of certain tasks and higher-capability libraries than just the APIs of JDK 1.2 to open itself to the outside.

WebLogic, for its part, is an outstanding application server. At the top of the list in this area of comparison along with Oracle Application Server, WebLogic offers all the tools to be a distributed object client, in order to interface with databases or can be flexible enough from a deployment architecture point of view to support significant workloads. The only limitation involves the technical level required in the use of Java APIs. These are based on Sun recommendations, regarding the Java platform for business. However, WebLogic is no longer a viable object server for business projects. It must be said that the choice of EJB technology limits the possibilities considerably in this area.

➤ **Development Platform:**

Finally, **Symantec**, with the VisualCafé Database Edition, is the only environment that does not include an application server in its offer. This product is a development platform for two-tiered Java applications, with access to the databases via a client-side JDBC driver. The architecture is therefore quite different from the other environments evaluated in this report. For the Java interface, and for what is destined for, VisualCafé provides very good results. The weak points involve the inability to generate print jobs and HTML interfaces. It must be said that the three-tiered architecture does not correspond to the product's philosophy. VisualCafé can also be used as an IDE for application servers such as WebLogic (partnership being established) and GemStone/J. But here again, beyond the interface design wizards, it lacks extended integration with business processing, such as the presence of a debugger or server-side generation wizards. The next version of VisualCafé, which will support EJB, RMI and Corba, will provide more complementarity with WebLogic for example. Still, the EJB specifications will have to evolve so that this integration can be used in the concrete implementation of business applications.

## 5.2. Product Sheets

### 5.2.1. BEA: WebLogic Application Server 4.0.2

Identification	
<b>Product composition</b>	WebLogic Application Server
<b>Version</b>	4.0.2
<b>DBMS supported</b>	All bases with a JDBC driver (Oracle, Sybase, Informix, MS/SQL Server, etc.)
<b>Development platforms</b>	HP UX, Solaris Sparc, AIX being validated, Linux, NT 4, OS 400, True 64
<b>Deployment platforms</b>	Same as development platforms
<b>Editor</b>	BEA Systems

### Introduction



BEA is a young American company founded in 1995 by former employees of Sun and Pyramid Technology. BEA took off when it acquired the Tuxedo technologies from Novell in 1996. In addition, the Novell teams became BEA employees. Later, it similarly acquired the ORB technology from Digital ObjectBroker, along with close to eighty employees. This was also the case for the TOP END middleware from NCR. Finally, in 1998, the company acquired WebLogic, the editor of the Web application server "Tengah". Today, BEA, which has over 1,200 employees, has become a market leader in server-side software building blocks (TP monitor, ORB, MOM, etc.). The company's ambition is, correspondingly, to successfully make a name for itself in the market of application servers.

BEA offers two application servers. WebLogic Application Server is the product from the WebLogic acquisition, ranked in the mid-product palette by BEA, which we evaluated in this study. The other product, WebLogic Enterprise, is the new name of M3, a server ranked in BEA's top palette, and is based on Corba and Tuxedo technology. Unlike the WebLogic Application Server, it does not yet offer broad services for the implementation of Web applications. The two products, based on very different technologies, have only the name in common for the moment, but BEA's future strategy consists of integrating the capabilities of the two products and offering upward compatibility of WebLogic Application Server to WebLogic Enterprise. In fact, it is already possible to integrate these two products into the same architecture (for example, with WebLogic Application Server as the application server and WebLogic Enterprise as the object transactional server). We must nevertheless be wary of the complexity inherent to this type of architecture.

WebLogic Application Server does not include a development workshop, but does focus on the server part. However, a set of JavaBeans is provided to facilitate integration with third-party development workshops.

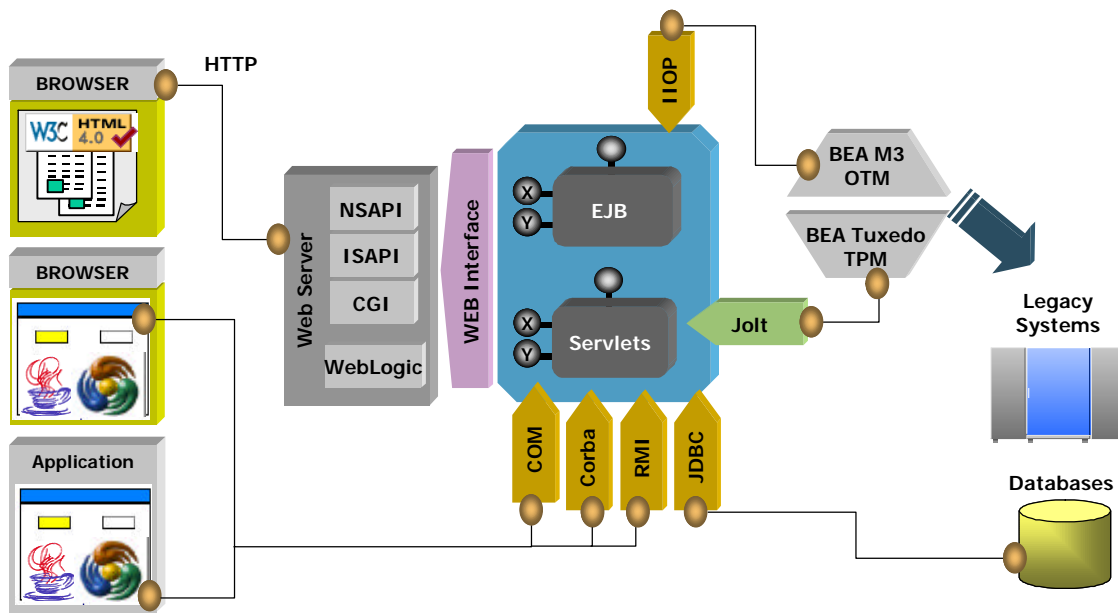
As middleware and a server-side application, WebLogic offers numerous features, most of which are based on the recommendations from Sun concerning the Java enterprise platform. Based on JDK 1.1.7, they integrate the connection pool, security, load balancing and EJBs in particular. Database connections are ensured by the editor's JDBC technology, which uses its proven experience here with middleware. We also note the crash tolerance features introduced in version 4, which is well ahead of numerous competitors in this area.

As for its Web features, the product offers its own HTTP server while making it possible to use other servers. There is also a servlet engine, which has been tested and validated in the field. The generation of servlets is based on a specific technology called JHTML, the editor having the intention of migrating to JSP in the future.

Regarding object technology, the distributed communication system is RMI. BEA offers the option of using a new RMI implementation developed by the vendor, since the standard one Sun delivered in the JDK is known for its poor performance, particularly with high numbers of users.

The proposed business objects model is that of EJB. In this area, the product must make substantial advances, because the technology available today does not provide an environment truly conducive to implementing a true business object development. We have especially noted that the development process turns out to be too heavy and the persistence features are insufficient.

## Architecture



BEA WebLogic Architecture

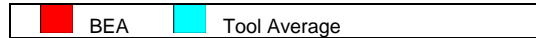
WebLogic Application Server is based on the company's APIs defined by Sun, and therefore definitely plays a part in its standardization card. The services offered vary, and we will especially appreciate the expertise of the editor regarding database access. Nonetheless, caution must be taken: since the product does not have strong integration with an IDE, the development phase turns out to be particularly heavy.

The Pros	The Cons
----------	----------

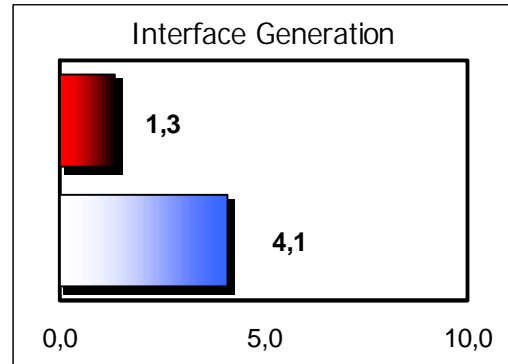
- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>▶ Openness toward standards</li> <li>▶ Tested servlet engine</li> <li>▶ Optimized model for distributed objects</li> <li>▶ Advanced JDBC support</li> <li>▶ Load-balancing and failover abilities</li> </ul> | <ul style="list-style-type: none"> <li>▶ No development workshop</li> <li>▶ Heavy development/deployment process</li> <li>▶ JDK 1.2 is not yet supported</li> <li>▶ Little support of standard object technologies</li> </ul> |
|---|---|



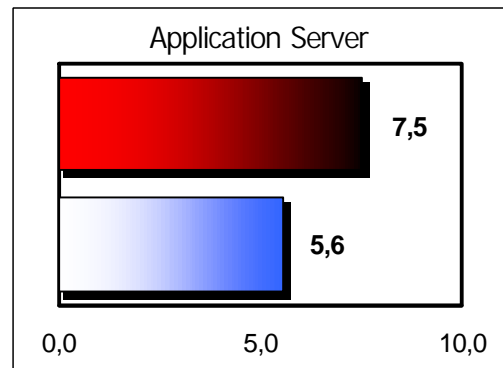
## Positioning



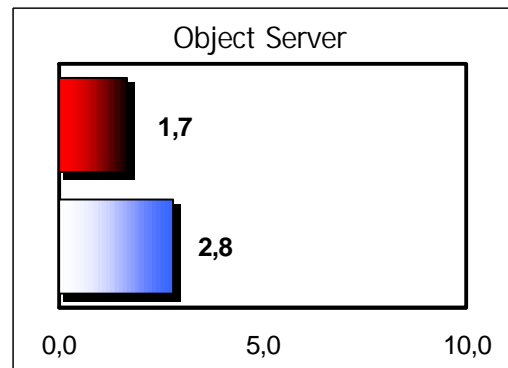
WebLogic Application Server does not offer an integrated development environment for the implementation of the user interface. Indeed, a set of JavaBeans is delivered, but, overall, the integration of WebLogic with third-party tools leaves something to be desired. The application server is open to event-driven as well as HTML interfaces. With respect to the latter, we saw the integration of a servlet engine and support of proprietary technology called JHTML, offering the same type of integration of Java with HTML, as well as with Java Server Pages.



In the area of application servers, WebLogic shares first place in this study with the offer from Oracle. Good database-connection abilities, well-designed clustering and failover model, good openness to the company's Java APIs or even a tested servlet engine, thus allowing WebLogic to obtain this very good result. A faster development process, for example, by offering an integrated development environment, would have enabled WebLogic to increase its lead even more.



In the area of Java object servers, WebLogic disappoints. BEA's product indeed offers optimized object distribution capacities, but it is not sufficient when compared with the competitor's products evaluated in this study. The object capacities offered by the WebLogic Enterprise JavaBean model are in fact immature, limited and of no use for slightly complex company projects.



## Technical Specifications

<b>Development license</b>	Approximately \$2250 per developer
<b>Deployment license</b>	Approximately \$11,800/CPU. Cluster option: approximately \$5,800/CPU
<b>Configuration for the evaluation</b>	WebLogic Application Server v 3.7 and 4.02
<b>Additional modules</b>	Additional JDBC pilots, technical support and subscription for updates



## 5.2.2. GemStone Systems: GemStone/J 3.0

Identification	
<b>Product composition</b>	GemStone/J
<b>Version</b>	3.0
<b>DBMS supported</b>	Oracle, Sybase, Informix, DB2, Open Ingres, SQL Server, etc.
<b>Development platforms</b>	Solaris , Windows NT, HP-UX planned for end of 1999, AIX planned for beginning of 2000
<b>Deployment platforms</b>	Same as development platforms
<b>Editor</b>	GemStone Systems

## Introduction



GemStone Systems, based in Beaverton, Oregon, is well known in the world of object technologies where it has played a major role for a number of years. Its object database for Smalltalk is recognized and widely used. With the arrival of Java, the Smalltalk market collapsed, causing the company to fundamentally shift toward this new language.

Version 3.0 of GemStone/J offers a set of services, which allows for the implementation of multi-level architectures based on an object database. As an application server, object server and database engine, GemStone offers significant coherence qualities.

The recommended resources for the optimal operation of the product are sizeable: between 192 and 512 MB RAM is required, depending on the hardware configuration, in order for the server to run comfortably. Add to this the resources required by the applications themselves.

As an application server, GemStone allows for, among other things, the implementation of transactional Web applications. Implementation of HTTP protocol is ensured, with context management.

GemStone also houses objects that can be accessed externally via RMI or Corba. These objects are persistently stored in the internal object database. With this said, it is also possible for a GemStone application to interact with relational databases via JDBC or mapping object/third-party relational tools.

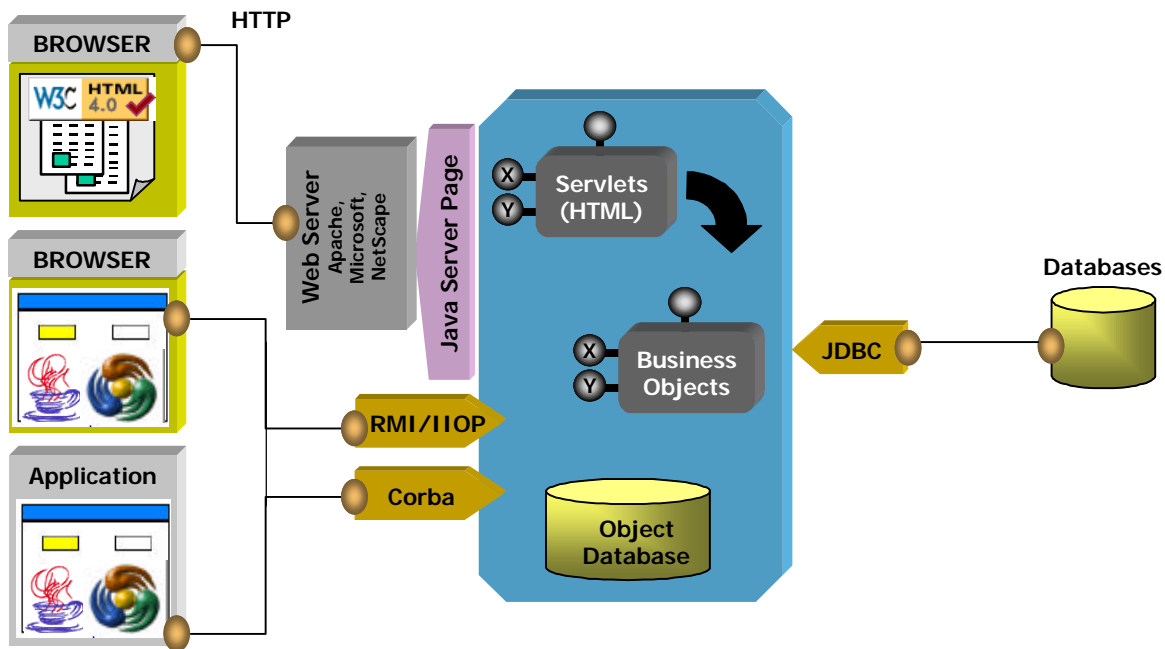
GemStone does not include a graphical development environment. A third-party environment should therefore be used, even with the limitations in terms of productivity and integration that it infers.

Many services are provided to implement the application. The database connection pool is ensured and can be configured using a graphics tool. Security management is a strong point of this product, which offers highly advanced and complete services in this area. For workload increases, load-balancing services are available which allow the work to be distributed over several GemStone servers.

As regards to its object technology, GemStone clearly constitutes one of the most advanced environments. A true modeling for business object can be implemented. The object persistence and transactions are ensured, and conflict management strategies resulting from multi-user operations are also taken into account. With respect to object distribution, we note the presence of an RMI implementation that is different from Sun's, and whose performance is notoriously poor. A particularity of this implementation is the use of the IIOB communications protocol, usually implemented by Corba, which corresponds to the last recommendations made by OMG.

Concerning EJB support, although we consider that it is not at the level claimed by the vendor, it nevertheless conforms to the 1.0 standard overall. However, development using GemStone with this model is much too heavy at the moment.

## Architecture



*GemStone/J 3.0 Architecture*

Based on JDK 1.2, GemStone/J is one of the most advanced Java tools regarding object technologies. It is one of the rare products that truly offers the elements required for a true modeling per object. These capabilities will indeed be used within the scope of applications based on the integrated database. We will leave GemStone/J to the teams of object technology experts.

The Pros	The Cons
----------	----------

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>▶ Advanced object technology support, particularly for persistence</li> <li>▶ JDK 1.2 support</li> <li>▶ Advanced security model</li> <li>▶ Load balancing</li> </ul> | <ul style="list-style-type: none"> <li>▶ Resource-intensive server</li> <li>▶ No integrated development environment</li> <li>▶ Heavy development process</li> <li>▶ Slow graphic interface</li> </ul> |
|--|---|

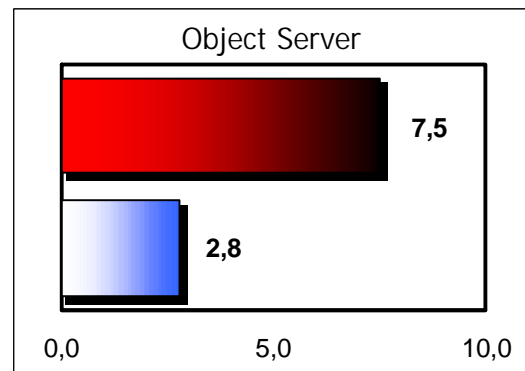
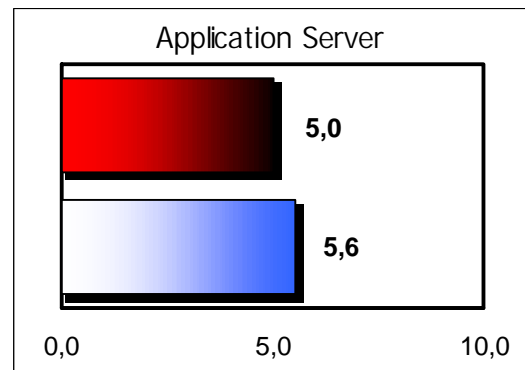
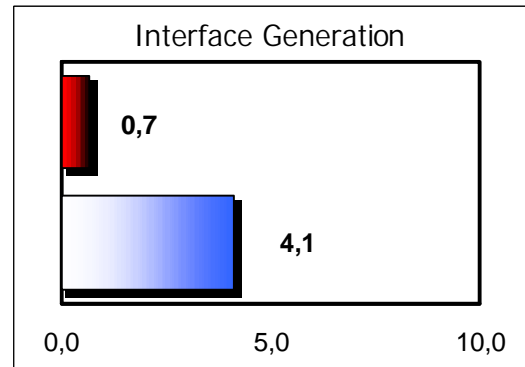
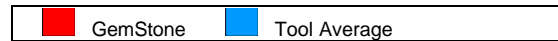
## Positioning

GemStone/J does not offer a tool to implement a graphical interface. Integration with existing tools is weak and not at a high enough level. We would wish to be able to implement an end-to-end object approach, including a graphical interface. To do this, a business object-mapping tool must be available on the interface.

Nevertheless, GemStone is open to event-driven type interfaces as well as to HTML interfaces. An integrated servlet engine generates the latter.

Although slightly below average compared with the tools evaluated, GemStone/J still offers interesting features, such as the connection pool or JDK 1.2 support. Unfortunately, the lack of a development environment, the heaviness of the administration tools, incomplete and poor-quality documentation, or the lack of support for multimedia applications lowers its grade. Overall, the development process is still too heavy and should be improved to make GemStone a leader in the area of application servers.

Taking first place in this study in the area of Java object servers, GemStone/J literally leaves the other products behind. It is the only product in the study to offer an effective environment to implement strong object-oriented applications. Characteristics such as persistence, distribution, object transactions and administration make up a coherent model that completely surpasses those of the other products evaluated that either offer poorly developed versions of these features or none whatsoever. You must nevertheless pay attention to the strong object abilities required by the developer.



## Technical Specifications

<b>Development license</b>	Approximately \$4,900 per developer
<b>Deployment license</b>	Approximately \$58,000/CPU. Starter Kit: approximately \$29,000 for 3 development licenses + 1 deployment license
<b>Configuration for the evaluation</b>	GemStone J/2.1 and GemStone/J 3.0. Windows NT versions.
<b>Additional modules</b>	Security module



### 5.2.3. IBM: VisualAge for Java Enterprise Edition 2.0, WebSphere Advanced Edition 2.0.2

#### Identification

<b>Product composition</b>	VisualAge for Java Enterprise edition (+Enterprise update), IBM WebSphere Advanced Edition
<b>Version</b>	2.0 (VisualAge), 2.0.2 (WebSphere)
<b>DBMS supported</b>	JDBC, ODBC
<b>Development platforms</b>	Windows 95, 98, NT, AIX
<b>Deployment platforms</b>	Windows NT, AIX
<b>Editor</b>	IBM Software

#### Introduction



IBM's strategy currently revolves around e-business. The technical infrastructure provided by IBM for creating e-business applications is the "Network Computing Framework" using a unique development language -- Java. IBM's huge investment in Java technologies has resulted in a variety of tools including the Java WebSphere application server and IDE VisualAge for Java.

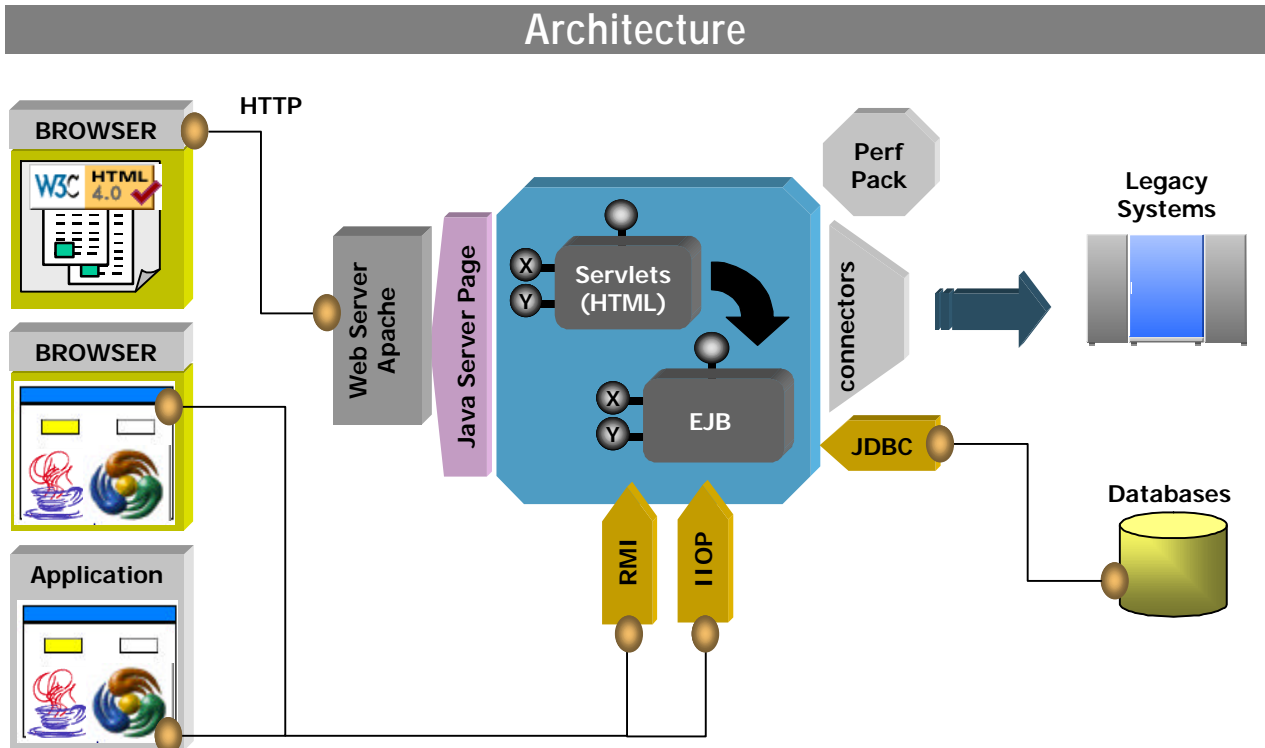
Due to their massive investment in Java technologies and their experience with AGLs as part of the VisualAge line (Cobol, SmallTalk, C++), IBM has made VisualAge the benchmark for Java development tools. Although IDE is confusing at first, it is very productive when mastered and pleasant to work with. All development is stored in a repository to ensure coherence between source files and compiled files, manage development versions and provide a high-end view to the developer by making it possible to not have to deal with files and directories. The repository may be shared for development in teams. Its visual design workshop is a tool that fully supports the philosophy of JavaBeans development. Swing components are also supported. By supplying a large number of JavaBeans in their Enterprise version (Data Access Builder, CICS Beans, SAP R/3 Beans, RMI Access Builder, Domino, Encina Access Builder), VisualAge makes it easier to develop applications that access enterprise systems.

IBM has gone the standardization route by providing support for servlets, JSPs and EJBs in WebSphere. Administration is accomplished through an ergonomic Java applet and has rich features. The recommended front-end HTTP server is Apache, but the user may select another server during installation. The server does not have a Corba ORB; this has been announced for the next version. The WebSphere application server includes the MVC (Model View Controller) development model. It is very interesting; the EJB-JavaBeans, JSPs and servlets have the roles of model, view and controller respectively. WebSphere has a pool of JDBC connections to improve performance and uses JavaBeans to hide the complexity of the JDBC 1 API and make up for its shortcomings.

However, it is unfortunate that the tools are not better integrated. JSP and HTML development must be accomplished using third-party tools provided in the WebSphere Studio product (NetObjects ScriptBuilder and Fusion). For the time being, there is no common universal set. One is announced for the next version. It is known as WallOp.

WebSphere itself does not have a workload solution. An additional product, the Performance Pack, must be obtained from IBM. While it does provide load-balancing services for Web servers, it does not have any solutions for the application server itself. Balancing for several servlet engines from a Web server has been announced for version 3 of WebSphere.

The object modeling support in WebSphere is the EJB standard. The EJB standard, in its current state, does not provide object modeling, because it does not support inheritance or referential integrity. It is also impossible to create EJB-compatible components from existing databases since relational-object mapping was not specified in standard 1.0.



IBM VisualAge for Java Enterprise architecture

The IBM product is promising, but it must mature and provide better integration. At the heart of IBM's strategy, WebSphere has benefited from heavy investments and involvement from IBM in terms of fine-tuning standards. This will help it to survive. Many additional functions are expected in the next version.

The Pros	The Cons
----------	----------

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>▶ Completeness of the product</li> <li>▶ Quality and maturity of VisualAge</li> <li>▶ Very strong commitment from IBM in terms of standards and huge investments in the entire product</li> </ul> | <ul style="list-style-type: none"> <li>▶ Lack of integration of the various tools</li> <li>▶ WebSphere's lack of maturity</li> <li>▶ Complex choices for designing applications (multiple options)</li> </ul> |
|--|---|



## Positioning

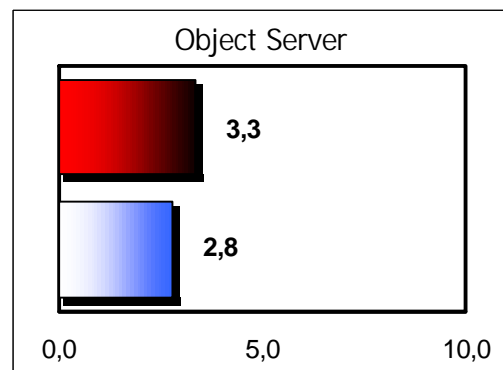
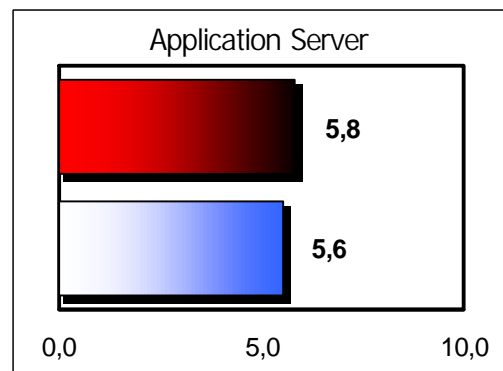
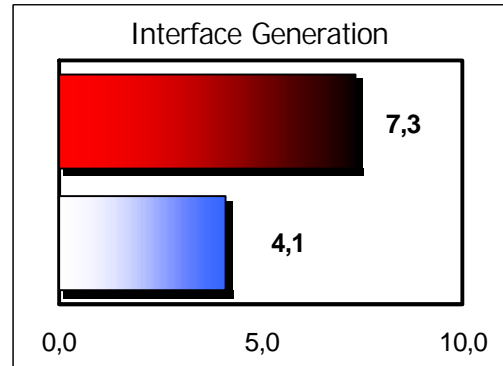
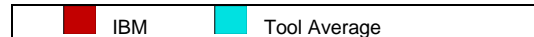
VisualAge for Java is the benchmark in Java development tools. Java interfaces are created by drag and drop. The IDE includes a debugger, a compiler, a source editor and an object inspector. While the programming method may seem confusing/complicated at first, it is productive. The repository handles management of development versions. The tool makes it possible to program visually and fully respects the JavaBean philosophy.

Its weakness is in the lack of help for creating HTML interfaces for which use of a third-party tool is suggested.

WebSphere is centered on two major components: a servlet/JSP engine and an EJB server. The installation process proposes the installation of the IBM HTTP server based on Apache and allows the choice of other Web servers. It is administered by a Java applet. The recommended programming model is MVC (Model/View/Controller for EJB-JavaBean/JSP/servlet) in which Java plays a central role. Numerous Java components facilitate application development (DBMS access, CICS systems, Encina, SAP/R3, XML, etc.) and a pool of JDBC connections is provided. Unfortunately, integration could be better between the development tools and the application server. As well, there is no Corba ORB.

Support for object models recommended by IBM for their WebSphere server will be provided through the EJB standard. Much effort has been put into providing a productive EJB development environment. However, the EJB standard and its implementation in WebSphere are still in its early stages and do not make it possible to create enterprise object models.

Nevertheless, IBM does provide an interesting alternative with its Persistence Builder module in VisualAge. However, it does not integrate into the WebSphere server.



## Technical Specifications

<b>Development license</b>	Aproximately \$3120 (VisualAge 2 Enterprise Edition)
<b>Deployment license</b>	Approximately \$830 (WebSphere 2 Standard Edition) Approximately \$4900 (WebSphere 2 Advanced Edition)
<b>Configuration for the evaluation</b>	Windows NT 4 sp 3 (DELL Bi xeon 1GB RAM)
<b>Additional modules</b>	Approximately \$7600 (Performance Pack)



## 5.2.4. Inprise: Inprise Application Server – JBuilder 2 for Application server

### Identification

<b>Product composition</b>	Inprise Application Server, JBuilder for Application Server
<b>Version</b>	1.0 (Application Server), 2.0 (JBuilder)
<b>DBMS supported</b>	JDBC, ODBC, Interbase
<b>Development platforms</b>	Windows 95, 98, NT
<b>Deployment platforms</b>	Windows NT 4 sp3, Solaris 2.5.1, HP-UX, AIX
<b>Editor</b>	Inprise

### Introduction



Borland was founded in 1983. The new name of the company is Inprise, following its recent acquisition by Visigenic Software. By combining Borland's inheritance with the development tools and ORB technology of Visigenic, Inprise launched its new flagship product Application Server to provide an overall response that covers development, deployment and centralized management of distributed applications.

Strengthened by its experience in development tools under the Borland name, Inprise includes the JBuilder 2 development tool in its product. The tool provides all elements indispensable for developing Java applications, including a debugger, source editor, a rich palette of components, a database connection assistant, support for JavaBeans and a manager for integrated versions (PVCS). Its special feature is its ability to function in both directions. In fact, it can be used to integrate existing Java developments. This feature makes the tool very close to the code and will satisfy developers who want to maintain total control over their developments. However, since it does not support visual connections, the tool is not well adapted to JavaBeans development. Nevertheless, considerable effort was made to integrate the development tool in the application server by the Inprise development teams: Visibroker is integrated with JBuilder 2 and publishing components on the server is quite natural.

Founded on the Visibroker ORB, Application Server provides rich functions for supporting Corba applications and makes it possible to deploy applications on many servers. The deployment of applications on target workstations as well as naming service and Corba transaction management are accomplished through a graphics console. While it may require a bit of time to adapt to it, it is productive once it is mastered. Inprise provides transaction services, ITS (Integrated Transaction Services) that conform to the OTS specification of OMG. In addition to transactions on Corba objects, ITS can integrate calls to JDBC data sources and central systems (CICS, Tuxedo, IMS/TM and MQSeries). While developing transactional applications with ITS provides interesting prospects, it involves rather cumbersome development by writing IDL interfaces, for which the integration of the JTA API could have simplified. Regarding databases, the functionality of the pool of JDBC connections is unfortunately too limited. Security services are available as an option in Visibroker (SSL on IIOP).

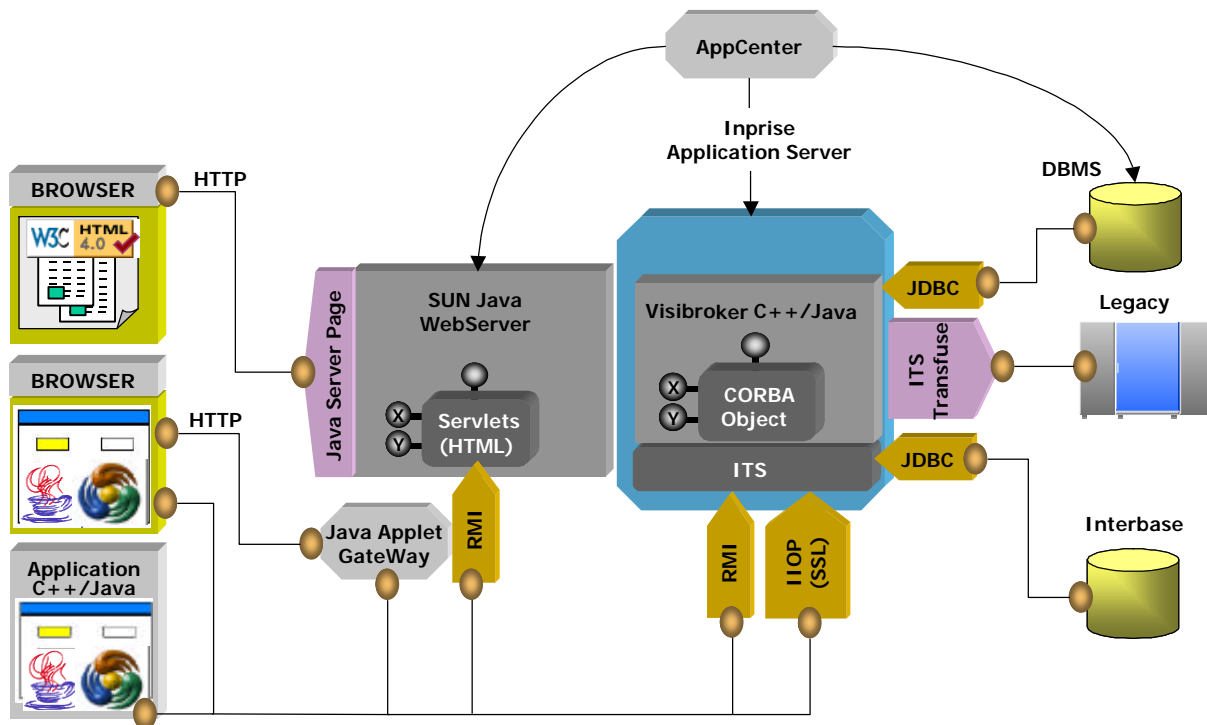
On the administration side, the AppCenter tool provides a wide palette of functions with a user-friendly graphical interface. Following a fully assisted initial configuration phase of the application, AppCentre provides a centralized view on the operating status of various portions of a distributed application (Corba and Entera objects, middleware services, databases, executable programs, etc.). An application can be stopped or started with a simple click, and scripts can be launched on events or the calendar. AppCenter can also be used to configure the implementation of load balancing and failover.

The effort put forth for Corba has unfortunately been to the detriment of Web technology integration. The product includes Sun Java Web Server, which supports servlets and JSP, but the assistance provided by JBuilder for the development of servlets is scant and no JSP or HTML development tools are provided.

Inprise is hoping to orient itself toward EJB technology for object modeling. The integration of an EJB server is planned for Visibroker (code name Kodiak) for the summer of 1999. The product is missing a high-level object development tool. In fact, if assistants are used to quickly develop Corba applications, the only way to create true object models while benefiting from the framework is through IDL interface development.

The Inprise product is one of the best integrated solutions and the most complete for development, administration and deployment of Corba applications. While Inprise took full advantage of Borland's expertise in development and Visigenic for Corba middleware, it did not make much of an effort to provide help for using Web technologies. The implementation of such a solution, even if it has a productive and high-quality graphical environment requires heavy investment in terms of ability, and therefore must be justified.

## Architecture

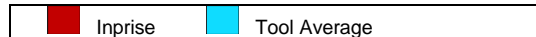


*Inprise Architecture*

The Pros	The Cons
----------	----------

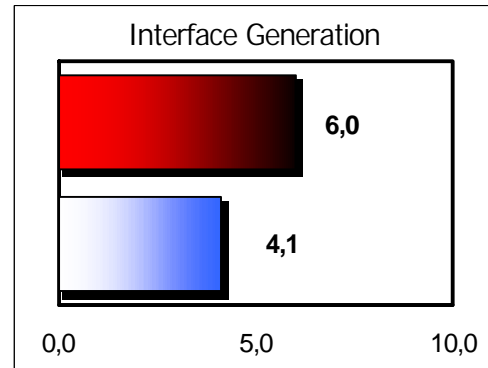
- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>▶ Complete transactional environment (ITS)</li> <li>▶ Good integration of the various tools</li> <li>▶ Completeness of the product</li> </ul> | <ul style="list-style-type: none"> <li>▶ Ineffectual integration of Web technologies</li> <li>▶ No object persistence</li> <li>▶ The JTA API (Java Transaction API) is not supported</li> </ul> |
|--|---|

## Positioning



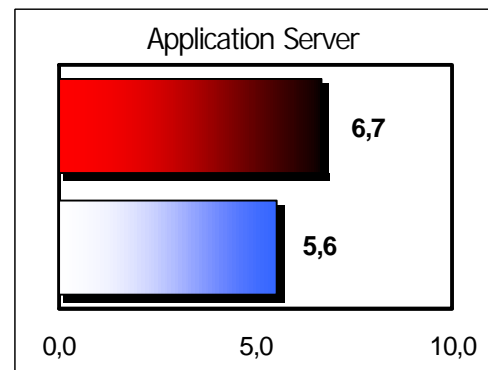
By providing all elements indispensable for developing enterprise applications, JBuilder 2 appears to be a good development tool. The PVCS version management tool is integrated into the environment. This IDE will satisfy developers who want to stay close to their code and properly integrate existing Java developments.

