



Product Review
WebLogic 4.5.1
(with Visual Café Enterprise Suite)

TABLE OF CONTENTS

1. PRODUCT PROFILE.....	3
2. PERFORMANCE MEASUREMENTS.....	8
2.1. INTRODUCTION	8
2.2. RESULTS.....	12
3. FUNCTIONAL EVALUATION	19
3.1. PRESENTATION.....	19
3.2. QUALITY AND RICHNESS OF THE DEVELOPMENT ENVIRONMENT	20
3.3. HTML INTERFACE GENERATION.....	25
3.4. JAVA GRAPHICAL INTERFACE GENERATION.....	28
3.5. IDE PRODUCTIVITY FOR SERVER PROCESSING	31
3.6. APPLICATION SERVER	33
4. ANNEXES.....	45
4.1. WORKLOAD METHODOLOGY.....	45
4.2. FUNCTIONAL EVALUATION CRITERIA	49

1. Product profile

Identification	
Product composition	BEA WebLogic Server and Visual Café Enterprise Suite
Release	4.5.1(WebLogic), 3.1 (Visual Café)
DBMSs supported	Informix and SQL Server (type 4 JDBC), Oracle (type 2 JDBC) and all other ODBC bases. Moreover, any other database with a JDBC driver can be, in principle, used with WebLogic Server.
Development platforms	Windows NT
Deployment platforms	Windows NT 4, Sparc Solaris, HP 9000 HP-UX 11, Red Hat Linux for Intel, RS 6000 AIX 4.3, Alpha Tru64 Unix
Editor	BEA (WebLogic) and Symantec (Visual Café)



BEA is a young company founded in 1995 by former employees of Sun and Pyramid Technology. BEA took off when it integrated Novell's Tuxedo technology and teams in 1996. BEA then continued its course, acquiring along the way Digital ObjectBroker's ORB technology, as well as almost 80 employees, and NCR's Top End Middleware. Finally, in 1998, the company purchased WebLogic, editor of the Tengah Web application server. Today, BEA employs over 1,200 people and has become a leader in the server-side building block (TP monitors, ORB, MOM, etc...) market.

The company hopes to be just as successful in making a name for itself in the application server and EAI (Enterprise Application Integration) markets. Acquiring technology, especially in such a dynamic sector, is a rather tricky task. There are many pitfalls: understanding the new technology and integrating it with existing technology, holding on to head engineers, bridging cultural and technical gaps between old teams and the new arrivals, and so on. With so many stumbling blocks in the path, few companies succeed—but BEA has.

For the moment, WebLogic is currently on the same road to success as its illustrious predecessor Tuxedo. In the past few months, this Java-based application server has established itself as one of the major players in this turbulent market. The recent acquisition of The Theory Center has endowed BEA with a whole range of e-commerce-oriented server components and a component-generating tool, both of which have already been integrated with WebLogic Server.

WebLogic Server is BEA's second venture into the application server market. Last year the editor presented M3, which was later renamed WebLogic Enterprise. This solution, which is based on Corba and Tuxedo technologies, was positioned as an upmarket server with some polished workload capacities. At the time, the WebLogic Server was presented as a middle-market product and an integration of the two technologies was promised with allowing ascending compatibility.

Today, however, there has been a change in strategy and positioning. WebLogic Server has now become BEA's key product while WebLogic Enterprise, while not completely abandoned, is now destined to only a relatively small market consisting of those using Corba in business-oriented computing. This was a wise move: while M3/WebLogic Enterprise was struggling to find its market, bright prospects were popping up before WebLogic Server as companies were attracted by its capacities to support Web standards. Add to this is the fact that while Corba, the technology on which WebLogic Enterprise is based, is heavily used in some industrial sectors, it is used much less frequently in business-oriented computing.

It is worth noting that these solutions can now be integrated in the same product. For example, WebLogic Server could be used as the application server (servlets, etc.) while using WebLogic Enterprise as the

object-oriented server. Nonetheless, be careful of the complexity such architecture entails and the relatively uncertain future of WebLogic Enterprise. WebLogic Server is a Java-based application server. It enables implementation of HTTP Web applications just as well as three-tier client-server applications with Java clients. One of the product's notable characteristics is that it revolves entirely around Java 2 Enterprise Edition (J2EE).

This specification, which is promoted by Sun, is recent but very promising. It is an extension of the standard Java platform and aims to meet the specific needs of enterprise information systems. In particular, the specification includes a dozen object frameworks such as servlets for generating dynamic HTML pages; Java Database Connectivity (JDBC) for low-level database access; Java Messaging Service (JMS) for asynchronous message exchange; and Enterprise Java Beans as business component model.

WebLogic is firmly positioned as an ardent supporter of J2EE and integrates most of the elements of this ever-evolving specification (in particular, support for the frameworks mentioned here). This support is one of the product's major assets since J2EE is backed by a growing number of editors, despite Sun's financial demands. A powerful product in many respects, WebLogic incorporates BEA's middleware know-how. We think that companies looking to choose a J2EE platform must consider WebLogic as one of the most serious contenders.

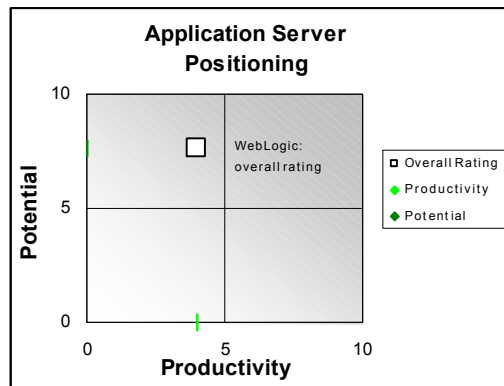
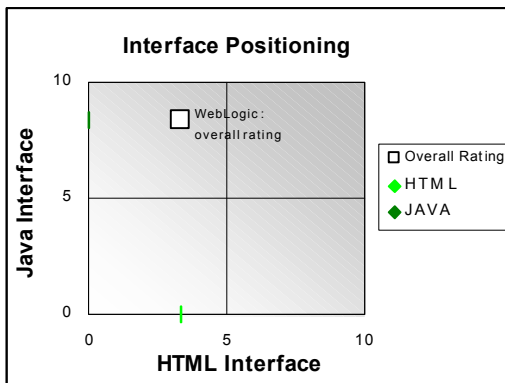
However, WebLogic proves less suitable in two particular situations: projects with strong time-to-market constraints, and projects taking a business object approach to development. In the first case, the lack of development tools and low level of integration with other workspaces, as well as the low-level and sometimes extreme immaturity of some J2EE elements, do not allow WebLogic Server to offer strong development productivity. Today, projects requiring quick deployment remain the privilege of RAD application server pioneers. With regard to the second point, projects based on a business object approach; WebLogic suffers because of its choice of EJB technology, which is still immature and unsuitable for large-scale object modeling. In this field, major players offering a tried-and-true solution are few and far between. Nonetheless, BEA does offer a solution that was not evaluated in this report, WebLogic Commerce Server, which offers components designed to facilitate implementation of e-commerce applications. This solution could offset the problems linked to J2EE's immaturity as it includes elements that act as higher level frameworks.

WebLogic concentrates on the server part and does not include a development tool. Nevertheless, it provides various JavaBeans that facilitate integration with third-party tools. In this study, we will evaluate the couple formed by WebLogic Server 4.5.1 and Visual Café Enterprise Suite 3.1. This choice is completely arbitrary but was strongly influenced by the close ties between Symantec and BEA. Note that other Java IDEs such as VisualAge for Java could serve as development tools for WebLogic. The latter is a Java development environment that includes a source code editor, a powerful debugger, a Java graphical interface builder and a tool enabling creation of HTML pages.

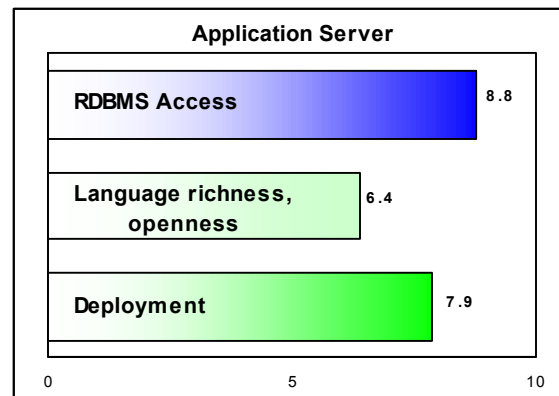
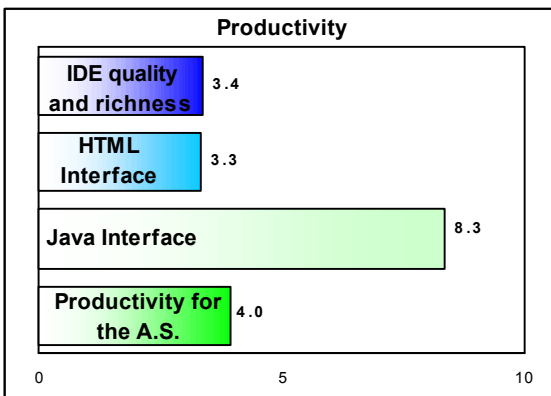
BEA has just purchased Visual Café, formerly edited by Symantec, via a joint venture. Visual Café is one of the leaders in its market. It offers a specific opening for developing components destined to be deployed with WebLogic. All the ingredients that make up a Java IDE worthy of this name are present. However, you must integrate an external version controller to manage group work. In addition to Swing components, Visual Café offers rich palettes of specific components (multimedia, database components...) to satisfy demanding developers. Although not as accomplished as that of its competitor VisualAge, WebLogic's visual programming tool offers some good possibilities. Another interesting characteristic: the environment is based on Open API and the IDE can be broadened for specific developments. Furthermore, Java source code can be compiled in native Win32 in order to obtain better performances.

However, Visual Café is not well suited for generation of dynamic HTML pages from servlets. The product does contain an HTML development tool, VisualPage, but this tool enables creation of static pages only. WebLogic Server and Visual Café remain two very different products from different editors. No miracle here, integration is minimal and does not in any way offer developers the same productivity found in

Functional evaluation



Visual Café has a rich WYSIWYG graphical interface editor, which allows you to configure the various controls and establish connections with processing objects. However, when it comes to helping with implementation of HTML Web interfaces, WebLogic/Visual Café does not enjoy such a good position. Indeed, it offers only a static page editor, which lacks a lot of functionality compared with offerings of the leaders in this domain. On the server side, WebLogic is based on a servlet engine and Java Server Page (JSP) technology, which offers another approach to servlet generation.



Regarding application server capacities, WebLogic achieves a good overall rating. Implementation of J2EE frameworks, an object development language and database access capacities make this product one of the most complete solutions in this field. Furthermore, even if the load balancing and failover capacities still have room for improvement, they are some of the best in the application server market. The rating would have been even higher had connectivity to enterprise systems been included in the offer or had higher-level services been furnished.

On the other hand, in the area of business object servers it's quite a disappointment. Certainly BEA's product offers optimized object distribution capacities, but the object capacities offered by WebLogic's EJB model prove too immature and limited to implement business object modeling. Therefore, WebLogic must be considered principally when taking a relational approach to modeling of information system entities.

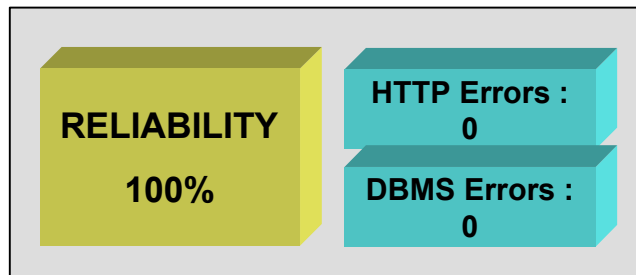
Furthermore, WebLogic obtains a mediocre rating for productivity. Integration between Visual Café and WebLogic is not tight enough and does not in any way permit BEA to compete with totally integrated solutions on this point. Beyond these integration limits, WebLogic lacks many tools required to offer a high level of productivity. Finally, even if J2EE frameworks offer powerful functionality, they remain at a rather low programming level and you need very good Java skills to implement them.

Performances

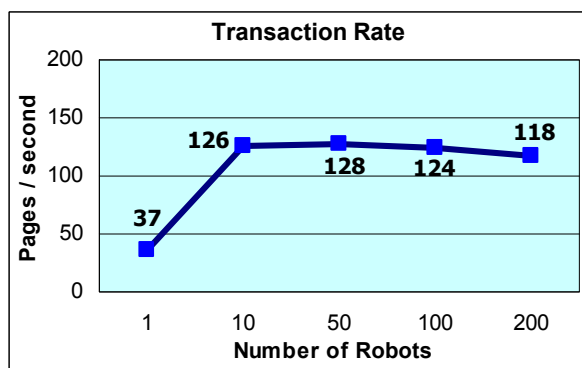
118

This is the number of dynamic HTML pages delivered per second by WebLogic application server during execution of a "mixed" scenario (stringing together of 8 modules or 24 pages) with concurrent use of the application by 200 robots.

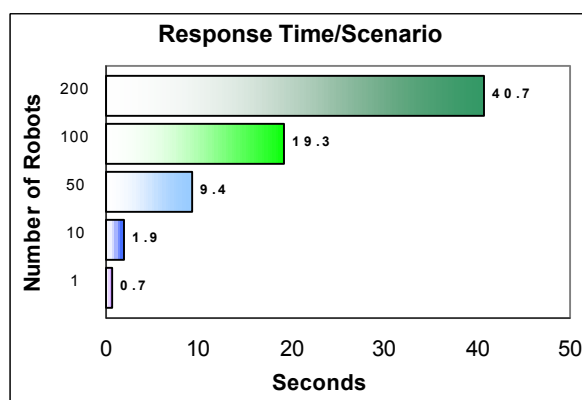
The WebLogic application, which was in line with our TMBench specification, proved to be an entirely reliable application server when heavy workload is placed on it. This was confirmed throughout the tests, even when it was delicately configured so as to optimize the measurements. The solution is mature and proves reliable at all levels.



We can see how robust the WebLogic application server is when looking at the analysis of server behavior during successive workload increases. The transaction rate remains steady with anywhere from ten right through to 200 concurrent robots.



Although the response times are rather average in relation to rival solutions, this stability reassures us of the product's potential and enables us to envisage quite serenely much more considerable workloads.



Response time debasement is linear in relation to the number of robots, which proves optimal use of server resources no matter the load.

Technical profile

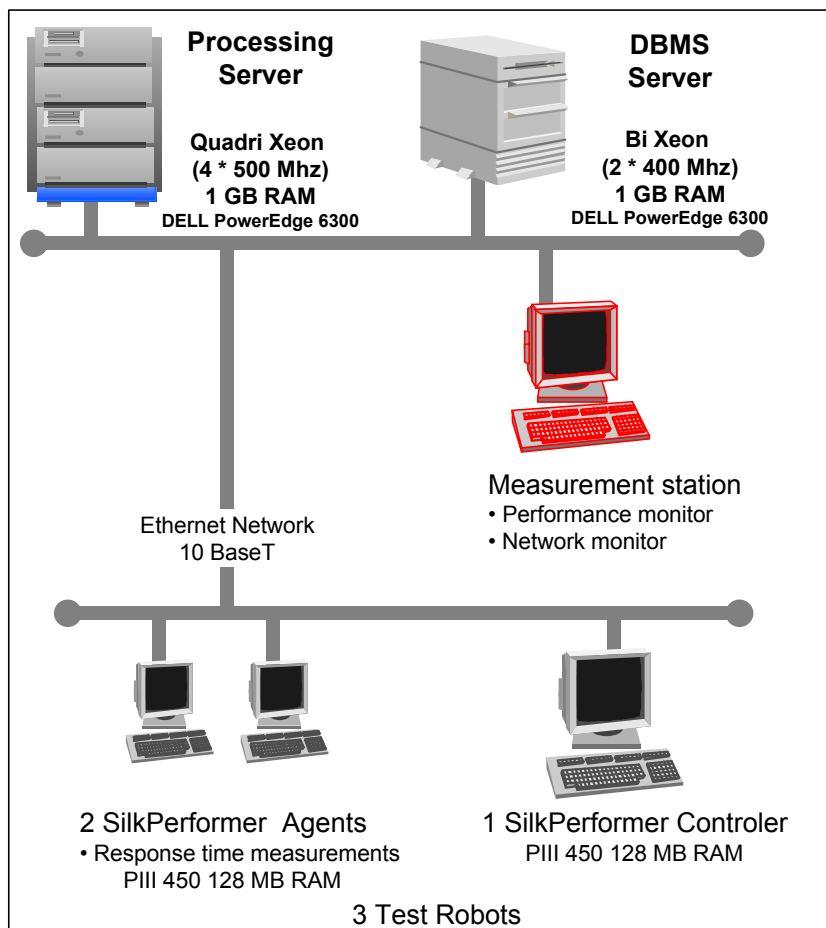
Development license	Price per developer for WebLogic Server varies between countries.
Deployment license	WebLogic Server: price based upon the total number of CPUs. WebLogic Server Cluster Edition is designed for implementation of automatic load balancing and failover solutions: price based on the total number of CPUs.
Configuration used for evaluation	WebLogic 4.5.1 for Windows NT 4 with database driver module; Visual Café Enterprise Suite
Complementary modules	Enterprise version database drivers

2. Performance measurements

2.1. Introduction

2.1.1. Carrying out of measurements

The performance measurements are carried out on an application developed by the editor. This application is installed on the TechMetrix platform and subjected to sustained workload increases in order to assess and analyze the application server's behavior in extreme conditions. All measurements are carried out by TechMetrix consultants. After that, optimizations made on the premises by the editor are evaluated. The complete specifications of the application tested can be downloaded from the TechMetrix Web site.



Platform used for performance measurements

➤ **Two types of measurements:**

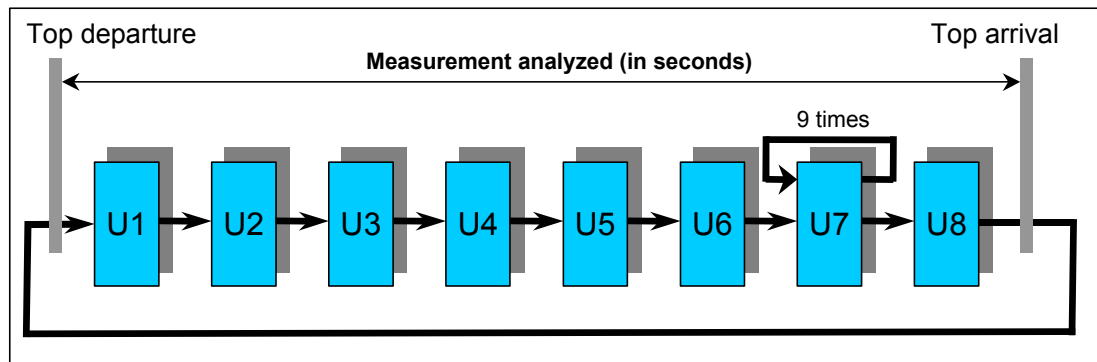
The application is broken up into eight distinct modules (or units), each of which represents one of the main needs of transactional intranet applications. Each model is characterized by its focus on a specific intranet need (simple search, update, calculations...) while remaining as simple as possible in order to put forward, in a precise manner, each of the elements to be analyzed.

A robot, which simulates an application user, carries out the modules in a predefined order several times. The term “scenario” refers to the unitary stringing together of modules; repeated many times in order to obtain an overall average time that is representative of the length of time required to carry out a scenario. During workload increases, each robot carries out the same scenario to simulate a sustained load equivalent to several concurrent users.

The “mixed” scenario:

In this context, the scenario consists in stringing together each module of the application. Each robot carries out the first module, followed by the second module and so on and so forth until the eighth module. Once module 8 is completed, the total time needed to carry out all of the modules is noted. This is the time required to complete this scenario. We should note that step 7, which only consists in storing information in memory, is carried out nine times.

Once the total scenario time has been recorded, the robot carries the scenario out again and again for several minutes. The average time is then recorded. This represents the single-user measurement.



Description of “mixed” scenario with eight modules

For workload increases, the same operation is carried out successively with 10, 50, 100 and 200 robots that concurrently launch the same scenario (sequence of eight modules). The average measurements are then recorded and correspond to the average response time needed per robot to complete a scenario.

The “independent” scenarios:

The “mixed” measurements can have some fringe effects on measurement analysis by saturating part of the application server (memory, HTTP server response, CPU consumption...) and thus resulting in artificially long response times for some modules. To underline the product’s real potential regarding certain criteria, workload was only increased in modules 4, 5 and 7. In this context, each robot will repeatedly complete only module 4 several times. Then once the times have been recorded, all of the robots are started again and carry out the next module, 5, and then only module 7.

For each module, load increases are carried out with 10 and 50 concurrent robots. At one instant “t”, all of the robots carry out the same module, in other words, first 4, then 5 and lastly 7.

2.1.2. Analysis of the results obtained

All of the measurements, times, behaviors, etc. are deduced from the physical configuration of the TechMetrix platform and according to the application tested. It is for this reason, that even if this context tries to be as representative of real world business needs as possible, all of the results must nonetheless be situated in our specific context (See the sections Platform and Annexes).

➤ Reliability (HTTP / SQL)

This is by far the most important point, since it indicates whether the application deployed was able to support sustained workload increase without showing sign of weakness. To make this assessment, we analyze the number of HTTP requests that are simulated by the robots carrying out a scenario. We then compare this number of HTTP requests with the number of those actually sent and check the database to see if all the database transactions were indeed recorded.

➤ Measurements (time / transaction rate and throughput)

The measurements obtained are presented in three ways:

- The response time: The average times obtained for a robot to complete a scenario. Detailed measurements for each module are also given for information purposes. These times are those of a robot, in other words, the average time a user will have to wait to complete a scenario in its entirety.
- The transaction rate: Knowing the number of HTML pages that are produced to run a scenario, the number of pages per second is calculated using the average time needed to carry out a scenario. Here, the number presented is the number of pages delivered or the number of scenarios completed by the application server in one second. This measurement gives a good idea of the number of dynamic pages delivered by the application server and of the number of pages that a user receives per second.
- The transaction throughput: Throughput provides information on the processing time of one dynamic HTML page. The “user” axis indicates the latency period for the display of an HTML page while the “server” axis represents the amount of time that the server needs to deliver a dynamic HTML page. As soon as there are several robots, the results on these two axes differ.

➤ **Configuration (database connections, number of processes, number of threads per process)**

Here we provide information on the program architecture that is implemented by the editor to get the most out of his application server. These are interesting points that can be used to judge the application server's potential on this level and to better analyze the results obtained. In addition, they make it possible to assess the potential behavior of a tool when faced with much larger workload increases and identify potential bottlenecks in different physical configurations.

➤ **Consumption (memory, CPU)**

The CPU and memory consumption of the two servers used is presented here. One physical machine is used as both an HTTP server and an application server, while the other acts as the database. CPU and memory consumption make deducing the physical configuration required for the application server tested quite simple, in relation to the number of expected concurrent users.

2.2. Results

2.2.1. Synthesis of the results

The application run by the WebLogic motor, which entirely conformed to all of the TMBench specifications, was developed in the form of servlets. Deployment on TechMetrix Research's platform did not pose any particular problem: the application was installed and functioned very quickly.

The first measurements taken yielded acceptable results, without having to delicately reconfigure the application server. In fact, during these first tests the application remained steady—even when significant workload was placed on it. Neither increasing/decreasing the number of connections to the Oracle database in the pool, nor toying with the number of threads destabilized the application. This is definitely an argument in favor of the product, which did not require very fine-tuning in order to guarantee acceptable results and complete reliability.

One of the product's strongest characteristics concerns the a priori deployment configuration. Indeed, all of the workload tests were carried out following the same deployment scheme, no matter the number of robots accessing the application. This is an extremely interesting point because in numerous intranet and Internet contexts, the number of users cannot be estimated. The measurements taken with WebLogic in this study can thus be applied to restricted or large-scale applications.

With regard to connection pooling, TechMetrix Research's teams certainly noticed the finesse of WebLogic's administration management. Indeed, a 17-connection pool to the database was created when the application was started, as an initial test had pinpointed this need with maximum workload. At this level, it is easy to set the number of connections when the application is started, as well as the maximum number of connections that cannot be surpassed, no matter the application server's needs. The administrator is thus « master » of all deployment at this level. This proves indispensable when the same physical server hosts several transactional applications at the same time.

➤ **Reliability**

Average	1 robot	10 robots	50 robots	100 robots	200 robots
HTTP reliability	100%	100%	100%	100%	100%
Database transactions	100%	100%	100%	100%	100%

We did not encounter any problems at any point during the tests. No matter the configuration, the WebLogic application server proves robust when put to the test.

On the user side, no « Web » errors thwarted the measurements. We checked the Oracle database after workload sessions; it did not show even the slightest sign of weakness: all of the database transactions were carried out successfully.

➤ **Analysis of user-side results**

Average	1 robot	10 robots	50 robots	100 robots	200 robots
Response time per 24-page scenario (in seconds)	0.7	1.9	9.4	19.3	40.7
Response time per page (in seconds)	0.03	0.08	0.39	0.80	1.70

Up until 10 concurrent robots, the response time needed to display a page on the user side was practically instantaneous. Indeed, in this situation the final user waits less than one tenth of a second between the moment a page is requested and the moment this page is received.

With 200 robots (which is of course equivalent to a much larger number of concurrent users in a real-world production context), the average waiting period remains less than two seconds. Although far from negligible, this time is more than acceptable in a professional context.

➤ **Analysis of server-side results**

Average	1 robot	10 robots	50 robots	100 robots	200 robots
Transaction rate (pages delivered per second)	37	126	128	124	118
Transaction throughput (time in seconds between delivery of two pages)	0.027	0.008	0.008	0.008	0.008

WebLogic is quite impressive when it comes to performance consistency during successive workload increases. From 10 to 200 robots, there is practically no drop in the number of pages delivered by the HTTP server and the decrease perceived on the user side is directly proportional to the number of robots. This means that, for all of the measurements taken, it was impossible to notice even the slightest fringe effect and that the response time obtained is in direct proportion to the number of robots.

This point is encouraging with respect to more intensive workloads since WebLogic is obviously capable of managing parallelism perfectly.

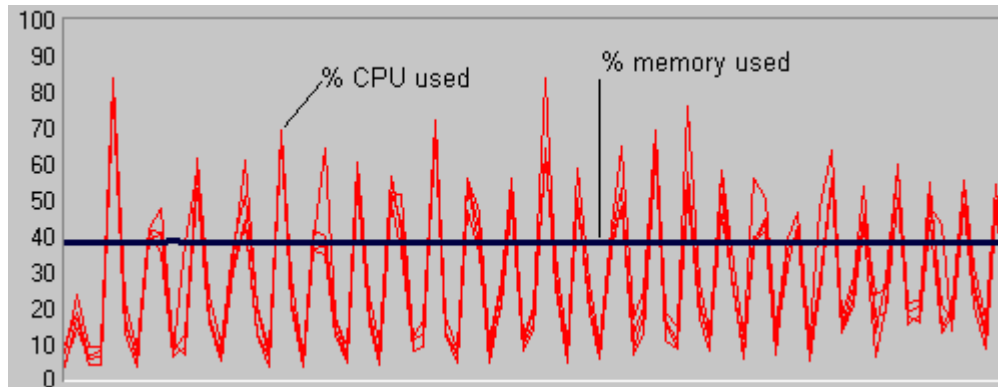
2.2.2. General characteristics

Configuration	1 robot	10 robots	50 robots	100 robots	200 robots
Number of open connections to Oracle database	17	17	17	17	17
Number of launched instances (processes)	1	1	1	1	1
Number of threads per instance	20	20	20	20	20

The first test with 200 concurrent robots was carried out with an initial pool of 10 database connections, and the maximum number of open connections was set at 30. During this test, WebLogic opened seven additional connections, which meant that 17 connections were opened initially for all of the tests. WebLogic's connection pool management finesse makes it possible to proportion the database correctly and have total control over the number of open database connections. We should note that some third-party tools require one physical database connection per user session.