Unfortunately, there are no HTML or JSP development solutions, which does not make it easier to develop Web applications.



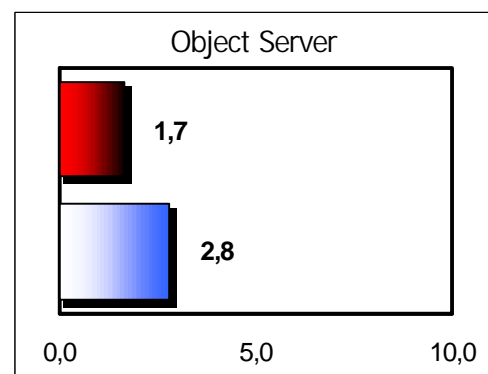
Based on the Visibroker ORB, IAS provides full Corba framework, including name, event and transaction services. The management of these services and deployment of components is accomplished through a graphics console. Applications are administered and monitored by AppCenter. This tool has very rich functions (configuration of load balancing, failover, launching actions from the calendar or events, etc.) and is user-friendly.

However, the pool of JDBC connections is unfortunately too limited in functionality and integration of Web technologies is too fast.



The transactional environment provided by the server is irreproachable. It controls areas as diverse as the OTM, connections to databases and calls to central systems (CICS, Tuxedo, IMS/TM, MQSeries) in a unique transactional environment that is managed graphically.

By passing on an object modeling tool and persistence framework, the product makes module development difficult. Nevertheless, the product does provide a solid infrastructure for the EJBs, which should eventually be supported by the server.



## Technical Specifications

<b>Development license</b>	Approximately \$6000 (with JBuilder) \$3,400 (without JBuilder)
<b>Deployment license</b>	From approximately \$13,700 to \$19,400 (rate per processor per server)
<b>Configuration for the evaluation</b>	Windows NT 4 sp3 (DELL Bi Xeon 1GB RAM)
<b>Additional modules</b>	Contact the editor



## 5.2.5. Oracle: JDeveloper 1.1 – Oracle Application Server 4.07

Identification	
<b>Product composition</b>	JDeveloper, Oracle Application Server, Oracle 8
<b>Version</b>	1.1 (JDeveloper), 4.07 (OAS)
<b>DBMS supported</b>	Oracle, ODBC, JDBC
<b>Development platforms</b>	Windows 95, 98, NT
<b>Deployment platforms</b>	Windows NT, Unix
<b>Editor</b>	Oracle

### Introduction



Oracle resolutely turned toward Java from the first hints of intranet developments. However, the strong base of Oracle database servers installed, linked to the PL/SQL language, and the immaturity of Java technology regarding the graphical interface were and still are consequential obstacles for the editor when trying to convince its clientele to change technology and architecture from one day to the next.

Nevertheless, to give everyone the opportunity to make the move toward Java, Oracle integrates this new language and its associated technologies into all levels of the architecture and into most development and deployment environments.

The two key actions regarding the repurchase of Borland's JBuilder technology and the addition of Java in Oracle relational databases resulted in Oracle 8i. Borland-Inprise, with JBuilder, provides a mature development environment for the design and creation of applications with Java interfaces. Oracle drew on this IDE by adapting it to its product. This is how JDeveloper came about.

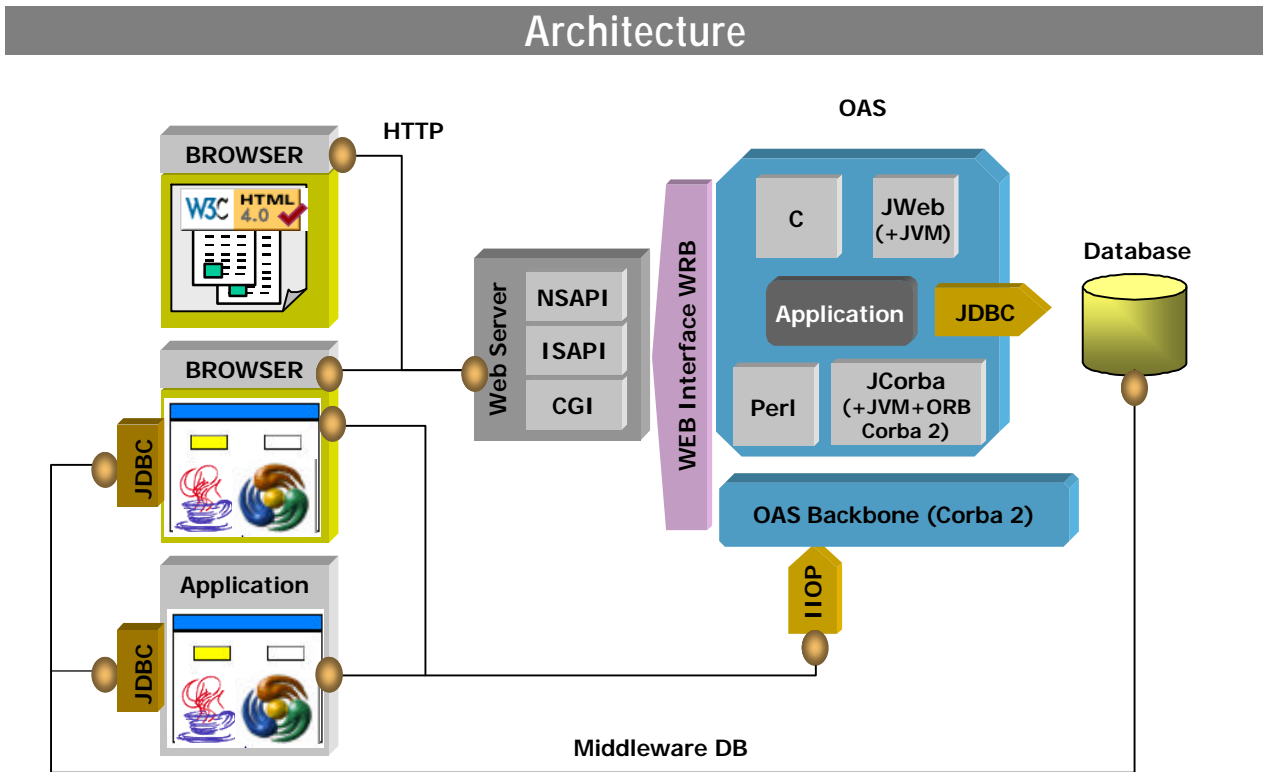
On the server side, Oracle now places Java on the same level as PL/SQL in its RDBMS. While PL/SQL is still present, since it provides both upward compatibility in version 8i of Oracle with all previous versions and guarantees in terms of potential and performance, the addition of Java is an undeniable benefit because this language adds new development abilities due to rich API and its object concepts.

Oracle 8i was not evaluated for this report because the current versions of JDeveloper and Oracle Application Server do not allow users to fully benefit from the new potential of Oracle 8i database-object server. For a truly integrated product, we will have to wait for version 2 of JDeveloper and version 4.08 of Oracle Application Server.

Oracle Application Server, which will act as middleware between the interface and server processes (routing, load-balancing, administration and failover services), is therefore currently the central element of the NCA architecture from Oracle. With an application server and Corba and EJB object servers, Oracle Application Server 4.07, evaluated here, contains all the applicative server logic located in front of the relational database.

Derived from Web Application Server 3 created by the same editor, Oracle Application Server 4 clearly positions itself as an HTML and JAVA transactional application server. With the help of third-party development tools, or with WebDB or JDeveloper, OAS 4 provides the necessary functionality for generating dynamic HTML pages with its automated context management. To accomplish this, the application server relies on PL/SQL, Java, C, etc. processes, which are interpreted by Oracle modules known as cartridges.

The other concern is about Java interfaces. At this level, Oracle has a mature development environment with JDeveloper in its product. In this type of architecture, JDeveloper, which can also create two-tier Java applications, will help the developer design the interface and generate the server processing code which will be placed in Oracle Application Server. It will rely on JCorba cartridges for this and the runtime motor relies on a JVM and a Corba ORB, which are included in the product.



Oracle Architecture

While waiting for the arrival of 300% Java from Oracle (JDeveloper/OAS/Oracle 8i), OAS is Oracle's application server and contains all server processes, except for the procedures stored in the database.

Oracle provides various types of architectures for the interface. For HTML intranet mode, the user will be able to choose the development language from among those provided by Oracle. Note that if PL/SQL is used, the Oracle database engine will execute the code instead of OAS. Oracle now makes it possible to use HTTP servers other than its "in-house" products. For Java interfaces, the application may be based on two-tier architecture, or in the form of a Java applet or a Java application that accesses the database. In multi-level architectures, Oracle Application Server will act as application server with the dialog between the interface and the application server handled via Corba/IIOP.

### The Pros

- ▶ Complete product
- ▶ Covers all types of architecture
- ▶ Supports Corba 2
- ▶ Rich product from the editor

### The Cons

- ▶ Clarity missing in the product
- ▶ Lack of maturity of Jdeveloper 1.1
- ▶ Limited support for object server functions
- ▶ Problems with integrating between the various components of the architecture



## Positioning



JDeveloper is a development workshop for Java applications. A WYSIWYG tool is used to design the Java interface. As a foundation, it includes all the functionality of an IDE (debugger, compiler, source editor, object inspector, assistants, etc.). In addition, JDeveloper opens naturally to Oracle Application Server making it easier to develop multi-level applications.

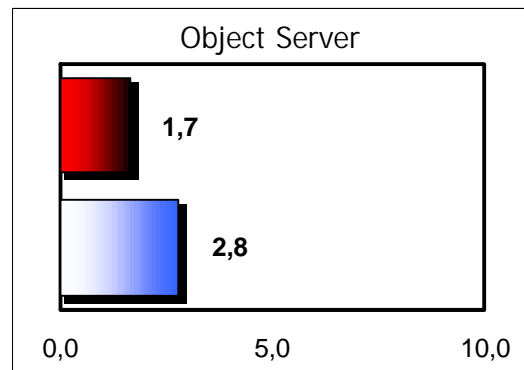
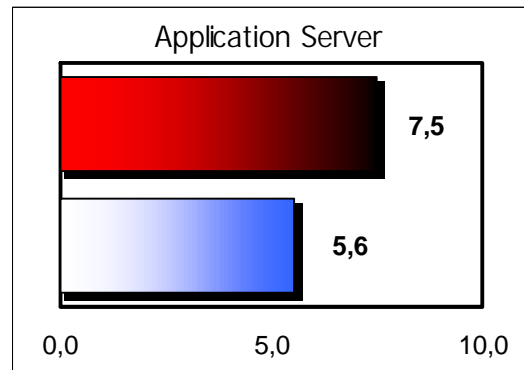
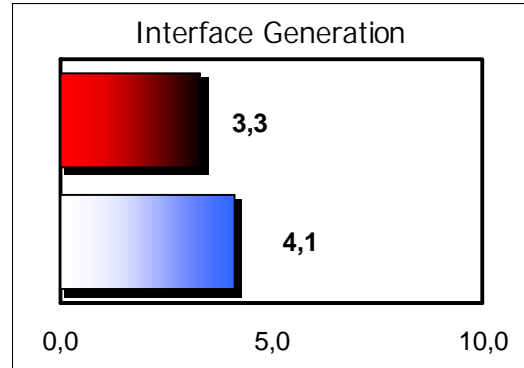
Its negative points include the youth of the product (version 1.1), which only has minimal support of HTML interfaces due to the absence of Swing components and assistants that are still incomplete. Version 2.0 corrects many of these shortcomings.

The strength of the Oracle product is in the potential of its Oracle Application Server. Formerly known as Web Server and Web Application Server, Oracle's application server distinguishes itself with its extended palette of programming languages.

While PL/SQL and Java are the most popular languages, nothing stops developers from using C or Perl to create their applications. Another strong point is the openness of the application server. This is due to the richness of APIs provided, even though additional tools may sometimes be required.

In the absence of Oracle 8i, Oracle Application Server only provides limited functionality for object servers. OAS is nevertheless a Corba 2 object server because a proprietary Corba ORB is included in the product and because the JCorba runtime engine instantiates and manages its objects. It also includes an EJB container.

However, despite the implementation of OTS, Oracle Application Server does not provide assistance with object persistence support and therefore too little for transactional. JDeveloper does not currently help the developer with modeling of business object graphs.



## Technical Specifications

<b>Development license</b>	JDeveloper 1.1: approximately \$2400 per developer workstation
<b>Deployment license</b>	Oracle Application Server 4.07: approximately \$725 for 8 users
<b>Configuration for the evaluation</b>	JDeveloper 1.1, Oracle Application Server 4.07
<b>Additional modules</b>	Oracle 8: approximately \$11,300 for Windows NT



## 5.2.6. Symantec: VisualCafé 3 Database Edition

### Identification

<b>Product composition</b>	Symantec VisualCafé Database Edition
<b>Version</b>	3.0 for VisualCafé, 1.1 for dbANYWHERE
<b>DBMS supported</b>	ODBC, JDBC
<b>Development platforms</b>	Windows 95, 98, NT
<b>Deployment platforms</b>	JVM
<b>Editor</b>	Symantec

### Introduction



Symantec was founded in 1983. With sales of \$575 million in 1998, Symantec is the seventh largest editor of PC software in the world. Symantec sells three categories of products: security and help, remote productivity solutions and Internet tools.

Symantec VisualCafé was one of the first Java development tools. Its environment is well integrated and learning it is easy. All the ingredients of an IDE worthy of the name are included: graphical interface editor, source editor, debugger and assistants. An external controller is however required to manage development. In addition to Swing components, Symantec provides a rich palette of proprietary components (multimedia, database components, etc.) to satisfy demanding developers. The visual programming tool is not very mature, but it is useful for development. Other interesting characteristics include an environment based on an open API (OpenAPI) and the ability to extend the IDE to create specific developments. It is also possible to compile Java sources in native Win32 code for increased performance.

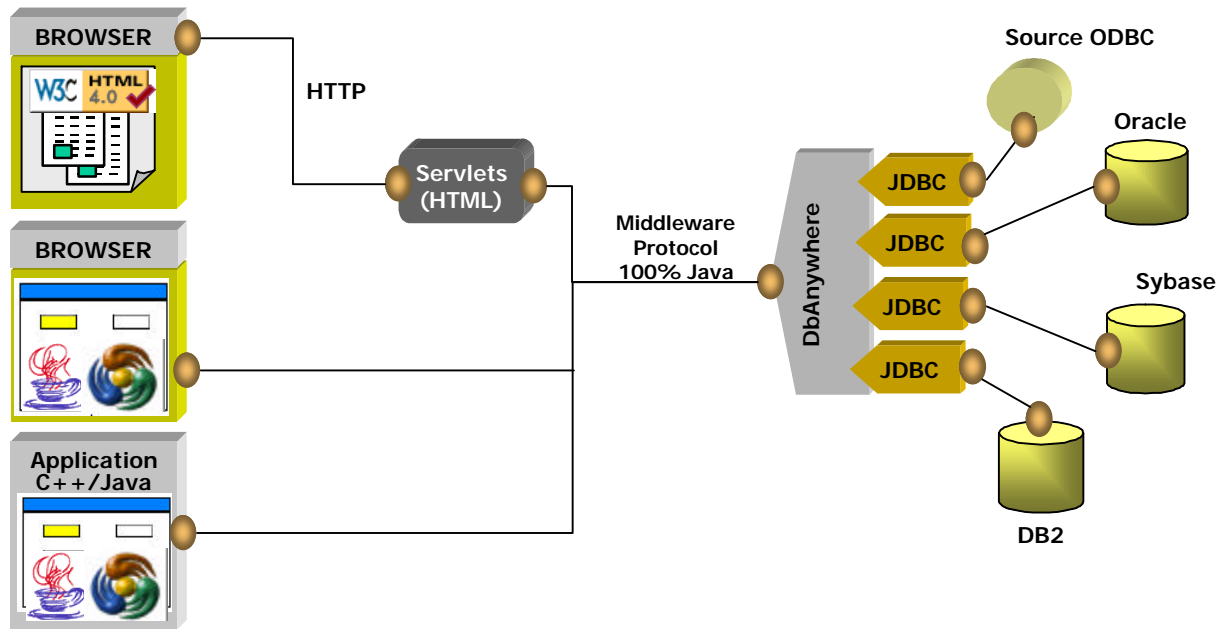
The advantage of this product is undeniably the generation of Java interfaces for accessing databases. Creation of these interface is entirely assisted and the resulting developments are easily maintainable and extendable with JavaBean components carefully designed by the editor.

VisualCafé is not well suited for generating dynamic HTML pages from servlets. However, the product does include an HTML development tool known as VisualPage, but it is primarily for creating static pages. The developer must handle context management.

A JDBC server, dbANYWHERE, based on a type 3 JDBC driver is provided. Based on three-tier architecture, it facilitates database access for Java interfaces. It also optimizes the traffic generated between the server and the presentation level. Its administration console is easy to adopt. However, dbANYWHERE is not recommended for loading and does not provide a solution for error recovery.

The next version of the product will include support for Corba, RMI and EJBs as well as a remote debugger (VisualCafé Enterprise RAD).

## Architecture



*Symantec Architecture*

As a Java development tool from the outset, VisualCafé was received well by developers. Symantec only provides one server-side software application -- dbANYWHERE. This is a JDBC server for small architectures.

The absence of an application server in its product has forced Symantec to develop partnerships with the appropriate players who, for the most part, did not have an integrated IDE. Symantec works with BEA and Sun to integrate their IDE with WebLogic and NetDynamics.

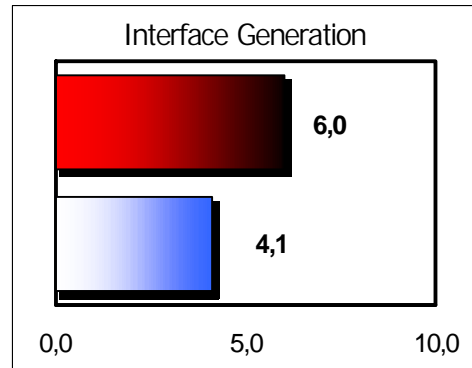
<h3>The Pros</h3>	<h3>The Cons</h3>
-------------------	-------------------

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>▶ Productive IDE</li> <li>▶ Development of DBMS interface access very elaborate</li> </ul> | <ul style="list-style-type: none"> <li>▶ No support for distributed objects</li> <li>▶ No application server</li> </ul> |
|---|---|

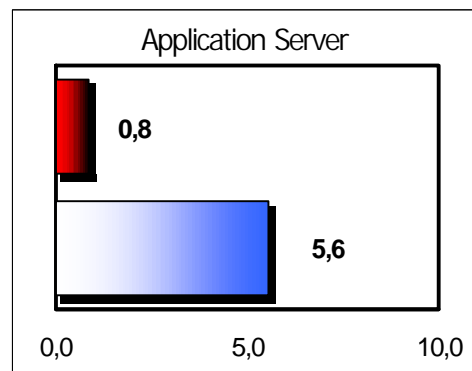
## Positioning



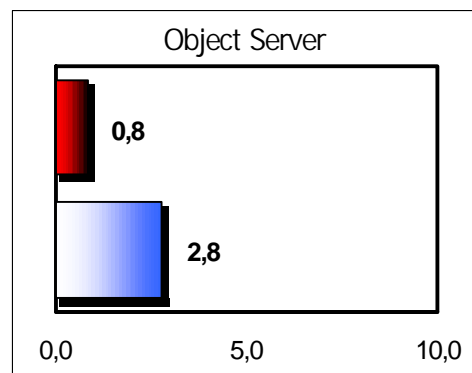
The IDE interface will be very familiar to Windows developers, since it includes the form, object inspection window and project concept. The design of Java interfaces is done entirely in WYSIWYG. Numerous assistants are included. Very good assistance is provided for the creation of Java interfaces for accessing databases and it proves to be extremely rich from a functional point of view. There is too little support for designing HTML interfaces from servlets.



The low score obtained by Symantec in this category is due to its JDBC server based on a type 3 JDBC driver. This is not an application server at all, but is used only to optimize access to JDBC sources. This JDBC server is easy to implement, but is not designed for heavy architectures.



While VisualCafé does not have an object server, it does provide a graphical view of the hierarchy of Java classes, which can help developers when they create object models. Symantec is working with editors of application and object servers to interface their IDE with their products (NetDynamics and WebLogic).



## Technical Specifications

<b>Development license</b>	Approximately \$895
<b>Deployment license</b>	N/A
<b>Configuration for the evaluation</b>	Windows NT 4 sp3 (DELL Bi Xeon 1GB RAM)
<b>Additional Modules</b>	



## 6. Evaluation of the Market Leaders

### 6.1. Evaluation of WebLogic Application Server 4.0.2

---

#### 6.1.1. Introduction

---

WebLogic is primarily an application server. It offers a wide variety of methods for providing access to business methods in user applications. Its objective is not to provide a workshop for development with which it is possible, for example, to create HTML, Java or C++ user interfaces. WebLogic provides the runtime environment for code on the server side. To implement a graphical interface, technologies able to communicate with WebLogic can be used (HTTP, RMI, Corba, DCOM, JDBC, Java Sockets, T3, etc.).






For HTML interfaces, an engine of servlets and a specific technology, known as JHTML, are used to generate servlets. JHTML lets developers place Java code in HTML pages directly. This Java code is run on the server when a browser requests the page. As output, it produces HTML. This dynamically generated HTML takes the place of the Java code in the page, which is then sent to the browser.

At runtime, the requested JHTML pages are dynamically compiled as servlets, unless a compiled version already exists, in which case this is used. There is an equivalent of the JSPs (Java Server Pages) from Sun in this technology. It is likely that in time BEA will migrate toward this technology and abandon JHTML.

Another tool for generating HTML supplied by WebLogic is htmlKona. This is an object framework used to represent HTML pages through Java graphics objects in memory. This API supplies the corresponding Java objects to the various elements of an HTML page (header, button, field, form, table, etc.). These objects are capable of producing the appropriate HTML code. HtmlKona thereby provides an API that is used for modeling (and generating) an HTML user interface at a high level on the server side.

To help implement Java user interfaces, WebLogic includes a tool named "BeanBar", which groups JavaBean components that implement graphical interface elements (text fields, tables, etc.) created in AWT and able to connect and interact with the application server. To do this, an event-based distributed communication model is provided. Here again, a technological migration of the event model to the JMS (Java Messaging Service), defined by Sun in cooperation with other editors such as BEA, is to be expected. Since the two models are quite different, we recommend not relying too strongly on the event model that is currently provided to avoid a heavy migration phase. The BeanBar can be used with JBuilder 2 and Symantec VisualCafé, 3.

## Ratings:

Generating the interface	WebLogic
Step 1: Event interface	
Step 2: Graphic component creation	
Step 3: HTML interface	
Step 4: Tabular data display	
Step 5: Print Management	

Score from 0 to 3 (☆☆☆: 3; ☆☆: 2; ☆: 1; ☹: 0)

## 6.1.2. Application server

## Step 6: Adapted for distributed objects

In this section, we will contemplate the development of distributed object access. To clearly identify the areas we are exploring, we must know the runtime environment in which the objects we want to access are located. In our case, we want to develop code that runs on the WebLogic application server and accesses remote distributed objects. These remote objects are hosted on an object server. We will place ourselves in the situation of a client in regards to these objects. The capacities of the WebLogic object server are described in another section.

From WebLogic, we can access remote objects via RMI, Corba, COM and a number of specific technologies grouped under the name "T3".

## ➤ T3

Since T3 is a WebLogic model, we can only use it if remote objects are hosted by another WebLogic server. It includes an optimized connection model named "Rich Socket" on which all traffic for accessing high-level services provided by WebLogic may circulate.

A Rich Socket is formed by T3Connection class object. The traffic between the client and the remote WebLogic server moves through this connection. Dependent on the services used, these would be RMI, JDBC, Event or T3Servlet requests that travel over the Rich Socket.

This model provides maximum integration for communication with a WebLogic server and provides access to specific functions such as Events or Workspaces. The latter are used to define rich memory storage environments for server-side remote objects. As an example of a service specifically accessible via T3, the following is a portion of an API for using a Workspace:



```
// adds an object to the workspace by associating it to a key
void store(String key, Object o)

// returns an object based on its key
Object fetch(String key)

// retrieves a record (object/key combination) from the workspace
void remove(String key)
```

## ➤ RMI

In addition to T3, the most natural object communication model from WebLogic is RMI. This model is used to communicate with Java distributed objects. If we master the runtime environment of remote objects, we could use the specific implementation of RMI provided by BEA, which is better optimized than the benchmark implementation of Sun.

Development is done in several stages. The service files for remote communication must be obtained first. These files, which contain the RMI stubs, would have been produced when the remote objects were developed using an RMI compiler (rmic.exe if the RMI implementation from Sun is used, rmic.class if the RMI from BEA is used).

The following step consists of writing the code for using the remote object:

- You must start by connecting to a directory of distributed objects. The remote object should have been saved to the object directory on the network. To connect, you may use Java Naming and its Java registry or JNDI (Java Naming and Directory Interface). Naming is the old interface provided by JavaSoft for saving remote objects, while JNDI is the new standard recommended for enterprise applications. This technology lets you organize objects in a tree, similar to how files are saved in traditional file systems. WebLogic only implements a portion of the functions defined in the JNDI specification. More specifically, the "directory" portion, which is used to assign additional attributes to each object in the directory (for example, a date or permissions), is not implemented. For our development, we have decided to use the "modern" API, or JNDI.
- When you are connected to the directory service, you query it to obtain a reference to the remote object from a public name. In our case, we want to retrieve a reference for an object whose public name is "bargain\_of\_the\_week". The system then supplies a stub instance that represents the remote object in local space.
- Using the stub, you can invoke methods on the remote object. In fact, every method invoked in the stub is sent to the remote object. We must test the possibility of a RemoteException, which indicates a communication problem (always possible).

```
Hashtable ht = new Hashtable();
ht.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.TengahInitialContextFactory");
ht.put(Context.PROVIDER_URL, "t3://localhost:7001");

Context context = new InitialContext(ht);

Automobile auto = (Automobile)
context.lookup("RMICContext.bargain_of_the_week");

System.out.println("Bargain of the week : " + auto.slogan());

System.out.println("id          : " + auto.idaut());
System.out.println("version     : " + auto.version());
System.out.println("cylinder   : " + auto.cylinder());
System.out.println("price      : " + auto.price());
System.out.println("warranty period : " + auto.periodWarranty());
```

### ➤ **COM**

COM objects can be accessed, which will be very useful if you want to use components implemented according to this model in a WebLogic program. Unfortunately, the documentation provided for developers to implement this solution is lacking.

### ➤ **Corba**

Corba objects can be accessed, but this is not documented extensively. It is useful for accessing objects developed in languages other than Java and to connect to the numerous existing systems using a Corba interface. The interaction between WebLogic Application Server and WebLogic Enterprise (formerly known as M3) can also be accomplished with Corba.

For development, a Corba ORB is required (WebLogic does not provide one). BEA took certain precautions to ensure that its Visibroker product functions properly.

Programming with Corba is traditional and resembles RMI:

1. Initialize the ORB
2. Obtain a reference pointing to the naming service
3. Obtain a reference to the "bargain\_of\_the\_week" object from the naming service and ORB
4. Use the remote object that represents the bargain of the week by sending it messages

However, development is more complex than with RMI. IDL language must be used to describe objects and integration with the Java language leaves a lot to be desired.

## Step 7: Database access

Since it was created in 1995, WebLogic, now an integral part of BEA, was involved in accessing databases with Java. The method used to access is JDBC, for which a number of drivers supplied. Also included in the package is an evaluation version of the CloudScape database, which is a hybrid object-relational database. Naturally, it is also possible to connect to other databases.

JDBC is an API for accessing data that resembles ODBC and is used with Java. It is used to send SQL queries to a database and to manipulate the results that are returned in ResultSet objects.

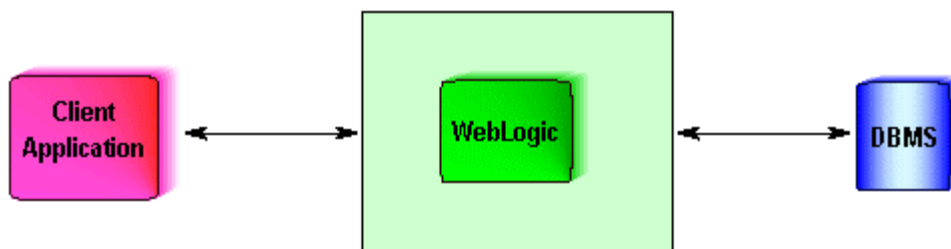
When developing JDBC with WebLogic, you must select the architecture and type of JDBC driver to be used. There are four types of JDBC drivers.

For two-tier development, the client application accesses the database directly and runs applicative processes. In this model, the client application must recognize the relational model in the database. SQL handles the interaction. This architecture is illustrated below:



*Two-tier architecture: The client communicates with the database using SQL language.*

For three-tier development, the client application communicates with the WebLogic server through interfaces for applicative services. These applicative services, which may be high level, can be used by various client applications and may represent business processes that are common to a group of applications. Factored in this way at the server level, the company's industry logic is more easily mastered and can develop without requiring deployment on client workstations. They communicate with the WebLogic server using various technologies including distributed objects, sockets, HTTP/HTML, type 3 JDBC, etc. In this model, WebLogic accesses the data sources:



*Three-tier architecture: The client communicates with the application server via RMI, HTTP, Corba, RPC, etc.*

To communicate with the database, WebLogic uses JDBC. During development, the first step consists of selecting the type of driver. The four choices are described in detail in Chapter 1.2.

### ➤ Implementation of pooling

The connection pool is a key function of WebLogic. Using this technique, an application can support a very high number of users while providing communication with the database over a limited number of connections.

The implications of pooling are not all beneficial. For example, you can no longer rely directly on the access control functions of the database because only WebLogic is truly a client of the database. The concept of user identity for the database is lost. Control must now be implemented for accessing the application server. However, the advantages of pooling far outweigh its disadvantages.

Configuring a pool of connections consists of defining a group of properties for the pool such as the initial number of connections, the maximum number, the method for managing unused connections, etc. ACLs (Access Control Lists) can be used and ultimately stored in an LDAP directory or in a database. To use the pool with WebLogic, a text file named "weglogic.properties" file must be created (this file is also used for many other tasks and quickly becomes a main element for development):

```
#definition of a pool of connections
#####

weblogic.jdbc.connectionPool.sqliPool=
    url=jdbc:weblogic:oracle,
    driver=weblogic.jdbc.oci.Driver,
    loginDelaySecs=1,
    initialCapacity=4,
    maxCapacity=10,
    capacityIncrement=2,
    props=user=pjav;password=pjav;server=pjav

#Definition of an ACL
#####

weblogic.allow.reserve.weblogic.jdbc.connectionPool.eng=sqli,administra
tor
```

### ➤ Accessing the database

JDBC use is traditional:

- Configure access
- Obtain a connection
- Create and execute a query
- Manipulate the results
- Free resources

```

Properties props = new Properties();
String driverClass = weblogic.jdbc.oci.Driver;
String driverURL = "jdbc:weblogic:oracle:DEMO_TCP";

props.put("weblogic.t3.connectionPoolID", "sqliPool");

Connection con = null;
Statement stmt = null;
ResultSet rs = null;

try
{
    //Load the driver
    Class.forName(driverClass).newInstance();

    //Retrieve a connection
    con = DriverManager.getConnection(driverURL, props);

    //Create the Statement object
    stmt = con.createStatement();

    //Execute the query
    rs = stmt.executeQuery("select * from automobile");

    //Display the result
    int count = 0;
    while(rs.next())
    {
        System.out.println(rs.getString(2));
    }
}
catch(Exception ex)
{
    ex.printStackTrace();
}
finally
{
    // free resources
    close(rs);
    close(stmt);
    close(con);
}

```

What happens if, when a connection is requested, all the connections in the pool are already taken and the maximum number of connections for the pool has been reached? The default action is to wait for a free connection. A call to the "DriverManager.getConnection()" method could therefore block connections for a while. Maximum waiting time can be configured over the maximum of the DriverManager if it could not provide a connection. For example, to wait ten seconds, the following instruction is added:

```

| props.put("weblogic.t3.waitSecondsForConnection", "10");

```

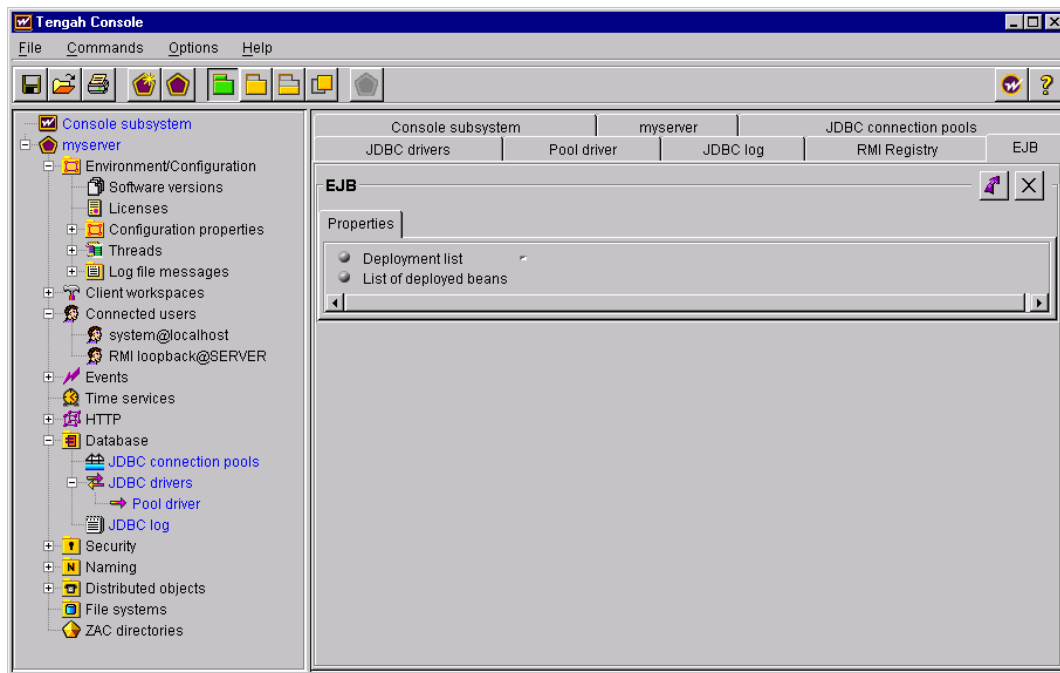
## ➤ Advanced functions

At the time of our tests, the drivers provided conformed to the JDBC 1.2 specification. For more complete ResultSet objects, drivers that conform to specification 2.0 are required.

The 2.0 drivers were being validated and may be available by the time this study is published.

The XA standard and two-phase commit were not supported.

However, the graphics administration console may be used to inspect the server configuration:



*The graphics console of WebLogic Application Server*

## Step 8: API richness

WebLogic does not provide support for multimedia or for sending mail beyond what is standard with Java.

## Step 9: Workload management

WebLogic Application Server is positioned as an enterprise solution that can support heavy loads. To do this, it includes numerous services and optimizations.

The database connection pool has already been mentioned. We will now have a look at other aspects related to supporting large-scale multi-user applications:

## ➤ WebLogic RMI

The implementation of RMI by SunSoft and supplied in JDK is notoriously slow and its performance degrades significantly as the number of users increases. To resolve this problem and provide a system of distributed objects for the enterprise, WebLogic provides its own implementation of RMI. However, the Sun reference implementation may still be used if desired.

BEA indicates that its implementation fully supports the RMI semantic. However, in practice this is not so simple. For example, up to version 4 of the product, the implementation from BEA did not use the distributed garbage collector even though it was an integral part of the RMI API. The session-based model provided was not adapted to all contexts. Without technology for managing the life cycle of objects adapted to a distributed environment, serious large-scale applications could not be developed. Based on its application, the WebLogic implementation meets the requirements.

WebLogic's RMI differs from the reference implementation in many ways. It provides more freedom and simplicity in development by removing some major design constraints imposed by the traditional RMI and recognized as very problematic (inheritance, exceptions, etc.). The following table summarizes the key differences in workload support:

Component	SunSoft RMI	WebLogic RMI
Connection management	Several sockets are adapted for communication between a client, a server and the RMI Registry.	Only one connection between the client and server. The RMI dialog travels over this connection as do JDBC, Remote Calls and Events.
Objects passed by value	Serialization	Optimized implementation of serialization.
Management of co-allocated objects	Objects allocated in the same VM can sometimes communicate via their stubs/skeletons.	Objects allocated in the same VM communicate directly.
RMI Registry	External application, separate.	Functionality handled by the application server.
Stubs	Standard RMI stubs.	"Smart Stubs", able to implement load distribution strategies on several application servers and provides failover.
RMI Compiler	A Stub/Skeleton pair class generated for each remote class.	A Stub/Skeleton pair class generated for each remote interface.

## ➤ Thread pooling, object pooling

Creating Java threads can be a slow process. To solve this problem, WebLogic uses a thread-pooling technique. When a thread terminates, it is again made available in the pool and can be reused.

In the same way, the various EJB components are managed in a pool. This is mostly transparent to the developer. Caution is however still required. When an EJB has been taken out of the pool to be used, its instance variables are not initialized to default values, but when an object is allocated, these variables are initialized. The developer should therefore always initialize all instance variables in object constructors.

➤ **ZAC (Zero Administration Client)**

ZAC is an infrastructure used to automatically deploy new versions of client Java applications. Client applications include technology that can test the update of an application on a server and automatically restore it over an intranet or the Internet in a manner that is transparent to the user. The communication protocol, HTTP Distribution and Replication Protocol, was designed to optimize communications by minimizing the size of data to be transmitted. The use of ZAC is not transparent to the user. On the client side, the application uses a specific API to control its update and, on the server side, new versions must be packaged in a specific manner.

➤ **The testing phase**

WebLogic applications are for all intents and purposes always to be used concurrently by multiple users. This is why development must include multi-user-testing phases. If an application is the least bit complex, especially if it is transactional, and if a number of users will probably consistently request, the only solution is to implement a testing tool that can simulate loading. There are two objectives: to validate programming (functionality, bugs, concurrency problems, deadlocks, etc.) and to provide the information required for optimization.

WebLogic does not provide any tools for this. A third-party tool must absolutely be purchased.

➤ **Clustering: load-balancing and failover**

Clustering is a major advance in version 4 and a key element for workload management. The technology is an important first step in BEA's ambitious vision in this area. It implements architectures composed of multiple servers that can distribute the work and replace each other in case of problems.

Such a group of servers, known as a "cluster", can be configured dynamically. This makes it possible for a server to join the group or leave using the TSAP protocol (WebLogic Service Advertisement Protocol).

Moreover, it is interesting to note that a server can run on specific architecture and a different operating system. Considering that WebLogic was written in Java, clustering services do not directly depend on the functionality provided by the OS, but on the Java platform itself.

Load-balancing (load distribution) is for the most part automated. In this version, "clusterizable" services include EJB, RMI and servlet. BEA has announced that additional services, such as connection pools, will be included in this architecture in the future. Administration tools are also in development.

Cluster architecture will be interesting in that it will improve overall system performance and support some types of breakdowns. From a theoretical point of view, the concept of tolerance for breakdowns is huge and includes simple monitoring services and automatic restarting, and even extended and extremely complex possibilities of continuing service based on, for example, state replication and synchronization. Therefore, when an application server is said to be "breakdown tolerant", this means nothing in itself. What must be determined is the types of breakdowns taken into account and how.



A distinction must be made between equipment and software breakdowns. In the case of equipment breakdown, the two basic elements in a tolerance system are the cluster (multiple machines) and a reliable method for shared storage. In the case of software breakdowns, cluster architecture can be insufficient by itself. Suppose that a portion of the code is incorrect and this provokes a crash of the virtual machine when it is executed. Re-executing the code on another server would provoke the same problem and the entire cluster would collapse. In such a case, an infrastructure that would allow implementing a robust software program is required. The robustness of a system obviously depends on the robustness of each separate component. However, it also depends on the system's ability to handle a lack of robustness in one of its components. Such considerations lead to the formalization of precise rules for designing and implementing subsystems such as the "rule for robust design". Since requirements for robustness are important, teams of specialists in this field should be called upon.

Let's return to WebLogic and tolerance for breakdowns. The system makes it possible to automatically replace a server that has broken down. Requests from clients made after the breakdown are re-routed (since they would have initially taken another path) to another server.

What happens to work in progress on the server side when it breaks down? In this architecture, the work cannot be completed. The client application must therefore be ready to receive notification of the error instead of the result of the request. It must be able to restart the request.

In a general sense, the issue of applicative coherence comes into play because the process requested when the request failed was:

- Not complete
- Completed partially
- Completed fully

In the case of a partial completion, serious coherence problems could result. To avoid this, a transaction programming model must be used such as, for example, the EJB.

Furthermore, a shared system with state persistence must be used (generally, we rely on the database).

If there is an error, you should check whether the transaction was completed on the client. To do this with WebLogic, you would need to install a parallel system to identify transactions and track those that have been completed. By accessing this system, the client should be able to obtain the desired information.

This is just about it for general cases, but there are still certain types of requests that are easier to manage. For example, an object method can be strictly a calculation that does not modify or save any type of state on the server. These are idempotent methods. In this case, the request can always be sent again if it failed without worrying about coherence. Better still, you can indicate to WebLogic that methods of a certain class are idempotent and, in this case, the WebLogic infrastructure handles resending the call itself.

The development of an application that implements the breakdown tolerance architecture provided by WebLogic requires specific programming. The documentation provides an outstanding introduction to the problem and very useful examples. Compilation is done using certain specific flags that, for example, indicate that the methods of a class are idempotent

(unfortunately, it does not seem possible to go down to the level of each method). Certain attributes of the deployment descriptor for EJBs are used to configure the failover operation.

As far as service continuity for HTTP servlet applications, WebLogic can make the session context persist in a database or file system (with some limitations). This context can be retrieved by the emergency server in case of problems. This ability is commonly known as "session-level failover".

In conclusion and confining ourselves to a theoretical viewpoint, we feel that WebLogic Application Server provides, due to its cluster architecture, good load-balancing capabilities and basic failover capabilities which are nevertheless quite good if compared to those provided in competitive products.

## Ratings:

Application Server	WebLogic
<b>Step 6: Openness for distributed objects</b>	☆☆☆
<b>Step 7: Database access</b>	☆☆
<b>Step 8: API richness</b>	☆
<b>Step 9: Load-balancing management</b>	☆☆☆

Evaluation from 0 to 3 (☆☆☆: 3; ☆☆: 2; ☆: 1; ●: 0)

### 6.1.3. Object server

#### Step 10: Modeling by business object

##### ➤ Choice of architecture

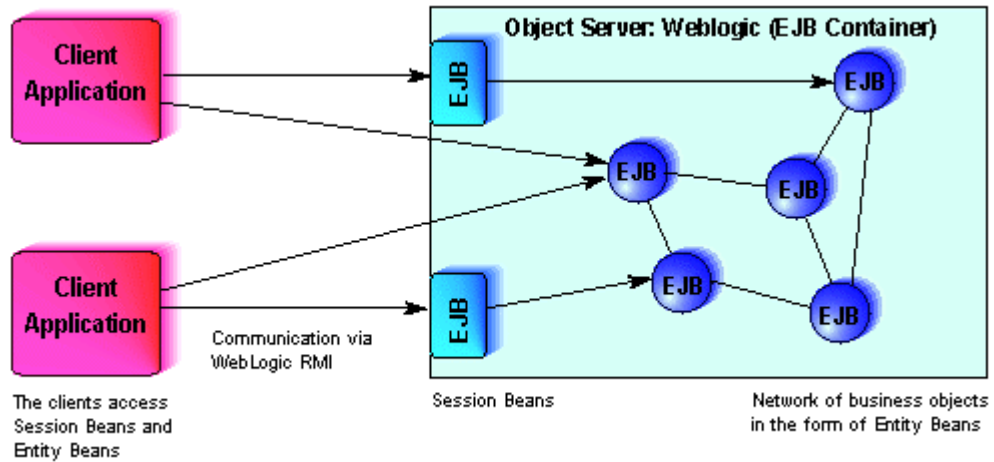
For object modeling, WebLogic uses the emerging EJB technology. The product supports specification 1.0 as well as some extensions such as Entity Beans, which are partially implemented.

The architecture that we chose includes several types of EJBs. Our business objects, Automobile, AutomobileNew, AutomobileUsed and Model, are implemented as Entity Beans. Clients can access these objects. Furthermore, Session Beans are also implemented and provide a certain number of business methods with more granularity.

For example, a specific automobile can be queried and its warranty period can be consulted using a method (fine-grained). A Session Bean can also be asked for a text description for all automobiles with a price lower than a certain value with a single method call (coarse-grained).

The granularity of implemented methods is an important characteristic for a distributed architecture. The impact on performance can be significant. As a general rule, the finer the granularity, the more network interactions there are between clients and the server.

The selected architecture is represented below:



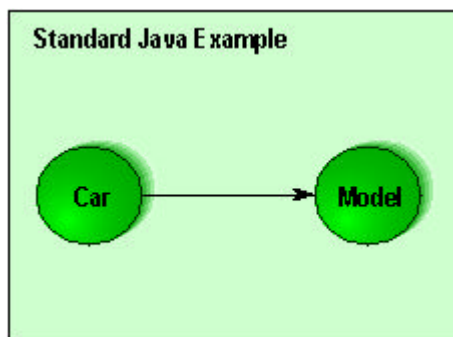
➤ **Creation**

WebLogic does not provide tools for designing and developing object models. The documentation does include some design and implementation advice.

Development of EJBs requires the implementation of several classes and service methods for each business class. It is also necessary to create a certain number of service files such as deployment descriptors or "manifest" files.

Java development is done with JDK 1.1.7. WebLogic has announced their intention to support JDK 1.2 in the future.

Using the EJB model involves specific development of the business object model, which is very different from traditional Java development. For example, the implementation of a relationship between objects is no longer accomplished by simply providing a Java reference; instead a specific EJB mechanism must be used.

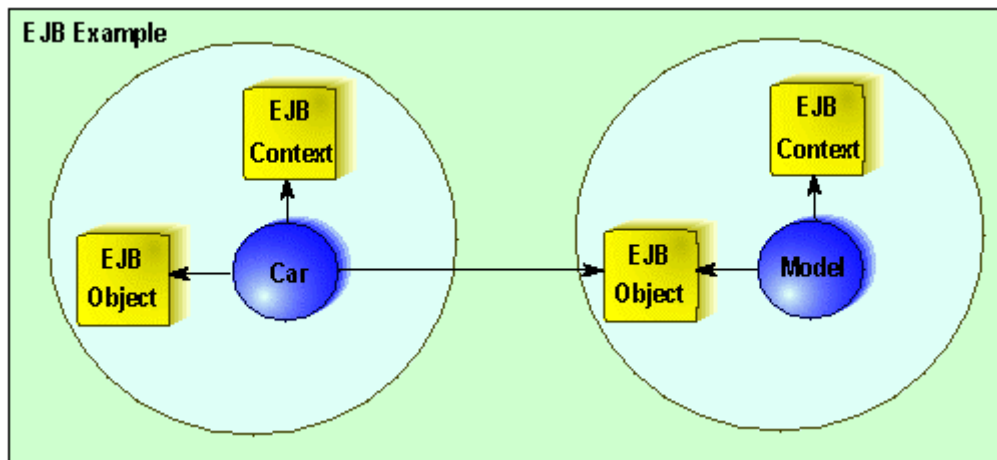


*Implementation of a 1:1 relationship between a car and its model*

To communicate with an EJB, the business object must not be referred to itself, but an object created by WebLogic, the "EJB Object". For each instance of a business object, an instance of an EJB Object is created.

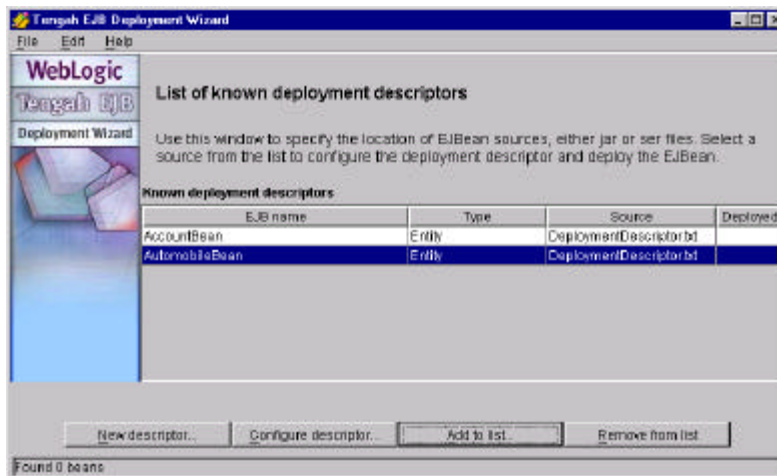
To obtain a first reference to this intermediate object, you must communicate with a third object named "Home Object", for which there is an instance by business object class. How can you access it? Generally, it should have been saved in a JNDI context and it is through this API that it is passed.

During development, you should be careful to never reference a business object directly. Instead, you should always go through its EJB Object. Another object is used for this purpose: the "EJB Context", from which a business object could request a reference to its own EJB Object. Each business object is assigned a specific EJB Context by the system.



Implementation of a 1:1 relationship between a car and its model

When the classes have been developed, they must be compiled and the deployment descriptor must be implemented. To create this descriptor, you must write a text file in a format specific to WebLogic. Next, a utility known as the Deployment Wizard is used to complete the final stages of development.



Openness of the deployment descriptor using the Deployment Wizard

These steps consist of querying various fields using the wizard. It can, through introspection of the classes that make up the EJB, adapt automatically to each specific case and suggest useful assistance through method menus or attributes.

**Configure the entity bean deployment descriptor**

Modify the attributes of the *EntityDescriptor*. Using the tabs, set the EJB and interface classes, the security access control, the transaction control descriptors, the entity bean environment, container-managed fields and persistence.

AutomobileBean | Access control | Control descriptors | Environment | Container-managed fields | Persistence | Errors

EJB class  
workshop.entityEjb.server.AutomobileBean

Home interface class  
workshop.entityEjb.interfaces.AutomobileHome

Remote interface class  
workshop.entityEjb.interfaces.Automobile

EJB home name  
ejb.AutomobileHome

Reentrant transactions

Primary key class  
workshop.entityEjb.interfaces.AutomobilePK

< Back   Next >   Done   Cancel

*Configuration of the EJB*

Access authorizations can be defined for each method:

**Configure the entity bean deployment descriptor**

Modify the attributes of the *EntityDescriptor*. Using the tabs, set the EJB and interface classes, the security access control, the transaction control descriptors, the entity bean environment, container-managed fields and persistence.

AutomobileBean | Access control | Control descriptors | Environment | Container-managed fields | Persistence | Errors

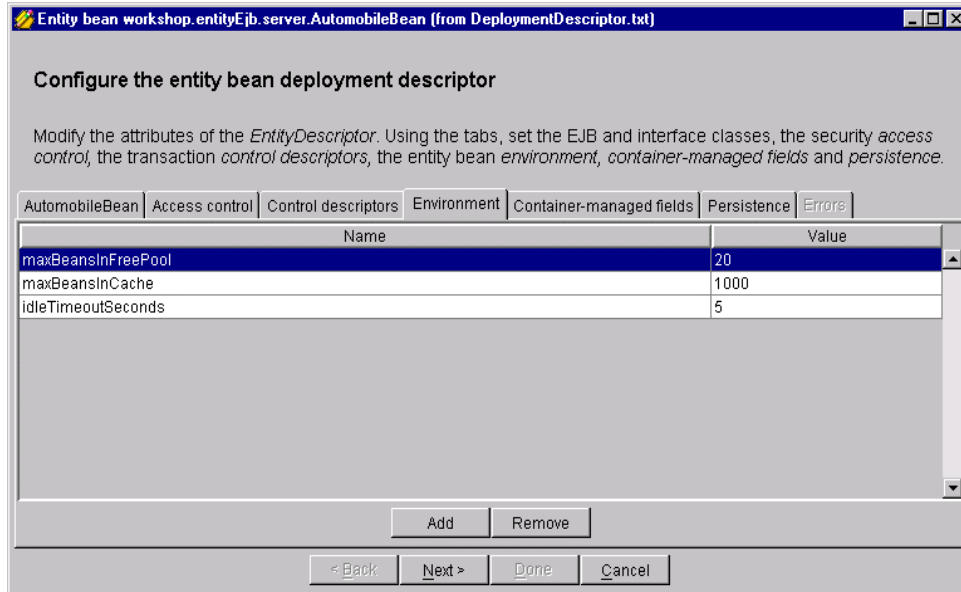
Method	Allowed identities
DEFAULT	
setPrice(int)	joe, karen

Add   Remove

< Back   Next >   Done   Cancel

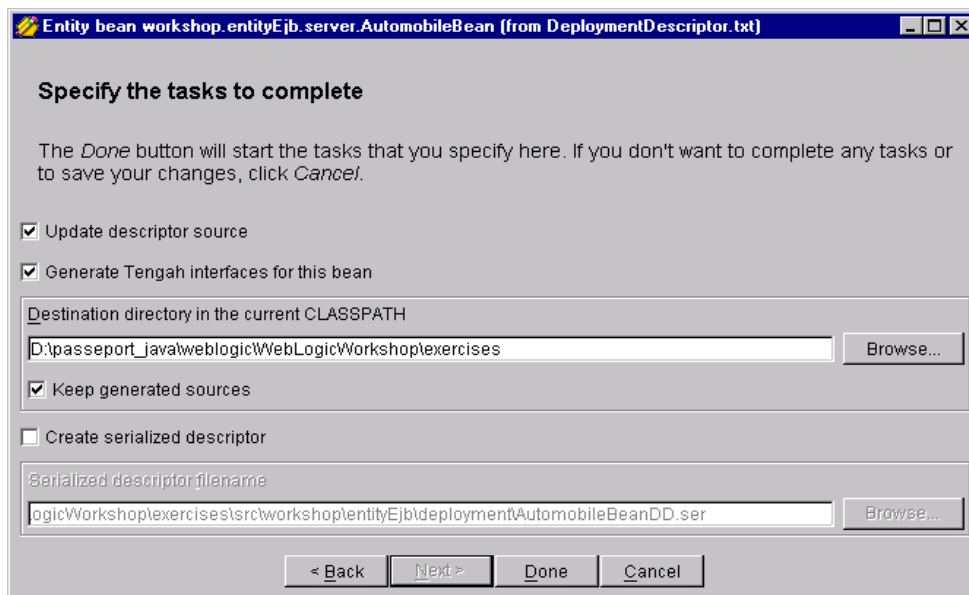
*Definition of an access control list for the setPrice() method*

A certain number of attributes can be implemented to configure the execution of the WebLogic server. For our development, we define a maximum number of Automobile objects in memory, a maximum number in the pool and a maximum duration for an inactive Automobile object to remain in memory before transferring it to disk (activation/deactivation).



*Definition of execution properties for the EJB*

After this configuration step, we will move on to generating files that are internal to WebLogic and which make the EJB function:



*Generation of runtime files*

The CLASSPATH system environment variable may need to be modified to specify the location of new classes.

Finally, for the server to take into account the EJBs developed, we must modify the configuration file to stop and restart the WebLogic server.

The client applications will then be able to interact with the EJBs on the server.

Overall, development of the object model appears very heavy with WebLogic. Every modification involves going through numerous steps again and this takes lots of time. Certain aspects of modeling with EJBs are still troublesome, particularly those involving inheritance.

## Step 11: Importing EJBs

Several steps are involved when importing EJBs:

- Opening the EJB (.jar file) with the Deployment Wizard from WebLogic, which consults the information from the deployment descriptor included in the EJB.
- Entry of deployment information specific to WebLogic. During this phase, which can be complex, we configure a mapping to the database for persistent EJBs.
- Generation of deployment files specific to WebLogic with the Deployment Wizard.
- Manual modification of the WebLogic server configuration file.
- Stop and restart of the server.

For our tests, we tried to import an EJB Entity created with JBuilder from Inprise. Unfortunately, it was impossible to complete the importing. WebLogic generates an error message with little information:

```
| Error loading '\\Perf\RepCom\pmougin\ejbAutmobile3.jar':  
| java.lang.IllegalArgumentException: Illegal value of  
| transactionAttribute
```

We conducted some more tests with JBuilder by specifying various transaction support models for our EJB, but better results were not obtained. We launched a utility named ComplianceChecker, provided by WebLogic, which is used to check deployment descriptors. It did not work. In the end, we were unable to determine whether the problem was with WebLogic or JBuilder. This test was conducted with WebLogic 3.5.

## Step 12: Persistence

Since we decided to develop using business objects, persistence is an absolutely essential element for implementation. Without it, our objects are useless. The issue of persistence is complex and, still today, much theoretical work is being done on the subject.

An important point to remember is that, at runtime, objects are not isolated entities. They mutually reference each other. For this reason, the issue of persistence does not only affect solitary objects, but also a complex network of objects.

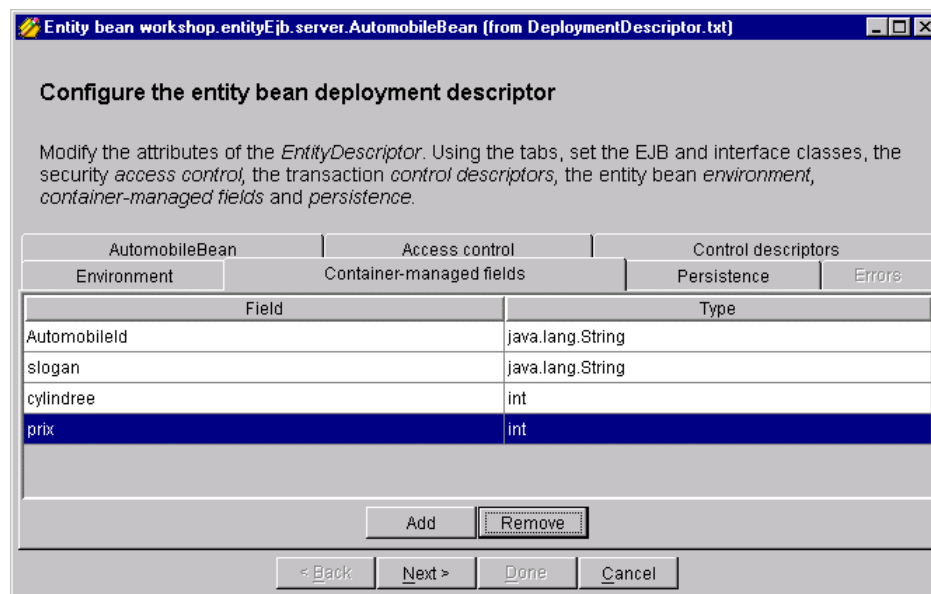
There are many problems to be resolved. How can a network of objects in memory be thrown to disk and vice versa? How can synchronization problems between the network on disk and the network in memory be resolved? How can only those portions of the network used at a given moment be loaded into memory? How can the network be saved to a relational database (technologies known as object-relational mapping)?

Persistence is one of the functional objectives of the EJB initiative. The EJB persistence semantic is based on the Java serialization semantic. The object server can take advantage of the extended introspection capabilities of the language to automatically manage instance variables. By using the serialization semantic, the developer can use a Java keyword to identify an instance variable as temporary data that should not persist.

A persistence engine is required. Its role is to host objects on disk and to provide certain diverse capabilities such as transaction support, recovery, querying, etc. There are currently two types of engines that are suitable for enterprise applications: relational databases and object databases. However, Hypercube technologies are still not well adapted to objects.

Choosing a relational engine makes it possible to include object modeling with existing enterprise data. An object engine provides more modeling freedom, more coherence and representation simplicity and potentially better performance.

The EJB model provides two types of persistence: Bean Managed and Container Managed. In Bean-Managed mode, the developer must implement the code for interaction between his EJB and the persistence engine. This mode should not be seen as a general development model, but rather as the possibility of having control over low-level persistence mechanisms when necessary (in very specific situations when specific actions must be performed). Using this model in a general manner for all EJBs is unrealistic due to the very heavy load and complexity of development.

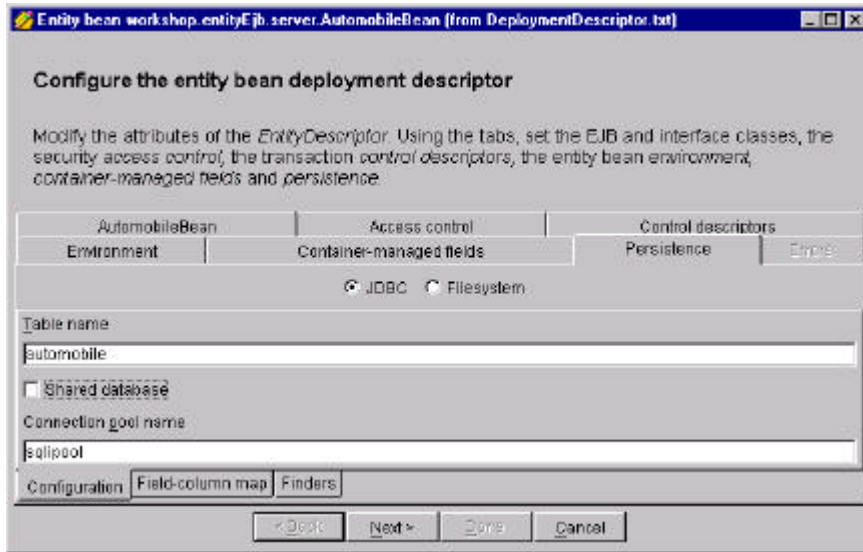


#### *Declaration of persistent fields for Automobile objects*

The Container-Managed mode, which provides automatic persistence, should be used as the general model for EJB development.

If a relational persistence engine is selected, object-relational mapping must be configured during development. Such mapping is an operational definition for objects to throw themselves to tables in the database and vice versa.



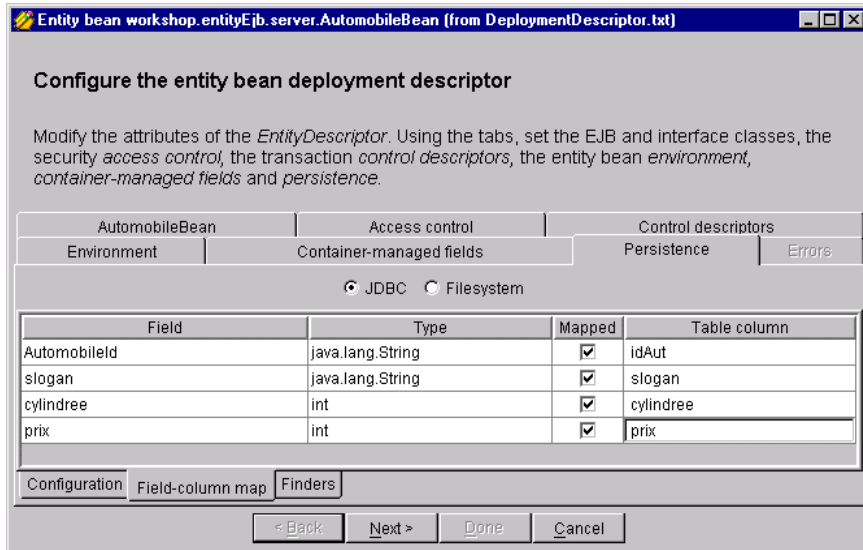


*Configuration of the table and of the pool of used connections*

For example, with O/R mapping, you must normally associate a column in a table to each instance variable of object model classes. Another example is that you must indicate how inheritance is represented in the database. An additional column may be used to code the class, or different tables, or other techniques.

With EJBs, mapping depends on the EJB container used. This means that to import an EJB into a new object server, you must go through this development phase again.

WebLogic provides some persistence capabilities, but they only apply to very simple object models.



*Mapping of instance variables and of columns in the table*

The Deployment Wizard is used to perform object-relational mapping. The mapping is stored in the deployment descriptor in a form specific to WebLogic.

The Deployment Wizard is also used to map "finder methods" from EJBs to database queries. Queries are specified using a specific language whose syntax resembles Lisp.

In practice, for enterprise applications, the object persistence capabilities of EJBs from WebLogic are not suitable.

Some third-party editors who are specialists in object persistence are working on developing object servers that can integrate with WebLogic. The most advanced development is that of Versant. They provide an EJB container based on a powerful and mature object database. However, Versant's future is questionable after a rocky entry on the stock exchange.

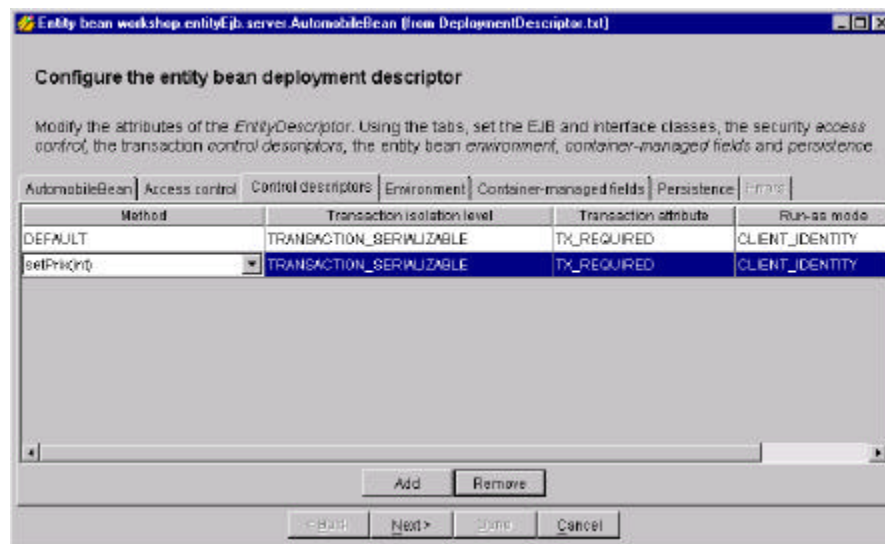
For its part, "The Object People" are developing a module that integrates WebLogic and TopLink, which is a framework for object-relational mapping. At the time this study was published, their products were not yet available.

### Step 13: OTM and object distribution

To distribute business objects with WebLogic, we would choose RMI, Corba or possibly DCOM.

Development is required for all. Specifically, skeletons and stubs must be generated to enable communication between remote objects. If the business objects are EJB components, the natural distribution model is RMI.

WebLogic uses the EJB model for OTMs. In this model, the methods used to start and manage transactions can be specified in large part in a declarative fashion.








*Specification of the transaction model to be used for a call to the setPrice() method for the Automobile EJB*

In Entity Bean mode, the "committed" state of the object network is stored in the selected persistence engine (usually a database). Objects in memory can therefore represent the states during the transaction, the eventual rollback executed when retrieving the last validated state from the persistence engine.

We believe that the technology provided by WebLogic Application Server for object transactions is not yet ready for use for true enterprise applications.

Ratings:

Object Server	WebLogic
Step 10: Modeling by business object	
Step 11: Importing EJBs	
Step 12: Persistence	
Step 13: OTM and object distribution	

Scores 0-3 (☆☆☆: 3; ☆☆: 2; ☆: 1; : 0 )



---

## 6.2. Evaluation of GemStone/J 3.0

---

### 6.2.1. Introduction

---

GemStone does not provide an environment for developing a graphical interface, but only focuses on the server portion. Therefore, in order to build an application's interface, third-party tools such as VisualAge, VisualCafé or an IDE suitable for creating HTML pages should be used. The link between the graphical interface and the server code executed within GemStone is established via Java communications protocols (RMI, socket, etc.) and this has to be done manually. Moreover, no advanced functionality is provided for print control: no generation of PDF files on the server side, no report editor, etc.

Version 3.0 is open to Web HTML applications. Once again, there is no environment for development but only an environment for deployment and execution. Thus, to set up an HTML application, the developer uses the standard servlet model. GemStone/J 3.0 integrates the ServletExec servlet engine from New Atlanta, a firm sharing leadership in this field with Jrun. A JSP compiler is also available for generating servlets.






This technology is simple in its concepts. Each Web page is associated with a Java class, which should implement a number of methods for interacting with the Web server. During a request, an object of the class corresponding to the requested page is instantiated and a method is called with optional HTTP arguments passed as parameters. The object should generate the HTML code corresponding to the page to be displayed in the browser.

Communication between a client and server via HTTP is achieved in a so-called "unconnected" mode, which means that there is no notion of dialog context. A Web application server must provide a mechanism for compensating this limitation, a mechanism commonly called "context management". GemStone provides such a mechanism through its servlet technology. Each servlet execution has access to an object corresponding to a client session that made the page request. This object provides a dictionary-type API, which is used for storing and reading values. By implementing GemStone, this object may be made persistent, which provides a basis for setting up a fault-tolerant system at session level ("session failover").

For developing applications, the choice is either directly programming servlet classes or using JSP technology (Java server pages). With the latter, pages mixing HTML and Java server code can be developed. This is the Java counterpart of Microsoft's ASP technology, itself based on VBA use. When a client requests a JSP page, it is automatically translated into Java code as a servlet, which is compiled and executed. The resulting HTML code is sent to the client. Subsequent requests for the same page do not lead to a repetition of this process: the compiled servlet is used directly, except if the source JSP page has been changed in the meantime. Using JSP technology simplifies development. It should be noted that this technology was promoted by Sun and is used in many products.

No HTTP server functionality is integrated into GemStone. For implementing Web applications, it must be interfaced with a third-party server. Three leading HTTP servers are supported: Apache, Microsoft IIS and Netscape (FastTrack and Enterprise, on NT and Solaris). Lastly, it should be mentioned that Gemstone servlets support the uploading of files.

## Ratings:

Interface generation	GemStone
Step 1: Event-driven interface	
Step 2: Graphic component creation	
Step 3: HTML interface	
Step 4: Tabular data display	
Step 5 : Print management	

Scores 0-3 ( ★★★: 3; ★★: 2; ★: 1; ●: 0 )

## 6.2.2. Application server

---

### Step 6: Openness to distributed objects

GemStone provides several facilities for accessing distributed objects as a client. Remote objects may thus be accessed via RMI and Corba. Accessing EJBs, based on RMI, is also supported.

Development using Corba is standard. It involves using a supplied Corba 2.3 ORB, based on Sun's implementation in the JDK. Interfaces of server objects, which we want to make accessible, should be described in IDL language, as in the example below:

```
interface Model
{
    int idModel();
    string manufacturer();
    string description();
    setIdmodel(in int idModel);
    setManufacturer(in int manufacturer);
    setDescription(in string description);
}

interface Automobile
{
    string idaut();
    Model idmodel();
    long capacity();

    /.../

    string slogan();
    int periodWarranty ();
};
```

GemStone provides an executable file named "IDLtoJava", which is responsible for the required service files producing from the IDL files in order to use the ORB. In particular, these service files contain stubs, objects used for accessing the remote object.

In our development scenario, we would like to connect to a remote Automobile object, named "bargain\_of\_the\_week".

The steps are as follows:

- Initialize the ORB
- Obtain a reference from the naming service
- Obtain a reference to the "bargain\_of\_the\_week" object from the naming service and the ORB, get a reference to the "bargain\_of\_the\_week" object
- Use the remote object that represents the bargain of the week by sending it messages

```
public class Client {
    public static void main(String[] args)
    {
        org.omg.CORBA.ORB orb;
        NameComponent nsname[] = new NameComponent[1];
        NamingContext namingService;
        java.util.Properties props = new java.util.Properties();
        Automobile auto;

        props.put("org.omg.CORBA.ORBClass",
"com.gemstone.PortableServer.ORB");

        orb = org.omg.CORBA.ORB.init(args, props);

        namingService = NamingContextHelper.narrow(
orb.resolve_initial_references("NameService"));
        nsname[0] = new NameComponent("bargain_of_the_week", "");

        auto = AutomobileHelper.narrow(namingService.resolve(nsname))

        System.out.println("bargain of the week : " + auto.slogan());

        System.out.println("id      : " + auto.idaut());
        System.out.println("version : " + auto.version());
        System.out.println("capacity: " + auto.capacity());
        System.out.println("price      : " + auto.price());
        System.out.println("proposed guarantee period : " +
auto.periodWarranty());
    }
}
```

## Step 7: Database access

To begin with, a basic feature of GemStone should be noted: it includes an integrated object database. Therefore, use of an external database may be avoided, whereby GemStone assumes the role of second and third tiers (application server + database). This feature gives GemStone an enormous edge over its competitors. We will further discuss this point when object-server capabilities are examined.

As for using relational databases, GemStone provides JDBC as an access method. An object-relational mapping tool may also be used. In this case, "CocoBase" from "Though Inc" will first be considered, as the preliminary results of its work with GemStone have been positive. Another mapping tool, "TopLink" from "The Object People", may also be used.

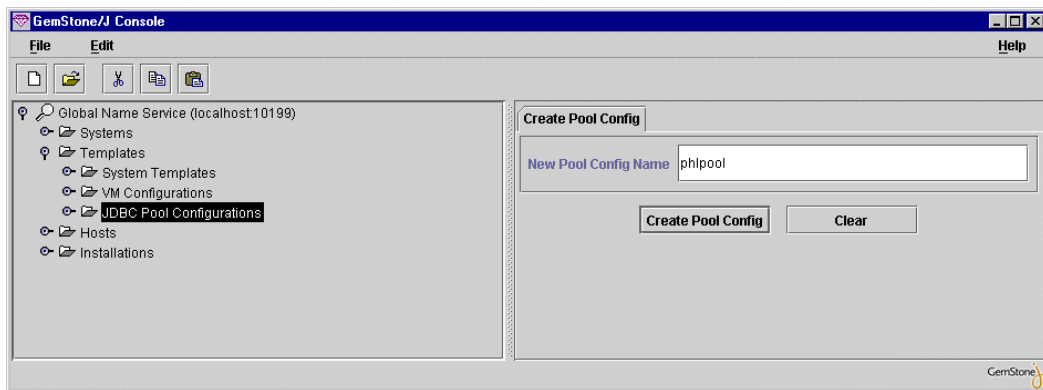
The technology used for development depends, in particular, on the choice of enterprise data modeling on the application server. The following table illustrates the options:

Relational modeling	Object modeling
JDBC	Gemstone's object database or Third-party object-relational mapping tool

It is not realistic to choose object modeling and implement it manually using JDBC, which is restricted to relational modeling.

Since GemStone does not include an object-relational mapping tool, we used JDBC to connect to our relational database. The first step consisted of installing the database driver on the application server. SequeLink drivers are provided as well as jConnect drivers for Sybase and Oracle JDBC Thin Driver.

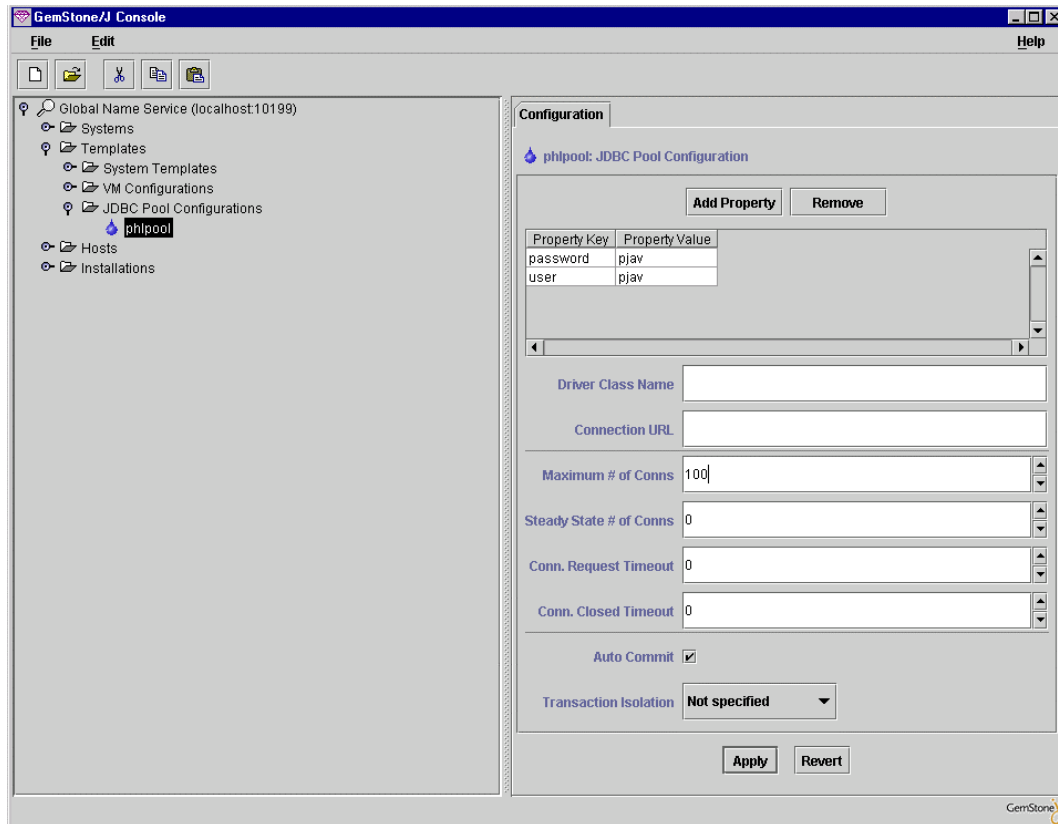
As should any application server deserving of its name, GemStone provides the functionality of pooling connections to the databases. Therefore, the second step consists of configuring the connection pools, which may be done using a Java program or the graphics console:



*Creating a pool of connections via the GemStone console*

Once the pool is created, all that remains to be done is to configure it by entering specifically the database login and password.