The number of threads was globally defined in relation to the different measurements taken when changing this value. We should note that the application always functioned perfectly, no matter the configuration used. Moreover, the only differences we noticed were variations in response time.

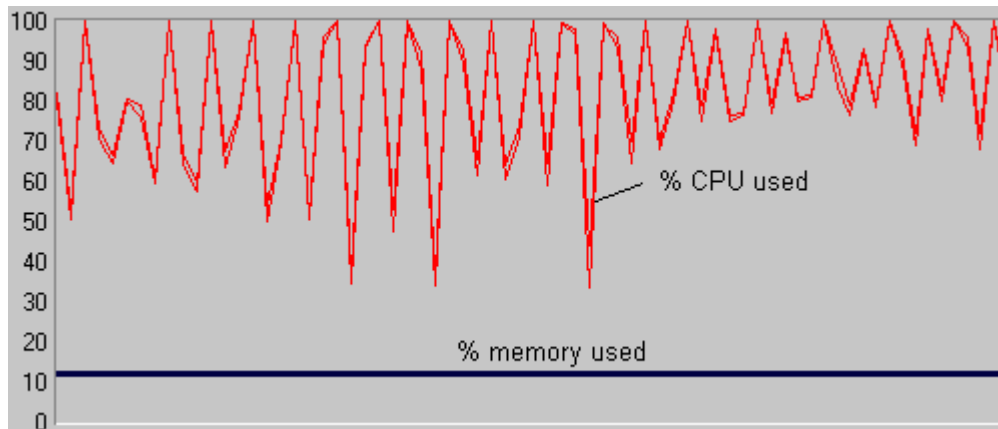
➤ **CPU consumption and memory used with 200 robots**



*CPU consumption (4*500 MHz) and memory used (1Gb of RAM) with 200 robots on the processing server (in %).*

The quadri-processor housing the WebLogic application server and the database was never saturated. Load balancing across the four processors was well managed. However, the load generated by 200 robots yielded both highs and lows, resulting in rather irregular CPU use. Using roughly 35% of the CPU for the entirety of the test, WebLogic leaves some room for other applications that can thus cohabit on the same platform without any difficulty.

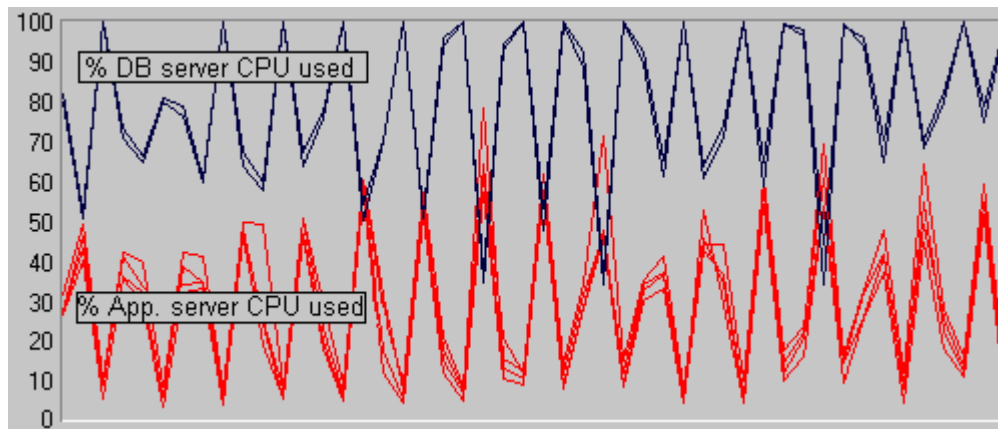
WebLogic's robustness shines through again when we look at the amount of memory used. WebLogic behaved very well in this regard, using roughly 400 Mb of memory throughout the entire length of the test, and presented undeniable guarantees of consistency. Obviously, WebLogic is not the type of application server to present flaws when it comes to memory resource management.



*CPU consumption (2*400 MHz) and memory used (1 Gb of RAM) on the database server with 200 robots (in %).*

The server, a bi-processor housing the Oracle database, is much more saturated than the processing server. However, we notice the same behavior characterized by consumption peaks, followed by troughs. On the contrary, memory use remains stable as WebLogic initiates physically all of the connections when the application is launched.

➤ **CPU use on the two servers with 200 robots**



*CPU consumption (4*500 MHz) on the processing server, and CPU consumption (2*400 MHz) on the database server, with 200 robots (in %).*

The above diagram highlights the behavior of the two servers with a maximum workload placed upon them in a 200 concurrent robot configuration. We notice that when one server is being used at 100%, the other is in a trough « waiting. »

These servers take turns as the bottleneck. When the DBMS' CPU is used entirely, there is a bottleneck at this level. The application server is thus on standby, and uses the CPU less. On the contrary, when the database responds correctly, nothing hinders the smooth running of the applications server, which can run at full throttle.

2.2.3. Table of results

➤ **Average time, in seconds, for each module of a “mixed” scenario**

The eight modules of the scenario allow us to center the analyses on basic features. The role of each module is described below:

- Module 1: large “select” (more than 5000 records returned without cache in a database)
- Module 2: sum of the small “select” without cache in a database
- Module 3: sum of the small “select” with cache in a database
- Module 4: multi-criterion search with dynamic request, without cache
- Module 5: database updates (simple insertion and updates)
- Module 6: algorithm
- Module 7: storage of user context in memory
- Module 8: mass insertion

Number of robots	Modules							
	1	2	3	4	5	6	7	8
1 robot	0.05	0.10	0.00	0.19	0.02	0.08	0.07	0.09
10 robots	0.14	0.28	0.01	0.43	0.12	0.10	0.09	0.70
50 robots	0.70	0.91	0.57	1.47	0.76	0.65	2.90	1.39
100 robots	1.48	1.88	1.57	3.08	1.63	1.42	6.07	2.10
200 robots	3.52	4.67	4.72	8.43	4.17	2.82	9.08	3.21

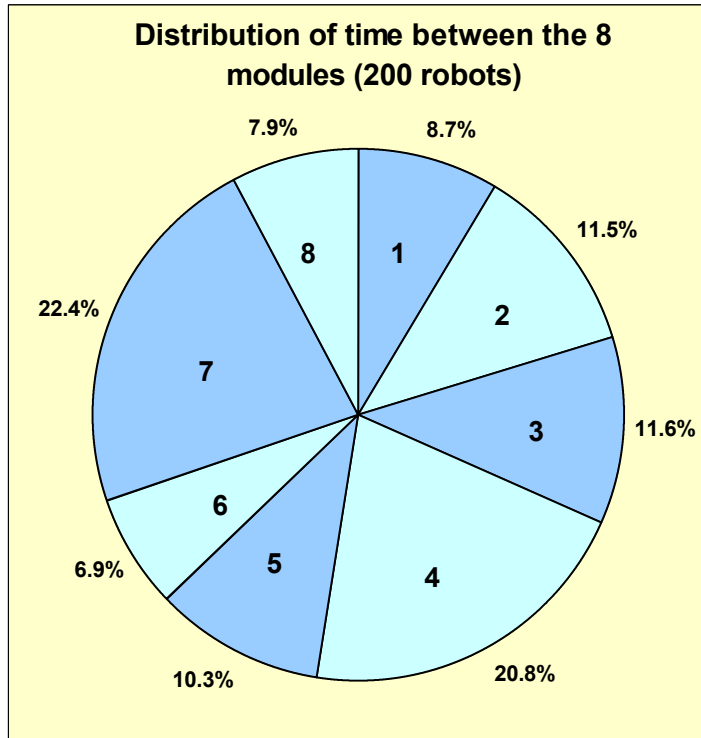
The above results that we obtained with WebLogic remain logical. The response time increases with the number of robots, and this is the case for each step.

➤ **Average time, in seconds, for modules 4, 5 and 7 (independent scenarios)**

Number of robots	Modules		
	4	5	7
1 robot	0.19	0.02	0.07
10 robots	0.85	0.06	0.23
50 robots	4.59	0.37	1.99

Even for the independent scenarios, the results obtained remain consistent. Here, we can also point out that the debasement in response times remains in proportion to the number of robots from 10 robots on. When there are fewer than 10 robots, the bottlenecks are not yet reached, which explains the non-proportional times in relation to the number of robots.

➤ **Distribution of time per module**



Distribution of response time between the 8 modules of a scenario with 200 robots (in %)

Without a doubt, module 7 takes up the largest amount of the total time, but with nine times the number of pages than the other modules. Thus, in proportion to the number of pages, it is the module that uses up the least amount of time. This is entirely logical because this module is the only one that does not carry out a large calculation or database access.

In fact, module 4, which entails a multi-criterion search in the Oracle database, requires the most time. For the most part, the other steps yield rather homogeneous results.

3. Functional evaluation

In this section, we will take a look at our evaluation of the WebLogic Server 4.5.1/Visual Café Enterprise Suite 3.1 couple in different areas of functionality. Firstly we will discuss the quality and richness of the development environment, followed by openness to third-party tools and application server potential.

3.1. Presentation

➤ Description of the solution

The WebLogic Server solution is made up of the following modules:

- WebLogic Server, which is the runtime environment.
- WebLogic Console, which is the server administration console.
- EJB Deployment Wizard, which is a graphical tool that enables configuration of the deployment characteristics of each EJB.
- ZAC Publisher, which is a graphical tool that makes it possible to create, publish and manage ZAC (Zero Administration Client) packages. These packages provide opportunities to deploy and update files automatically on several machines.
- Cloudscape v1.5 is an object/relational database engine. BEA provides an evaluation version of Cloudscape, which has restricted storage capacities.
- Java Run-time Environment (JRE), which enables execution of Java applications.

Visual Café Enterprise Suite 3.1 is made up of:

- Visual Café IDE, which allows you to create and perfect code and the Java graphical interface.
- Visual Page, which is a WYSIWYG editor for creating static HTML pages.
- Oracle Lite 3.5, a database engine.
- IBM On Demand Server, a tool that simplifies administration of Java Web applications.
- Netscape Communicator 4.5.
- dbANYWHERE Server, a collection of database drivers.

3.2. Quality and richness of the development environment

3.2.1. Installation

➤ Prerequisites

The Visual Café development environment can be installed on Windows 95/98 or Windows NT 4.0. However, WebLogic Server requires NT or Unix since it does not support Windows 95/98. The minimum recommended RAM configuration is 128 MB for Visual Café plus 128 MB for WebLogic. You should also plan on 540 available MB on the hard drive for Visual Café and an additional hundred MB or so for WebLogic. You must be connected to the Internet should you wish to access WebLogic documentation. It is not necessary to have an HTTP server, as one is included in the solution. Moreover, it is possible to begin development using one of the database engines shipped.

➤ Procedure

An InstallShield package makes it easier to install WebLogic Server on the NT platform as it installs the different files automatically and configures several different parameters. However, with Unix WebLogic must be installed manually.

After this first phase you must install a JDK. Should you choose to use an HTTP server other than WebLogic, you must configure it manually and then install the database drivers.

Once the above steps have been completed, you can then move on to installation of Visual Café Enterprise Suite. The Java IDE and the HTML tool are installed automatically, but separately.

➤ Documentation and tutorials

The good quality documentation suggests a certain level of maturity. There is no paper documentation, so you must connect to BEA's Web site should you wish to consult or download the online documentation. Online you will also find a relatively complete search engine. The documentation covers a wide range of subjects: a collection of high-level, conceptual documents enables a good approach to and understanding of the product. A second series of documents outlines implementation of different features.

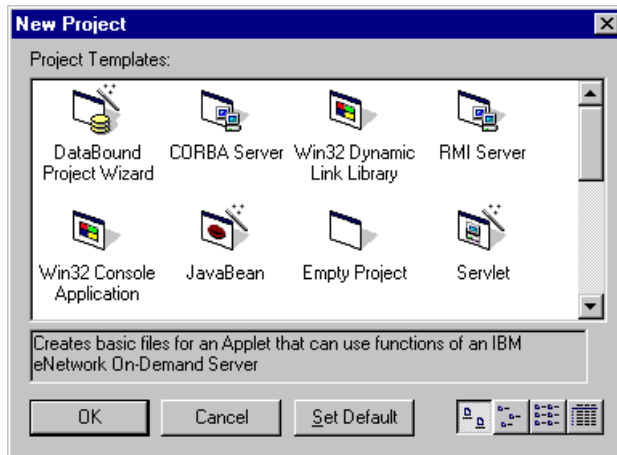
Visual Café's documentation can be installed in PDF format; a specific guide concerning integration with WebLogic's EJBs is included.

There isn't a WebLogic tutorial, but examples of the various server functionalities are given. As for Visual Café, both a tutorial and examples are available.

3.2.2. Development environment

Visual Café Enterprise Suite 3.1 is the development environment evaluated in this study. While it is entirely possible to use other environments to develop a WebLogic application, Visual Café is the tool offering the tightest integration. Moreover, BEA has just bought Visual Café from Symantec via a joint venture.