*Configuring the connection pool*

Subsequently, conventional development is possible using JDBC. To get a connection, the object representing the pool is directly referred to, and the “getConnection()” method is applied to it. When the “close()” method is applied to the connection, the latter is not actually closed, but instead merely made available again in the pool.

GemStone supports JDBC drivers types 1-4, based on the JDBC 1.0 specification. Drivers based on Specification 2.0 may be used but the connections established will not in any way implement the 2.0 specification. Nevertheless, supported services offer more than those required by JDBC 1.0 and notably include the two-phase commit and distributed transactions. These functionalities are based on the implementation of the Object Transaction Service (OTS) 1.1 provided by GemStone as specified by the Corba standard.

## Step 8: API richness

GemStone does not provide support for multimedia or for sending mail beyond what is standard with Java 1.2.

## Step 9: Workload management

GemStone's product is intended for large-scale applications. By itself, the GemStone server requires very large computer resources. Accordingly, a minimum of 192 MB of RAM will be needed for running GemStone on NT, and 256 MB on Solaris. For certain hardware configurations, GemStone even recommends up to 394 MB for optimal operation (512 MB on Solaris). Resources required for the applications themselves should be added to this.

The editor's experience in terms of technologies supporting large workloads is consistent. Its product, GemStone for Smalltalk, has been recognized in this field for a number of years. Moreover, and from a theoretical point of view, using an object-persistence engine affords many advantages in terms of performance, particularly by removing the object-relational mapping process.

Many services provided by the product are for large-scale multi-user applications:

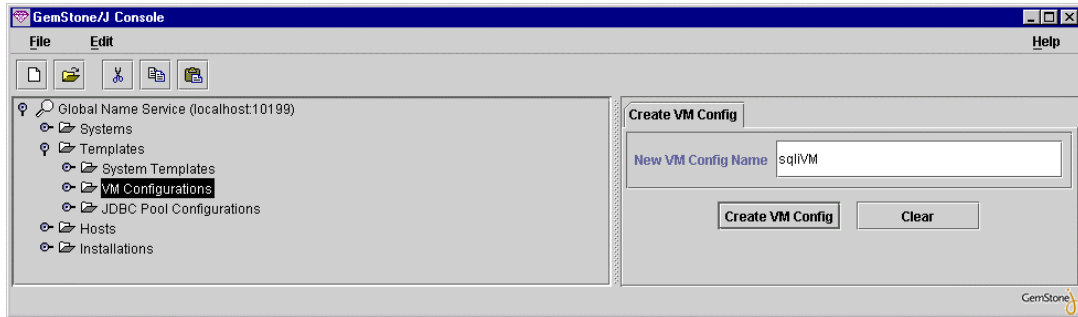
- The very complete security service provides good access control.
- Transaction services are for designing OTM applications, ensuring data consistency. Different strategies for dealing with conflicts during concurrent access (optimistic, pessimistic, etc.) are provided and integrated into the product.
- A set of classes specially designed for meeting needs from heavy workloads. New collection classes such as EnhancedCollection are found, which provide extended functionalities and implement optimized algorithms. However, unfortunately, a number of these classes have not yet been validated, and for the moment it is advisable not to use them in production.
- A service allows the management of groups of objects in common virtual memory pages in order to enhance referential proximity. The principle consists of pooling objects that frequently communicate with one another in order to reduce swapping. Since this type of technique has proved its effectiveness in other object systems, their availability with GemStone is very attractive.
- The database connection pool, imperative for workload management, is also present.

With Java, unnecessary objects are automatically destroyed, thus freeing up memory space. This mechanism, called the Garbage Collector, is important for workloads. Here, GemStone provides attractive management functionality. It should also be noted that this mechanism is documented, which is exceptional but useful for the developer.

The JVM (Java Virtual Machine) used is a modified version of Sun's standard JVM. It includes specific mechanisms for distributing workloads, fault tolerance and object persistence from shared cache.

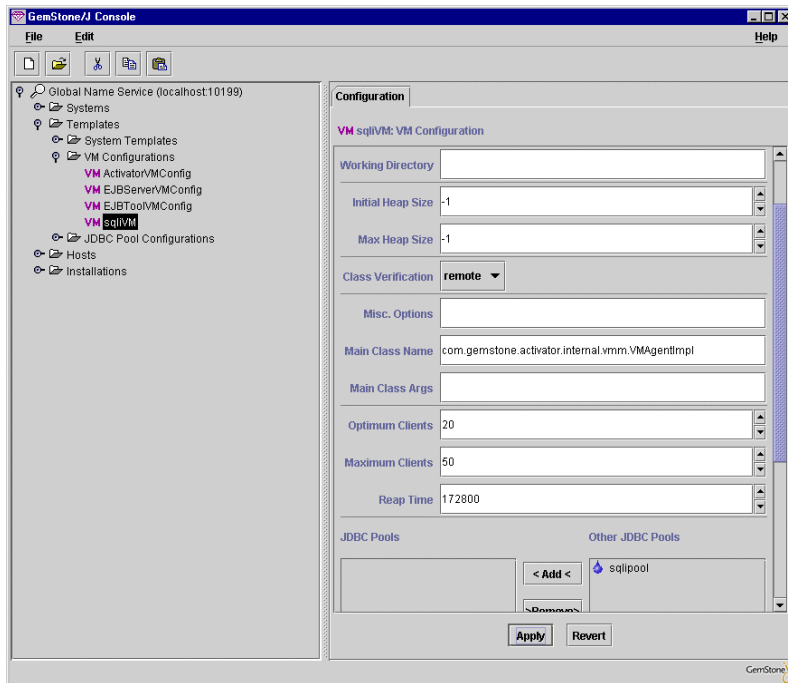
Within the scope of its multi-user management, GemStone is based on the session concept. A session is created on the server for every user, bearing in mind that each virtual machine can cope with several sessions.

Virtual machine pools may be also defined and distributed on different physical machines, the system assumes the task of allocating the loads among them. Moreover, with GemStone, virtual machine configurations can be defined which are automatically used by the system to create virtual machines while running. The GemStone console is used for this purpose:



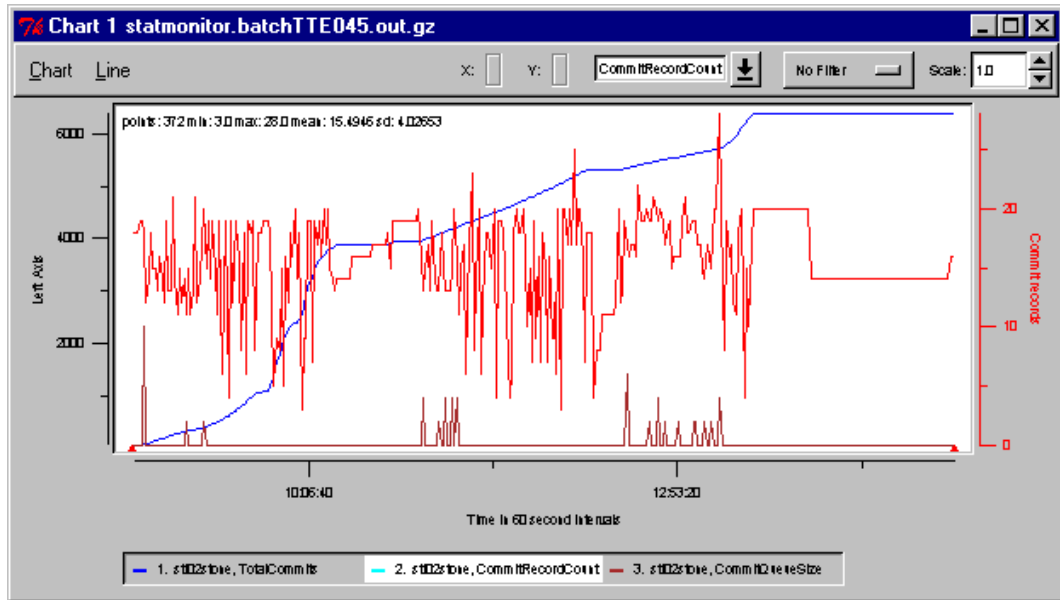
*Creating a virtual machine configuration*

Various details may now be provided such as reservation of memory space, recommended and maximum number of users, or the connection pools to be used.



*Entering information for configuring a virtual machine*

With other tools, the statmonitor and VSD, allow to closely follow server activity and to detect potential optimizations. In this way, more than a hundred data items are available in the statmonitor in order to monitor the server in real time or at a later time.

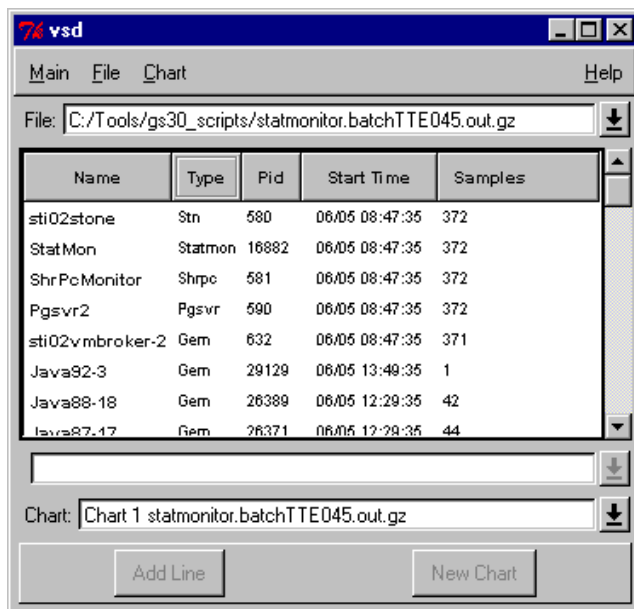


Statmonitor tool: View of three statistical data items related to commits

VSD's main window displays the entire set of server processes for a GemStone system.

By selecting one or more of these processes, and then selecting a data item, its can be plotted in an existing statmonitor diagram ("Add line" button) or in a new diagram ("New Chart" button).

Templates may also be used involving data sets with the same theme (for example, validation activity, disk transfers, "garbage collection", etc.).



The VSD's main window

Furthermore, in version 3.0, an HTML interface is provided for configuring the servlet engine (context management strategy, security, logs, etc.).

During our development activities, we were not pleased that the graphical interface of the different tools, and, in particular, the console was so slow. Even with powerful hardware, the use of these tools is cumbersome.

## Ratings:

Application server	GemStone
<b>Step 6: Openness to distributed objects</b>	☆☆
<b>Step 7: Database access</b>	☆☆
<b>Step 8: API richness</b>	●
<b>Step 9: Workload management</b>	☆☆

Scores 0-3 (☆☆☆: 3; ☆☆: 2; ☆: 1; ●: 0)

### 6.2.3. Object server

#### Step 10: Modeling by business objects

Modeling by business objects is a critical step in setting up an application. Choices made at this point greatly affect developments because the business object layer becomes the entity on which nearly all the processing operations are carried out. Whatever the tool, developing an object model should be carried out by experts. Indeed, in order to benefit from object technology, the model should be designed for factorizing out generic processing operations by relying on Java "interfaces", common to multiple concrete classes. In this way, by capitalizing on Java's polymorphic potential, modular and extendable applications may thus be obtained. This type of modeling may only be accomplished by a team who is very experienced in object technology.

#### ➤ Choice of architecture

Before starting to build a model, the architecture and an implementation technique must be selected for the object model. But what does this mean? What options do we have? These depend, of course, on the tool, the object server and the application to be built. With GemStone, for instance, should we choose the Corba, EJB, DJB or even the Java standard component model? The development process, the services which may be used (for example, Corba transaction services are different from EJB transaction services) and many other items are determined by selecting one of these models.

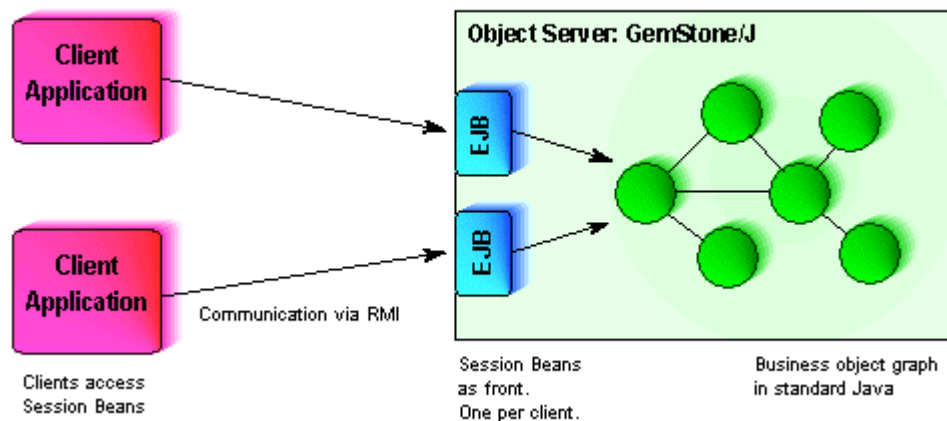
#### ➤ Selecting an implementation technique or why EJB Entities are not used

With GemStone/J, by selecting an implementation technique for our object model, it should be kept in mind that a distributed object mechanism must be able to make our objects persistent and accessible. Using EJB Entity technology for implementing business entities might appear as the solution: each business class (in our case: AutomobileNew,

AutomobileUsed, Automobile and Model) is implemented as an EJB Entity component. However, this solution was not retained for our developments, for the following reasons:

- Support of Entity Beans technology: contrary to a widespread belief, Entity Beans are not supported in a satisfactory way by GemStone/J 3.0. For instance, automatic persistence is not guaranteed.
- GemStone recommends another solution: this consists of implementing business objects as Java standard components and using the Session Beans model for delivering application services to clients. This solution actually avoids the cumbersomeness of the EJB model at the business object graph level, while providing the ease of having this graph persist in GemStone's object database or in a relational database by using a third-party mapping tool.

The technique we decided to use is therefore the following: it consists of using standard Java language for modeling business entities, keeping in mind that Gemstone JVM extensions will be available, such as automatic persistence. Session-type EJBs group together application services, services that use the business object graph. EJB sessions are made accessible to the clients and thus form the server's public interface to client programs.



## ➤ Implementation

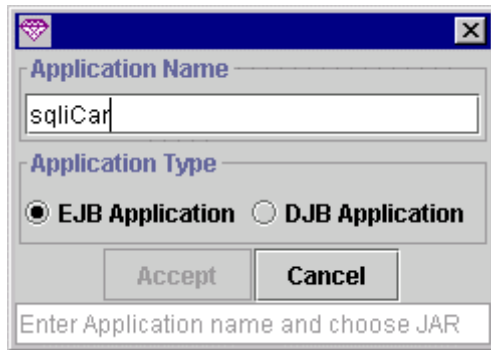
Having selected the technology for the business objects, it is now time to implement it. GemStone does not provide any object modeling tool, not even a development environment (code editor, etc.). So a third-party tool must be used or a simple text editor.

Developing Session Beans requires implementation of a certain number of methods and production of service files as deployment descriptors or "manifest" files.

GemStone supports JDK 1.2, and has therefore a significant advantage over environments, which are still based on JDK 1.1. Business APIs are implemented and a rich palette is available to use our objects' functionalities optimally.

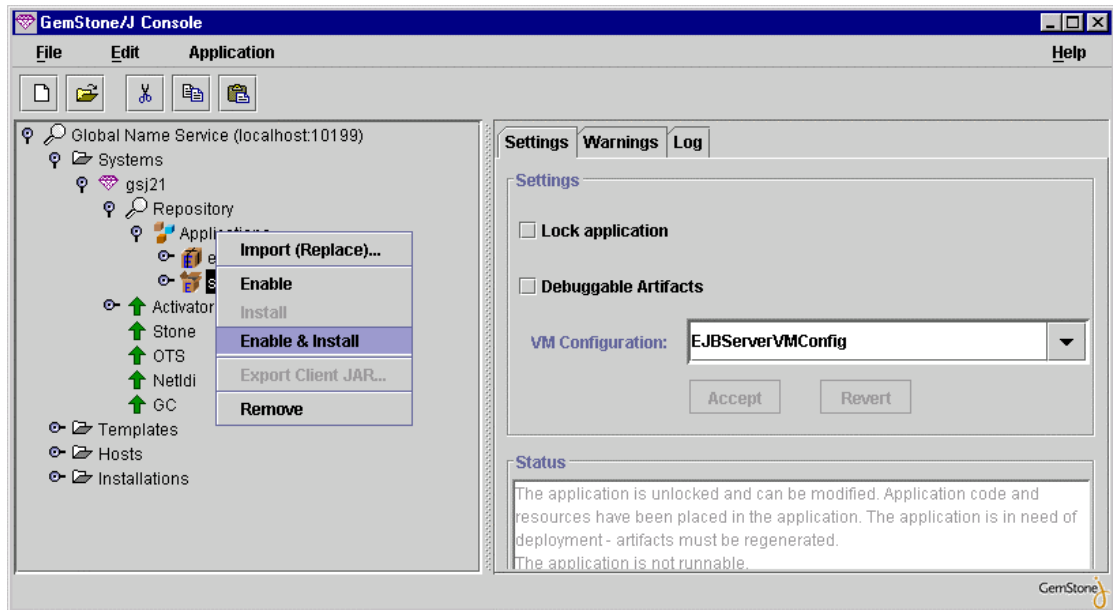
Once the code has been developed, it must be compiled and then imported into GemStone. To accomplish this, the compiled files and the service files must be grouped into a “.jar” file (Java Archive) via its preferred development tool or the jar command.

The next step consists of prompting the console to create a new application, and then importing our jar file and activating it.



*Creating a new application in the GemStone repository*

During creation, we are prompted to select between an EJB and a DJB application. DJBs (Distributed JavaBeans) correspond to an old component model specific to GemStone. Therefore, we chose EJB.



*Initializing the application*

These steps produce service files, containing, in particular, stubs which clients must use to access server objects. Here, the term client refers to any process connecting to our server objects, including another server. The service files generated must be made accessible to the clients, which basically means that they are locally deployed on each client machine with changes in CLASSPATH.

Finally, client application code is developed.

The application can then be tested. Generally, the third-party development tool may only be used for coding and compiling. Application testing must be conducted in GemStone, which is normal since the runtime environment in GemStone is different from that in the development tool. Fortunately, GemStone has a debugger, which can be used to debug the code executing on the server.

The console integrates an object browser for navigating within instantiated objects on the server side. Viewing of values for instance variables of each object is provided by this inspector, and if these variables are themselves objects, they may in turn be examined in the same way.

The development/packaging/import/deployment/test cycle is very cumbersome, as we have seen. A lot of manipulation is required and we are nowhere near the model of “exploratory programming” where code can be changed and effects observed immediately. This problem is again more pronounced because GemStone does not offer a development environment.

## Step 11: Importing EJBs

For importing EJBs, the steps are the same as those described in the preceding chapter for importing an application.

### ➤ **Importing a EJB Session created using IBM's VisualAge/Java**

Running a Session for importing a simple EJB was not easily accomplished. Thus, on the first attempt, GemStone, after a certain number of validity tests, reported that it was rejecting the EJB.

In order to import our EJB, we had to take it out of its package, change the “manifest” service file, and then manually repackage it by using the jar command. However, this time, importing was a success and we managed to connect to the EJB hosted by GemStone from our client program.

### ➤ **Importing a EJB Session created with JBuilder from Inprise**

This time, importation succeeded without any problems, whereby GemStone acknowledged the EJB developed with JBuilder as being valid.

## Step 12: Persistence

### ➤ **Selecting persistence technology**

Our business objects representing business entities must be provided with persistence capability. In our case, the objects were the AutomobileUsed, AutomobileNew and Model classes.

Object persistence has been studied for a long time and is known to be a complex problem. Numerous tools have followed one another over the last ten years in this field, which are now beginning to be mastered from a theoretical point of view. One of the challenges to be met is that of performance. Since objects are organized in graphs, it is absolutely necessary to have



a tool that restricts the loading of objects into memory to only the portion of the graph used at a given time. Moreover, this type of optimization must be transparent to the programmer.

The following table reviews different technological options for implementing persistence with GemStone.

Technology	Comment
JDBC	This option involves manually developing the persistence mechanism. For slightly complex business applications, this is totally unrealistic because this is too cumbersome and difficult.
Third-party object-relational mapping	This option consists of using a relational database for storing the objects. Mapping, which must be configured, then automatically takes place at runtime. It is a good solution if the data to be accessed are already in relational databases. This technology must be purchased separately. It is not provided by GemStone.
Gemstone's Persistent Cache Architecture	This option uses the object database located in GemStone. It is a very good solution because it ensures consistency, transparency and performance.
EJB Bean Managed Persistence	This option consists of using EJB components of the Entity type and of developing the persistence mechanism itself based on the EJB specification. For this purpose, an API is used for accessing the database, for example JDBC. It is an awkward solution and hard to implement.
EJB Container Managed Persistence	This model consists of using EJB components of the Entity type and leaving the server to cope with persistence possibly after a mapping configuration phase. This technology is not satisfactorily supported by GemStone 3.

For our developments, we decided to use PCA (Persistent Cache Architecture).

GemStone, which has long been one of the most respected experts on object persistence, provides a product with a real object database that handles persistence in a highly transparent way. This technology enables applications running within GemStone to share persistent objects and ensures multi-user consistency through transaction support. This is one of the most important aspects distinguishing GemStone from its competitors.

## ➤ Development

To implement PCA, an object only needs to be entered in the database. It becomes a persistence root: all objects pointed to by this root, either directly or indirectly by transitive coverage, are made persistent.

In our program, we handle objects that represent automobiles and associated objects that represent models. Each automobile points to a model object. Therefore, if an automobile object is made persistent, the associated model will automatically be made persistent. But how can the whole set of automobile objects be made persistent? Very simply: they only need to be referenced in a collection-type object, and this object should be defined as a persistence root. At the application level, we may consider our collection object as a catalogue, which groups together our automobiles.

The following code connects to GemStone and implements the persistent business object graph:

```
import java.util.*;
import java.lang.*;
import com.gemstone.vm.*;
import com.gemstone.ejb.*;
import gemstone.services.*;

public class BootStrap
{
    public static void main(String[]args) throws Throwable {
        String appName = args[0];
        String rootName = args[1];

        // Connecting to GemStone
        GsSessionIF gsSession =
        SessionManagerImpl.getDefault().createSession();
        VM vm = VMImpl.getCurrentVM();
        vm.bind("testSession", gsSession);

        // Creating the application in GemStone
        EJBApplication app = new EJBApplication(appName, gsSession);

        // start the GemStone transaction
        gsSession.begin();

        // creating the catalogue
        Vector catalogue = new Vector();

        // the catalogue is made persistent
        gsSession.bind(rootName, catalogue);

        // the available cars are created and recorded
        // in the catalogue
        catalogue.add(new Automobile(1, "a spacious passenger car"));
        catalogue.add(new Automobile(2, "a sedan for young people!"));
        catalogue.add(new Automobile(3, "the sportscar that takes your breath
away!"));
        catalogue.add(new Automobile(4, "More than a car"));
        catalogue.add(new Automobile(5, "Safety above all"));

        // end of the GemStone transaction
        gsSession.commit();
    }
}
```

### Step 13: OTM and object distribution

Up to this point, we have set up the application logic on the server side. It is structured by our object modeling and is implemented in attributes and methods of our server-side business objects. For example, every Automobile subclass provides a method for calculating the warranty period, based on different management rules: the warranty period for an object of the AutomobileNew class is calculated in a different manner from that of a AutomobileUsed object.

The goal is to now make the application services available outside the server. Applications running on other machines must be able to use them. For this purpose, we shall resort to a distributed object technique.

In addition, our services should support concurrent accesses. The common problems of handling multi-users in databases are encountered, but appear at the object level. Here, we are entering a field that is far from being mastered theoretically, so it is the subject of much discussion. GemStone is based on the concept of transactions in order to ensure object consistency, atomicity and independence of sequence of operations. Durability is assumed by persistence. Technologies provided by GemStone in this advanced field are considered as state of the art.

The following table reviews various techniques for distributing objects that may be considered for implementation with GemStone. Some of them directly integrate transactional capabilities; others require that intermediate items (fronts) be placed between business objects and clients.

Technology	Comments	Transactions
Corba	The main interest for this choice is that the server objects may be made accessible from non-Java-based applications, which could be written in C++, Smalltalk, etc. However, if Java/Java communications are being considered, a RMI-based model will be preferred as option because it provides much better integration into Java, greater simplicity in development and better transparency.	Business objects may be directly accessed in a transactional context by implementing the Corba OTS supplied by GemStone.
RMI	RMI provides the most natural means of communication with server-side distributed objects. Proper integration into Java distinguishes it from Corba: passing of objects by value is virtually possible (regarding this point, Corba is at a theoretical stage), distributed garbage collector simplified development without IDL, etc. Its major drawback is that it does not provide transaction management. Consequently, one of the enhanced variations shown later in this table will be used instead.	No transaction management. Objects must be installed on the server side, which will trigger transactions by using GemStone's services.
RMI with EJBs of the Session type	This is the model described in the chapter "Modeling by business objects". It requires that EJBs of the session type be interposed between clients and pure Java business objects. Communications between remote objects are provided through RMI. Implementing GemStone's RMI for EJBs internally uses the IOP protocol (generally used for Corba) instead of the conventional RMI internal communications protocol. However, semantics and capabilities are clearly those of RMI.	The EJB transactional model is used. Transactions are handled by EJB Sessions, automatically if desired. This handling may be configured in a declarative way.
RMI with DJBs (Distributed JavaBeans)	This model specific to GemStone provides RMI communications semantics and offers further capabilities. It is to be noted in particular that in this model, a distributed object is not constrained, as in a conventional RMI, to a particular inheritance. Globally, this is a more flexible and more powerful extension to RMI. But GemStone points out that the presently recommended model is EJB, which is expected to replace this DJB model.	In this model, transactions are supported. GemStone's transactional services are available.

The model that we selected for our developments is that of session-type EJBs as a front to our business objects. Preferably, this model is suitable for implementing methods of large granularity. Therefore, instead of making our automobile objects available (fine granularity), we may define methods our Session Beans that perform complex processing tasks on the server side without repeatedly interacting with the client. For example, a method might be provided for searching automobiles according to criteria and returning results by value.

Communications between client applications and server objects are provided via RMI. GemStone has its own implementation of RMI. It differs from the one provided by SunSoft in

the JDK, which is generally recognized as not being very effective. However, it provides the same semantics.

GemStone's implementation is characterized by the following points:

- Communications protocol between objects
- Marshalling mechanism
- Management of remote references and object life cycle

The communications protocol used is not the conventional JRMP, but IIOP, a protocol usually used by Corba. This choice is consistent with current guidelines from OMG and Sun, which involve finding ways to improve Java and Corba integration. However, be careful: using a specific technology restricts communications to only those objects using this implementation.

### Ratings:

Object server	GemStone
<b>Step 10: Modeling by business objects</b>	☆☆
<b>Step 11: Importing EJB</b>	☆☆
<b>Step 12: Persistence</b>	☆☆
<b>Step 13: OTM and object distribution</b>	☆☆☆

Scores 0-3 ( ☆☆☆: 3; ☆☆: 2; ☆: 1; ●: 0 )

## 6.3. Evaluation of VisualAge for Java 2.0 - WebSphere

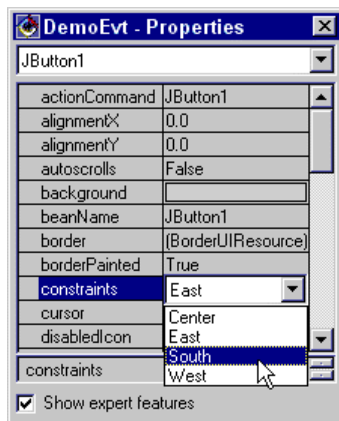
### 6.3.1. Presentation

#### Step 1: Event-driven interface

The evaluation of the event-driven interface will be conducted during the development substeps of the Euro-Franc converter:

- the creation of the graphical interface,
- the definition of the methods and variables,
- the creation of visual connections between components.
- Creating the graphical interface

With its “Visual Composition” tool, VisualAge for Java offers a very rich and highly user-friendly graphical interface. This editor consists of several palettes of graphic objects, classified by theme (AWT, Swing, data access, servlets, etc.) together with an assembly surface where the components are placed. To illustrate our example, we shall use the Swing components palette. It is complete and supports all of the components in the standard way. However, the IDE does not allow for the parameterization of Look&Feel, this must be hand coded.



*The objects inspector*

Concerning the assembly of components, the tool does not offer a placement grid, but all layout functions are present (alignment, size, arrangement). The components are assembled in a consistent manner, with the arrangement control in the containers (the layouts) being especially effective.

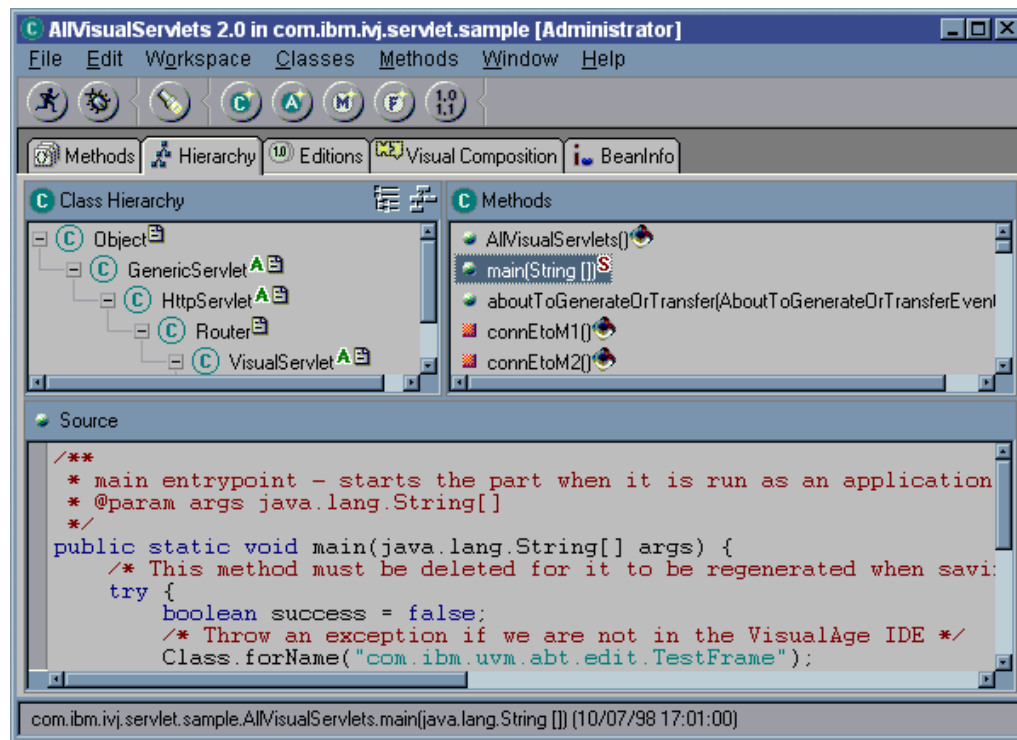
Graphic component properties are accessible by means of the object inspector and all modification is handled in the graphical interface. The inspector offers two levels of property display: a standard display combining the essential properties and a comprehensive expert level display. For each component, it is possible to store the names of character strings in external files (practical for multi-language management).

The code generated by the tool is very clean, but if one wants to export it in order to execute it outside the VisualAge environment, it then becomes necessary to modify the imports of JFC classes so that they are compatible with the latest version of JDK 1.2.

### ➤ Definition of methods and variables

All of methods and properties of objects are accessible via the objects explorer, in which the generated source code can be displayed and edited. VisualAge indicates to the developer where his code can be inserted.

The tool offers wizards for creating methods and variables for an object. The wizard checks to see that each parameter is properly filled.



*The objects explorer*

For our currency converter, the following methods were coded:

- the “double convertEtoF(double e)” method,
- the “double convertFtoE(double f)” method,
- the Rate variable (rate of conversion between currencies).

For the Rate variable, it is possible to have the wizard generate the read and write accessors `setRate()` and `getRate()`. The wizard also allows generating the body of the methods so that the user need only code the calculation for conversions.

### ➤ **Creating visual connections between components**

Event-driven interface management is handled graphically in the interface editor. This management is based on the graphic connectors principle. Indeed, the developer “stretches” links between the different components.

Let's take the example of our Euro-Franc converter. The action performed event of the `convertEtoF()` must trigger a call to the `convertEtoF()` method. The value to be converted must be a parameter of this method and the result of this method must be displayed in the same field where the data was entered.

To implement our example, VisualAge does not offer us a wizard. On the contrary, the developer must understand the tool's visual connection mechanism and must break down his problem according to an operating mode, which is similar to the Java event-driven model (Source-Event-Delegate).

In practice the principal phases for our example are as follows:

- Choice of the sender of the event:

For our converter, this is a button. A right-click on this component provides us with a list of methods available to the object. The method `convertEtoF()` is chosen which then retrieves a graphic representation from the assembly surface. Visually, at this stage, the button is linked to the arrow attributed to the method `convertEtoF()`.

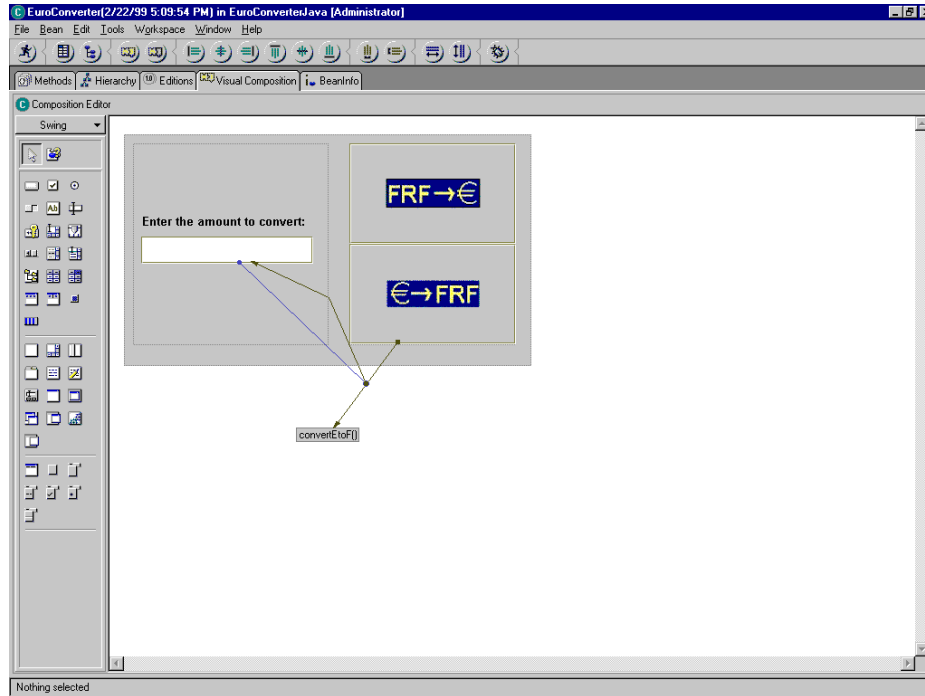
- Choice of the source of the argument passed to the `convertEtoF()` method:

This argument corresponds to the amount entered in the field. A right-click on the component provides us with the list of accessible properties. From these properties, we choose the ones named “text”, which correspond to the entered data. It is then necessary to connect this value to the `convertEtoF()` method by clicking the arrow set up in the first phase. During this connection, the list of arguments of `convertEtoF()` (here there is only one argument) is provided. The argument “e” is therefore chosen.

Visually, an arrow is displayed between the data entry field and the connection button method.

- Display of the result of the `convertEtoF()` method:

To accomplish this, by clicking the `convertEtoF()` method connection, we obtain the NormalResult option, which represents the return of the method. It is then necessary to link this return to the component that will display it, in other words the data entry field. A right-click on the data entry field provides a list of modifiable properties. In choosing the text property, a visual link is established between the button-method and the data entry field connection.



*Event management using visual connections*

Once this method is understood, event management becomes natural.

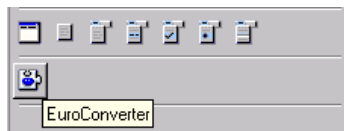
## Step 2: Creating graphics components

In this step, we now wish to make our currency converter reusable. It must therefore be transformed into a JavaBean component in order for it to be inserted into a palette of graphic objects. The VisualAge wizards offer all the necessary options for creating a Bean.

For each property (in our example of conversion rate), the read and write accessors are generated automatically and it is possible to specify the type of each property (linked, indexed or constraint).

VisualAge also allows to parameterize/configure the BeanInfo interface in Bean, which renders methods and properties visible to the exterior. This interface is paramount for all tools that must internally display the properties and methods of our component.

Once this stage has been reached, our Bean EuroConverter can then be inserted into the graphics palette and be reused in another application. This operation is handled naturally with the VisualComposition utility and, once the Bean is placed on the assembly surface, its methods become easily accessible.



*VisualAge palette with Bean "EuroConverter"*

It is also possible to export our Bean. VisualAge provides export functions in "jar" format notably, including the manifest file.



### Step 3: HTML Interface

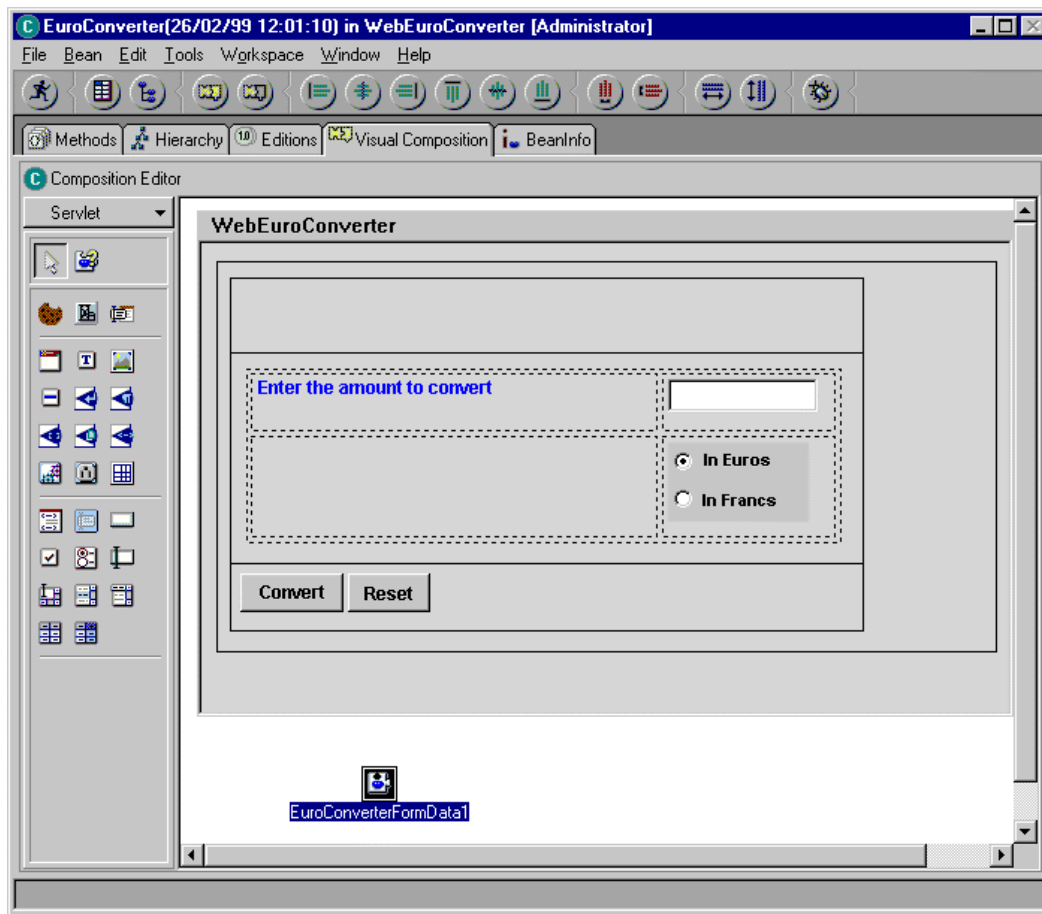
The IBM Java global product offers a development tool/utility, VisualAge, and an application server, WebSphere, for the deployment of components. VisualAge likewise offers the possibility of testing servlets generated using an integrated HTTP server.

#### ➤ The development environment

In this step, we wish to offer the currency conversion service on an intranet. Therefore, we must therefore offer our converter in HTML format.

The Web version is divided into two pages. The first page consists of a form in which the user enters an amount and chooses a currency (Francs or Euros). Validating this form using a submit button displays a page containing the result of the conversion together with a return link to the form.

The Visual Composition tool has a servlet palette for HTML interfaces. All the HTML 3.2 components are represented as graphic objects (tables, forms, buttons, etc.). VisualAge also offers high-level objects in this palette such as the Cookie Wrapper and the SessionData Wrapper, which control the user context across several pages.



*HTML interface management in the Visual Composition tool*

The generation of the HTML interface is handled in a similar way to classic Java. The user need only place the graphic components on the assembly page.

In our example, we place the form object on the page. At this level, it is necessary to tell the action method of our form the name of the servlet associated with the results page.

For the layout of components, we will use a table. The servlet palette uses the HTML table object, which in turn has numerous properties. Unfortunately no cell merging functions are offered. However, it is possible to use the overlap layer of tables to an advantage.

With regard to components, buttons, and radio buttons, all properties can be set. Once the graphic components are arranged, it is then necessary to add a "FormData" object, which is the high-level proprietary component that allows data to be stored in the form.

The next step consists of building the results page. To retrieve the data entered in the form, the user need only add a FormData object for the form to this page (in our case, the currency and the amount to be converted). For the calculation of the conversion, VisualAge leaves it up to us. It is possible to develop a specific method; however the user may also call an external component.

We are going to use our JavaBean component built in the previous step. There is no problem accessing the methods of our Bean. Since everything is handled on the server side, all that is required is to specify that our Bean will not be visible.

To return to the form, the user need only indicate the link target by specifying the name of the form's servlet. The generated servlets can be tested in the WebSphere test environment, which is integrated into VisualAge.

Another alternative would be to use this same JavaBean on the server side and to call it using a JSP application. The JSP development tool, included in the WebSphere Studio, intended for this purpose is NetObjects ScriptBuilder. This alternative seems more consistent since the HTML interface can be modified in any HTML editor on the market and the JSP engine allows context management. The previous example builds an HTML interface from JavaBean components, and is only modifiable from within VisualAge. It will likewise be difficult to maintain control of HTML code under these conditions, and this method would not be suitable for large-scale sites encompassing a large number of dynamic pages.

### ➤ **The deployment environment**

With WebSphere Application Server 2.0, IBM offers a servlet runtime platform. Upon installation, the product offers its HTTP service (based on Apache) by default. However it is possible to interface it with Netscape and Microsoft HTTP servers.

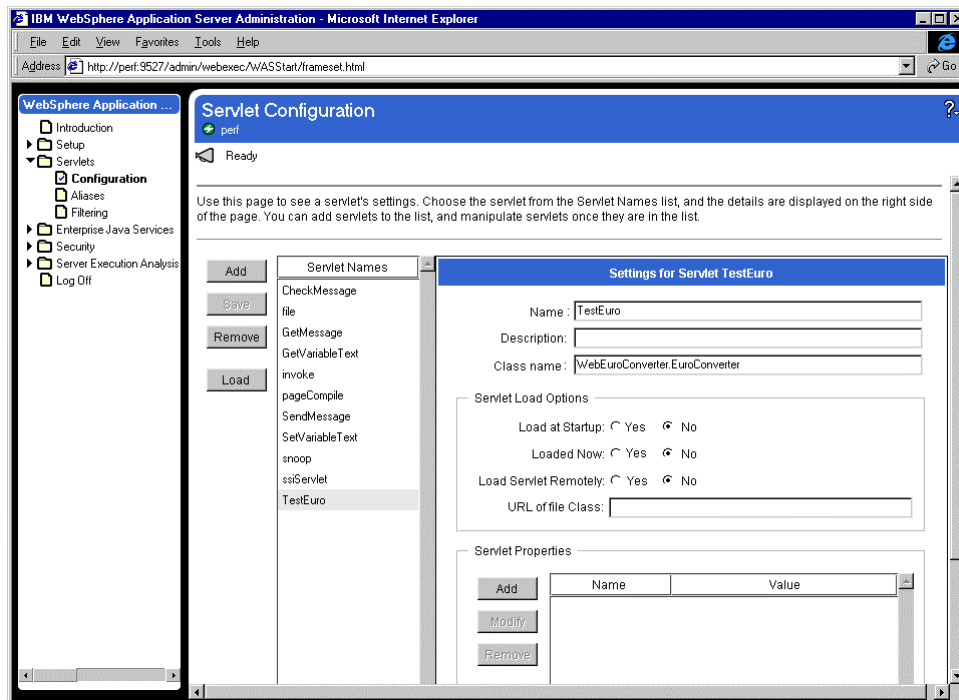
WebSphere is administered from a Web client by connecting to a port on the application server. This console relies upon an applet, which works in Internet Explorer 4 and Netscape Communicator browsers (version 4.5 and later).

As for the Web server, parameters for context management can be set. In particular, it is possible to select the system from "long URL" or cookies. The product also provides default user profile management by interfacing with the database. The execution of servlets can be handled on different servers since it is possible to specify the names of virtual machines and to position the servlets on remote machines.

Using the administration console is highly intuitive, but it has the disadvantage of not being completely independent of the file system. It is in fact impossible to browse the disk tree in order to specify the location of the servlet.

For servlet instance management, the console offers loading and unloading options. However these functions appear to be completely independent of the system, and in order to stop the execution of a servlet, the user is required to physically remove a file from its execution directory.

Finally, for servlet management, the console only provides for associating logical aliases with servlet Java classes and for passing initialization arguments to the servlet.



*WebSphere administration console*

The product also offers a monitoring console from which it is possible to display the servlet instances in progress as well as a history of invoked instances.

The advantage of this console is that it is accessible from a Web client. However, it has proven to be somewhat unstable during prolonged use and sometimes requires restarting the browser.

## Step 4: Tabular Data Display

The Visual Composition tool also offers a palette for data access. It consists of high-level objects in which the user can set the names of drivers and data sources.

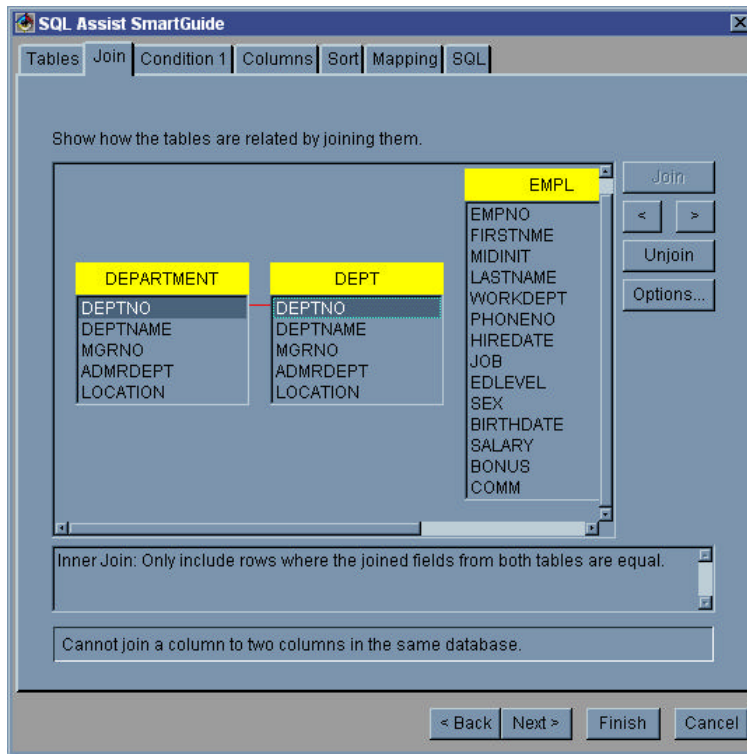
It is thus very easy to handle browsing of root tables as well as columns.

With regards to query management, the tool offers a wizard, the SQLAssist SmartGuide, which graphically generates SQL queries. This wizard is able to graphically handle links

between tables and generate conditions and sorts. This wizard generates the query code and even offers to test it.

Once the query is entered, all that remains to be done is to link its result to a graphic component (such as a Jtable or a Jlist). A DBNavigator component is also offered, which allows browsing the records in the table.

IBM also uses proprietary components to access other data sources. It is possible in fact to be connected to CICS or MQSeries sources.



The SQL Assist SmartGuide and table join management

## Step 5: Print Management

It is possible to import the JDK 1.2 printing API into VisualAge, although this tool provides neither a wizard, nor a printing function. In this case, the developer takes full responsibility for the code.

### Ratings:

Generation of the interface	IBM
Step 1: Event-driven interface	☆☆☆
Step 2: Graphic component creation	☆☆☆
Step 3: HTML Interface	☆☆
Step 4: Tabular data display	☆☆☆
Step 5: Print Management	●

Scores 0-3 (☆☆☆: 3; ☆☆: 2; ☆: 1; ●: 0)

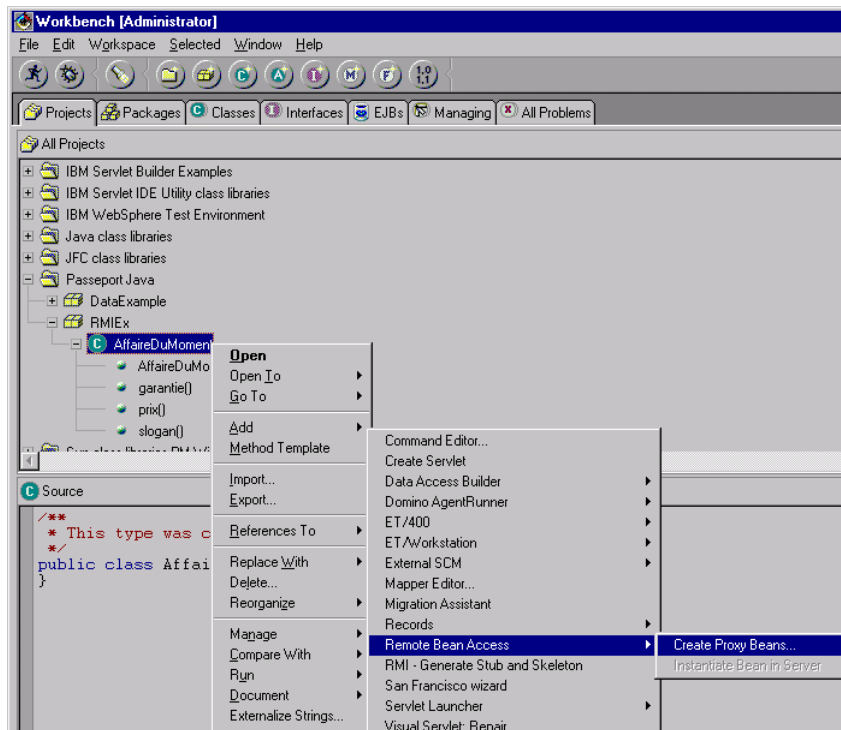
## 6.3.2. Application server

### Step 6: Openness to distributed objects

The IBM product provides several access possibilities for distributed objects. VisualAge integrates several tools in order to assist in the creation of distributed object clients: RMI Access Builder for the development of Java RMI-distributed applications, IDL Development Environment for the development of Corba, and EJB Tooling applications for the development of EJB and EJB clients.

#### ➤ Openness to COM objects

Concerning the integration of DCOM objects, IBM offers a third-party product named Jintegra (Linar Ltd), not provided with WebSphere. It allows Java objects to communicate with COM objects via an automatically generated Java proxy. The properties, methods and COM events are recognized by Java objects as Java properties, methods and events.



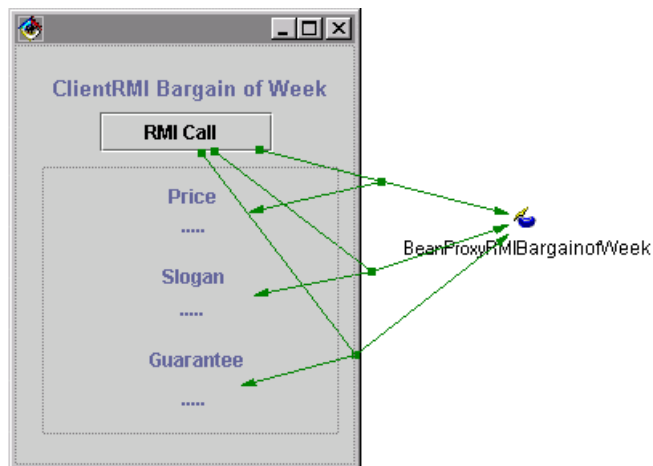
Generation of "Bargain\_of\_the\_week" interfaces and related Bean Proxies

### ➤ Openness to EJB

The IBM product is definitely oriented toward the Sun Enterprise JavaBeans specification. Upon completing the development process for an EJB component, VisualAge can generate a client interface in order to access the properties of the object. The tool is unfortunately not interfaced with an existing component in WebSphere and it provides neither for internally displaying the object, nor for generating a client interface.

### ➤ Openness to Java objects from RMI

The RMI Access Builder tool allows to generate and test a distributed Java application based on the RMI mechanism with incredible facility. The first step consists of defining the class that the user wishes to make accessible to RMI clients. The “generate Proxy Beans” command then allows for the generation of the interfaces, stubs and skeletons necessary for remote access. The interesting aspect is the generation of a Bean that can be integrated into the visual composition tool, the properties of which can be specified during development. RMI application testing can be handled from within VisualAge.



*Building interface to access to the remote “Bargain\_of\_the\_week” object*

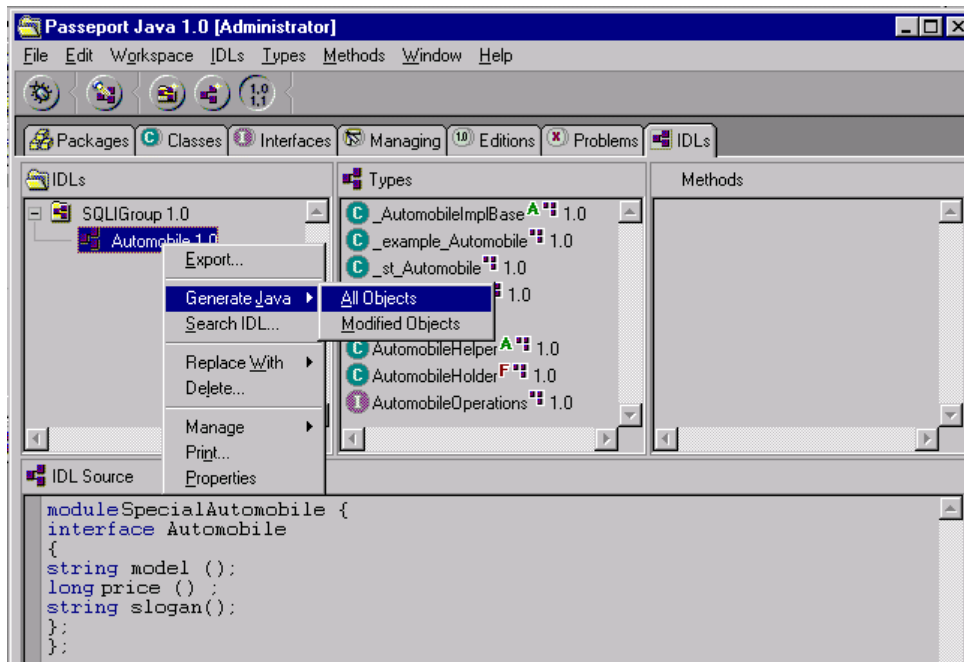
Invoking Java objects with RMI is facilitated with the tools provided by IBM. The applications under WebSphere could be RMI clients. This protocol's own services and efficiency are nevertheless limited.

### ➤ Openness to Corba objects

The development of Corba applications requires the installation of the IDL development environment, and the integration of Corba 2 ORB classes, of a third-party product, as well as its IDLtoJava compiler (IBM Component Broker, Visibroker or Orbix). The tests were performed using Visibroker for Java 3.4. The IDL development environment provides us with minimum functionality:

- IDL development version control
- The organization of IDL files in the logical tree
- Invoking the IDLtoJava compiler from within IDE

Unfortunately, it is not possible to start or debug the application from within VisualAge. The absence of JavaBeans for this part limits the developer's ability to harness the power of visual composition, and moreover the generation of the Corba client remains entirely the developer's responsibility.



*Generation of stubs and skeletons by invoking the IDLtoJava compiler from VisiBroker*

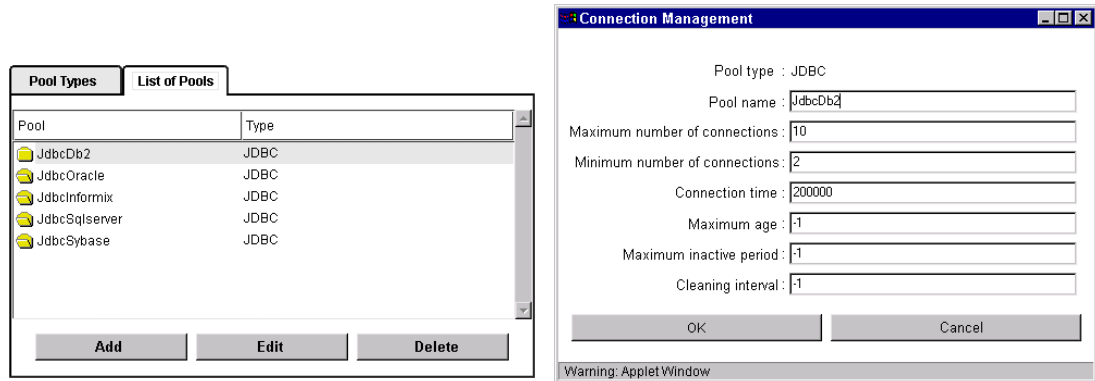
It is important to note that the absence of Corba ORB in WebSphere makes it impossible to integrate existing Corba objects from a WebSphere server.

## Step 7: Database Access

An application server worthy of the name should facilitate the implementation of interfaces for database access. IBM offers JavaBeans to facilitate the development of such applications and suggests pooling of connections to handle the workload.

### ➤ The connection pools for JDBC sources

IBM provides the possibility to define connection pools within WebSphere for JDBC sources. With this functionality, the application server is able to control access to databases through a reduced number of connections. Entirely configurable, these accesses allow to reduce and control the use of system resources. The administration console allows to define and configure these pools. The connection manager is very well documented.



*Management and configuration of connection pools for JDBC sources*

Here is an example of servlet code for implementing a pool of connections:

```

public class IBMConnMgrTest extends HttpServlet
{
    static IBMConnMgr connMgr = null;

    static IBMConnSpec spec    = null;
    static String  dbName     = null;

    public void init(ServletConfig config)
    throws ServletException
    {
        try
        {
            // creation of a JDBC connection specification.
            spec = new IBMJdbcConnSpec
                (MonPoolJDBC, // Name of pool specified in WebSphere
                true, // Wait or not to free connection
                "oracle.jdbc.driver.OracleDriver", // Driver JDBC
                "jdbc:oracle:oci8:@PJAV", // URL of the JDBC source
                user,
                password);

            // Creation of a reference to the IBM connection controller
            connMgr = IBMConnMgrUtil.getIBMConnMgr();
        }
        catch(Exception e)    {}
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
    {
        // Initialization of connections for a new query
        IBMJdbcConn cmConn    = null;
        Connection dataConn    = null;
        Vector firstNameList = new Vector();
        try
        {

```



```

        // Creation of a reference to the JDBC IBM pool connection
        conforming to the specification within the IBM connections controller.
        cmConn = (IBMJdbcConn)connMgr.getIBMConnection(spec);

        // Recovery of a standard JDBC Java connection within the
        connection pool IBM JDBC connection.
        dataConn = cmConn.getJdbcConnection();

        // Send and SQL query and retrieve the result
        Statement stmt = dataConn.createStatement();
        String query = "SELECT * FROM TCARS WHERE PRICE < 10000";
        ResultSet rs = stmt.executeQuery(query);

        // Retrieve results
        while(rs.next())
        {   firstNameList.addElement(rs.getString(1));   }

        // close the object stmt and release the resources. The object
        DataConn cannot be entirely closed, it is saved in the pool and can be
        reused later.
        stmt.close();
    }
    catch(Exception e)   {}
    // release the pool connection
    finally
    {   if(cmConn != null)
        {
            try { cmConn.releaseIBMConnection(); }
            catch(IBMConnMgrException e)   {}
        }
    }
}
}

```

## ➤ Data Access Beans

Out of the double concern for hiding the complexity of the development of database access interfaces and addressing the lack of functionality in the Java SQL API, IBM developed Data Access Beans. These components offer the following functions:

### *Write results to cache memory*

The results of queries are retrieved once only and written to cache. The application can move back and forth within the result, which is different from the java.sql package, which retrieves lines one by one, erasing the previous one.

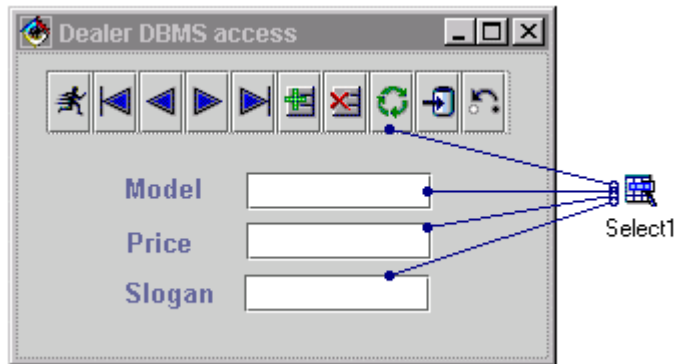
### *Update via result cache*

Changes to records are handled in cache through the use of standard Java instructions and are reflected in the database. The changes are handled in meta-data, thus avoiding having to manipulate java.sql instructions, which are often tedious.

### *Support of query parameters*

It is possible to specify queries using parameters available at runtime.

The Data Access Beans can also be used with VisualAge. Development has been radically simplified. The example below shows a pre-configured Data Access Bean whose properties corresponding to the columns of tables are linked to Swing components.



*Access to dealer's DBMS facilitated by a Data Access Bean*

The developer can control the transactional features of database access using the JDBC API.

It is best to use the Data Access Bean components in the VisualAge visual composition editor in order to enjoy the benefits of internal display. It however is possible to use them with servlets or JSPs. The Data Access Beans can also be used together with the pooling of WebSphere JDBC connections.

## Step 8: API Richness

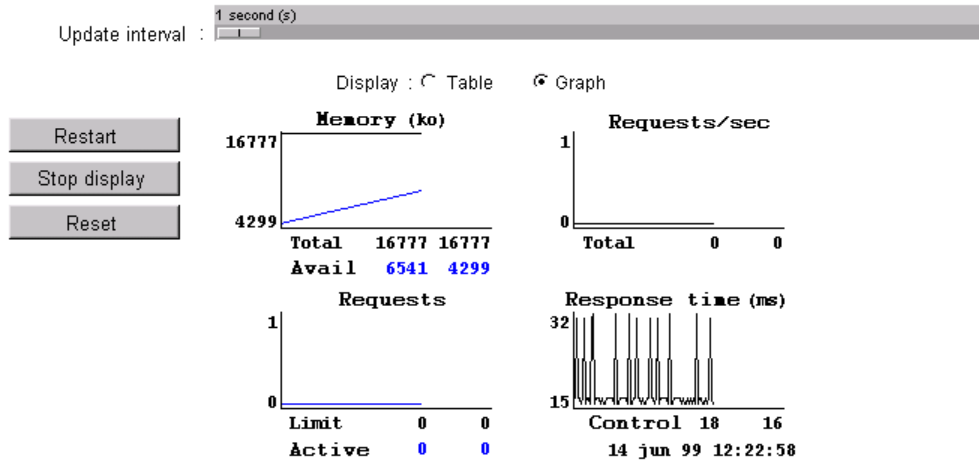
WebSphere Application Server does not offer a standard API to facilitate the implementation of certain services such as the blob management, or the generation of PDF files. The generation of server-side images is left to up to the developer, who however may use third-party components.

VisualAge for Java includes a Lotus API in Java to develop and debug Notes agents. These agents can access the Notes environment, among others, and trigger processes (consultation of a document database, sending mail, etc.).

Numerous JavaBeans are provided with IDE to facilitate the development of specific applications (C/C++ Beans access, SAP/R3, Tivoli, XML)

## Step 9: Workload management

In its administration console, the WebSphere application server has a portion reserved for system resources. It is possible to thus see the size of the execution stack as well as the available system resource. The use of system resources can be displayed graphically.



WebSphere's resource utilization console

On the contrary, nothing standard exists in WebSphere to manage workloads. For this function it is necessary to use a special product from IBM, the Performance Pack.

The Performance Pack includes three components that allow to manage "hardware" workloads:

- IBM SecureWay NetWork Dispatcher: it allows the balancing of loads for HTTP, FTP and other TCP-based services.
- IBM Web Traffic Express: Web Traffic Express plays the role of an HTML proxy. It allows front-end storage of Web pages in a cluster of WebSphere servers.
- IBM AFS Enterprise File System: AFS is a file management and replication system for distributed environments. With this system, the developer wishing to deploy files across a cluster of WebSphere servers does not have to have multiple FTPs on each of the servers. He deploys his files once, and AFS takes care of the publication on the different servers while ensuring consistency. It is also possible to configure security and to create backups.

The administration of a group of WebSphere servers is handled via Tivoli modules. A set of JavaBeans is provided in VisualAge to facilitate the implementation of this type of administration.

## Ratings:

Application server	IBM
Step 6: Openness to distributed objects	☆☆
Step 7: Database access	☆☆☆
Step 8: API Richness	☆
Step 9: Workload management	☆

Scores 0-3 (☆☆☆: 3; ☆☆: 2; ☆: 1; ●: 0)

### 6.3.3. Object server

---

#### Step 10: Modeling by business objects

The IBM product offers two modeling by business objects tools. The first, EJB tooling, is based on EJBs. The use of this tool in the development of complex object models can prove to be an adventure given the immaturity of the standard. The second, Persistence Builder, is based on proprietary IBM technology. It is based on frameworks and offers transaction and persistence services. Persistence Builder is fitted with a set of object modeling tools as well as with object-relational mapping.

##### ➤ EJB Tooling

The IBM strategy with regard to business components is specifically oriented toward Sun's Enterprise JavaBeans model. For the development and the modeling of these components, the VisualAge Enterprise version for Java should be updated with a special extension available for downloading from an IBM site. Upon completion of this update, VisualAge IDE will include an EJB tab and an executable environment (the WebSphere Test Environment).

IDE does not offer a visual development environment but is based on EJB class creation wizards.

EJB development is based on a group component concept. The first step consists of creating a group, in which EJB components are inserted.

The creation of components is handled in the same manner as with classic Java classes. The developer focuses only on the business methods and their implementation.

He must first choose the type of EJB component that he wishes to implement. VisualAge offers the support of Session Beans and Entity Beans. For Entity Beans, it is possible to choose between CLP Beans (Container-Managed Persistence) and BMP (Bean-Managed Persistence). These two options define the type of persistence selected for the component. It should be noted that support of Entity Beans is only optional in the 1.0 specification and that it will become a requirement in version 2. The developer then adds the methods and the fields that he wishes to see appear to the Remote Interface. Once the business methods are defined, it becomes necessary to deploy the component code. The product automatically generates all of the EJB interfaces and file stubs and skeletons necessary for communication between the client application and the server. The Deployment Description is also supported.

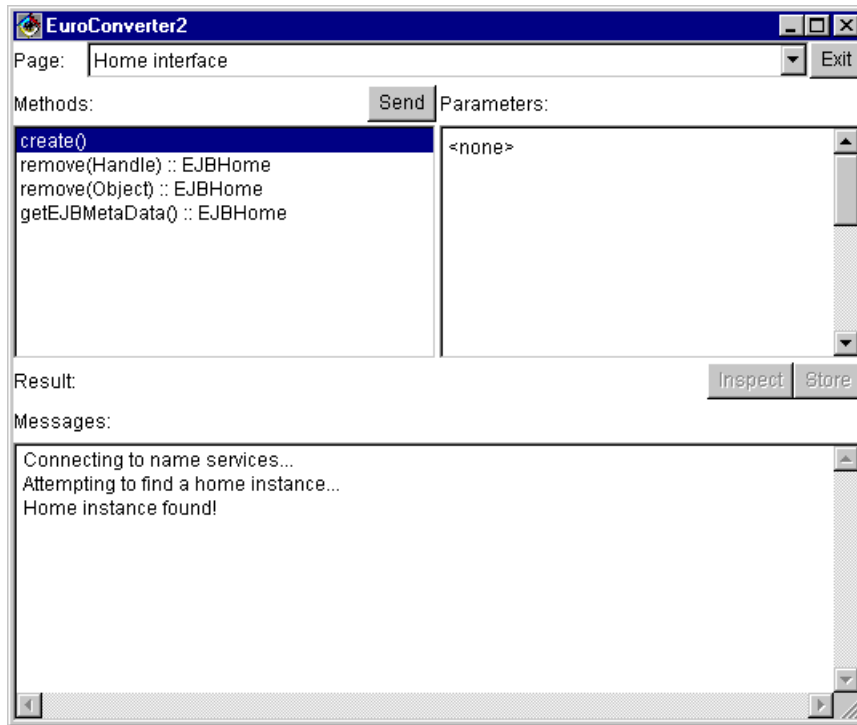
At this stage, it is always possible to add methods to our Bean, but it is necessary to deploy the code again so that the changes can be taken into account in the Deployment Descriptor. The development environment also provides the generating of the generic client application code.

Once these steps occur, it becomes necessary to import the group of our EJBs into the WebSphere test environment. This module is not an application server such as WebSphere; it is there only to test of the validity of the code being generated. The test interface consists of the console server: the EJB Server Configuration.

Two servers are present in the console by default: the Location Service Daemon and the Persistent Name Server. They are needed for functioning of the environment and must be

started. The user then starts our server. If all goes well, the VisualAge console displays the message that the Server is listening.

The runtime environment is ready and the user must then start the generic client with the command run test client. This client application consists of a completely configurable generic console. It gives access to all methods of our Bean and displays the return of these methods.



*Generic client interface*

Unlike the development phase when familiarity with the EJB concept was not mandatory, it becomes necessary during execution that the developer be familiar with the order in which methods are to be invoked. In other words:

It is first necessary to establish a connection in order to initialize the context of the naming space. Once this connection is established, the client retrieves the Home Interface corresponding to the requested name and the client interface displays the list of methods. It is then possible to call the create() Bean build methods. The interface then displays the page pertaining to the Remote Interface, on which the user can select the business method to test by filling in the parameters.

This client interface is very practical. However it is nevertheless not portable and requires access to IBM proprietary classes. It cannot be used to access another EJB server. The WebSphere application server itself must also be modified in order to be queried by the client.

IBM also offers samples of code for calling EJBs from either a servlet or an applet.

Debugging and thread management is possible in the test environment. As for deployment, VisualAge allows for exporting EJB components in standard “jar” format. This tool handles the generation of the manifest file as well as the Deployment Descriptor file, which must be serialized. In theory, compatibility and portability of the jar file is not complete. In fact, VisualAge adds identification and authorization information in the manifest file, information that is not recognized by other EJB servers (notably GemStone). There is also another export format. In this way, it is possible to export an EJS. It is a jar file that contains all of the classes on the server side. The EJS fulfills the role of the EJB container and can be imported into WebSphere.

After having covered the development cycle of an EJB, we will now address the development of the object model specified in the scenario. It has a 1-n type relationship and 2 inheritance relationships. For reasons of persistence and simplicity of development, the selection of CMP Entity EJBs becomes necessary.

Theoretical limitations will rapidly appear concurrent with the design of the model with Entity EJBs:

- Inheritance is not supported by the EJBs. This implies that our model of a new automobile will not be typical of the automobile class and that the retrieval of all the automobiles will not include either the new automobiles or the used ones.
- It is impossible to specify the links for this type of 1-to-1 or 1-to-n relationship. In other words, the referential integrity at the object level cannot be assured and the retrieval of associated members is impossible (in this case retrieving cars corresponding to a certain model).
- If the user wishes to build an object model from an existing database, it becomes necessary to use the relational mapping tool offered in VisualAge: Persistence Builder. Since mapping is not specified in the standard for CMP EJBs, it is necessary to export these EJBs with deployment classes containing the IBM-specific mapping code. It will not be able to be exported to another EJB server. The container will create a table and its columns in order to ensure the persistence of its EJB. However, this solution in no way solves the lack of inheritance and the entity-relation links.

It is impossible to implement the object model with EJBs in view of the limitation of the standard. The fulfillment of the model would have been assured owing to the development of proprietary extensions to implement inheritance and entity-relation links. It however appears wiser to wait for the release of specification 2 of the EJB standard, which should standardize the entity-relation links, inheritance and relational object mapping.

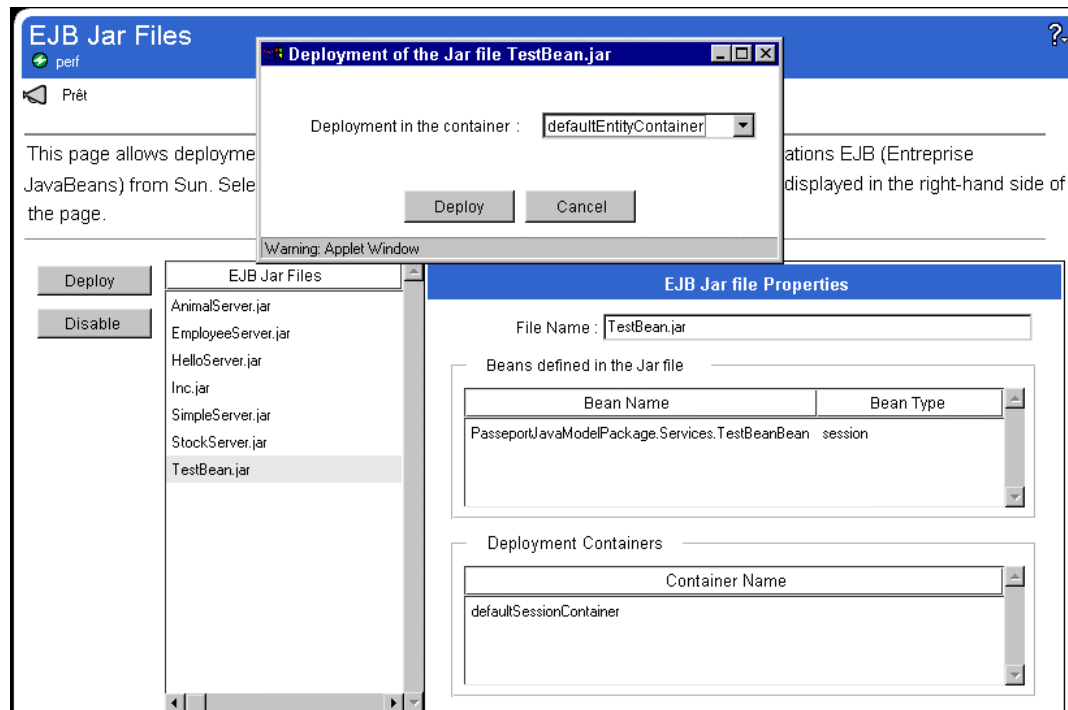
### ➤ **The Persistence Builder**

The Persistence Builder is covered in Step 12.

## Step 11: Importing EJBs

For EJB management, the console is quite simple and allows for the display of available components as well as deployed components, but, as is the case for servlets, the console does not permit access to the file system. In reality, it becomes necessary to copy its EJB components into separate directories. On the one hand, there is the DeployableEJBs directory, which contains the set of available EJB components and, on the other hand, there is the DeployedEJBs directory, which contains the deployed components.

Importing is handled in quite an intuitive way. All that is necessary is to select the container into which one wishes to import the EJB. The server suggests two containers by default: one container for the Beans Session and one for the Entity Beans.



*EJB administration console*

Importing is success, but it has proven impossible to access imported EJBs. Nor can the user directly use the client generated by VisualAge. It is thus necessary to develop a new client. The console does not provide specific monitoring functions for the EJB components. Nor can the user control the instances of these components.