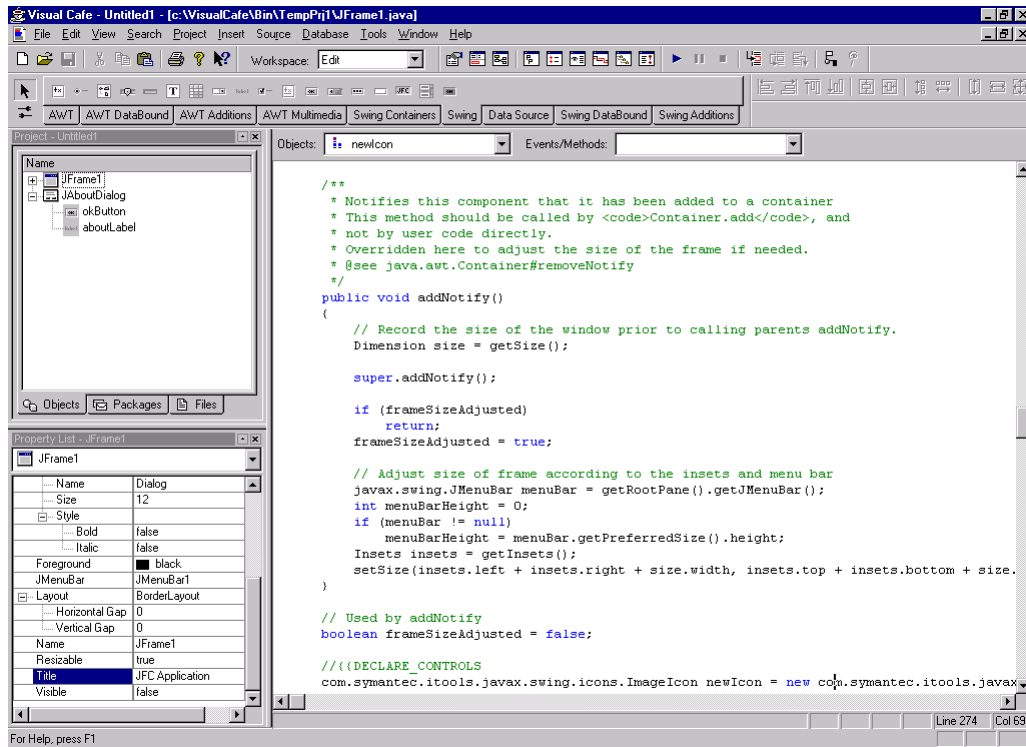
When beginning a project, Visual Café proposes project-specific assistance.



Selection of project type.

With the exception of the HTML part, the development environment is integrated.

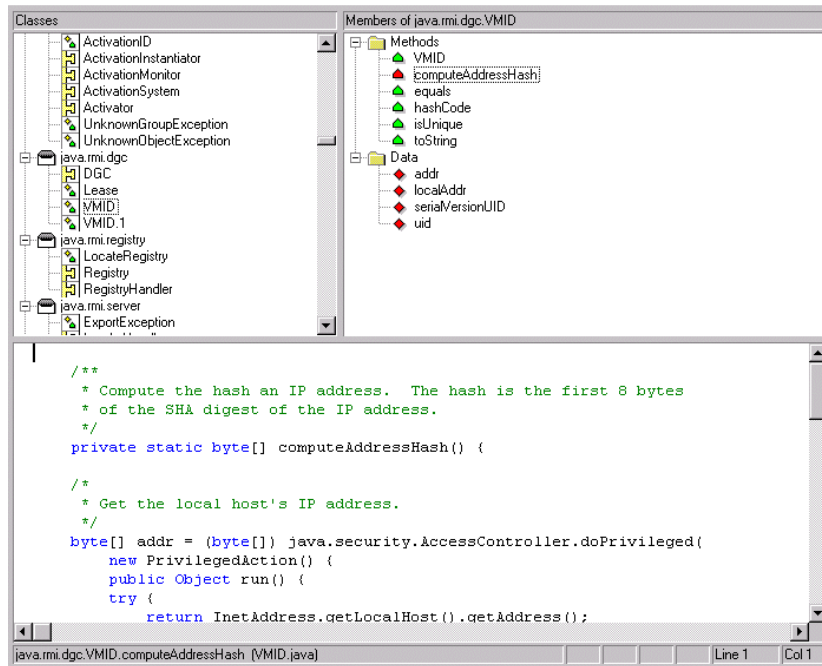
It is possible to develop the Java code corresponding to either the client part or to the server part destined for execution within WebLogic. It is worth noting the possibility of developing client-side Java applications, applets or JavaBeans on the client side using AWT or JFC graphical toolkits. On the server side, it is possible to create servlets or EJBs.



Visual Café integrated environment.

Visual Café is resolutely destined for Java programmers. It offers numerous tools such as a class browser, a module facilitating creation of multilingual applications; and a powerful debugger. Even if it does not offer a group work manager, it is nonetheless able to integrate with repositories that support the SCC interface. On the other hand, it is unfortunate that there is neither a modeling tool nor a report-generating tool. With regard to modeling, integration with Rose via RationalLink, a third-party tool from Ensemble, didn't seem tight enough.

Because of its various database access functions, especially its powerful graphical components, Visual Café has positioned itself as a product to be used primarily for implementing applications based on a relational model. But apart from its support for EJB development, Visual Café does not provide much help with the business object approach.



Class browser.

The class browser is ergonomic; hierarchical structure enables quick access to the methods and variables of the environment's different classes.

3.2.3. Evaluation

➤ Summary

With Visual Café, you have at hand one of the best Java development environments on the market. The interface is productive and the developer has access to good quality services: debugger, database access, graphical components, various wizards, and so on. However, the development environment obtained only a mediocre overall rating.

The three main reasons for this, which we will detail, are as follows:

- Complex development.
- Lack of functionality.
- Poor integration between the tools and the application server.

➤ Powerful server, but complex development

WebLogic offers one of the most powerful Java platforms around today. More specifically, this power lies with J2EE frameworks and such product-specific frameworks and technologies as T3 middleware, load balancing and failover technologies. With regard to development, implementing this power is somewhat complex. The frameworks are not hidden behind high-level tools but constitute the programming model. This is a very interesting approach, but it requires teams of development experts skilled in all the latest Java technologies.

➤ **The environment's lack of functionality**

Although Visual Café Enterprise Suite offers an exceptional Java development environment, it does not constitute a complete solution for business application development. Report-generating, modeling and testing tools need to be added along with all the difficulties these tools imply in terms of global integration of the development environment. Furthermore, servlets must be programmed directly in Java. Indeed, Visual Café does not yet offer the highest-level support for developing dynamic HTML pages. Moreover, the lack of integration between Visual Page (HTML tool) and Visual Café prevent the latter from offering a common repository to all of the elements of a Web application.

➤ **Limited environment/server integration**

With regard to integration between Visual Café and WebLogic Server, BEA does not succeed in winning on all the fronts, which are independence in relation to the development tools and seamless integration. The major difficulty is that in order to have high-level integration, the environment must know the concepts that are specific to the application server. In the case of Visual Café, integration takes place at a rather low level. On one hand, it implements EJBs, which are customized in WebLogic and on the other hand, a group of JavaBeans shipped with BEA that integrate with Visual Café. But again, the situation is problematic. Indeed, we discovered that Visual Café Enterprise Suite 3.1 supports the previous version of the WebLogic Server (version 4) only and not version 4.5.1, which is evaluated in this report. Nonetheless, it is possible to use some functions, but Symantec states that support for version 4.5.1 should be considered as only a « preview ».

IDE quality and richness	Rating/10
Installation, learning process, simplicity	3.8
Product completeness, openness to other tools	3.0

(More details about the above ratings can be found in section 4.2.1, page 49)

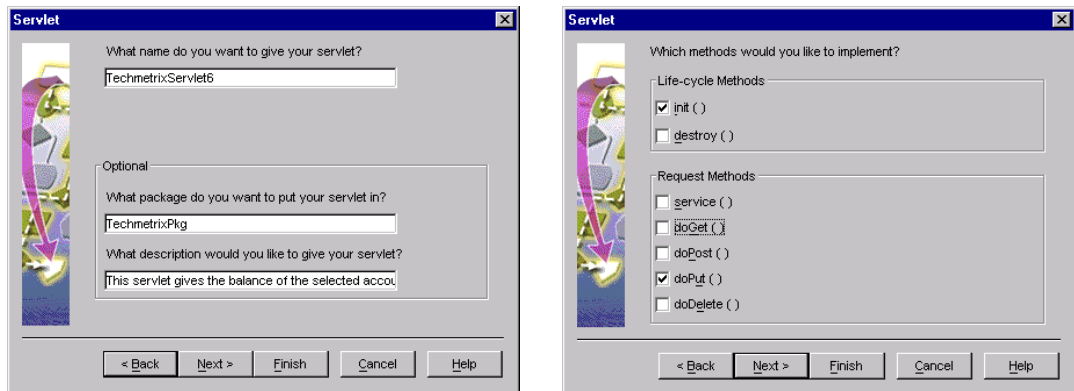
3.3. HTML interface generation

3.3.1. HTML interface generation: design and optimization

WebLogic Server integrates a servlet engine (servlet API version 2.1) and JSP technology (v. 1.0). Thus, there are primarily two possibilities for developing the HTML interface:

- Use of Visual Café to create servlets.
- Use of JSP technology.

In the first case, a wizard enables construction of a code skeleton for the servlet. In this way, it is possible to specify the servlet type (HTTP-specific, Generic or Custom) within the wizard. The servlet will then be developed to be thread-safe or not. You then give a list of the servlet API's methods that you would like to implement, possibly along with all of the additional servlet parameters.



Servlet creation wizard.