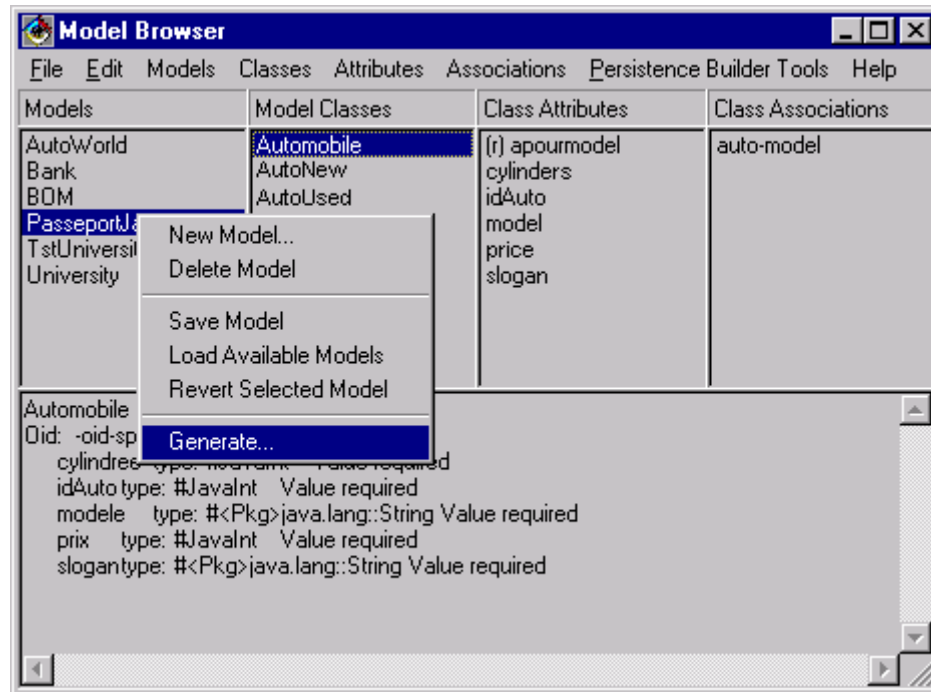
## Step 12: Persistence

As we have seen, the IBM product relies upon Enterprise JavaBeans. In this standard, two types of components can be made persistent: The CMP Beans (Container-Managed Persistence) and the BMP Beans (Bean-Managed Persistence).

Persistence management and development occurs in the VisualAge EJB development environment and relies upon the IBM persistence framework. This framework is based on an object-relational mapping technology, which provides for associating and synchronizing an object model in memory with a database.

The Persistence Builder product provides three types of object-relational mapping:

- First of all, the “top down” approach, which offers standard mapping within the definition of EJB components. Standard SQL code is then generated and allows for the creation of the tables required for mapping components.
- The “bottom-up” approach to mapping corresponds to the same operation in reverse. Starting with an existing database, the system imports the map of database tables and then creates the corresponding Beans.
- The third method, “Meet-in-the-middle”, provides mapping between an existing database and previously developed components. For this method, it is necessary to map the layout of the database by using the browser and then defining mapping between the existing EJBs and the database tables.



*The diagram browser*

Simple mapping is offered for these three methods, however it is possible to personalize this mapping function in order to achieve mapping suitable for more complex object models.

Although Persistence Builder Frameworks are functionally very rich, and reaching a certain maturity, the developments thus far cannot be integrated into WebSphere. IBM is making a lot of efforts in order to integrate the EJB development environment with Persistence Building for object-relational mapping. This tool is however not mature enough for development of EJB-based applications.



## Step 13: OTM and object distribution

The EJB distribution possibilities are interesting in theory. The naming service relies upon the JNDI API, and the distribution of an EJB to several servers is foreseeable (the implementation of JNDI in WebSphere relies upon the Corba naming service offered by the ORB integrated into the WebSphere EJB server).

Since the EJB server can be independent from neither the servlet engine nor the administration console, it is necessary to install several WebSphere servers in order to distribute the EJBs over several servers. EJB distribution to several servers is therefore not really integrated in version 2 of the product. IBM is announcing the possibility of distributing EJBs in WebSphere 3 to several machines, in particular to offer increased load functionality on the EJBs.

Regarding transactions, WebSphere provides two transaction management options for EJBs:

### *Transactions marked by the developer (Client-Demarcated Transactions)*

By using the `javax.jts.UserTransaction` API, the developer can specify the limits of the transaction in his code. The EJB in question must use the parameter `TX_BEAN_MANAGED` in the deployment descriptor so that the container does not manage the transaction in question. It is also possible to specify the level of isolation of the transaction in view of the inconsistencies (jumbled read, non-repeatable read and phantom line).

### *Transactions marked by the container (Container-Demarcated Transactions)*

When the development of EJB is completed, the developer will associate an isolation level with each method as well as a transactional attribute in the deployment descriptor. This attribute defines the transactional mode following which the container invokes the method (e.g.: `TX_REQUIRED`. The method invoked should be carried out in a transactional context; otherwise a transaction context is created). The advantage to this solution is that it prevents having to include the transaction code in the business code and to declare the transactional behavior at the time of deployment.

The server does not yet support embedded transactions.

If the implementation of EJB standard transactions in WebSphere simplifies the development of transactional applications based on components, it must still be proven since certain problems remain to be resolved:

- several databases could not be updated during a transaction (two-phase commit),
- support of all transaction isolation levels,
- concurrent transaction management.

## Ratings:

Object Server	IBM
<b>Step 10: Modeling by business objects</b>	☆
<b>Step 11: Importing EJB</b>	☆
<b>Step 12: Persistence</b>	☆
<b>Step 13: OTM and object distribution</b>	☆

Scores 0-3 (☆☆☆: 3; ☆☆: 2; ☆: 1; ●: 0)



## 6.4. Evaluation of the Inprise JBuilder 2 for Application Server



Version 3 of “JBuilder for Application Server” was released while we were carrying out the Java report and could not be evaluated in time. The features planned for JBuilder 3 include JDK 1.2 support, better Swing component support and a remote debugger. The major new feature for the Inprise Application Server involves EJB support.

### 6.4.1. Introduction

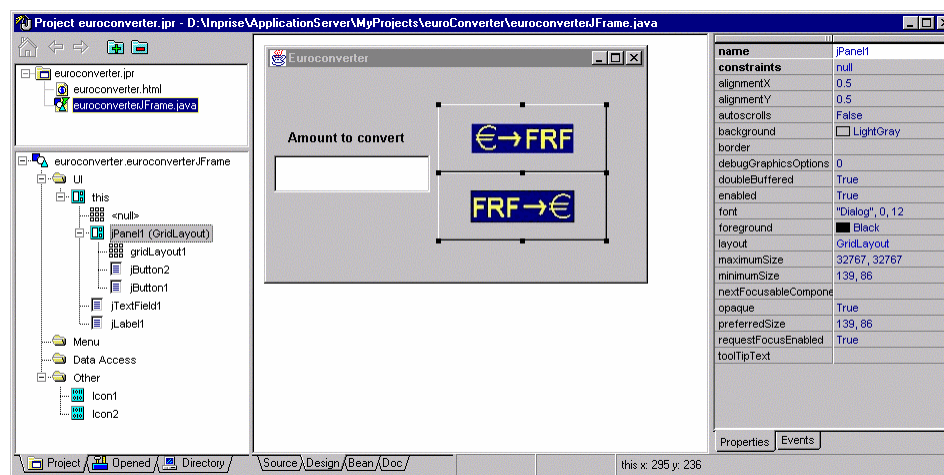
#### Step 1: Event-driven Interface

The evaluation of the event-driven interface will be conducted through the secondary steps for developing the Euro-Franc converter.

- The creation of a graphical interface
- The definition of methods and variables
- The creation of visual connections between components

#### ➤ Creation of the Graphical Interface

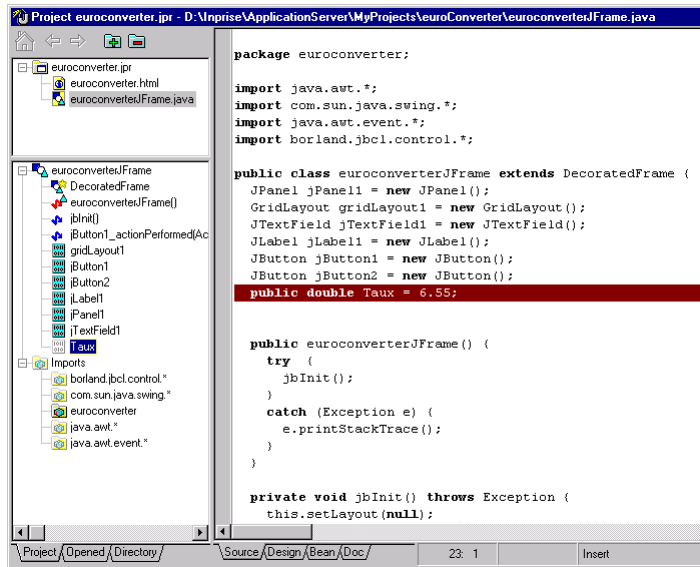
GUI is equipped with a palette containing Swing 1.0.1 components included in JDK 1.1.6 from Borland and delivered with the product. Palettes of Borland proprietary components complete the standard Java palette. The graphical components are inserted by drag and drop on the form and are stored in a graphic tree structure. Parameters can be set using the property editor located by default on the right-hand side of the screen.



*Euro-Franc Converter project*

## ➤ The Definition of Methods and Variables

The addition of methods, variables and objects is done manually. When a variable is manually created, it is filed in a tree structure of class components. By being directly connected to the source code, this tree structure is created from any Java file and is instantly synchronized with the source code when writing the code. The selection of a tree structure component (method or variable) directs the developer by selecting the corresponding code.



Source Code for Euro-Franc Converter

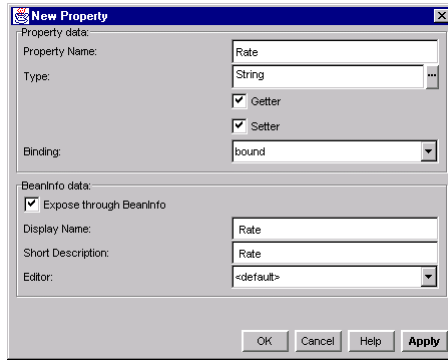
## ➤ Creating Visual Connections Between Components

The properties editor contains two tabs: the first lists the properties of the component and the other its events. To launch the conversion of Euros into Francs by pressing the button, simply select the button and click the event “ActionPerformed.” This operation displays the source code for the application and generates a method that will be called upon at the click of a button. The method is written entirely manually using an advanced syntax editor.

The increased productivity offered by this tool is average. There is no actual visual programming. Developers looking to maintain absolute control over the generated code will be very pleased by the approach put forth by Inprise.

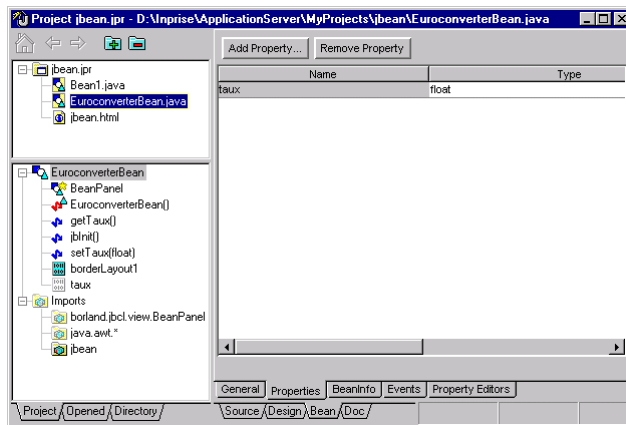
## Step 2: Creation of graphical components

JBuilder proposes a JavaBeans creation wizard called “BeanExpress.” It helps in the creation of Bean properties.



### *Addition of Bean Properties*

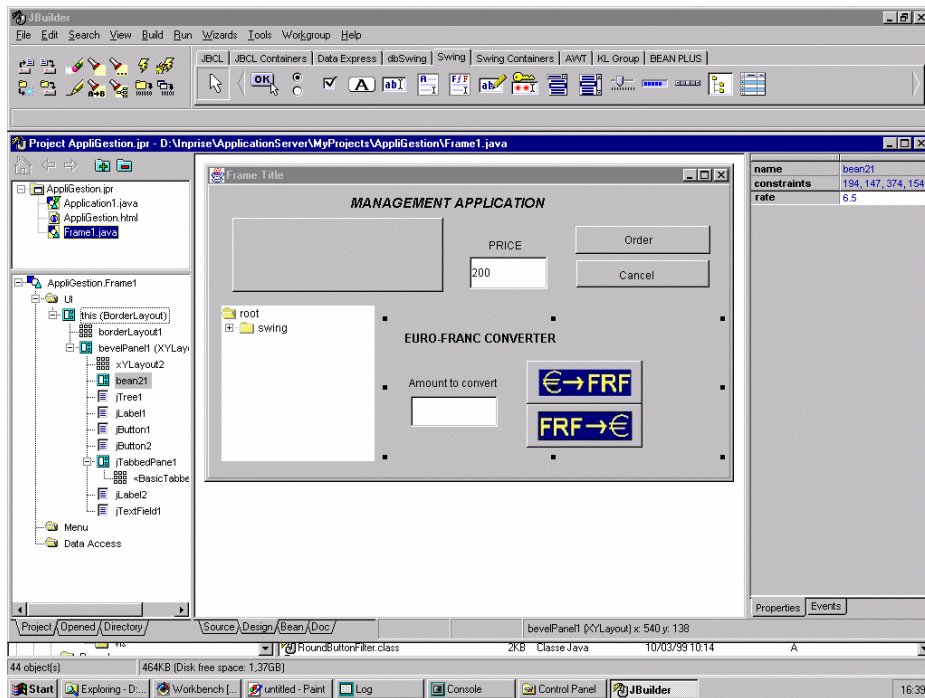
The Bean tab provides the means to edit the properties, events and BeanInfo class and to add or create property editors. The BeanInfo tab includes a button for generating the BeanInfo class. The Event tab indicates the type of event that can be notified or heard by the Bean. The necessary methods and interfaces are then added to the Bean. When this Bean is inserted into another application, the connection of an event to a method or property is done completely manually.



### *Bean Tab*

The insertion of the JavaBean into the palette involves two steps:

- Deployment of the JavaBean in a jar file (assisted by the deployment wizard)
- Loading of the file into the palette

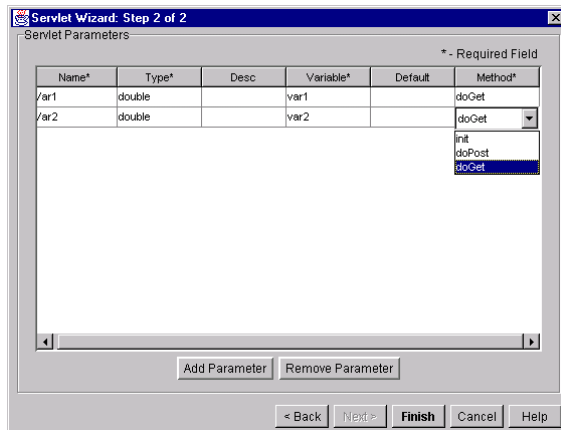


Management application

The above screen shows a management application in which the JavaBean converter has been inserted. The properties editor allows us to modify the converter's exchange rate using the introspection mechanism.

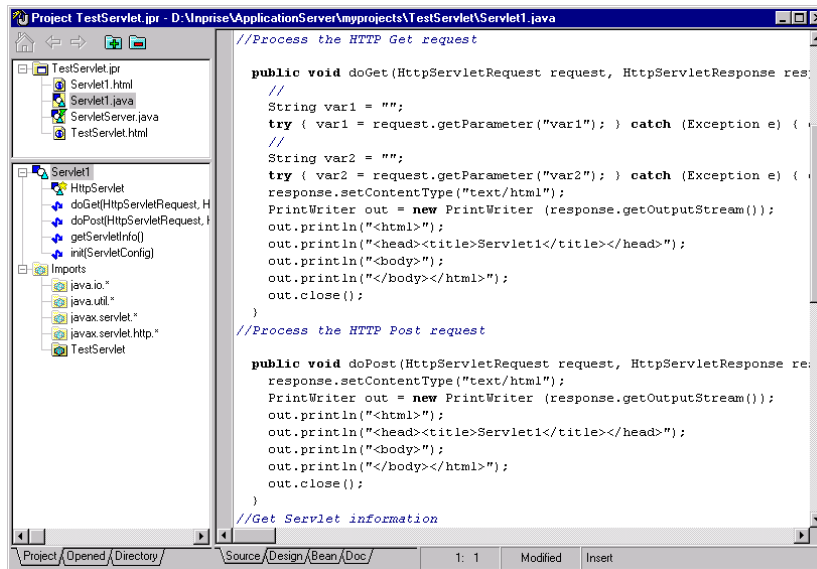
### Step 3: HTML Interface

Making a Euro-Franc converter available to people via simple Internet access can be done using a servlet. JBuilder proposes a wizard for the creation of such a class. After having chosen the methods implemented by our servlet (doGet(), doPost(), etc.), the parameters of the HTML form are added using a wizard.



Addition of parameters to the servlet

Clicking the “Finish” button launches the generation of servlet code and a minimum of HTML code loaded on the servlet. The remaining development is done entirely manually. No servlet visual programming is provided. The ServletRunner runtime environment is included in the tool's parameters and provides the means to conduct tests.



```

Project TestServlet.jpr - D:\Inprise\ApplicationServer\myprojects\TestServlet\Servlet1.java
TestServlet.jpr
  Servlet1.html
  Servlet1.java
  ServletServer.java
  TestServlet.html
Servlet1
  HttpServlet
  doGet(HttpServletRequest, H
  doPost(HttpServletRequest, I
  getServletInfo()
  init(ServletConfig)
Imports
  java.io.*
  java.util.*
  javax.servlet.*
  javax.servlet.http.*
  TestServlet

//Process the HTTP Get request

public void doGet(HttpServletRequest request, HttpServletResponse res:
  //
  String var1 = "";
  try { var1 = request.getParameter("var1"); } catch (Exception e) {
  //
  String var2 = "";
  try { var2 = request.getParameter("var2"); } catch (Exception e) {
  response.setContentType("text/html");
  PrintWriter out = new PrintWriter(response.getOutputStream());
  out.println("<html>");
  out.println("<head<title>Servlet1</title></head>");
  out.println("<body>");
  out.println("</body></html>");
  out.close();
  }
}

//Process the HTTP Post request

public void doPost(HttpServletRequest request, HttpServletResponse res:
  response.setContentType("text/html");
  PrintWriter out = new PrintWriter(response.getOutputStream());
  out.println("<html>");
  out.println("<head<title>Servlet1</title></head>");
  out.println("<body>");
  out.println("</body></html>");
  out.close();
  }
}

//Get Servlet information

```

*Servlet source code generated by the wizard*

This environment has only had limited use for the development of a Web converter. The HTML code must be introduced manually and the context management is the responsibility of the developer.

The product includes the Sun Java Web server and supports JSP technology. The development of dynamic HTML interfaces is therefore possible. Unfortunately, no JSP or HTML development tool is supplied and the joint use of JSP technology with the IAS server is not documented. However, JSP does provide the means to automatically manage HTTP sessions.

## Step 4: Tabular Data Display

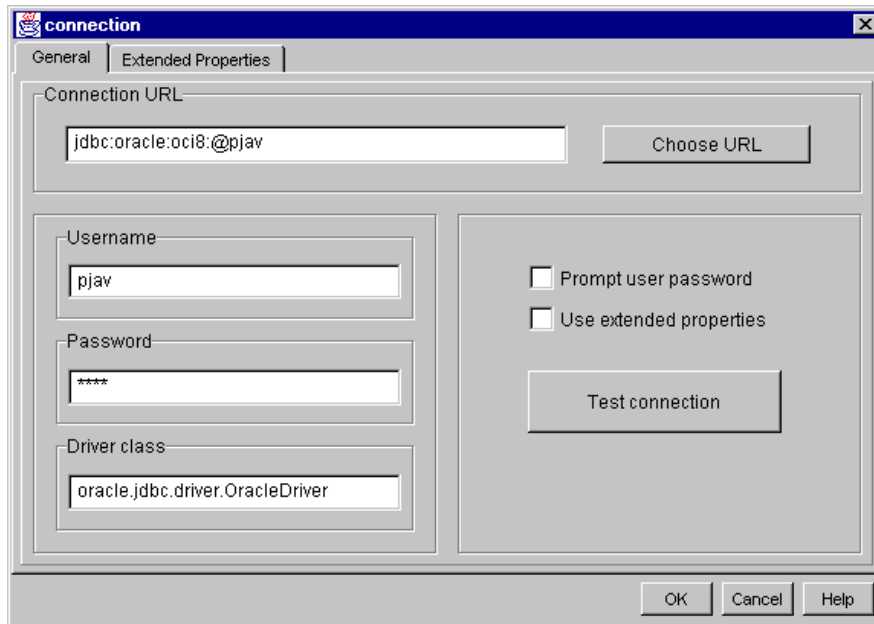
In this step, we are going to proceed with the creation of an access interface for our dealership database. No wizard is available for two-tiered architectures. Components specific to the databases are nevertheless provided:

- The DataSet components (non-visual elements): these components include the data source connections and data manipulation.
- The dbSwing components (Swing elements for databases): these Swing components committed to databases combine with Dataset components to post the data.

The step will be divided into two parts. The first will be connected to the presentation of tabular data. Assistance in the generation of SQL queries will then be dealt with.

## ➤ Presentation of Tabular Data

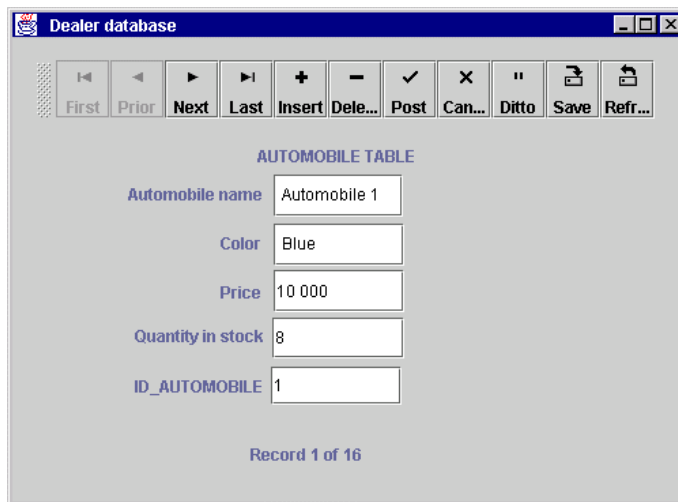
The first operation involves creating an object database. The object introspection mechanism allows us to specify the characteristics of the database and the JDBC drivers used, as well as to test the connection.



*Database Object Configuration*

The creation of the QueryDataSet object will allow us to create an SQL query (here “select \* from tautomobiles”) relative to the Database object created previously.

The construction of the graphical interface will be done using dbSwing components. Their Dataset property must then be connected to the different parts of the database. A navigation bar and a status bar are provided in the palette.



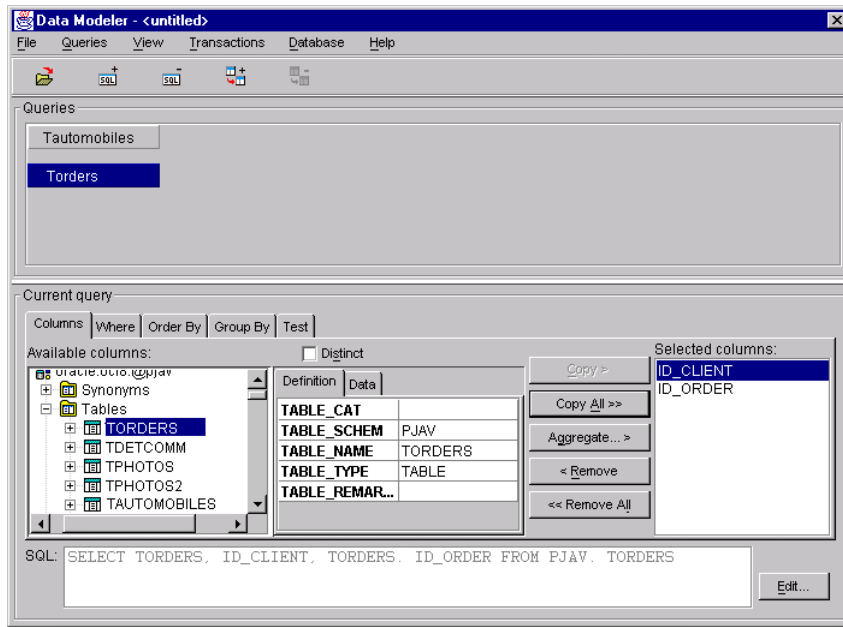
*Dealership form-type database access interface*



Some problems were run into during the creation of the interface when presenting data in the Jtable dbSwing. The scroll bars are not present, the width of the columns is not adjustable and the column names are not posted. Better Swing components support for databases is to be included in the next JBuilder version. The Borland proprietary GridLayout JBCL component compensates for this deficiency by correctly posting the data in a tabular way with the column names and scroll bars.

➤ **Generation of SQL Queries**

The “IDL DATA MODULE CORBA” wizard proposes to create a data access interface (HTML or Java) and a server using an IDL model based on a table, a view or an SQL query. It includes an SQL query generator spread out over five tabs and provides the means to define master/detail relationships.



Generation of SQL queries

**Step 5: Print Management**

By supporting JDK 1.2, JBuilder includes the print API, but provides no assistance in this area. The management of headers, footers, page breaks and other printing features are entirely the developer’s responsibility.

**Ratings:**

Interface Generation	Inprise
Step 1: Event-driven Interface	☆☆
Step 2: Graphic component creation	☆☆☆
Step 3: HTML Interface	☆
Step 4: Tabular data display	☆☆☆
Step 5: Print Management	●

Scores 0-3 (☆☆☆: 3; ☆☆: 2; ☆: 1; ●: 0)

## 6.4.2. Application Server

---

### Step 6: Openness to distributed objects

As a client, the accesses offered by App. Server for access to distributed objects is done via Corba.

Corba development is very well assisted. It must first of all go through the writing of the object server's IDL access interface. The writing is done using a simple syntax editor.

```
Model interface
{
    int idModel();
    string manufacturer();
    string description();
    setIdmodel(in int idModel);
    setManufacturer manufacturer (in int manufacturer);
    setDescription(in string description);
}

Automobile interface
{
    string idaut();
    Model idmodel();
    long cylinder();

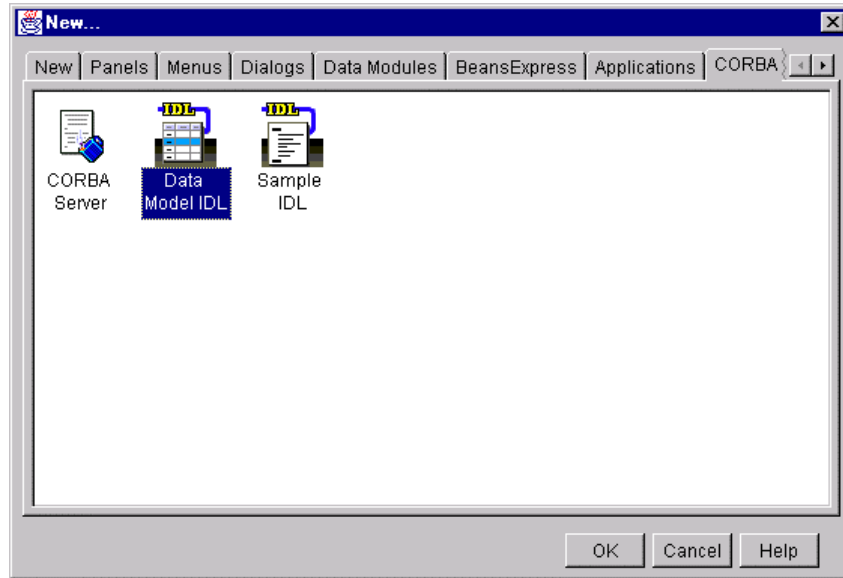
    /.../
    string slogan();
    int dureeGuarantee();
};
```

A tool called an application generator will then take care of the rest. It generates the Corba service files of the object server and a client equipped with a graphical interface that provides access. The ORB Visibroker integrated into JBuilder provides the means to test the client and the server. Once the application is finished, the object server publishes itself in the application server and the client gets easy access.

With JBuilder, it also would have been possible to generate the client based on the IDL contract and to get access to the object server in question already activated in the application server.

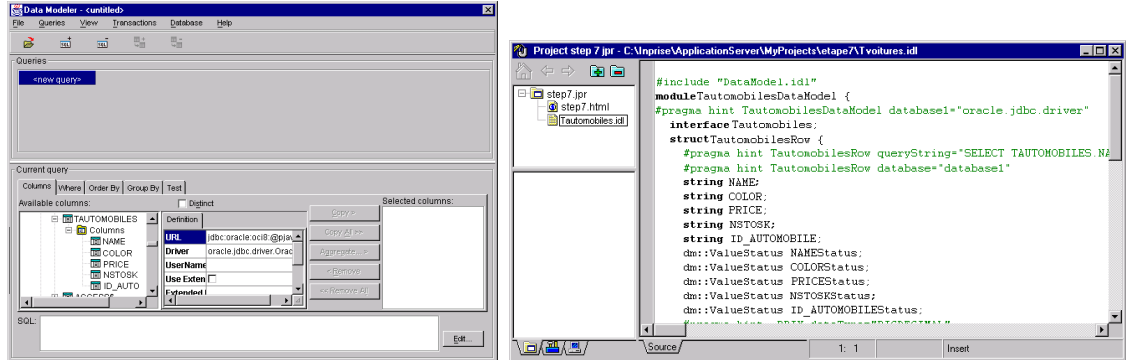
### Step 7: Database Access

The IDL Data Module wizard ensures the creation of a database access interface. This wizard provides the means to generate an IDL file based on a table, view or SQL query and thereby generate a three-tiered application focused on data.



*IDL data module wizard*

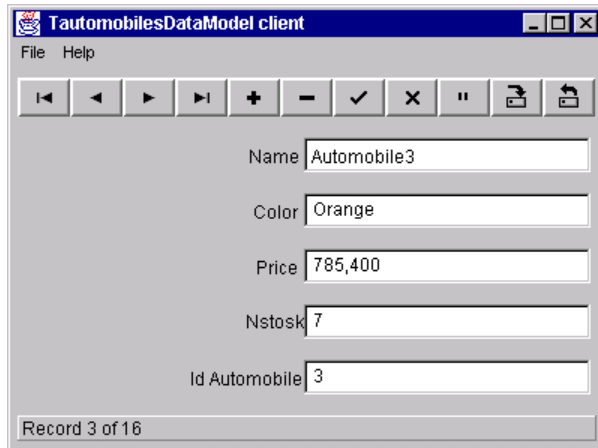
The first operation involves choosing a data source. A JDBC explorer is provided and gives access to the data source structures available on the machine. Users can then construct an SQL query or choose a table on which they want to work. The entire generation of an SQL query is done with the help of a wizard. The query can also be tested and modified manually. An IDL file is then added to the project.



*Generation of an IDL file based on the "Automobile" Table*

A wizard called an application generator can create the server and client components relative to the IDL file. Users can choose the client type: Java, servlet/HTML or both. There is a specific tab reserved for defining the graphical interface of each client. It should also be noted that the graphic components as well as their positioning can be modified.

The Java client provides the means to navigate through the table's records, as the form data synchronizes well with the basic records.



*Java Client of the dealership's database*

Inprise implements the Corba transaction services with Visibroker ITS (Integrated Transaction Services). The transactional aspect of this scenario is managed at the Corba object level. The IDL definitions of Corba objects that manage transactions will receive an interface called "CosTransactions::TransactionalObject." The Java objects stemming from these definitions will be capable of using the COMMIT/ROLLBACK mechanism using the ITS transactional context. The integration of JDBC calls at the non-XA data sources within ITS transactions is also possible with the JDBC DirectConnect driver. For XA data sources, ITS uses the ITS Resource Director.

A pool of JDBC connections is also proposed by JDBC Direct Connect. However, it offers only one option of parameters: the life span of a JDBC connection after a timeout period. The fact that we cannot define several pools with a minimum and maximum number of connections significantly limits the options of applications with access to the databases. It is impossible to reserve a certain number of connections for a given application and a large number of connections to a database in a small period of time can overload the server. Users will be disappointed by the absence of tools for the management of resultset lines. This is done with the help of the API JDBC 1.0, along with its functional limitations.

## Step 8: API richness

No API regarding the sending of mail is proposed in the Inprise product. However, it is possible to import outside classes to the product, notably API Javamail. No feature is provided for blob management or image processing.

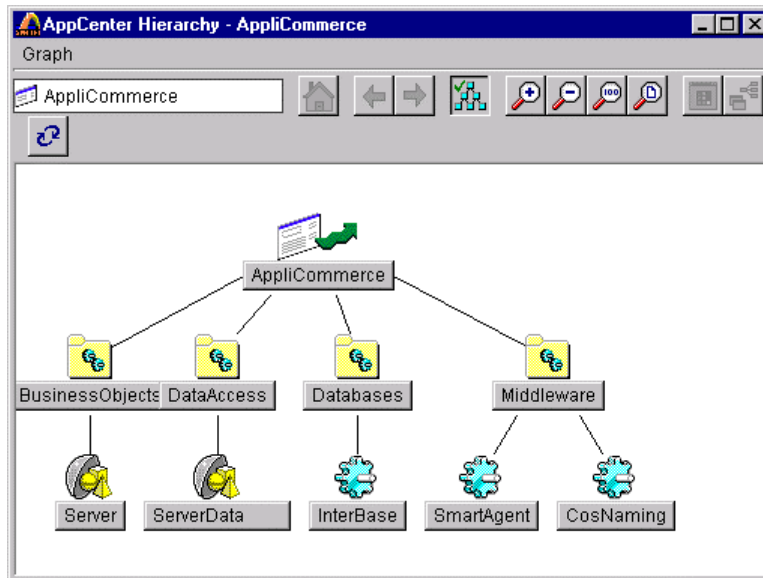
## Step 9: Workload management

Inprise has integrated into its Corba application server a distributed application management solution called AppCenter. Users can view and manage in a centralized manner an application composed of components as different as databases, Web servers and Corba or DCE components. Its subtle integration with IAS will provide the means to design management solutions for workloads and error recovery.

In order to properly understand the implementation of such solutions, the AppCenter management model is first of all described. The response to workload problems, fault tolerance and performance supervision will be dealt with afterwards.

### ➤ The Inprise Application Center Management Model

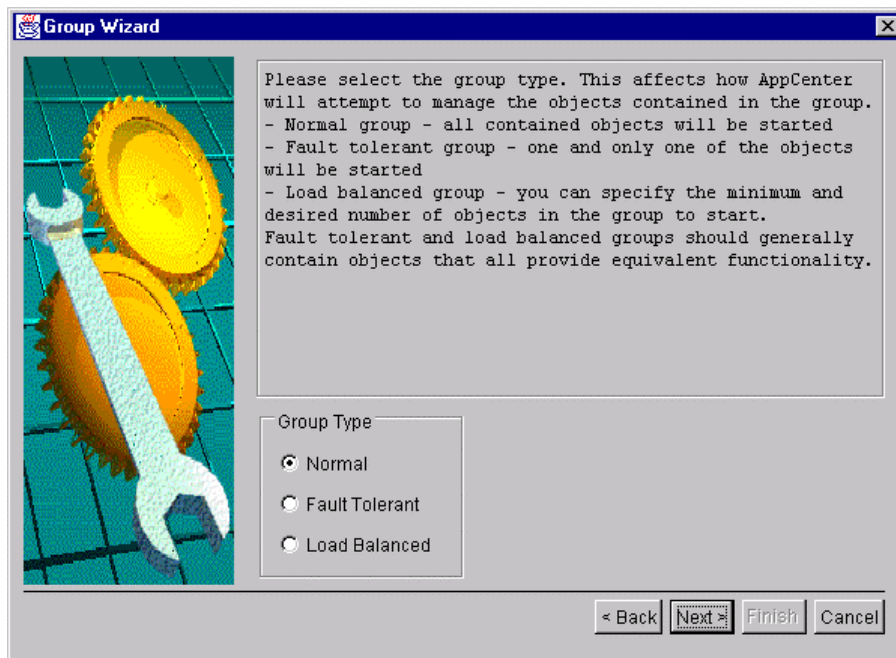
In this section, we are going to deal with the distributed applications management model proposed by AppCenter. It proposes a centralized view of an application's various components (middleware, object servers, databases) and their operating status. The definition of dependence links between these components provides the means to bring them together in groups and to launch the application in one unique operation. The screen shot below illustrates an AppCenter configuration during a launch. All the components of the tree structure have been launched.



*Representation of an AppCenter configuration during launch*

AppCenter provides a group of wizards for the creation of objects, Visibroker servers and Entera DCE objects. It is also possible to represent any executable program in the console. The user can then monitor the execution of a Web server or database manager.

## ➤ Workload and Error Recovery Management



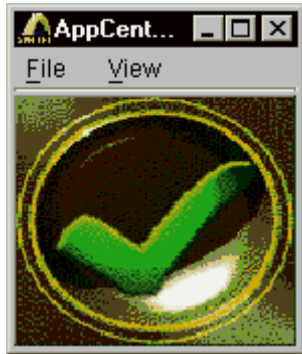
*Creation of an component group for fault tolerances and workload management*

In order to deal with the numerous connections on our electronic commerce application, AppCenter provides load-balancing groups in which are defined several instances of available objects. These objects are spread out on various physical servers. The IIOp queries routed by Visibroker are spread out between the various defined objects. All the parameters of the load balancing mechanism can be entirely set.

Error recovery is also ensured by fault tolerance groups. The first operation involves defining the group, the number of objects to be maintained in execution and the number of available objects. We then group together the various queries within the container. The launching of the application will automatically activate the number of objects wanted. If, for one reason or another, one of the objects is deactivated, the Visibroker middleware will redirect the IIOp queries towards the other waiting object. Another object will then be activated in order to deal with future incidents. It is important to note that if an object fails for some reason, its state will not be saved, persistence management being solely the responsibility of the developer.

➤ **Administration, Test and Monitoring Solutions**

Several monitoring tools provide the means to view the activities of objects present in AppCenter. These views can post readings in relation to the process in various graphics (electronic posting, graphs, counters, event logs, etc).



*AppCenter monitoring in icon mode*

Monitoring operators provide the means to view the performances of several processes (in the form of sums, averages, etc). The matrix chosen differs depending on the process it is analyzing. For example, the number of calls could be posted on an RPC procedure. The "icon" mode provides the means to ensure at a glance the proper operation of all applications. The slightest of incidents will change the appearance of the icon.