After that, the servlet skeleton is generated:

```
public class TechmetrixServlet6 extends HttpServlet
{
    public void init(ServletConfig config) throws ServletException
    {
        super.init(config);
        // to do : code goes here.
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
    {
        resp.setContentType("text/html");
        PrintWriter out = new PrintWriter(resp.getOutputStream());

        // to do : code goes here.

        out.println("<HTML>");
        out.println("<HEAD><TITLE>TechmetrixServlet6
Output</TITLE></HEAD>");
        out.println("<BODY>");

        // to do : your HTML goes here.

        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```

The developer can then insert his own code. Nevertheless, this type of programming remains rather unproductive as the HTML flow is generated directly by the servlet code, which must combine presentation elements with calls to business objects.

The second possibility is based on the use of JSP technology, which enables developers to place Java code directly in HTML pages. This Java code is run server-side when a browser requests the page. In the end, it must turn out HTML. This dynamically generated HTML then takes the place of Java code in the page, which is finally sent to the browser. During execution, the requested JSPs are dynamically compiled in servlet form, except if the compiled version already exists, in which case it is used directly. WebLogic also provides another HTML generation tool, htmlKona. In fact, this tool is a framework that enables HTML pages to be represented by a Java object graph in memory. This API furnishes the Java objects corresponding to the different elements of an HTML page (header, button, field, form, table, etc.). These objects are capable of producing adequate HTML code. HtmlKona thus offers a useful API for generation and modeling of a high-level, server-side HTML user interface.

There is no workspace dedicated to JSP development, but developers can rely in part on the HTML editor, Visual Page, which enables creation of static HTML elements.

3.3.2. Evaluation

➤ **Summary**

The development environment offers little assistance with HTML interface generation. The next version of Visual Café should offer better support for JSP. It is unfortunate that there is not an HTML component model within the workspace. Moreover, the static HTML editor has its limits: no cell breakdown, no support for ActiveX components, no help with direct input of HTML, and lack of support for client-side scripting languages.

HTML interface generation	Rating/10
Automation of HTML interface generation	3.3
HTML interface creation: assistance and potential	3.3

(More details about the above ratings can be found in section 4.2.1, page 51).

3.4. Java graphical interface generation

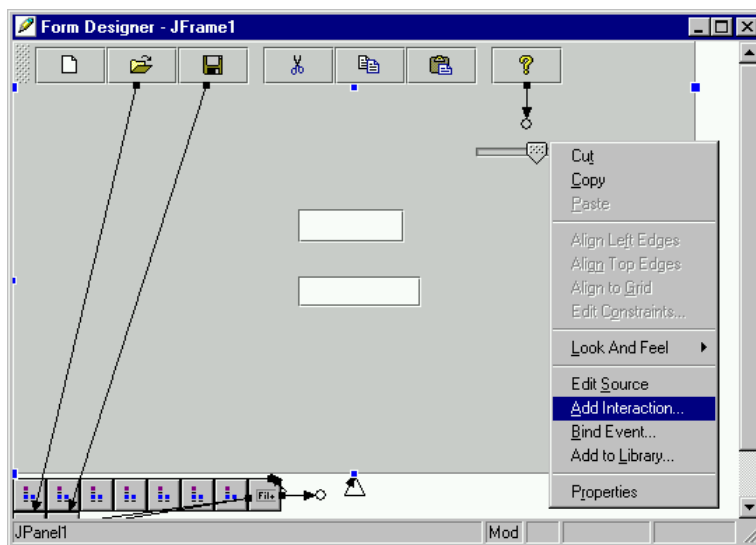
3.4.1. Java graphical interface: design

In this domain, the WebLogic/Visual Café couple offers a very high-quality environment.

Visual Café provides thorough assistance right from the initial stages of development. The tools propose a specific wizard for each type of development. Regarding Java interface development, it is worth noting the following options:

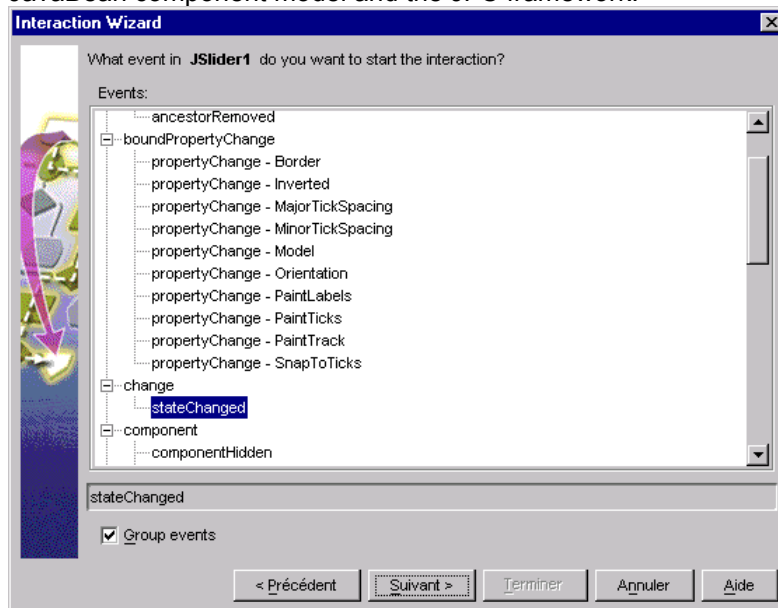
- Databound project, which helps with the creation of database access applications.
- JavaBeans.
- AWT Application.
- JFC Application.
- Win32 AWT Application.
- Win32 JFC Application.
- JFC Applet.
- AWT Applet.

Once the wizards have put the application skeleton in place, Visual Café offers a powerful and productive interface editor, which functions in WYSIWYG mode. Visual Café's graphical components palette is extremely rich. Swing components are supported in their entirety. The addition of Symantec graphical components that are 100% Java-specific (AWT Multimedia, Swing addition) will satisfy even the most demanding developer. To add a graphical component, you simply drag-and-drop; a sensitive grid similar to that of Visual Basic facilitates positioning components. Component properties are easily modified with the help of the property editor.



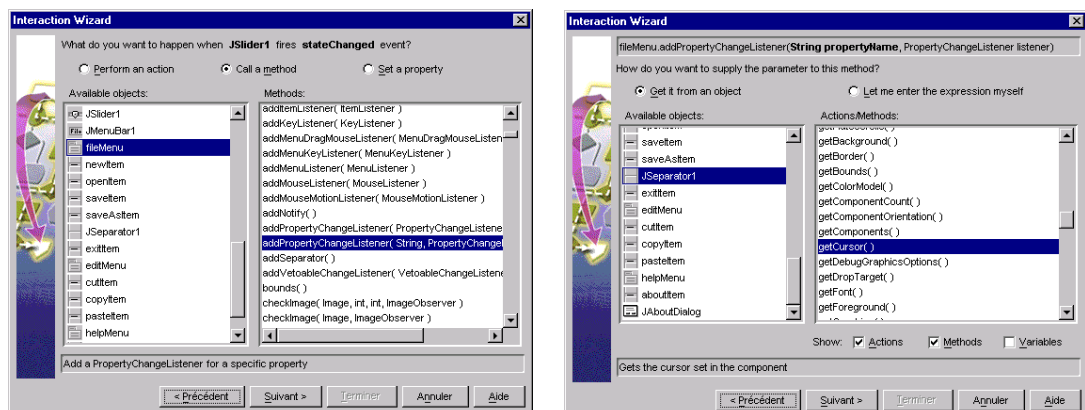
Construction of a graphical interface with Visual Café.

Visual Café not only allows visual implementation of graphical interface elements, it also enables configuration of interactions between these elements and with the business code. On this point, Visual Café takes the most advantage possible of the mechanisms present in the JavaBean component model and the JFC framework.



Visual Café's interaction wizard.

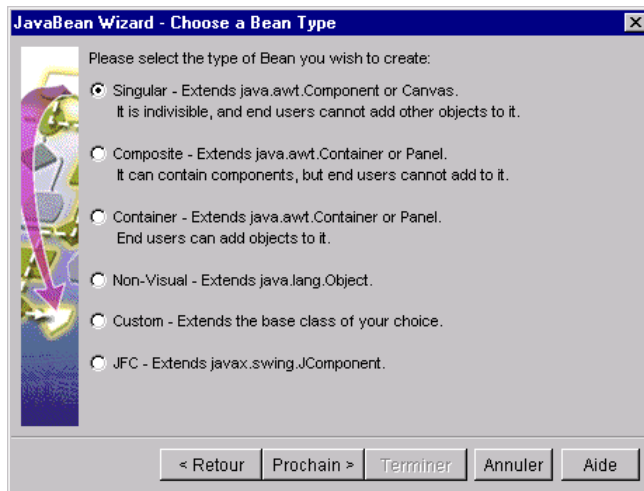
This tool offers some remarkable possibilities. The interaction wizard, which enables configuration of event flow or method calls between objects, provides great help: graphical connections, lists of the objects and methods that can be called, assistance with configuration of the various arguments with which these methods are provided, etc.



The interaction wizard offers extensive assistance.

At each step of the interface construction process, it is possible to display the corresponding source code generated by Visual Café. In certain conditions, it is even possible to edit this code; the visual interface wizard automatically takes any modifications made into consideration.

With Visual Café, it is also possible to create your own reusable JavaBean components. These components can then move into the IDE at the same level as the natively furnished components. Again, some very useful help is given.



A step in the creation of a JavaBean.

To help with implementation of Java user interfaces, WebLogic proposes a BeanBar, a tool that brings together all of the JavaBean components that implement graphical interface elements (text fields, tables, etc). Created in AWT, these Beans are capable of connecting and interacting with the application server. To do so, an events-based distributed communication model is provided. The BeanBar, which can be used within Inprise JBuilder and Visual Café, allows developers to create applications very quickly since communication with the WebLogic server is managed, for the most part, automatically. Nonetheless, there are two reasons for which we advise you to remain prudent when using these components in production. Firstly, during use we ran into some display problems using these components. Secondly, the programming model used, « Client T3 » is based on old WebLogic APIs, which have since depreciated.

3.4.2. Evaluation

➤ Summary

The WebLogic/Visual Café couple obtains a very good rating in this area of evaluation. Visual Café offers one of the best Java application development environments around. Integration with WebLogic could be tighter, but the possibilities are already quite good. It is also worth noting that WebLogic offers HTTP tunneling for RMI, as well as other very useful technologies for client/server interaction. In the case of an applet, there is even HTTPS, but this may result in a decrease in performance.

Java interface generation	Rating/10
Java graphical interface: automation and assistance with elaboration of Java graphical interfaces	8.3

(More details about the above ratings can be found in section 4.2.1, page 52).

3.5. IDE productivity for server processing

3.5.1. Assistance with server-side code generation

Development of WebLogic applications is based entirely on Java and thus enjoys all the advantages that are associated with it such as portability, security and object development.

WebLogic offers powerful application security capacities, which are based on the implementation of Access Control List (ACL) that can be supported by servlets, JNDI contexts, JDBC connection pools, EJB methods, JMS files and subjects and T3 workspaces. Unfortunately, this security system cannot be configured in the IDE and requires a text file.

The development environment does not include transaction application creation wizards that can be configured. But, help is given with creation of Corba, EJB and RMI objects (but not ActiveX).

WebLogic does not offer any assistance with two very common Web needs: sending e-mail and uploading files.

Management of the unique HTTP session identifier is automatic and based on the servlet API. You could choose to use cookies or full URLs. In the latter case, you should use the URLEncoder() API at servlet level during generation of the HTML corresponding to hypertext links. The context for each session is stored on the server side. No history recording mechanism is provided.

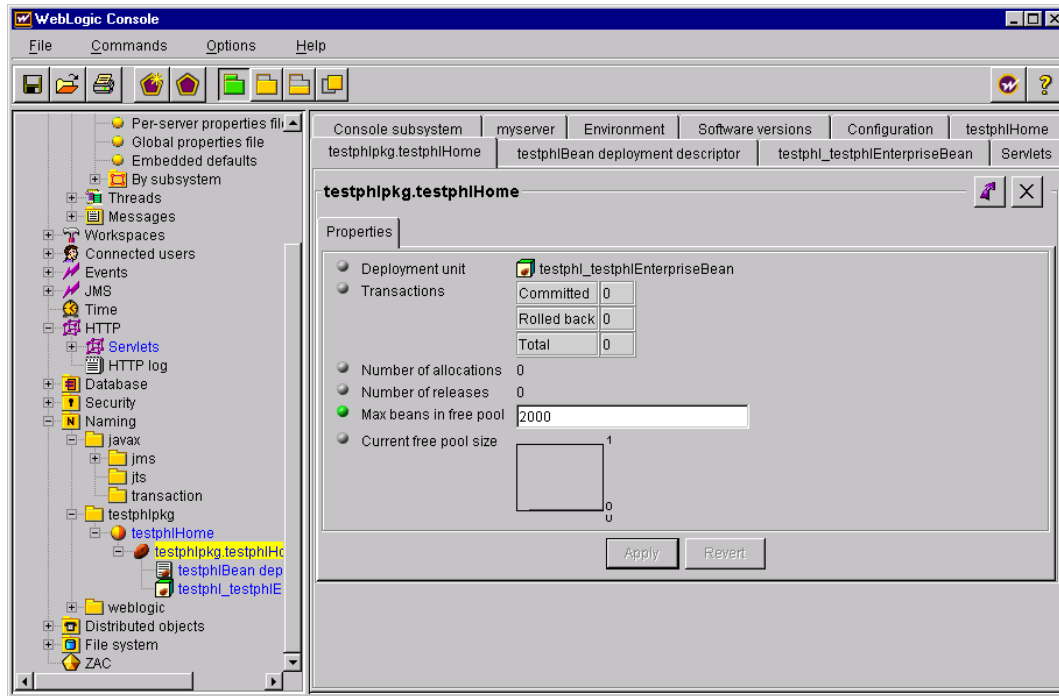
3.5.2. Tools and automation provided by the IDE to improve productivity

The Visual Café environment offers a complete and powerful Java code editor that includes numerous features: syntax analysis tool, method selection menus, multi-file find/replace capacities, etc.

Visual Café's remarkably high-level debugger is capable of running in distributed mode, which enables automatic passage from one virtual machine to another and thus makes it possible to keep track of the control transfers that occur between the different processes. However, the WebLogic Server version evaluated in this study is not completely supported yet. Moreover, WebLogic does not support debugging if the application being tested is deployed in cluster mode.

There are Java interface components that make it possible to link the graphical interface and data sources, but this option is limited to Java interfaces (no help with the HTML interface). You will not find any possibility for mapping between the interface and business objects, in the sense of object modeling. Moreover, WebLogic remains based principally on the relational model.

WebLogic's administration console enables display of some server characteristics and, in some cases, modification of these characteristics. But generally speaking, administration is handled via direct modification of configuration text files.



The administration console gives access to all of the components deployed on the server.

You will not find a sophisticated statistical tool for analyzing application use and server consumption, but the administration console does furnish several useful statistics. A workload test tool must be purchased separately.

3.5.3. Evaluation

➤ Summary

Good assistance with Java programming, offset by some functional shortcomings would have resulted in a decent rating in this area for WebLogic/Visual Café if only integration between these two products had been finalized.

IDE productivity for server processing	Rating/10
Assistance with server-side code generation	3.8
Tools and automation provided by the IDE to improve productivity	4.2

(More details about the above ratings can be found in section 4.2.1, page 52).

3.6. Application server

3.6.1. Database access

In three-tier development, client applications interact with the WebLogic server via interfaces that propose application or presentation services. These services, which can be high-level, are used by various client applications and can represent the business processes common to all applications. Once a company's business logic has been factorized at server level, it is easier to master and can evolve without needing to be deployed on client stations. These clients interact with the WebLogic server via several technologies: distributed objects, sockets, HTTP/HTML, type 3 JDBC, etc. In this model, it is WebLogic that accesses data sources.

➤ JDBC support

Since its creation in 1995, WebLogic, which is now an integral part of BEA, has been particularly involved in database access with Java, in which the access method is JDBC. An evaluation version of the CloudScape database is also included in the package. Of course, it is possible to connect to other databases.

JDBC is a data access API similar to ODBC that can be used from Java. With JDBC, it is possible to send SQL queries to a database and use the results, which are returned in ResultSet objects.

WebLogic supports four types of JDBC drivers, but they are sold separately (see the Product Profile for pricing). BEA offers type 2 JDBC drivers for Oracle and type 4 for Informix and SQL Server. For Sybase, JConnect is used (sold separately by Sybase). These drivers guarantee good performances. In general, any JDBC driver can be used.

➤ Connection pooling

Connection pooling is one of WebLogic's key functions. Thanks to this technique, an application can support a large number of users and at the same time maintain interaction with the database via a limited number of connections.

But not all of the consequences of connection pooling are beneficial. For example, with connection pooling it is no longer possible to rely directly on database access control features because WebLogic is the only true database client. In fact, as far as the base is concerned, you lose the notion of user identity. Control features must be implemented at application server access level. Nonetheless, in terms of scalability, the advantages of pooling are so significant that it has become one of the strategic technical characteristics of application servers. In this sphere, WebLogic is completely satisfying as it offers a powerful pool and very good configuration possibilities.

Connection pool configuration consists in defining a group of properties associated with the pool such as initial and maximum number of connections, unused connection management mode, etc. ACLs (Access Control Lists) can be put in place and possibly stored in a LDAP directory or database.

It is worth noting that even if access to pooling services currently remains specific to each application server, and is thus non-portable; the situation is in the process of changing within the Java world as the new JDBC 2.0 API takes care of access to pooling services.

➤ **Support for database specificity**

Database support is very complete. Binary-type columns (Long Raw, Blob) are supported and their use is documented. Moreover, it is possible to implement specific SQL commands for each base or use the JDBC's Extended SQL technology. Stored procedures and in/out parameter transfer are supported and documented.

➤ **Restart after failure**

With regard to DBMS failure and restart, WebLogic is capable of reconnecting to the base. Nonetheless, connections from the pool that were given to application code before failure will no longer be valid. A valid connection is obtained when a new connection request is made. During our tests with a servlet application, connections in the pool were restarted with each page; the failure did interrupt service. However, service was not reestablished when running tests with a Java-client application developed with WebLogic BeanBar components.

➤ **High-level framework**