**Ratings:**

Application Server	Inprise
<b>Step 6: Openness to distributed objects</b>	☆☆☆
<b>Step 7: Database access</b>	☆☆
<b>Step 8: API richness</b>	●
<b>Step 9: Load-balancing management</b>	☆☆☆

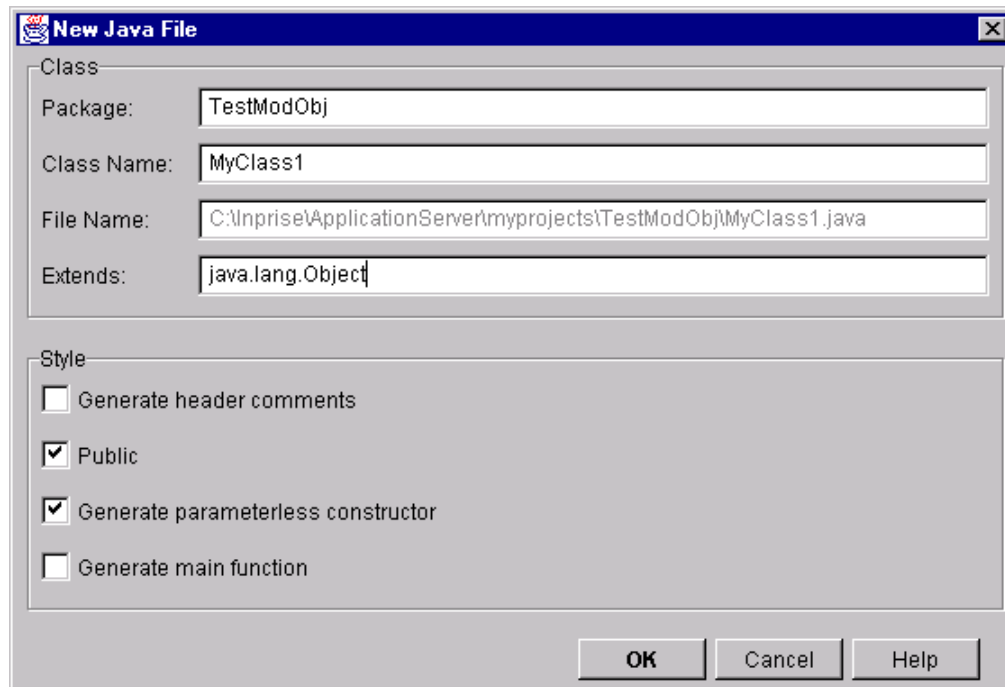
Scores 0-3 (☆☆☆: 3; ☆☆: 2; ☆: 1; ●: 0)

### 6.4.3. Object Server

---

#### Step 10: Modeling by Business Objects

The general Inprise product provides two options for modeling through business objects.



*Java Class creation wizard*

The first option involves completing a model composed of standard Java classes using JBuilder 2. A wizard provides the means to create a class skeleton (specification of the parent class, generation of the main() method, etc). Unfortunately, there is no tool allowing to view the hierarchy and classes. It is important to note that few services exist within IAS for using this type of model:

- The transactions are not supported
- Object distribution is done through Sun's RMI implementation, with the limitations that involves
- Security is the developer's responsibility through Sun's API security standard
- Persistence is the developer's responsibility through JDK's API serialization and a storage tool (JDBC, file system, etc)



Modeling with business objects is not viable with the help of Java standard classes.

The second solution involves implementing the model using IDL interfaces. This solution appears the best adapted to the product, the component model proposed by IAS being Corba. No modeling tool is proposed. IDL coding is therefore conducted using a syntax editor. If the implementation of such a model bogs down development, ORB Visibroker, to compensate, offers several services:

- Transparency of object location
- Presence of an events service
- Transaction management ensured by ITS, which has many complete features
- Security services ensured by the SSL option

The management of object persistence, a fundamental service in the completion of an object model, is unfortunately not offered for a Corba object model in App. Server and must be ensured by another tool.

### **Step 11: Importing EJBs**

The support of EJBs by IAS is scheduled for release in June 1999. An EJB beta version is currently available.

### **Step 12: Persistence**

No object persistence management solution is proposed in IAS. Inprise development teams are currently working on the support of EJBs and notably Entity Beans, which will provide the means to resolve this problem in normal circumstances. A beta version of the EJB server (Kodiak EJB Server) is currently available. Persistence management is the developer's responsibility by using Java serialization and the use of some sort of support (file, database, etc) or another object-relational mapping tool.

### **Step 13: Object transaction processing and distribution**

Based on ORB Visibroker, IAS offers the possibility of distributing C++ and Java objects across a network of machines equipped with IAS servers. The IAS server is administered from a graphic console. Transactions to Corba objects are supported by the ITS Visibroker product, which is integrated onto the server. Two versions of this console are available:

#### *Application Server Console*

This console allows to manage the Corba infrastructure and associated Corba services, notably naming services, events and ITS transactions (Integrated Transaction Services). The ITS service is administered using a console.





#### *Application Server Developer Console*

In addition to the services provided by the Application Server Console, this console offers the possibility of deploying applications in full graphical mode on the servers. The distributed applications are deployed within applications domains in which the different applications components are visible among themselves.

While the Inprise product allows distribution and transactions to Cobra objects using a well-integrated graphic environment, it fails to offer anything in the area of a DCOM model.

Development of an EJB server is underway and it will enjoy the benefit of a solid Cobra infrastructure. Once it matures, the EJB standard will allow the Inprise product to simplify transactional development for distributed objects a task that remains relatively complex with Corba.

**Ratings:**

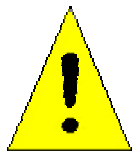
Object Server	Inprise
<b>Step 10: Modeling by business objects</b>	
<b>Step 11: Importing EJB</b>	
<b>Step 12: Persistence</b>	
<b>Step 13: OTM and object distribution</b>	

Scores 0-3 ( ★★★: 3; ★★: 2; ★: 1; 🚫: 0 )

---

## 6.5. Evaluation of JDeveloper 1.1 – OAS 4.07

---



Version 2 of JDeveloper and Oracle 8i were released during the development of Java report, and could not be evaluated in time. The principal enhancements to JDeveloper concern the integrated versions of JDK (1.1.7, 1.2 and the option to select the version of JDK), support of Swing components and more complete wizards, for both Java and HTML interfaces. Oracle 8i, also present on numerous platforms, offers among others the presence of an “embedded” JDBC driver, an EJB container, and a Java persistence API called JPublisher. Oracle 8i can, beyond its customary database server role, act as an application server, notably for very close processing of data.

---

### 6.5.1. Introduction

---

#### Step 1: Event-driven interface

In the Oracle product, the JDeveloper product was evaluated as a development platform. If the editor offers other tools or environments intended for or facilitating the production of HTML or Java transactional applications, only JDeveloper provides a sufficient degree of maturity to both graphically format the interface and produce the code required for the creation of business processing.

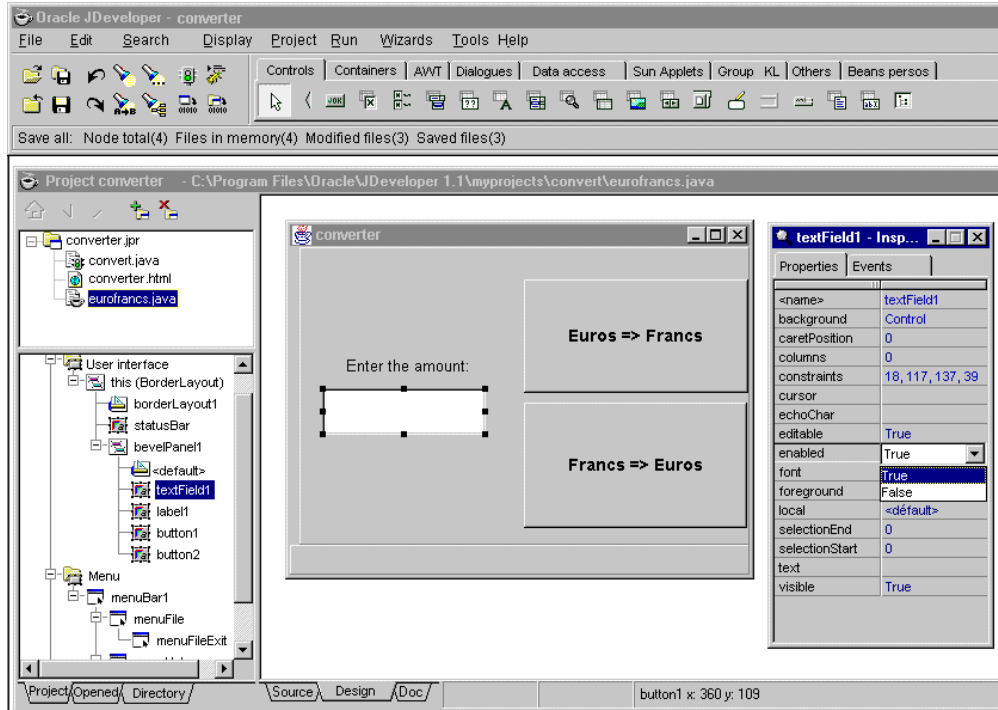
Thus, although other tools such as WebDB, Visual Page, Developer 2000 (Developer), and Designer 2000 (Designer) are found elsewhere, only JDeveloper is a true development platform for the development of transactional intranet applications.

In this first step, we wish to create a Java interface intended to convert a number of Francs to Euros. The points brought forward in this step concern the capacity of the IDE to assist the developer in the design of the interface, but also concern the possibilities offered to allow the objects to interact between themselves.

JDeveloper grew out of the acquisition of the Borland-Inprise JBuilder technology by Oracle. Adapted to its Oracle Application Server, the Oracle development platform relies upon Borland's experience in the implementation of L3G-L4G (Delphi, JBuilder, C++ Builder, etc.) development platforms, all of which interface with its own product and offer a set of complementary products, which cover in their quasi-totality the applications deployment-administration development cycle.

JDeveloper has a WYSIWYG graphic editor for the creation and editing of a Java graphical interface. Wizards already allow to create certain graphical skeletons. Thereafter, it is possible to subtly modify its window, either by providing it with new components by drag and drop from the palette of objects, or by repositioning the objects that are present using move, alignment, etc. functions. On the contrary, it lacks a positioning mask to increase the productivity at this level.

Even if the behavior of the graphical editor is not always predictable (difficulty in increasing the size of certain objects, difficulty in precisely positioning objects in certain locations), the designer will nevertheless manage to achieve the desired interface.



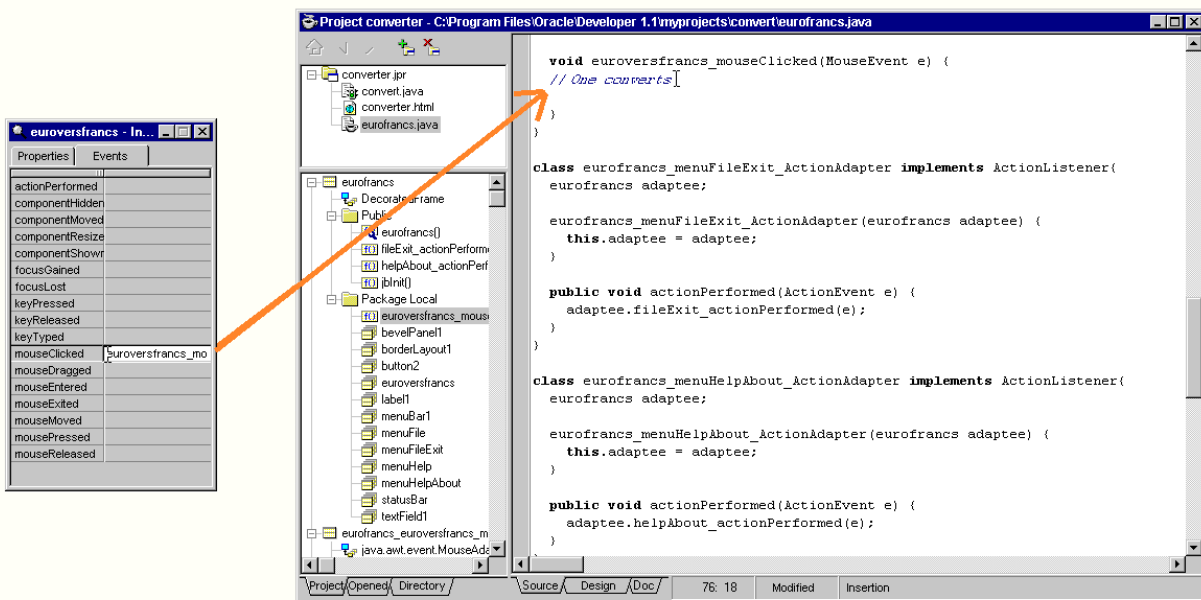
*The JDeveloper development platform is directly derived from Borland-Inprise technology*

However, it should be noted that JDeveloper version 1.1 only offers support of JDK 1.1.4. We should also point out the absence of Swing components and therefore the use of more cumbersome graphical components, but also the fact it is more “rigid” from a graphical point of view (linked to an interface with its own platform). So as not to be confined to the use of AWT package objects, JDeveloper offers a set of additional components from the JBCL package (JavaBean Component Library).

Provided by Borland, the set of components being offered covers areas as varied as graphic controls, database access, or event management. A hundred or so components are integrated into the JDeveloper development platform, considerably facilitating the task of designer but also of developer for anything related to database access management.

An objects inspector simply allows the properties of graphical objects in use to be edited. The events associated with each type of object are also offered. However, the principle strength of this inspector is the strong correlation between the object and the associated Java class source code. Thus, a double-click, for example, on an event of a button will automatically generate the code for an associated method and position the developer in the source editor in the corresponding location. Nothing particularly new here for developers experienced with Borland IDE, such as Delphi for example. In fact, the inspector and the Java sources are synchronized. Any change made in the inspector will automatically be made in the Java source and vice versa.

“Low level” developers will appreciate this programming mode, which, while completely automating a portion of code generation, nevertheless lets the programmer have a hand in all of the generated code in order to not limit the sphere of activity from a programming point of view. Unfortunately however, the interactions are not more extensive. If a wizard allows linking an action to an object event, the representation is not graphically formatted, and most importantly is not displayed using graphical links. It is therefore impossible to quickly see the entire set of interactions generated among the different components.



*A double-click on an inspector method automatically generates the associated method and positions the developer at the appropriate location in the source editor*

The hierarchical list represented in the IDE structures the content of Java files and allows rapid retrieval of all of classes, methods and properties present in the object or in the Java file being used.

## Step 2: Graphics component creation

The purpose of this step is to capitalize on the effort provided to create the converter interface, and to create a standard reusable component for JDeveloper developers, but also for developers dependent upon third-party products. The component model selected to respond to this type of requirement is JavaBeans. JavaBean specifications are widely known and this model offers edit possibilities through the use of inspector objects during the design phase.

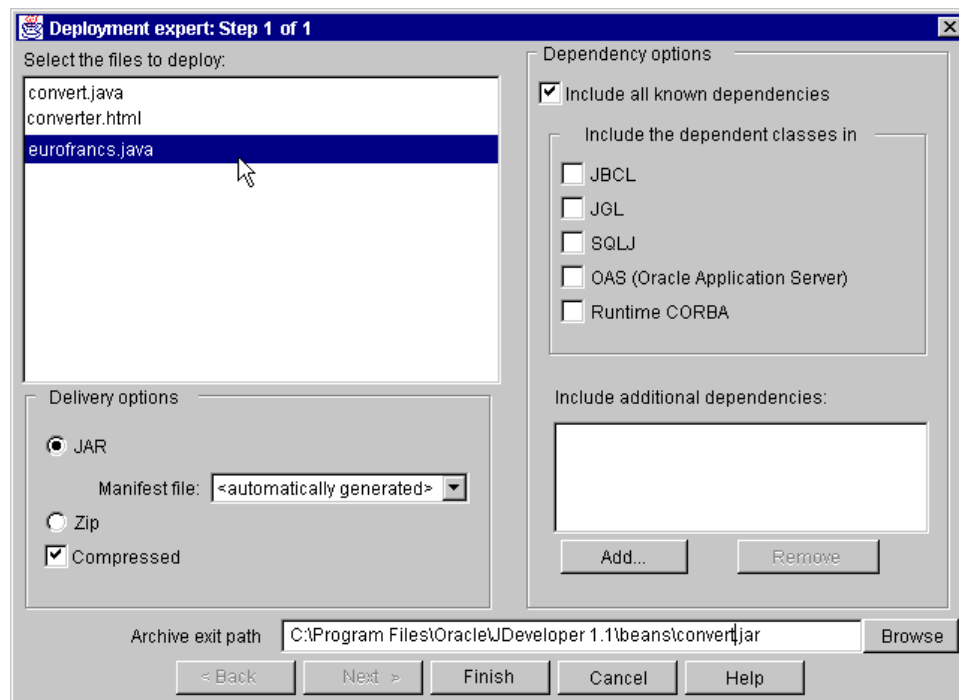
A wizard is offered with JDeveloper for the creation of JavaBeans. It generates the code necessary for the formalization of a component while adhering to the JavaBean specifications. The JavaBean skeleton is created and it then becomes possible to enhance it with objects or additional code (properties, methods, and events).

In addition, a BeanInfo wizard allows easy generation of a BeanInfo class. On the contrary, use of the BeanInfo interface is not really integrated, and it is not simple to enhance its JavaBean component to internally display newly added methods and events. It is necessary

to copy/paste, with the BeanInfo wizard merely generating a BeanInfo class implementation skeleton.

This is also the case for adding properties, methods, or events for the JavaBean component. Even during its creation, no option is available to enhance the component. Moreover, once the component is created, it is absolutely necessary to enter in the source editor all characteristics pertaining to the component. A graphics wizard would have been useful.

However, once the JavaBean is created, it becomes easy to add it to the components palette, and thus increase productivity. On-line help is associated with the component in order to offer the same ease of use as for the hundred or so pre-defined components delivered with JDeveloper.



*Deployment of JavaBean is assisted*

Exporting its own JavaBean component is also quite simple. The generation of a jar file with its manifest is handled without difficulty using the deployment wizard integrated into the JDeveloper development platform.

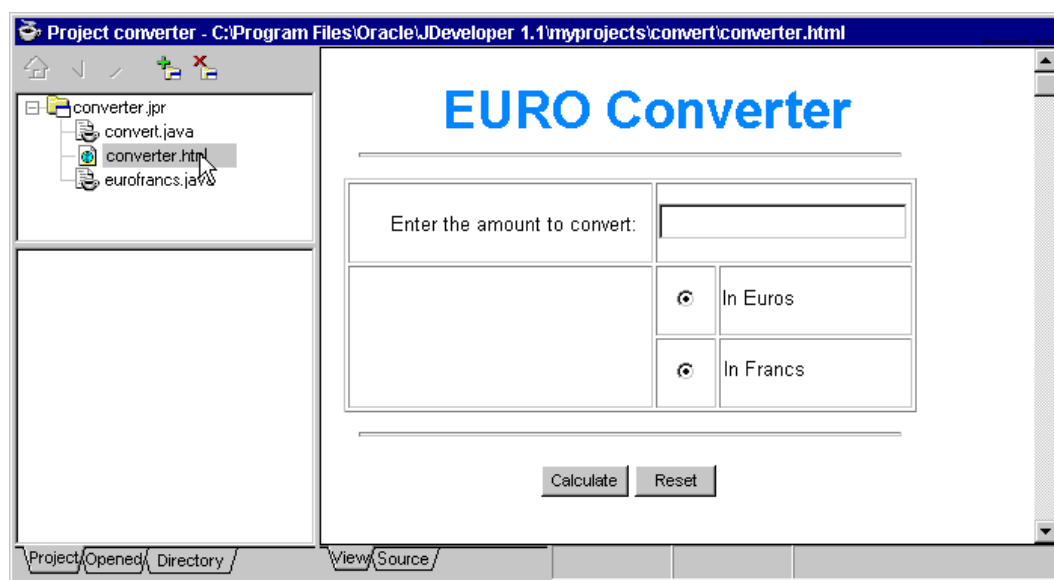
### Step 3: HTML Interface

This Java interface for the Euro-Franc currency converter is interesting because it provides high quality at the event-driven graphical interface level. However, it would also be a good idea to offer a lighter and more open solution such as HTML. The latter solution enlarges the sphere of use of the converter by providing a more nimble interface, which offers better response time, and allows each user with a browser to use the converter without the problems inherent to the richness of the Java interface (different functions possible depending on the platforms and browsers for the applets).

At this level, JDeveloper is still very much lacking. It should be noted that IDE is, above all, exclusively intended for the development of Java applications or for HTML interface applications, but dependent upon the use of Java applets.

The HTML wizard therefore only generates the basic HTML page skeleton, and no assistance is provided to go beyond that. A simple HTML “viewer” is integrated into JDeveloper in order to be able to have an idea of the appearance of a page in the browser. However, this editor does not offer the option of modifying the hypertext interface (no object moves, no text modification), or the option of graphically placing HTML objects. For all of these operations, it is necessary to go into the HTML source code, and enter the corresponding tags.

Once the HTML “template” is created, it is possible to generate the Java package, which will handle the interface between the static HTML page and the server components. The wizard, which generates this Java class therefore relies upon the available HTML template file, and will call it at runtime. The role of this class, which acts as a Java servlet, is to be able to retrieve the values entered by the user in the HTML objects, handle the corresponding processing, and return a new dynamic HTML page. To do so, the developer need only add the proprietary <WRB\_INC> Oracle tags, which will be interpreted on the server side by the application server.



*An HTML “viewer” is all that is present in JDeveloper*

In this regard JDeveloper relies upon the JWeb cartridge (Java) integrated into Oracle Application Server 4. In this way, the developer can count on the objects provided by the in-house application server, notably for user context management.

If JDeveloper is primarily a development platform dedicated to two-tier architectures (graphical interface in the form of an application or an applet accessing a database), Oracle Application Server on its side, allows for multi-tier transition. With this solution, deployments of HTML transactional applications are certainly attainable, and it is rather in the area of OAS 4 where it remains necessary to look for solutions. Other products in the Oracle product line rely upon this server application, and although JDeveloper is not yet the most mature product at this level for HTML interfaces, it already provides a first level of compatibility, which will

allow the developer, even if he will still have to enter many lines of code, a way to develop and deploy intranet applications.

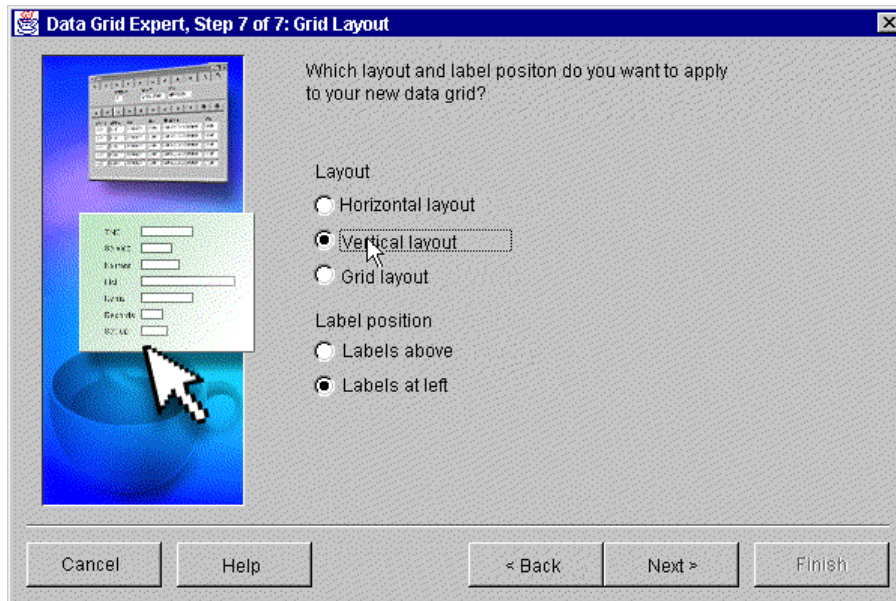
JDeveloper 2 will offer another alternative with the recycling of DBServlets available in Oracle 8i and in Oracle Application Server 4.08. These servlets rely upon databases, and run via HTML interface style sheets, with the option of browsing and updating data.

## Step 4: Tabular data display

Throughout the entire transactional application, it is necessary to display data from information databases. Most of the time, this information is stored in relational databases; however it may be in flat files, object or other databases. In all cases, the tool must allow the system to represent the blocks of data, which are to be displayed, and to easily navigate among them.

First of all, it should be noted that the absence of Swing components in JDeveloper reduces the potential for tabular data display objects, since DBSwing components are not offered. Nevertheless, JavaBean substitution components are integrated into the objects palette. Such non-visual components such as datasets are thus found, which allow the defining of database connection features. In addition, there are graphics components such as grids provided by JDeveloper, which allow to display the result-sets from databases in grids.

In this area, JDeveloper offers wizards which, from within a relational database, allow a developer to select one or more tables used to display, insert, delete or edit data. These wizards are quite concise, and offer two levels of appearance:



*The records display mode is configurable from within the wizard*

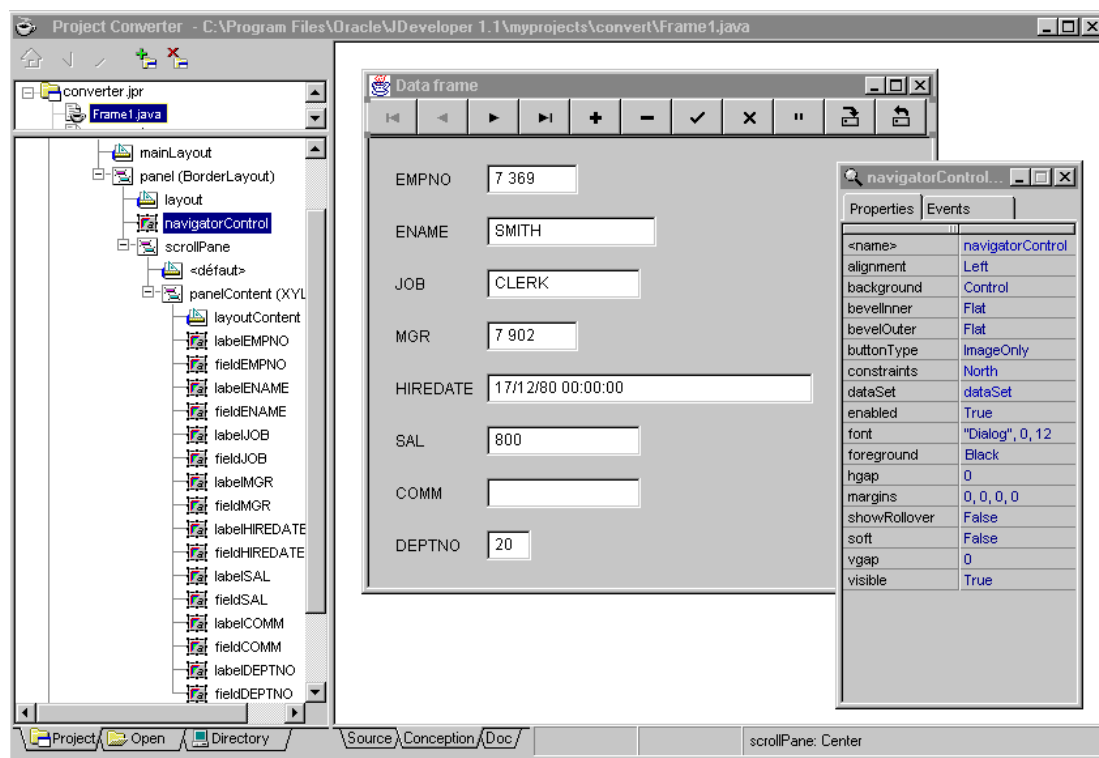
- simple representation: this is the form type or table display of the contents of a table or of a view. While several possibilities are offered from a final representation point of view, it is unfortunate to only be able to rely upon a single table or view (no join is possible), and to not, for example, be able to more specifically configure the request to execute a stored procedure, selection filters or sorts as needed.



- master/detail representation: Here, the window will be split into two parts. The first will show either the result from the table or the master view, while the second part will display the detail of either another table or view accessed by join. Everything is handled using wizards, and the result is fairly convincing. Here, likewise, the meager ability to configure at the wizard level is unfortunate.

Once the wizard has finished, it is possible to edit all generated code in order to fine-tune the request if need be. As for display in the tables, the option of running sorts by clicking columns, and the possibility of resizing columns are useful. Nor are there problems in terms of navigation among the different elements and the master components.

The JDeveloper tool does not offer a dynamic SQL query feature. Nevertheless, the wizards, even in the design phase, “graft” themselves to the database. Thus, once the wizard finishes, the results window displays the data in the first record in the table. Manually editing the query source code is immediately transferred over to the Java component WYSIWYG result. This means there is first-level query capability, even if a real SQL editor would offer more options and greater ease of use. It is necessary to exit the environment and use the SQL Worksheet (a utility provided with the Oracle database) in order to manage or query the database.



Even in the design phase, the graphics objects access the database

## Step 5: Print management

Printing is one of the essential requirements of any transactional application, regardless of its underlying architecture. The presence of a report generator is an undeniable asset, even if the principal expectation is the option to print documents on the server side.

JDeveloper does not support the Java print API provided with JDK 1.2 (version 2.0 of the IDE). Nevertheless, Oracle provides, with its "Report" product, the option to print documents on the server side in HTML or PDF formats. The greatest difficulty involves interfacing the different cartridges furnished by Oracle, in this case the Java cartridges and the Report cartridges. This integration is still far from optimal, and significant effort will be required before being able to easily reuse reports generated with Oracle Report and launched using JWeb or JCorba cartridges, which are interfaced with the Report cartridge.

Oracle Report, from its perspective, offers numerous possibilities. It allows to define headers and footers, and the body of the page, as well as to generate requests to fill text fields. However the great strength of the tool is its ability to create HTML and PDF formats. Oracle Report is nevertheless not integrated into the JDeveloper Suite product.

## Ratings:

Interface Generation	Oracle
<b>Step 1: Event-driven interface</b>	☆
<b>Step 2: Graphic component creation</b>	☆
<b>Step 3: HTML Interface</b>	☆
<b>Step 4: Tabular data display</b>	☆
<b>Step 5: Print management</b>	☆

Scores 0-3 (☆☆☆: 3; ☆☆: 2; ☆: 1; ●: 0)

## 6.5.2. Application server

---

### Step 6: Openness to distributed objects

A distributed object will often be reused by new applications. It is therefore important that the development tools offer the necessary wizards to facilitate the reuse of these objects. Wizards rely upon the internal display of objects, which is therefore necessary.

JDeveloper multi-level architecture, supported by Oracle Application Server, can be interfaced with Corba objects. Using an Oracle technology called JCorba, it is possible for OAS to communicate with distributed Corba objects. A Corba Oracle ORB is integrated into the product for this purpose so as to be able to use IIOP protocol.

However one of the other interesting features of this product is that Java applications, or Java applets, or even applications using JWeb cartridges, Perl or LiveHTML, for example, can also be clients of the Corba server application. Thus, this type of interface or the selected server technology allows all applications to use Corba for business processing on the server side. Oracle Application Server 4.07 likewise offers openness to Enterprise JavaBeans with a first level of support of EJB specification. An EJB container is integrated into the runtime engine, and for the time being, it exclusively supports Session Beans. There currently is no openness to DCOM objects, let alone via Java-based cartridges.

The Corba 2 model with IOP dialogue is currently supported by examples and explanations in the documentation. Oracle 8i, a veritable object server, will itself also offering Enterprise JavaBeans and ORB Corba VisiBroker. The OAS 4.08 application server product and the JDeveloper development tool will have to rapidly be updated in order to better support EJBs.

## Step 7: Database access

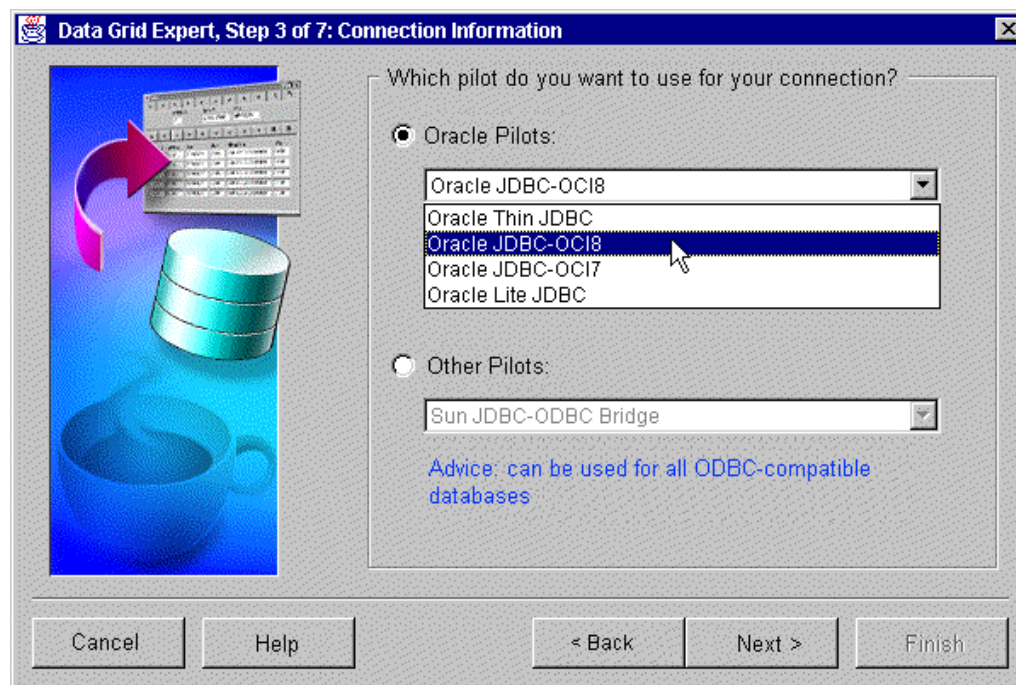
Database access is one of the essential ingredients for transactional applications. The application server must provide all indispensable elements to prevent this feature from causing deployment-level difficulties.

JDeveloper allows to deploy applications in two-tier architecture. In this functional mode, database access is handled exclusively by JDBC drivers. JDeveloper provides two JDBC drivers to access Oracle databases:

The drivers based on OCI and the “Oracle Thin JDBC Driver” or JDBC Lite driver.

The JDBC Lite driver will be the driver of choice when it is necessary to access the database from a Java applet. In fact, by itself, it can be executed directly from a browser. This driver will also be used when it is required to indiscriminately access an Oracle 7 or Oracle 8 database from either an application or from a Java applet. In other situations, meaning for a Java application exclusively accessing a database (Oracle 7 or 8), the JDBC OCI7 or OCI8 driver will be used, and will offer more features linked to the selected data.

- “Oracle Thin JDBC Driver” is a type-4 JDBC driver.
- “Oracle JDBC-OCI8” and “Oracle JDBC-OCI7” are type-2 JDBC drivers.



*Several type-1, 2 and 4 drivers are integrated into the JDeveloper Suite product*

However, JDeveloper also integrates a type-1 drive, meaning a JDBC-ODBC driver. With this driver, it is possible to interface with other databases such as SQL Server, Sybase, Informix, Access, etc. or even Oracle if the database in deployment phase is not first defined in the design phase. We should note that JDBC-ODBC driver cannot access a database from Java applets either. In this situation, it becomes necessary to either pass through a Java application or to provide oneself with an appropriate JDBC driver.

The connection pools are controlled by the application server. Oracle offers the most optimal management in this case by opening connections based on the number of simultaneous accesses and users.

For access to Oracle databases, JDeveloper offers all the functionality necessary for navigation in the results-sets. The objects that are provided allow navigating without any problem among these records, notably due to the presence of modules such as "data access" components, which allow for the definition of all elements from database connection to the query for access to this database.

It is unfortunate, however, to not be able to display the contents of the database at the JDeveloper level. No graphical representation is offered, but most importantly no hierarchical display of the contents of the database is available.

For concurrent access management, the developer relies upon the JCBC offering.

At the Oracle Application Server level, the interface with the databases is notably dependent upon the cartridges being used. Through the use of a PL/SQL cartridge, access to the Oracle database is native since the "application" code is merged into the code linked to the databases. To access databases other than Oracle, it is necessary to use an ODBC cartridge.

For JWeb or JCorba cartridges, database access is via the JDBC drivers delivered with JDeveloper.

Depending on the cartridge being used, the developer will be able to access the databases via PL/SQL or by using the JDBC API for Java development. However Oracle, for the latter solution, facilitates the developer's task by offering a much less tedious technology than JDBC: the SQLJ, which is ANSI standard.

The SQLJ code allows to directly integrate SQL database access within the Java code. Less complex than JDBC, and much more concise in terms of lines of code, this solution will be a very interesting alternative in the majority of cases when using the JDBC API. In fact, as long as the SQL query level is not very specific, this solution will be favored.

An example of SQLJ code:

```
Int price;  
String nameAutomobile = "Voyager";  
  
#sql { SELECT price INTO :price FROM Automobile WHERE idAut =  
:nameAutomobile};
```

The system is simple. A precompilation tool is provided with JDeveloper, which converts the compilation of SQLJ Java classes to JDBC code. We should note that the debugger adapts to SQLJ, which strengthens the choice of this technology. In addition, it is possible to reconcile SQLJ and the JDBC API within the same program, which thus allows for better enjoying the benefit of both solutions according to the required problem set.

Finally, we should note that for the reuse of procedures stored in Java, JDeveloper is furnished with a wizard, the PL/SQL Java Wizard, which generates Java code to address any stored PL/SQL procedure.

## Step 8: API Richness

Numerous specific requirements are currently needed for application servers. Sending mail on the server site, possibly owing to multi-level architectures, the generation of images and the support of blobs are frequent problem sets in the context of an enterprise application.

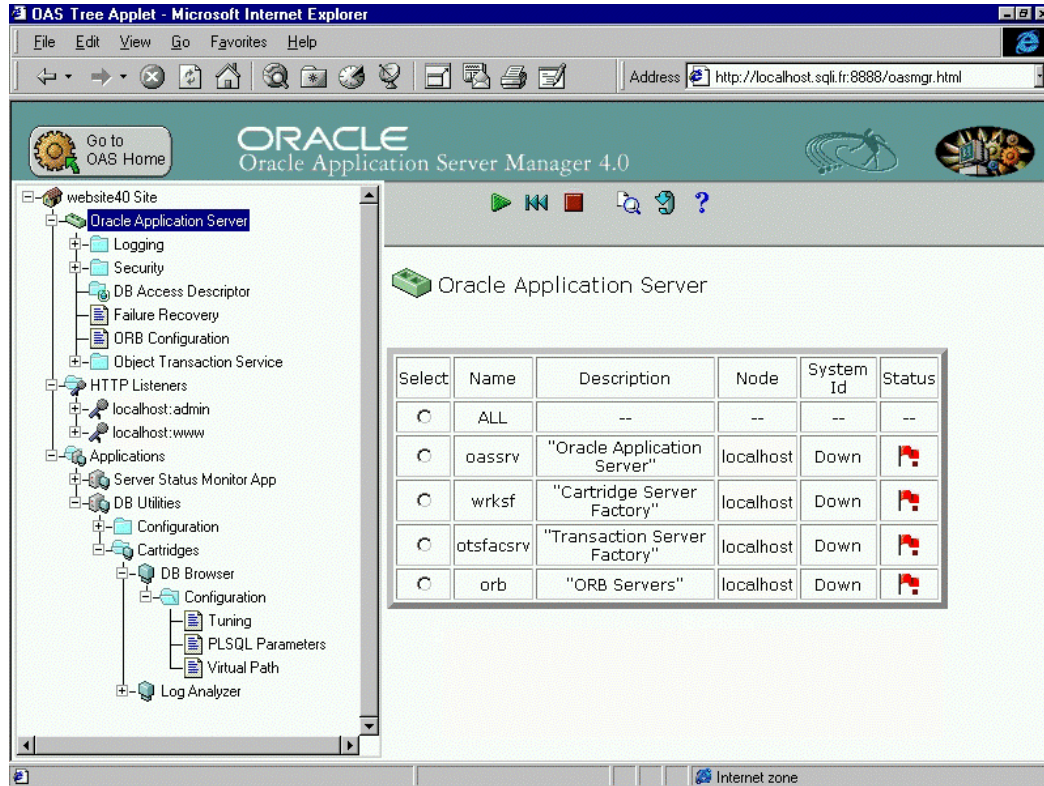
Oracle Application Server offers numerous solutions for blob management. With a strong interconnection with Oracle, the possibilities at this level are de facto sufficiently natural, and flexible. Examples of the code are furnished for reading and writing blobs. It is unfortunate, however, at the JDeveloper level, that automation of blob control is unavailable, as this would be useful, for example, to represent images in windows.

For sending messages, it is necessary to be interfaced with a messaging server. While this functionality can be implemented with Oracle, it nevertheless requires considerable coding using the Java APIs. No assistance is offered at this level other than perhaps the possibility of retrieval with the use of another Java package such as, for example, JavaMail.

For generating images on the server side, for example from a data source, Oracle does not offer any options at the OAS level. For this, it becomes necessary to be interfaced with a specific cartridge, not integrated into the JDeveloper Suite, with the customary inter-cartridge interconnection problems.

## Step 9: Workload management

In an intranet environment, the real essence of the application, at least for multi-tier architecture, is concentrated on one or more processor servers (application servers). It is at this level that most processing will be executed, which can rapidly pose problems in terms of performance (CPU consumption, memory, etc. See *“Comparative Study of Intranet Development Tools”*) as soon as the number of users becomes significant. The application server must therefore offer mitigating solutions at this level, as much to guarantee proper response time as for reliability of applications.



*The administration tool provides a view of all currently running applications*

The latest version of the Oracle application server, Oracle Application Server 4, has advanced considerably in the area of load balancing and in the administration and support of failover. Still a single server processor in version 3 (Web Application Server), OAS 4, now allows to divide applications processing across multiple machines.

In order to accomplish this, an administration utility (Oracle Application Server Manager), which may be used as a Web application in a browser, provides for a logical display of all of the servers being used as well as of the deployed application instances. This tool allows stopping and starting these applications objects. The administration tool interface nevertheless uses Java applets, which reduce performance in situations where they are used intensively.

This administration utility also provides applications use information, thus allowing easier tuning between them. The statistics offered are nevertheless not very abundant nor can they be configured, but they do provide first-level assistance.

Load balancing can be configured, even if selecting the algorithm is not possible. The administrator can weigh several levels of use for one resource or another. It thus becomes possible to distribute, by percentage according to the power of the servers, the number of cartridges being used in order to maximize or minimize the number of cartridges on each of the servers, etc. Recovery from incident (failover support) is also exists in the Oracle Application Server, and it will automatically restart each stopped cartridge.

**Ratings:**

Application server	Oracle
<b>Step 6: Openness to distributed objects</b>	☆
<b>Step 7: Database access</b>	☆☆☆
<b>Step 8: API Richness</b>	☆☆
<b>Step 9: Workload management</b>	☆☆☆

Scores 0-3 ( ☆☆☆: 3; ☆☆: 2; ☆: 1; ●: 0 )

**6.5.3. Object server****Step 10: Modeling by business objects**

Modeling of classes using JDeveloper is simply handled by writing the Java classes from the code editor. To facilitate this feature, JDeveloper furnishes wizards, which now generate page skeletons. Thereafter, no visual representation of its objects graph is presented, which does not facilitate the task of designer.

With Designer and a new version of JDeveloper, Oracle hopes to soon be able to offer the option to build its objects graph using UML.

**Step 11: Importing EJB**

Nothing is expected at this level for the time being. The arrival of JDeveloper 2 and Oracle 8i will probably improve things in the EJB area, and, thus, their importation.

**Step 12: Persistence**

The majority of time, data persistence in client-server projects is ensured by relational data. With an object model, this functionality, indispensable for application reliability, should also be provided in one way or another.

At this level, Oracle does not offer any persistence support for Corba, its distributed object model. It is necessary to handle this oneself, with programming using, for example, a relational database.

JPublisher will be part of Oracle 8i. JPublisher is a Java-persistence API. This feature has not been evaluated as part of this study.

**Step 13: OTM and object distribution**

From the moment when the application depends upon memory data stored in objects, it becomes necessary to offer the same level in terms of transactional processing as is offered during database access. For its own account, the distribution of objects intervenes to facilitate transversal deployment at the application server level.

The distribution of objects in Oracle depends on the Corba 2 model, through the use of JCorba cartridges and through the presence of a Corba ORB integrated into the product. Oracle Application Server implements the JNDI service thus providing for the retrieval of Corba objects. The editor nevertheless does not offer a graphics tool to simply partition its applications.

Among the features facilitating the distribution of objects, it should be noted that JDeveloper allows the generation of a deployment file for Corba objects. This file, JCO.APP, contains characteristics specific to JCorba applications such as the name of the application, the number of instances to be started, or even the inactivity time before generation of a time-out. EJB technology at this level is drawing nearer.

For object transactions management, the OAS Corba ORB implements OTS (Object Transaction Service). Conversely, for persistence management, which must be handled manually, the OTM will pass through thorough persistence management in order to offer consistent and reliable object transactional processing.



*The JCO.APP module is a deployment descriptor for a JCorba application*

In version 4.07 of Oracle Application Server only the distributed object model, Corba 2, works well. Possibilities are nevertheless offered at the EJB level, but the implementation in the 4.07 version of Oracle Application Server is far from complete.

**Ratings:**

Object Server	Oracle
Step 10: Modeling by business objects	●☼
Step 11: EJB Import	●☼
Step 12: Persistence	●☼
Step 13: OTM and object distribution	☆☆

Scores 0-3 (☆☆☆: 3; ☆☆☆: 2; ☆: 1; ●☼: 0)



## 6.6. Evaluation of Symantec VisualCafé 3



The new version of VisualCafe, “VisualCafe Enterprise Suite”, came out during the preparation of Java report, and could therefore not be evaluated in time. The new functions planned for VisualCafe are notably RMI, Corba and EJB support, as well as the presence of a remote debugger.

### 6.6.1. Introduction

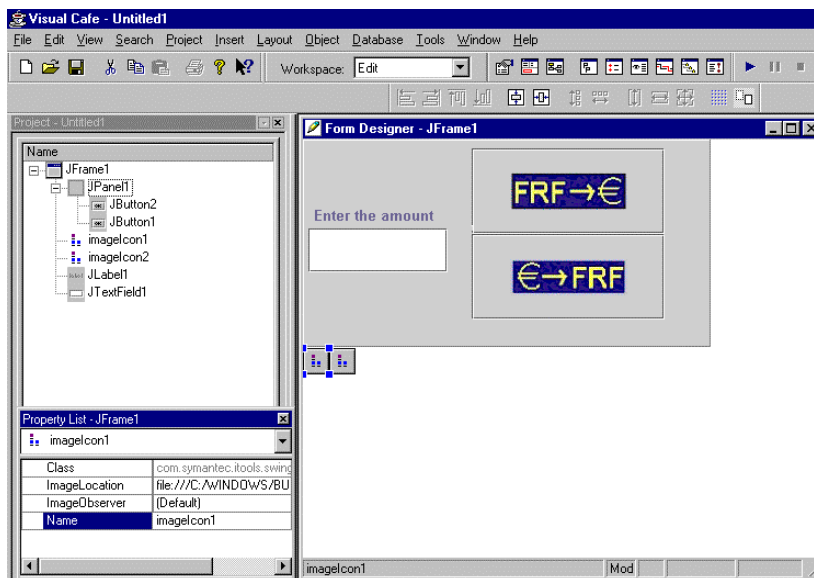
#### Step 1: Event-driven interface

The evaluation of the event-driven interface will be done during the substeps of the development of the Euro-Franc currency converter.

- the creation of a graphical interface,
- the definition of methods and variables,
- the creation of visual connections between components.

#### ➤ Creating the graphical interface

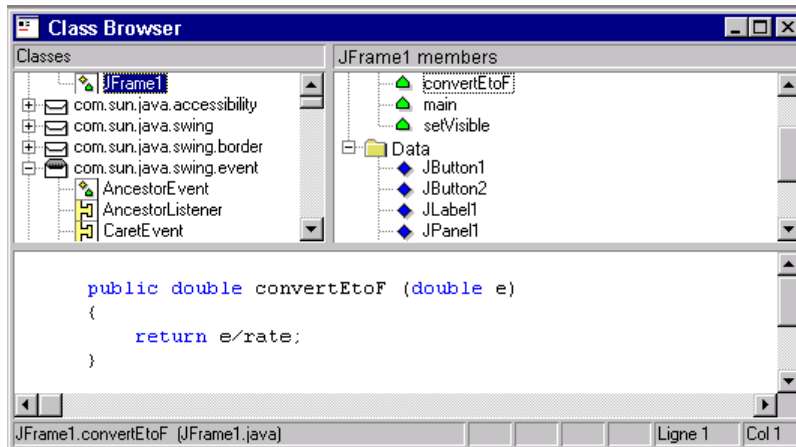
The graphics component palette offered by VisualCafe is very rich. The Swing components are supported in their totality. The addition of Symantec 100% proprietary Java graphic components (AWT Multimedia, Swing addition) will satisfy the most exacting users. The addition of a graphic component is handled by drag and drop. A grid similar to the one found in Visual Basic facilitates the positioning. The properties of components are easily modifiable through the properties editor.



*Implementation of the currency converter graphical interface*

## ➤ Definition of methods and variables

The consultation, creation and editing of methods and classes are handled through the classes navigator. It offers a display of environment packages, classes and their associated methods. It also offers variable- and method- creation wizards. The method-creation wizard requires method declaration and provides a choice between several modifiers (public, private, etc.). After this, the coding of the method is handled manually.



*Classes navigator*

The classes navigator is ergonomic, and its hierarchical display allows access and rapid editing of methods and environment class variables.

The following elements were created:

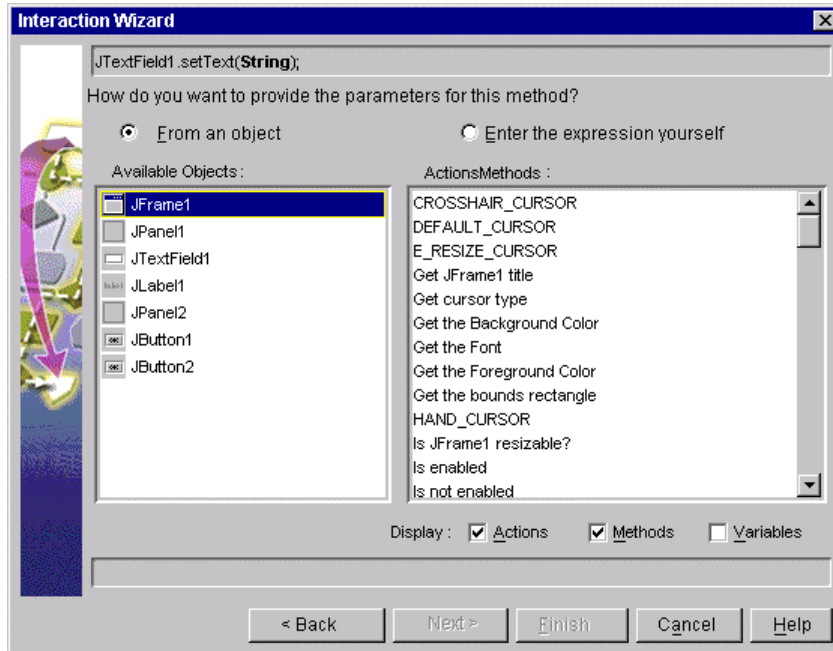
- the method "double convertEtoF(double e)"
- the method "double convertFtoE(double f)"
- the variable Rate (rate of conversion between currencies)

## ➤ Connection of visual connections between components

The following step consists of connecting events controlled by the user to methods created with good parameters. The interactions editor was designed for this purpose.

The link between clicking the Euro to Francs conversion button and the method convertEtoF is handled in the following manner:

The interaction editor asks what graphical event will trigger the rest of the operation. The type of element which will be associated to the action is suggested (method, action and property) and a list of available elements is suggested. In this case, it is the method "double convertEtoF(double e)". The editor will detect that a double-type parameter is present in the method being called, and will assist us in selecting it. In this example, it is the text in the "Sum to Convert" field.



*The interactions editor*

Validation of the connection will create an arrow connecting the button to the exterior.

That it is not possible to retrieve the result of the method visually is unfortunate. In this case, ideally it would be possible to retrieve the result of the conversion in order to display it in the text field. This is impossible with VisualCafé. An alternative is to use a `java.lang.Double` object in the forms editor. The simplest solution consists of editing the code generated by the interaction. The integrated syntax editor allows to retrieve this result to display it in the text zone.

VisualCafé, however, does not provide any means to sequence the visually programmed actions, which are triggered by the same event.

The application test gives us the following result:

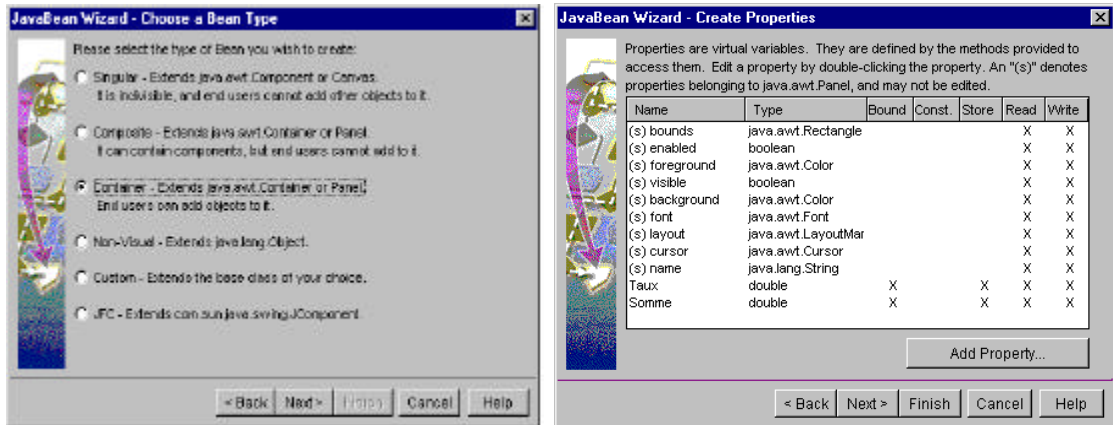


*Java Euro-Francs converter application*

## Step 2: Graphics component creation

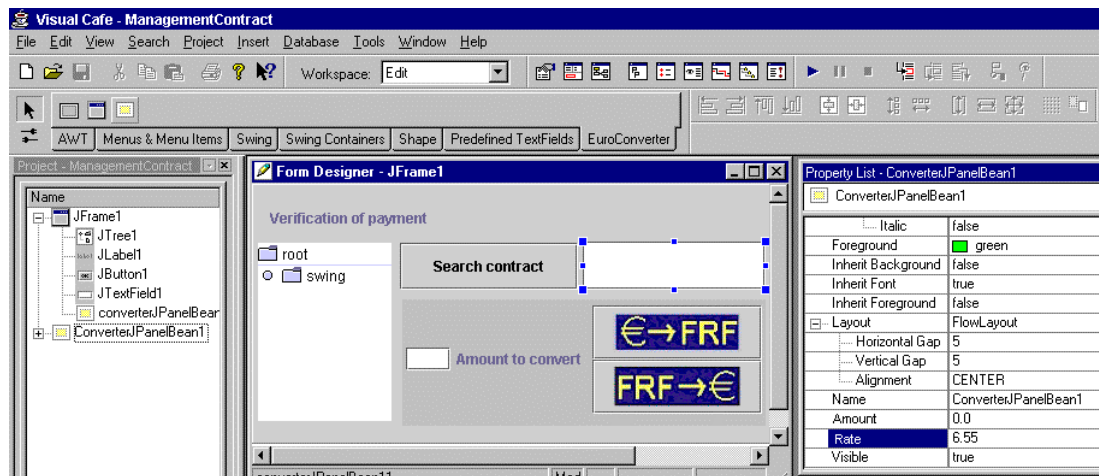
JavaBean format is used in all development tools. VisualCafe provides a wizard for their creation. In this case, we are going to create a panel type Bean, which will contain the converter for the preceding step. The conversion rate will be a property and the Bean will also be capable of generating an event if its rate of exchange changes in order to warn other receiving components of this type of event.

The creation wizard suggests which type of Bean should be created and assists with the creation of its properties. :



JavaBean components creation wizard

Once created, the Bean very easily inserts itself on the palette by drag and drop, and also places itself into another application without difficulty:



Management application for the Euro-Francs currency converter

Our currency converter is now integrated into a management application. The “Rate” property is found in the editor and is now modifiable in the properties window of the graphics component. A screen refreshment is however necessary to display the properties of the Bean (Update Project Beans).

Difficulties appear during the events connection from the new JavaBean to others of the application’s components. The FirePropertyChange event transmitted by the Bean Converter cannot be connected to other elements via the interaction editor. An alternative method would be to code this portion manually.

Our JavaBean now becomes exportable in jar format (Java Archive) through the use of a wizard.

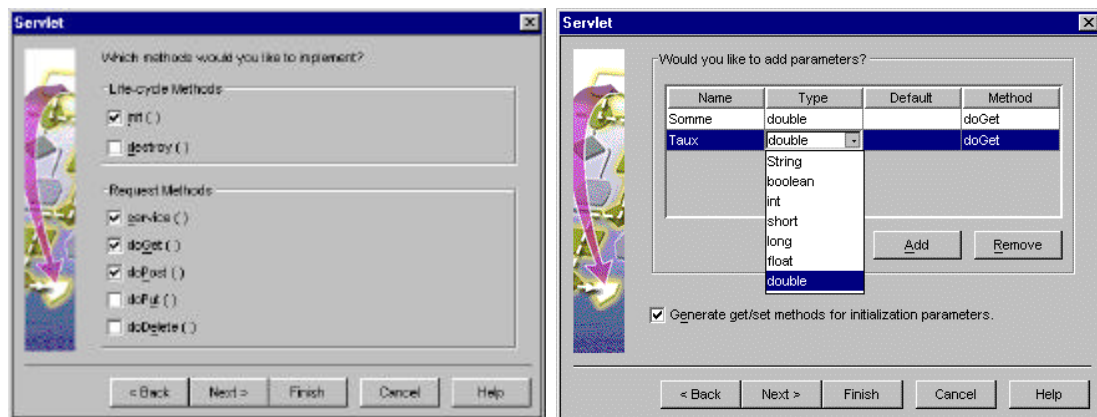
### Step 3: HTML Interface

We shall now re-create our converter and give it an HTML interface. The conversion parameters will be passed to the HTTP request, processed by a servlet and the conversion result will be returned to the client in the form of an HTML page.

The evaluation of VisualCafé of this part will be limited to its generation of the servlet. In fact, VisualCafé does not play a part in the generation and modification of the HTML code, or in the management of user context. VisualCafé is nevertheless provided with an HTML page creation program called VisualPage to fill this gap. This is not sufficiently well integrated with VisualCafé to assist in the implementation of a servlet (cutting/pasting will have to be done of the HTML code generated by VisualPage in order to place it in the VisualCafé editor).

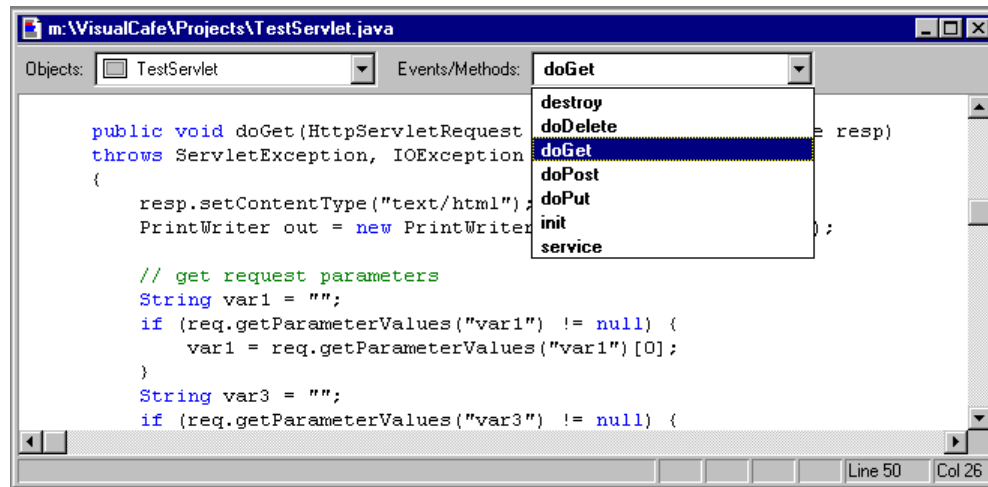
The servlet generation wizard allows choosing the variables of the HTTP request, which are to be intercepted. The test environment is composed of the ServletRunner and the default browser.

The servlet creation wizard simply allows generating the servlet code’s skeleton as well as the code to retrieve the HTML form parameters.



*Servlet creation wizard*

The rest of the development is handled entirely by hand (addition of variables, maintenance). This is also the case for context management. The code generated by the assistant is illustrated below.

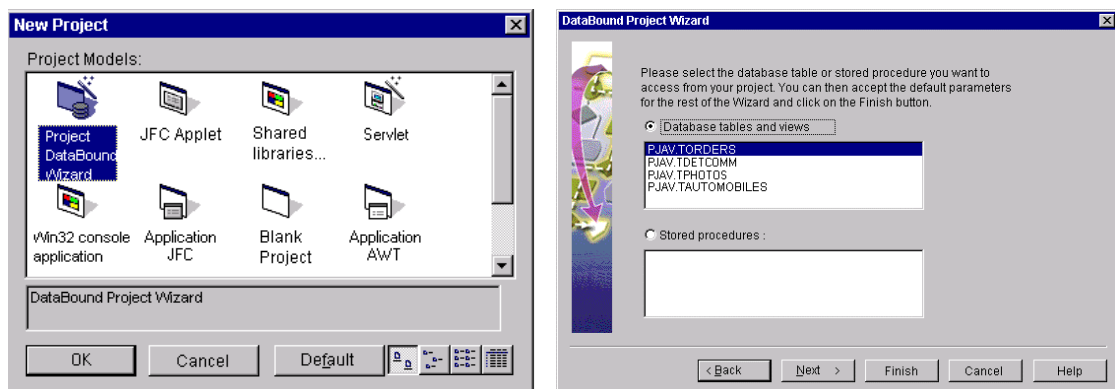


Maintenance of a servlet's source code

#### Step 4: Tabular data display

In this part, we are going to create both a Java access interface with dealer's database and queries of the tables.

The Databound Wizard assists in the creation of a data-oriented project. The wizard offers an interface either from an existing table or from a query originating, for example, from an ODBC data source.



VisualCafe Databound wizard

Once the choice of the data source is made, development of the application can be handled in several ways:

- creation of an interface from a table
- creation of an interface from several tables
- creation of an interface from a stored procedure

These options will be covered successively in this evaluation.

#### ➤ **Creation of an interface from a table**

The creation wizard gives us the choice of how to support the interface (JFrame, JApplet, etc.). There is a rich array of tables to allow the display of tabular data (JTable, Symantec Component, etc.). The layout is also suggested (display style form or table). At the end of the wizard, a personalized interface with the graphics components is generated for the table and is ready for use.

Other very interesting components are integrated:

- the JDBCToolBar assists the user in records navigation, maintenance and creation
- the JDBCStatusBar informs of the status of connections to the database and provides a journal of JDBC events.

The model of data used in VisualCafe uses a Model-View-Controller (MVC) structure. It will satisfy demanding developers with highly personalized tabular data display interfaces. For example, a text field can be dynamically associated with a storage column in a database through the use of a link object (Binding Model) without writing a line of code (binding model type objects appear next to their associated components in the above illustration).

The QBE option (Query By Example) is very interesting. It allows the user to query the table according to criteria without coding SQL queries. The query is handled for one or several tables with well-known and easy-to-manipulate operators (=,<,>,! =,<=,>=) (AND OR \* ? ...) directly from the graphical interface.

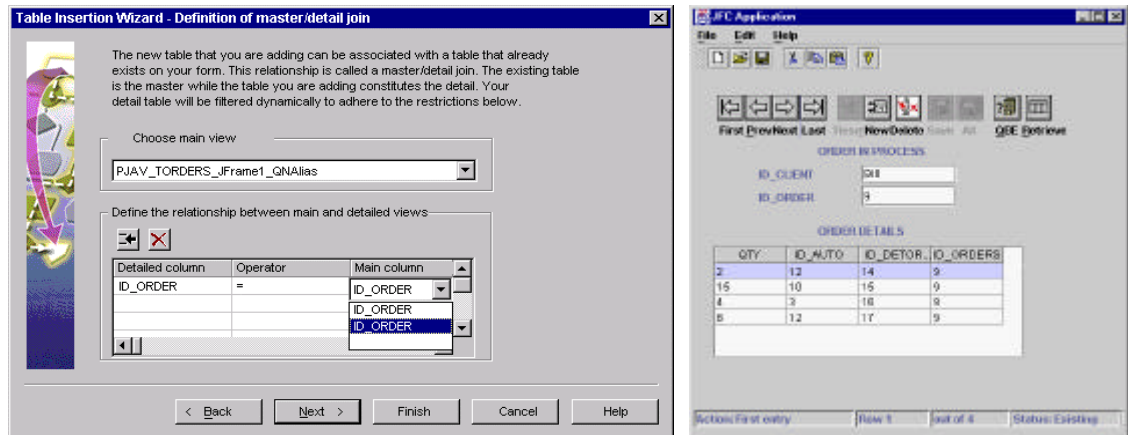
A tool allowing the user to generate an SQL query from a graphical representation of the database tables is unfortunately not available.

#### ➤ **Creating an interface from several tables**

A user interface for managing access to a single table is too limited in some cases. The database tables almost always include relationships. These relationships are represented by primary and foreign keys. In order to integrate several tables within a single interface and to manage joins, VisualCafe uses the concept of Master-Detail relationships.

Let's create an interface from the "orders" table. "Orders Detail" is another table in the database, which contains a column storing the identifier of orders relative to the unique identifier of orders and of columns relative to the detail of an order. The goal is to be able to browse through the orders and to have the related details be displayed dynamically.

In order to do this, we are going to add an “Order details” table to the project. The created table will be considered as a detailed view and the “orders” table will be the master view. For this, the wizard will suggest how to create the master/detail relationship. It then becomes necessary to choose columns related to the primary and foreign keys and to choose an operator for the relationship between the two columns.



*Master/Detail relationship between the ORDER\_ID fields in the two tables*

The above screen shots show the display of details related to each order. The details are displayed in a JTable. It is possible to select the lines and to easily adjust the size of the columns. The use of master/detail definitions is handled easily. This can nevertheless become tedious if many tables are involved in the project. The use of a stored procedure would also be prudent.

The use of the QBE option is also available for projects being run with several tables.

### ➤ Generation of interfaces from stored procedures

VisualCafe offers the possibility of generating a graphical interface based on an existing query. An attempt directed at a command insertion procedure stored in a database generated an interface with fields for data-entry parameters and an execution button.

## Step 5: Print management

As it supports JDK 1.2, VisualCafe includes the print API, but does not provide any assistance of this type. The management of headers, footers, page breaks and other printing features is entirely the responsibility of the developer.



## Ratings:

Interface Generation	Symantec
Step 1: Event-driven interface	☆☆☆
Step 2: Graphic component creation	☆☆☆
Step 3: HTML Interface	● <sup>+</sup>
Step 4: Tabular data display	☆☆☆
Step 5: Print control	● <sup>+</sup>

Scores 0-3 ( ☆☆☆: 3; ☆☆: 2; ☆: 1; ●<sup>+</sup>: 0 )

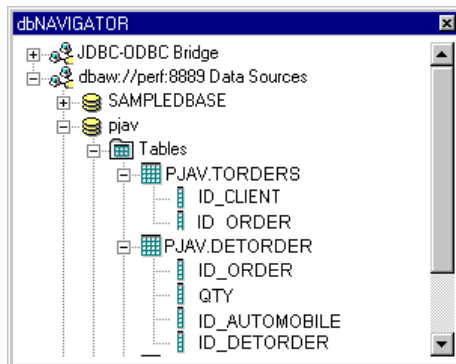
## 6.6.2. Application server

### Step 6: Openness to distributed objects

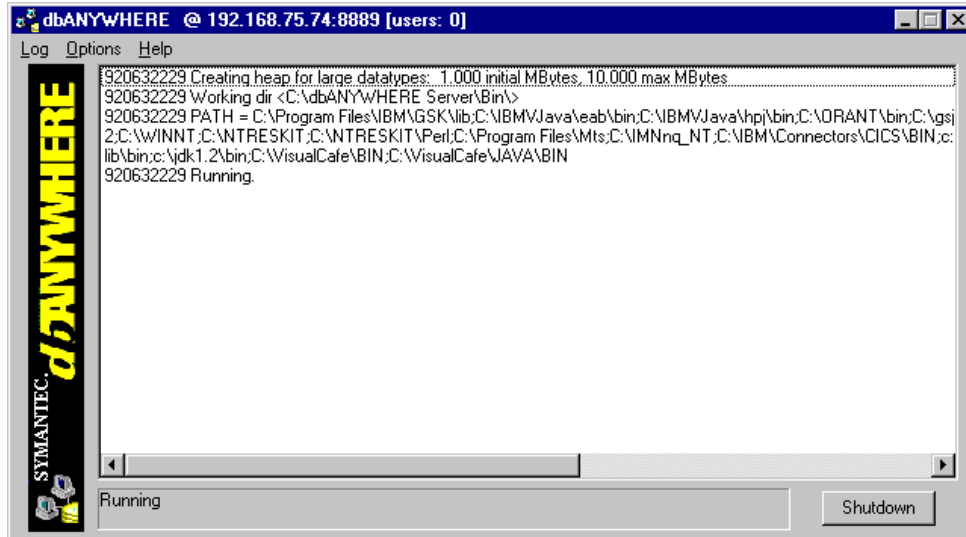
To integrate JDK 1.2, VisualCafe uses middleware necessary for distributed objects. However, no distributed object connection wizard is available in Database Edition. Symantec has announced their support of the next version, IRAD, which is expected in the near future.

### Step 7: Database access

The first operation consists of choosing the data source to which the connection is desired. VisualCafe provides a window called dbNavigator, which offers a hierarchical view of available data sources. It also displays tables, columns and procedures stores in the databases. An intermediate JDBC server for JDBC type-3 drivers is offered with the database edition.



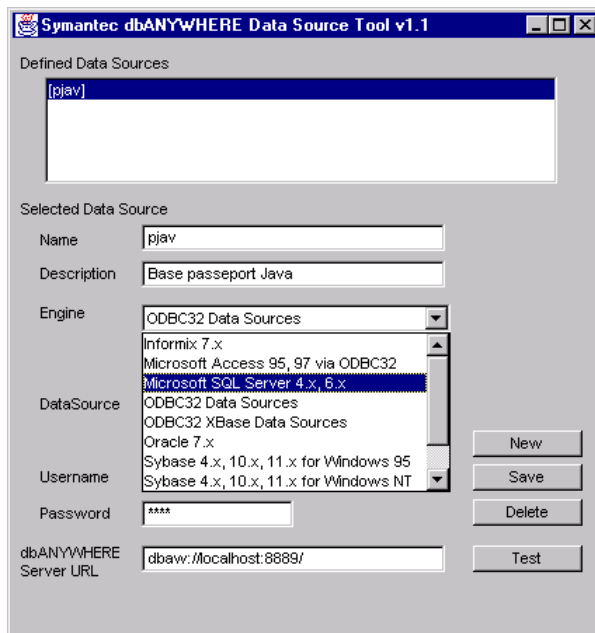
*Hierarchical display of data sources, including ODBC and sources defined in the Symantec JDBC server, dbANYWHERE*



*dbANYWHERE server*

The dbANYWHERE server displays the number of connections in progress and their respective information (JDBC errors, etc.). The “Configure Data Sources” tool allows defining the sources of data together with their JDBC drivers. Numerous JDBC drivers are provided (Oracle, Sybase, Informix, ODBC; etc.). Using the dbANYWHERE server has several advantages:

- Delegation of JDBC driver control,
- Optimization of network traffic owing both to the use of Java middleware protocol between the client workstation and the JDBC server and to cache management.



*Data sources configuration window*

The event-driven application generated to search for one's choice of automobile allows easily locating an automobile through the use of the QBE (Query by Example) option. Navigation in the result of the research is handled in front, at the rear, at the beginning and at the end, all through the use of the navigation bar.

The Databound components were implemented using the JDBC 1API. Symantec is currently working on the integration of the JDBC 2 API for the next version of VisualCafé.

Validation of a customer order is going to allow us to implement transaction support with the tool. The transaction will encapsulate an Insert in the orders table and an Update in the inventory table. VisualCafé does not provide any tool for transactions, meaning the developer must rely upon mechanisms provided by the JDBC 1 API to implement transactions.

On-line help is offered to explain how the "setTransactionIsolation(Value)" method of the jdbcConnection class works. This method allows to specify a level of isolation when faced with three types of inconsistency in the transactions (read jumbled, read non-repeatable and phantom line). The jdbcConnection setAutocommit(false) method allows the user to both deactivate the Autocommit mode by default and to define a transaction consisting of several operations. The Commit/rollback mechanisms are available through commit() and rollback() methods.

### Step 8: API Richness

VisualCafé does not offer wizards for generating images on the server side for an HTML interface. Blobs are not managed. Sending of mail is not supported, since classes for this purpose must be added to do so.

### Step 9: Workload management

Symantec positions the dbANYWHERE server as a " low cost JDBC server facilitating the management of connections to heterogeneous databases with optimization of the generated traffic" (ideal for a reasonable number of users). Regarding fault tolerance, the management of workloads and security, Symantec directs its customers toward third-party solutions sold at higher prices.

### Ratings:

Application server	Symantec
Step 6: Openness to distributed objects	
Step 7: Database access	
Step 8: API Richness	
Step 9: Workload management	

Scores 0-3 (☆☆☆: 3; ☆☆: 2; ☆: 1; ☹: 0)

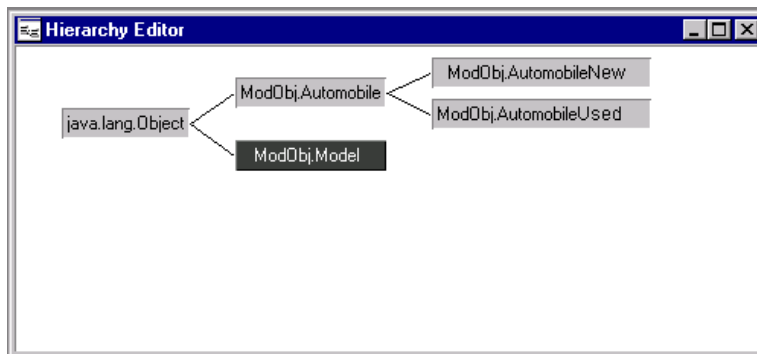
### 6.6.3. Object server

#### Step 10: Modeling by business objects

The modeling tool for business objects offered by VisualCafe uses Java classes. It represents classes in a tree structure based on their inheritance link.

The creation of the Automobile parent class is handled by using a class creation wizard. It allows for the creation of variables and method overloading. It also proposes the associated modifiers (public, private, etc.).

To create the AutomobileUsed class, the user need only drag and drop from the Automobile parent class to start the class creation wizard. The class browser presented in Step 1 ensures inspection of the members of a class.



*Hierarchy editor*

**Insert Class**

Class information

Name:

Source:  ...

Package:  ▾

Extends:  ▾

Type

Class

Interface

Access

Package

Public

Protected

Private

Final

Abstract

Bean

< Back   Next >   Finish   Cancel   Help

*Child class creation wizard*

### Step 11: Importing EJBs

VisualCafe offers neither an EJB development environment, nor an EJB server.

### Step 12: Persistence

VisualCafe does not offer a persistence management solution. The developer can nevertheless use the Java serialization offered by JDK.

### Step 13: Object transaction and distribution

Since it does not have an object server, VisualCafe does not offer any functionality in the area of either object distribution or OTMs. The next version of the product will address Corba, RMI and EJB development.

### Ratings:

Object Server	Symantec
Step 10: Modeling by business objects	☆
Step 11: Importing EJBs	☛
Step 12: Persistence	☛
Step 13: OTM and object distribution	☛

Scores 0-3 ( ☆☆☆: 3; ☆☆: 2; ☆: 1; ☛: 0 )



---

A report by  
TechMetrix Research

Franck Gonzales  
Frédéric Bon  
Jean Christophe Cimetiere  
Nicolas André  
Nicolas Farges  
Philippe Mougins

**In the US:**

TechMetrix Research  
6 New England Executive Park, Suite 400  
Burlington, MA 01803

Tel.: +1 (781) 270-7486  
Fax: +1 (781) 270-7489

<http://www.techmetrix.com>  
[info@techmetrix.com](mailto:info@techmetrix.com)

**In Europe (French headquarters):**

TechMetrix/Groupe SQLI  
Département R&D  
55/57 Rue Saint Roch  
75001 PARIS

Tel.: +33 1 44 55 40 00  
Fax: +33 1 44 55 40 01

<http://www.sqli.fr>  
<http://online.sqli.fr>  
[R&D@sqli.fr](mailto:R&D@sqli.fr)

**P u b l i s h e d : S e p t e m b e r 1 9 9 9**

**WARNING:** Intellectual Property Act.

Art. L 335 - 2: All publication of written works, musical compositions, paintings or any other literary output, printed or engraved, in full or in part, in defiance of the laws and regulations relative to the property of authors, constitutes forgery, and all forgery is a misdemeanor.

Forgery in France of works published in France or abroad is punishable by two years imprisonment and a fine of 1,000,000 F (L. no. 94-102, 5 Feb. 1994, art. 1st)

Art. L 335 - 8: Legal entities may be held legally responsible under the provisions of article 121 - 2 of the Penal Code for infractions defined under articles L 335-2 to L 335-4 of this act.