Rounding out WebLogic Server's development possibilities is a technology that enables even higher-level coding than JDBC. This technology, dbKona, comes in the form of a framework that proposes an abstract view of the database, which is independent of the particularities of each database engine. Beyond independence at RDBMS level, the presence of dbKona is interesting in terms of productivity. Indeed, JDBC imposes low-level coding and thus remains relatively unproductive (if you have already had to handle schema metadata via JDBC, you know how fastidious programming with JDBC can be). By offering higher-level development, dbKona broadens WebLogic's already wide range of database access possibilities.

➤ **Distributed transactions**

With regard to DBMS access possibilities, the only thing we regret is the lack of support for the XA standard and two-phase commit because it limits transactions to a single external resource manager (no transactions on several bases at the same time).

3.6.2. Language richness, openness

In choosing to implement the J2EE platform defined by Sun, BEA offers a rich, powerful and open runtime environment. But J2EE's strength does not stop there. In fact, this model is in the process of becoming an industry standard that is supported by such major players as IBM

and Oracle. The code developed using J2EE is more portable than code that requires specific frameworks.

Moreover, Java language, which is both easier and more powerful than C++, is on its way to becoming one of the most frequently used programming languages for new business projects. It is a true object language, which also offers good productivity thanks to such technology as Garbage Collection. The presence of object frameworks that cover several areas allows the developer to concentrate on implementing business logic. Several frameworks, which are especially adapted for implementation of business information, really stand out:

- The JDK for all basic frameworks (object collections, mathematical functions, graphical interface, object introspection, archiving and object replication, etc.).
- JDBC and dbKona for database access.
- JMS 1.0.1 as distributed message model (a new feature in this version of WebLogic Server).
- An optimized RMI version for communication between distributed objects.
- Implementation of JSP 1.0 technology (a new feature in this version of WebLogic Server) for servlet generation.
- API servlet (v. 2.1) and htmlKona for transactional Web interfaces.
- JNDI 1.1 for distributed naming services.
- WebLogic Files Services for accessing server-side files from client applications.
- EJB 1.0 as business component model.
- JTA and JTS 1.0 for transaction management.

➤ **Variable/session management model**

WebLogic handles variable management both at session level, which ensures context independence for each user; and at application level, which allows for sharing of information. There are several ways of implementing a variable at application level, depending on the context. For example, you could use instance variables at servlet class level. Indeed, a single instance of each servlet is created and shared.

Via the servlet API, WebLogic offers methods enabling dynamic configuration of HTTP session time-out. Therefore, it is possible to define a common time-out for all sessions but also to configure time-out specific to each session. End of session is managed perfectly. It is possible to force invalidation of a particular session as well as configure the application code call during end of session.

➤ **Openness**

WebLogic's adoption of J2EE and its respect of Web standards result in a good amount of openness.

WebLogic Server is based on the RMI object communication model. Openness to DCOM allows the client using this protocol to access, to a certain extent, the application services implemented within WebLogic. Be careful, development is fastidious when using this model. Moreover, WebLogic Server can be a Corba client; in other words, it can access Corba services outside of WebLogic. BEA has announced that the next version of WebLogic should be a Corba server via adoption of the RMI/IIOP model. BEA is getting ready to implement a validation campaign of interoperability with languages rooted in C++.

WebLogic, as a client, is capable of interacting with LDAP engines via JNDI. SMTP and POP3/IMAP4 should be supported via the JavaMail framework in the next version of WebLogic. WebLogic can also be an HTTP client, but it is then necessary to rely on a low-level framework that renders programming fastidious.

As SSL is taken into consideration, integration of a secure environment is made easier. Firewalls can be used thanks to the HTTP tunneling technology that WebLogic proposes. This technology makes it possible to transfer T3 connection frames on the HTTP protocol. HTTP tunneling is supported between a client application and the server. For an applet, it is also possible to do HTTPS tunneling, but be aware that doing so results in a significant decrease in performance in terms of exchanges.

➤ **EAI capacities**

The WebLogic Server solution does not provide openness to ERP. However, it is open to eLink, an EAI (Enterprise Application Integration) architecture. The product eLink, which is edited by BEA, is built around the Tuxedo engine and offers numerous adapters enabling access to different enterprise systems (ERP, MOM, etc.).

Openness to TP monitors and some integration middleware is good. More specifically, a high-level API above Jolt, Tuxedo's native Java API, provides openness to Tuxedo. It is also worth noting openness to BEA's other application server, WebLogic Enterprise, which was previously known as M3. CICS can be reached using a BEA product, Java to Host. As far as MQ Series goes, it can be used directly via its own Java APIs.

Although WebLogic has excellent openness potential, it does not yet offer enough elements to make it a true EAI platform. For this, BEA should work on integrating such technologies as Java connectors or XA transactions.

➤ **Object modeling capacities**

For object modeling, WebLogic is betting on emerging EJB technology. The product supports specification 1.0 and such extensions as Entity Beans, which are partially implemented.

WebLogic Server was one of the very first tools to offer EJB support. Today, it offers some interesting load balancing and failover features centered on this technology.

The goal of EJB technology is to surpass the basic Java object model by integrating new functionality considered important for business systems. Most notably, EJBs bring functionality in terms of:

- Server management of object life cycle.
- Object security: who can use which object and how?
- Object persistence: how are objects stored on a long-term basis?
- Transaction on the objects: how should one proceed with implementation of ACID transactions including Rollback possibilities while working with objects and not relational database tables?
- Distribution: how can remote applications access objects in the most transparent manner possible?
- Workload support: by implementing object-pooling technologies.

The components must be portable across different environments supporting EJB technology.

From a developer's point of view, an EJB is presented as a group of files bringing together:

- Java classes.
- Java interfaces.
- Deployment information.
- Metadata.

To best understand this evaluation, it is important to know specific EJB concepts and vocabulary. It is particularly helpful to specify the notions of Session Bean and Entity Bean, which describe two possible EJB types:

- Session Beans represent objects whose access supports security, transaction and distribution models. The Session Bean framework integrates the idea of context management, which enables external conversational interactions to be established in the case of multiple-client access. Session Beans can implement « business services ».
- Entity Beans also integrate security, transaction and distribution frameworks, but unlike Session Beans, they support persistence. EJBs are used to implement what are commonly called business objects; they are designed to allow business entity modeling. Therefore, Entity Beans are of the most interest to us in this evaluation because we want to explore the capacity of EJB technology to implement business object modeling with an information system.

WebLogic does not provide a tool for conception and development of the object model but Visual Café does provide development assistance. But again, we run into the same problem brought up before: Visual Café's lack of complete support for the version of WebLogic evaluated in this report.

EJB development requires implementation of several service methods and classes for each business class. It is also necessary to create service files such as deployment descriptors or manifest files.

Depending on the platform, Java development is done using either JDK 1.17 or 1.2.

Using the EJB model implies distinctive business object model development, which differs greatly from traditional Java development. For example, implementing a relationship between objects is no longer question of a simple Java reference; it is necessary to use an EJB-specific mechanism.

Despite the substantial technical and marketing investments made by numerous editors, EJB technology remains immature. With regard to WebLogic, you'll notice the use of Session Beans and Entity Beans. Session Beans are useful for making available distributed application services. We advise against using Entity Beans within the framework of business object modeling.

BEA's EJB implementation suffers due to limits that prevent productive implementation of an object model that is even the slightest bit complex. In particular, persistence limits, as well as limits in the management of object associations, render the solution ineffective. Moreover, BEA recommends looking to more sophisticated EJB frameworks that can integrate with WebLogic. We will detail some of these solutions below and highlight those based on an object persistence engine (object-oriented database) or object/relational mapping.

Versant is one of the most advanced solutions; it offers an EJB container based on a powerful and mature object-oriented database. The Object People, recognized experts for the

past several years, propose a model integrating WebLogic and TopLink, an object/relational mapping for EJBs.

Object/Relational Model	Object-based database
TopLink from The Object People	Versant
ROF from WaterShed Technologies	ObjectStore from ObjectDesign

EJB solutions that can be integrated with WebLogic.

Generally speaking, EJB technology will have to evolve independently of WebLogic on several points before being able to offer a framework truly adapted to object modeling. Among the current problems are:

- Functionality limited to large-grained objects.
- No support for reentry.
- Under-specification of associations between objects.
- Incompatibilities with the standard Java model.
- Under-specification of the distribution model and RMI's immaturity on IIOP.
- Cumbersomeness brought about by the use of static glue.

3.6.3. Deployment

Written primarily in Java and thus very portable, WebLogic Server can boast about being one of the application servers that functions on the largest number of platforms. In terms of support for the deployment environment, it is unfortunate that integration with an external Web server is limited to either NSAPI or ISAPI interfaces. This last point means that the only HTTP servers that can be used are those of Microsoft, Netscape and WebLogic.

WebLogic Application Server is positioned as a business solution capable of supporting considerable workload and has a simplified deployment mode, for which it proposes numerous services and optimizations.

➤ On the fly deployment

The possibility to deploy on the fly, in other words, without having to shut down and then reboot the application server, is one of the biggest additions to version 4.5. It is now possible to deploy, redeploy or delete the following components on the fly:

- JSPs.
- Servlets.
- JDBC connection pool.
- EJBs.

In some cases, it is still necessary to start and then restart the server. For example:

- ACL modification.
- Modification of SSL configuration.
- Configuration of system properties.

➤ ZAC (Zero Administration Client)

ZAC is an infrastructure allowing automatic deployment of new versions of Java client applications. Client applications include a technology that is capable of testing application updates on a server and then automatically retrieving this same application via an intranet or the Internet, and all of this is transparent to the user. The HTTP Distribution and Replication Protocol is a transfer protocol designed to optimize communication by reducing the size of data to be transmitted. ZAC use is not transparent to the developer. On the client side, a specific API controls application update. On the server side, new versions must be packaged in a specific manner.

➤ WebLogic RMI

The RMI implementation provided by SunSoft in the JDK is known for being slow and its performance decreases sharply as soon as the number of users increases. So as to get around this problem and offer a distributed object system, WebLogic proposes its own RMI implementation. Nonetheless, you can use Sun's implementation if you would like.

BEA states that its implementation supports RMI semantics entirely. But in practice, things are not so simple. For example, up until version 4 of the product BEA's implementation did not include distributed garbage collection, which is an integral part of the RMI API. The session-based model proposed was not adapted to all contexts. You must therefore make sure that WebLogic's implementation meets the needs of your application.

WebLogic's RMI differs from Sun's implementation in many ways. Most notably, it offers more development freedom and ease by removing a number of significant and truly problematic design restraints that are imposed by the traditional RMI (inheritance, exceptions). The table below summarizes the principal differences with regard to workload support:

Component	RMI SunSoft	RMI WebLogic
Connection management	Several sockets are opened for communication between a client, a server and the RMI registry.	A single connection between the client and the server. The RMI dialogue passes by this connection as does JDBC, Remote Calls and Events.
Object transfer by value	Serialization.	Optimized serialization implementation.
Management of co-allocated objects	Objects allocated in the same virtual machine can in some cases interact via their stubs/skeletons.	The objects allocated in the same virtual machine interact directly.
RMI Registry	Separate, external application.	Functionality handled by the application server.
Stubs	Standard RMI stubs.	« Smart Stubs », capable of implementing load balancing strategies across several application servers and proposing failover.
RMI compiler	A Stub/Skeleton class pair is generated for each remote class.	A Stub/Skeleton class pair is generated for each remote interface.

➤ **Thread pooling, Object pooling**

Java thread creation can be slow. WebLogic uses a thread pooling technique to get around this problem. Once a thread is done being used, it is once again made available in the pool so that it can be reused.

The various EJB components are managed in a pool in the same manner; this is largely transparent to the developer.

➤ **Clustering**

Clustering, which appeared in version 4 of the product, is a key workload element. The proposed technologies represent an important first step towards reaching BEA's ambitious goal in this field. It enables implementation of architectures made up of several servers that are able to balance workload and replace one another in case of failure.

Such a group of servers is called a cluster and can be configured dynamically. It is thus possible for a server to integrate the group or to leave it using the TSAP protocol (Tengah-WebLogic Service Advertisement Protocol).

EJB, RMI, servlet and connection pool services can be clustered.

A global naming service, which is distributed and replicated, presents clients with a unified view of server architecture. This service is extremely important and accessible via JNDI API.

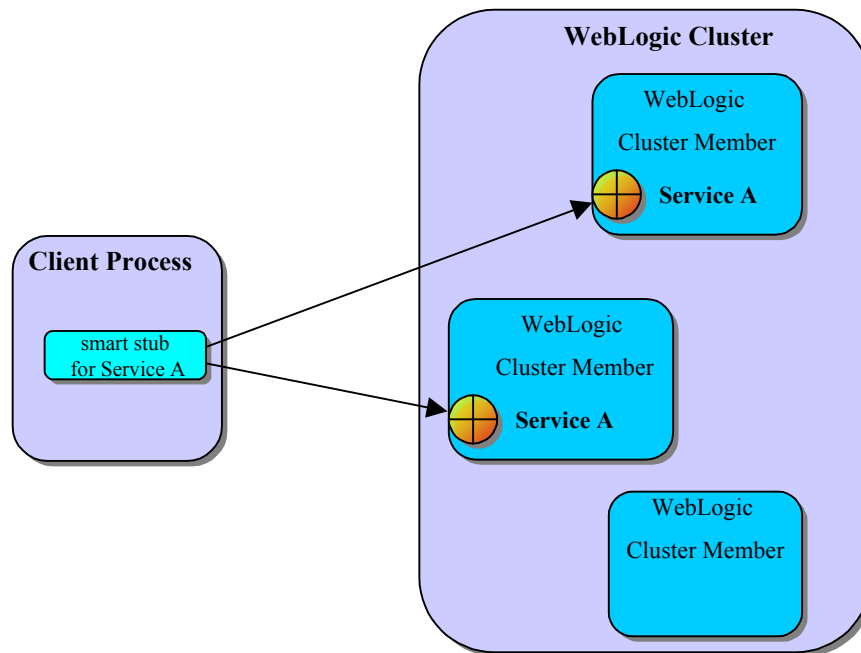
It is interesting to note that each server can run on a different physical architecture and different operating system. As WebLogic is written in Java, the clustering services do not depend directly on functionality provided by the OS, but on that provided by the Java platform.

➤ **Load-Balancing**

Load balancing (distributing workload across cluster members) is for the most part automatic. So that service load balancing can be put in place, the service in question must be housed by at least two cluster members.

Before being able to call a remote service, a client must obtain a reference to this service. In order to do so, a client calls on WebLogic's naming service. The reference obtained is brought about in the client's virtual space by a Java object named smart stub. This is the object that handles service requests (for example EJB method calls). The smart stub is aware of the cluster members housing the corresponding service. It handles load balancing automatically by sending the request to one of the available servers.

WebLogic generates smart stubs. Based on static-glue technology (generation of different stubs for each service), they must then be deployed on the client side.



To access a service, a client goes through a smart stub that handles load balancing.

Servlets stand out among all the WebLogic services offering a load-balancing mode, as they are indispensable for scalability of EJB and Web applications.

In the case of services possessing a state (stateful), the load balancing mechanism takes care of routing towards the cluster node containing the required state. For example, with a dynamic, servlet-based Web application, the load balancing mechanism will reroute all of a client's requests towards the cluster member having the session context that corresponds to this client. With regard to stateless services, a client request can be routed indifferently towards any one of the cluster members supporting the service.

When a stateful service is instantiated or when a request is routed toward a stateless service, the load balancing mechanism selects a cluster member from several candidates. This choice is based on algorithms proposed by WebLogic. The following possibilities have been available since version 4.5:

- Round-robin.
- Random.
- Weight-based.

The algorithm can be configured during stub generation or at runtime for all of the stubs that have not previously been configured.

➤ Failover

Thanks to its cluster-based architecture, WebLogic offers interesting fault-tolerant services.

From a theoretical point of view, the notion of failover is quite vast and covers everything from simple monitoring services and automatic restart to such extensive and extremely complex service continuity services that are based on, for example, replication and synchronization of states. Thus, an application server classified as « fault tolerant » doesn't really mean anything. In fact, this term simply means being able to determine the types of failure that are taken into account and in which manner.

It is interesting to distinguish the difference between material failure and program failure. In the case of material failure, the two fundamental elements of a tolerant system are a cluster (several machines) and a reliable means of shared storage. Should a program fail, cluster architecture can prove insufficient. For example, if part of a program's code is erroneous and triggers the crash of the virtual machine in a particular context, re-executing the code on another server will yield the same results and the entire cluster will collapse. In such a situation, you need an infrastructure that enables implementation of a robust program.

A system's robustness depends, quite obviously, on the robustness of each separate component. However, it also depends on a system's capacity to deal with a lack of robustness of one of its components. Such considerations lead to the formalization of such specific rules for the design and implementation of sub-systems as the « robust design rule. » If you have considerable robustness needs, you will have to call on teams specialized in this area.

The system WebLogic proposes enables automatic replacement of a failed server at logic level. Client requests made after failure are rerouted towards a different server.

But what happens to server-side work in progress when the server fails? In the proposed architecture, unfinished tasks cannot be completed. The client application must therefore be ready to receive not the results of its request, but an error signal. The application must be able to send its request again (we will see that in some cases WebLogic is capable of handling this problem).

Generally speaking, there may be a question of application coherence because the process requested during a failed request can be:

- Not at all completed.
- Partially completed.
- Entirely completed.

Should a request be only partially carried out, serious coherency problems can occur. To avoid such problems, you can use a transactional programming model, which guarantees process atomicity. At this level, WebLogic integrates the necessary machinery, whether it concern JDBC pools or EJBs.

Moreover, a reliable, shared state persistence system must be used. Generally speaking, WebLogic is based on the database.

It remains to be seen if a transaction is completed for a client should an error arise. For this it is necessary with WebLogic to put in place a parallel system that enables identification of the transactions and notes if they were realized or not. For example, the identifiers of completed transactions could be displayed in an ad hoc database table. When accessing this system, the client must be able to obtain the information sought-after¹. It is important to note that such a model encumbers coding considerably.

¹ The documentation is very clear on this point: "All method calls that are not idempotent must have some way of testing that the transaction was committed. The code for this can only be located in the client and the client must have a method of identifying the transaction."

Up until now, we have discussed the most common case scenario, but there are some requests that are easier to manage. For example, an object method made up exclusively of a calculation cannot modify or save any server-side state; this is called an idempotent method. In this case, you can always launch the request again if it fails without worrying about consistency problems. Even better, it is possible to let WebLogic know if the methods of a particular class are idempotent (they do not modify the state of the instance). If there are idempotent methods, the WebLogic infrastructure, namely the corresponding smart stub, takes care of rebooting the call in a transparent manner, possibly on another cluster element.

Development of an application implementing WebLogic's proposed fault-tolerant architecture requires specific coding. The documentation provides a remarkable introduction to this problem and some very useful examples. Compilation takes place using specific flags that enable, for example, to indicate that methods of a class are idempotent. Some attributes of EJB deployment descriptors are also used to configure failover.

With regard to service continuity of HTTP servlet applications, WebLogic is capable, to a certain extent, of duplicating session context. The backup server can then use this context should there be a problem. This capacity is commonly referred to as session-level failover. Duplication can take place in memory of another cluster node (nodes work in pairs) or in an external data manager (the context is thus rendered persistent). While session-level failover is activated, the mechanisms of duplication and synchronization of context are implemented automatically by WebLogic without requiring specific coding.

With regard to EJBs, it is important to point out that WebLogic does not support replication of EJB services that have an internal state. Therefore, only stateless services--namely Stateless Session Beans and Home EJBs--can be duplicated on several cluster members. Entity Beans are not duplicated but their state is persistent in the database; they can be reinstantiated on another server should a problem arise (be careful, this requires use of an indirect programming model, which consists in coding explicitly the reinstantiation of the EJB in the case of partial error).

Concerning Stateful Session Beans, WebLogic does not offer great failover ease since they are not replicated and their state is not persistent. Furthermore, serialization of EJB Handles is unreliable in cluster mode.

In conclusion, its clustered architecture, good load balancing and interesting failover capacities make WebLogic Application Server one of the best non-specialized application servers in this domain. Nonetheless, development of fault-tolerant code remains complex and the product must work on offering more transparency at this level.

3.6.4. Evaluation

➤ Summary

WebLogic achieves an excellent rating for database access capacities. Rich, powerful features offer such advanced possibilities as restart in case of failure or coding via high-level frameworks. Integration of drivers in the solution or support for the XA standard would have resulted in an even higher rating.

WebLogic's adoption of J2EE, as well as its respect of Web standards, results in a good degree of openness. The richness and power of the Java platform and WebLogic-specific frameworks enable implementation of large-scale applications. Its unfortunate that support for ERP, reporting and such technologies as SET, SMTP or XML is low-level, but no application server offers a solution covering all technical domains. In fact, WebLogic's only big weaknesses in this part of the evaluation concern the relative immaturity of J2EE and the product's very weak capacities for implementing a business object model.

Deployment capacities are very good. WebLogic Server runs on several platforms and is positioned among the best with regard to load balancing and failover services.

Application server	Rating/10
Database access	8.8
Language richness, openness	6.4
Deployment	7.9

(More details about the above ratings can be found in section 4.2.2, page 54).

4. Annexes

4.1. Workload methodology

➤ Introduction

Objective

To compare the performance and intrinsic behavior of intranet application servers, by testing (through the use of workload tests) the various implementations of TMBench 1.0 specifications presented here.

How?

- By relying on HTTP and SQL standards, rather than on products considered as references.
- By offering representative, independent intranet units instead of a complete integrated intranet application.
- By making the implementation of test units easier (simple and straightforward HTML layout).

Characteristics of the chosen intranet architecture

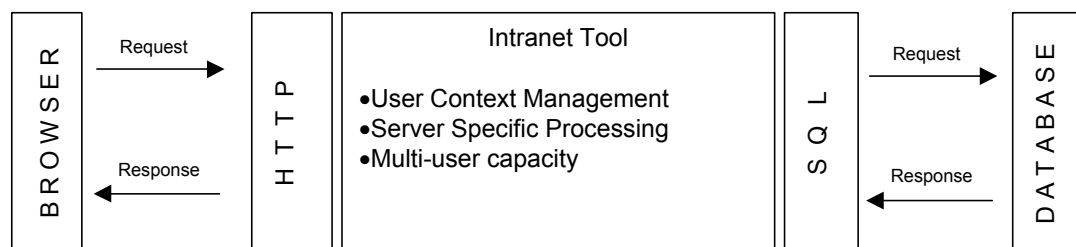
The application's clients are standard browsers (the application generates only HTML).

The development tools used to implement TMBench 1.0 must have the following intranet development features:

- The tool must have a programming or scripting language.
- The tool's application server must allow for algorithmic calculation and processing.
- The tool must allow access to the standard databases available on the market.
- User context management must be possible.
- Finally, the application developed by the tool must operate in multiple-user mode (for several concurrent users).

General principles of TMBench 1.0:

The tool specifications attempt to highlight the services provided by the intranet part by relying on two standards: HTTP and SQL.



A pair (request, response) defines a basic transaction. The request is an HTTP request (the size and format of the URL are not specified). The expected HTTP response is entirely standardized (the result and format of the answer are imposed).

In requests involving database access, DBMS access is normalized with a SQL query. Most SQL queries cited in this document can be used as they are, but they can also be adapted to a specific intranet tool, mostly concerning the data access possibilities that the tool offers.

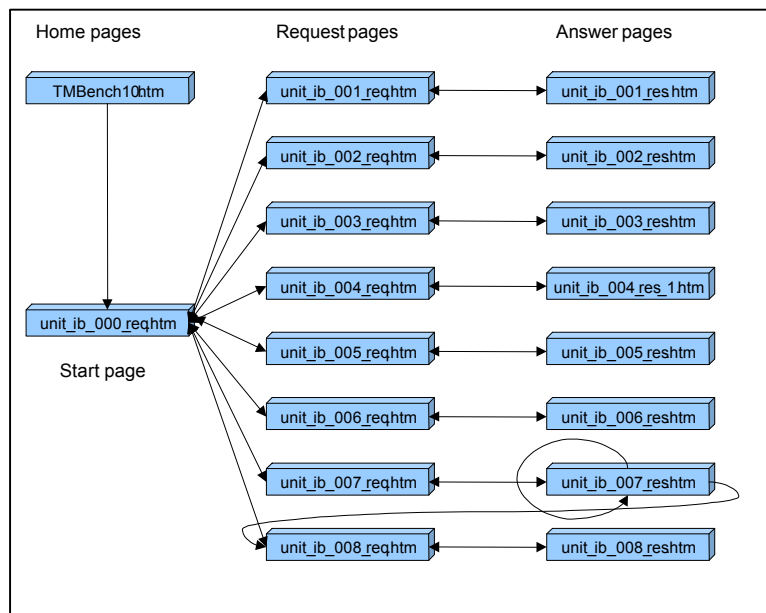
➤ **Database and SQL queries**

The database model used is of a relational type. No particular DBMS product is specified, and the choice of a database engine common to all TMBench 1.0 specifications is made at the last moment during the tests. The SQL queries must be as standard as possible (in other words, usable with the most popular DBMSs on the market).

The physical data model contains six tables, three relationships, 50 fields and a maximum of 60,000 records for a total data volume ranging from 10 to 100 MB. These contents may vary at any time in order to accommodate the specific needs of different tests.

➤ **Bench application**

The TMBench 1.0 application is made up of eight independent units, each including two pages (one request page and one response page). There is also a welcome page and a page that references the eight units. TMBench 1.0 is therefore made up of a total of 18 HTML pages. The application's kinematics is as follows:



TMBench 1.0 application kinematics (all of the HTML links)

Role of each module

The eight modules of the scenario allow us to center the analyses on basic features. The goal is thus to isolate each of the desired features and, using the dominant functional features of the desired application, to be able to anticipate the behavior of the deployed application in relation to the application server selected.

Modules	Role (basic feature analyzed)
Module 1	Large "Select" (+ than 5000 records returned) without cache in a database
Module 2	Sum of the small "Select" without cache in a database
Module 3	Sum of the small "Select" with cache in a database
Module 4	Multi-criterion search with dynamic request, without cache
Module 5	Database updates (simple insertion and updates)
Module 6	Algorithmic calculation
Module 7	Storage of user context in memory
Module 8	Mass insertion

The complete TMBench specifications can be downloaded from the TechMetrix Research Web site: <http://www.techmetrix.com/lab/tmbench/tmbenchindex.shtml>

Technical constraints

The editor must respect HTML page content as described herein as much as possible while keeping in mind the following constraints:

- HTML file names are given for example purposes only; you may choose others.
- At the top of each page there must be an HTML title and label (standard HTML text) indicating the test reference.
- A back to Start Page link must be found at the bottom of all "starting" HTML pages (requests).
- A back to IB-00X (previous page) link, which enables the request to be restarted quickly, must be found at the bottom of all HTML "answer" pages (request results).
- No JavaScript data input monitor is mandatory in the HTML pages.
- The request pages of IB-002 and IB-003 units possess "hard coded" SQL requests. These values can be modified at any time, thus making application evolution possible.
- The only freedom concerns URLs and their format (URLs of the HTML links and the SUBMIT buttons), as they cannot be imposed.

Format of normalized HTTP/HTML responses and requests

The format of HTML request and response pages must comply with the following guidelines as closely as possible:

- Resulting data is to be put in tables as soon as possible.
- The default table border must be equal to one.
- No specific format is to be applied to the data displayed.

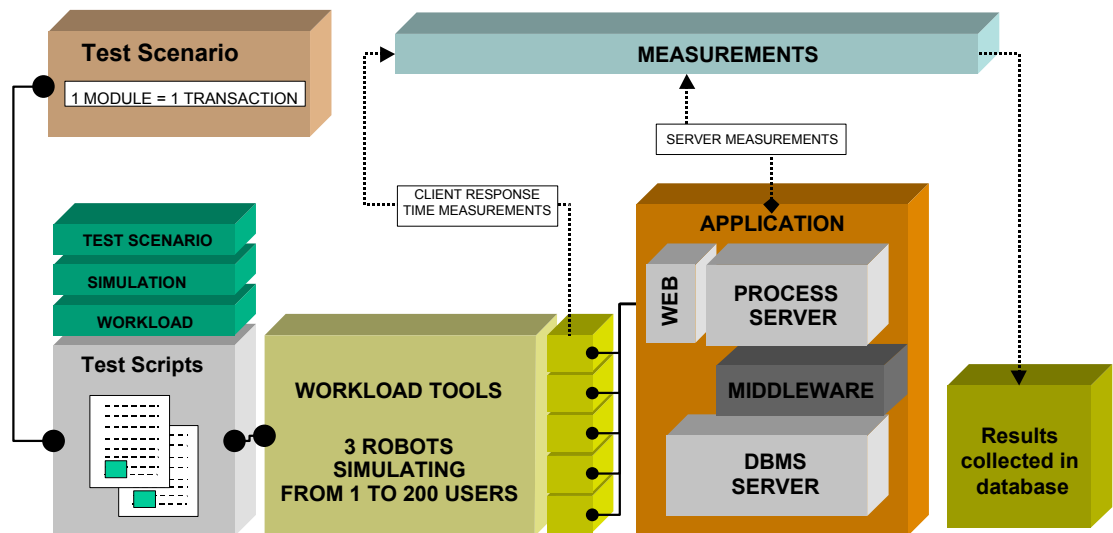
➤ Performance measurements

Once we have received and validated the integration complying with all of the TMBench 1.0 specifications, a workload test, simulating at least 200 concurrent users, is conducted. Breaking the application up into test modules makes it possible to define each unit as a test transaction independent from the others. Each transaction (called a scenario) is subject to a

measurement of client response times. When a unit is used several times in sequence, the whole of these iterations makes up a complete transaction.

Method used for workload tests:

Our method is based on pre-prepared test scenarios, which are conducted as shown in the diagram below. These tests, which involve the entire application architecture, are applied to each TMBench 1.0. implementation in the same way.



Workload test architecture

To have a better understanding of measurement analysis, note that each unit makes up a test transaction, and therefore a measurement in its own right. This measurement corresponds to a request and to the obtainment of its answer. The time required for a complete IB-001 transaction is equal to the sum of the amount of time taken to send a request and the amount of time taken to receive its result from the server. For unit IB-001, this measurement corresponds to the time needed to go from page *unit_ib_001_req.htm* to page *unit_ib_001_res.htm*.

Performance measurements and indicators

Measurements are made at three different levels:

- On the workload test robots, by collecting time values from HTTP server requests and answers with which the simulated users are interacting.
- On the processing server, with a measurement station which traces and logs the percentage of CPU time and the memory used, which expresses the load generated by the application server.
- On the DBMS server, by counting the number of connections opened by the application, and the load generated by the SQL orders.

4.2. Functional evaluation criteria

Each criterion is evaluated and then rated using one of the three following values:

- ☆☆ or YES: criterion supported correctly
- ☆ or YES Minus: criterion supported partially or is complicated to use
- or NO: criterion not supported or is much too complicated to use

4.2.1. Productivity of the development environment

➤ Quality and richness of the development environment

Step 1: Installation, ease of learning, simplicity

N°	Description	Rating	Comments
1	Quality of integration between tools (respect of unique window).	●	Visual Café provides an integrated IDE for Java development. The HTML editor is a separate product, but nonetheless ships with Visual Café. Integration between Visual Café and WebLogic Server, which are two different products, covers some aspects, but on the whole, there is little integration between the various tools.
2	Printed documentation and online help with search options.	☆	WebLogic Servers ships without paper documentation. Online documentation is found on BEA's Web site, in the form of HTML pages or PDF documents. BEA provides a search engine. Accessing documentation requires an Internet connection. Visual Café's good-quality documentation is accessible via the Help menu and several manuals are available in PDF.
3	Ease-of-use and installation. Installation mustn't call for too many prerequisites (browser, DBMS, HTTP server). The developer must be able to use the wizards quickly.	☆	WebLogic's installation process on NT includes an automatic stage. Then, other operations must be performed manually (JDK installation, installation and configuration of database drivers, configuration of environment variables, installation of IIS plug-in, etc.). These operations are fastidious and deserve to be more clearly described in the documentation. In the end, a previous version of WebLogic results in operation failure, but the product does not correctly diagnose this problem. On the whole, installation and implementation could stand a considerable amount of improvement. Visual Café's installation process is exemplary; the IDE can be used immediately. Wizards allow you to begin using the tool very quickly. Visual Page, the tool used for HTML page design, must be installed separately.
4	Richness and quality of the tutorials and examples provided. The examples must cover all internet-related problems (scripting language, DBMS access, context management, etc.).	☆	WebLogic does not provide a tutorial. However, the examples given cover numerous points. Visual Café offers a tutorial and examples.

Step 2: Completeness of the product, openness to other tools

N°	Description	Rating	Comments
1	Completeness of the product in terms of development, deployment, administration.	★	An editor that enables implementation of dynamic Web pages, and design- and test- level tools would favorably complete BEA's offer. Concerning administration, WebLogic Server provides a graphic console that essentially enables consultation of system information. An online command tool allows for EJB administration. Another tool, Publish Wizard, makes it possible to implement packages that are destined to be deployed by WebLogic's Zero Administration Client (ZAC) technology. The Deploy Wizard is used to deploy an EJB on the server.
2	Presence of a project and project resource manager, quality of the interface with PVCS, VSS, etc. (check-in/check-out, versioning).	★	Visual Café does not provide a project resource manager, but offers the possibility of integrating tools that support the SCC interface.
3	Tool for developing multilingual applications (listed in a dictionary, post-development extraction).	★	Yes for Java code and interfaces thanks to Visual Café's Localization Tool. No for the HTML interface.
4	Test HTTP server and a local development DBMS.	★★	BEA provides an HTTP server (WebLogic server) and a database (demo version of CloudScape). Furthermore, Symantec provides Oracle Lite and PointBase databases with Visual Café. You can thus begin developing if you do not have these elements.
5	Presence of report-generating tool, the possibility of interfacing with a report-generating tool (for DOC, RTF, HTML, or PDF printing). The report editor must allow for control breaks and offer a test mode.	☛	
6	Presence of a search engine and/or quality of the interface with a search engine.	☛	
7	Presence of a relational and/or object modeling tool, possibility of interfacing with Power AMC, Rational Rose, Mega, etc.	☛	Very limited integration with Rose via Ensemble's RationalLink.
8	SQL query editor, with support for stored procedures (run and display source code). The editor must allow for query input and provide, in real time, without passing by the application server, the result of this query. Richness of the authorized SQL would be a differentiating factor.	☛	
9	Tree structure and application display management tool. From a page, the tool must propose all of the called pages and the request pages. Any change in the page name must be monitored or taken into consideration by the calls. A graphical view of the pages and/or window is a plus.	☛	
10	Repository, cross object references (graphics, components, objects used by the IDE...), detection of unused objects, export of standard objects, etc. For example, during the deletion of a "database connection" object, the IDE must inform the developer of the different uses of this object.	★	Powerful repository within Visual Café, but no global repository that integrates HTML pages.

➤ HTML interface generation

Step 3: Automation of HTML interface generation

N°	Description	Rating	Comments
1	Presence of a WYSIWYG HTML editor. The developer does not need to know HTML in order to design his pages.	☆	The WYSIWYG HTML editor is a separate tool, VisualPage. It only supports static page construction (no support for JSP or other dynamic models). Thus, the designer must recover the HTML generated by VisualPage in order to integrate it in a JSP.
2	Numerous table manipulation possibilities (cell mergers, cell breakdown, nested tables)	☆	No cell breakdown.
3	Presence of a repository for HTML graphic components (objects or groups of objects).	●☼	
4	Possibility of creating style sheets, templates, and page models.	☆	It is possible to create style sheets.
5	HTML code generation during the insertion of Java or ActiveX components and images in HTML pages.	☆	No support for ActiveX.
6	Client-side generation of validation scripts.	●☼	

Step 4: HTML interface development: assistance and potential

N°	Description	Rating	Comments
1	Presence of an HTML display tool and/or an HTML source editor with syntax recognition.	☆☆	
2	Presence of dynamic HTML page generation wizards.	●☼	
3	Import/export of HTML pages and total command of HTML (static and dynamic, in other words, generating HTML code by programming).	☆	Neither Visual Café nor VisualPage supports dynamic JSPs.
4	Assistance with HTML input (tag list, completion, documentation...) and client-side scripting language (events, objects, authorized attributes and methods, documentation...)	●☼	
5	Presence of client-side scripting language components and Java applets.	●☼	
6	HTML 4.0, DHTML and DOM are taken into account.	☆	HTML 4.0. No specific support for DOM.

➤ **Java graphical interface generation**

Step 5: Elaboration of the Java graphical interface: automation and assistance

N°	Description	Rating	Comments
1	Presence of a Java WYSIWYG editor.	☆☆	
2	Presence of a components palette (AWT, JFC...).	☆☆	
3	Possibility of creating JavaBean components.	☆☆	
4	Possibility of creating Java applets.	☆☆	
5	Possibility of enhancing the Java graphics object palette and the presence of an object inspector.	☆☆	
6	Presence of dynamic transactional application generation wizards with a Java graphical interface.	●☆☆	

➤ **IDE Productivity for server processing**

Step 6: Assistance with server-side code generation

N°	Description	Rating	Comments
1	Management of user profiles with application and page access security.	☆☆	Implementation of access control lists concerning servlets, JNDI contexts, JDBC pools, EJB methods, JMS files and subjects and T3 workspaces. Security is configured by editing a text file.
2	Configurable wizards for creating transactional applications, with the possibility of modifying code after it has been generated.	●☆☆	
3	Assistance in creating standard distributed objects (ActiveX, CORBA, and EJB).	☆☆	CORBA, EJB, RMI. No ActiveX.
4	Server-side assistance in sending e-mail (high-level API, examples, documentation).	●☆☆	
5	Automatic management of the unique session identifier.	☆☆	

6	Freedom to choose the context management used for transfer of the identifier (full URLs, cookies, and hidden variables).	★	Cookies or full URLs. More complicated if you use full URLs, as it requires use of URLEncode method.
7	Freedom to choose between a client-side and server-side stored context, with the possibility of tracing history.	●	
8	Assistance in File Uploading (interface, code examples, explanations).	●	

Step 7: Tools and automatic functions provided by the IDE to improve productivity

N°	Description	Rating	Comments
1	Good quality source code editor, with find and replace features, context-specific help, completion, syntax parser, etc.	★★	
2	Complete debugger (logs, step by step, choice of the application or tracing in functions/methods/procedures, modification of variable content).	★	Visual Café's debugger is complete and powerful but integration with Visual Café 3.1 and WebLogic 4.5.1 is not completely supported. Moreover, WebLogic does not support debugging if it is deployed in cluster mode.
3	Assistance with mapping between business objects and the application interface.	●	There are Java interface components that allow the graphical interface and data sources to be linked. But, this is limited to only Java interfaces (no assistance for HTML interface). Moreover, there is no mapping toward business objects.
4	Presence of an application management console.	★	WebLogic's administration console enables some server characteristics to be displayed. Administration is generally handled by directly modifying configuration text files.
5	Presence of an application use/consumption administration tool with graphics and optimization solutions.	★	The administration console provides some useful statistics.
6	Presence of a testing tool and/or workload tool.	●	

4.2.2. Application server richness

Step 8: Database access

N°	Description	Rating	Comments
1	Support for ODBC and/or JDBC (type 1) driver.	☆☆	ODBC supported, but the driver must be acquired separately.
2	Native database access (SQL Net, Dblib-CTLib, Inet) via type 2 and/or type 4 JDBC.	☆☆☆	BEA drivers must be acquired: JDBC (type 2) for Oracle, JDBC (type 4) for Informix and SQL server. For Sybase, use Jconnect (can be acquired from Sybase). Other drivers can also be used.
3	Support for BLOBs/Long Raw (reading and insertion).	☆☆	Long Raw support.
4	Support for DBMS-specific SQL queries (Ex: select getdate (), select banner from v\$version...).	☆☆☆	Yes, also allows for use of JDBC's Extended SQL technology.
5	Support for stored procedures, with the transfer of in/out parameters.	☆☆☆	
6	Support for failure and reconnections to the DBMS. If the DBMS is stopped while there is an existing connection pool, the application will take care of restoring connections itself. At least a programming solution is possible.	☆☆☆	The application server can reconnect to the base after having lost the connection. Nonetheless, the pool connections with which the application code is provided before failure are no longer valid. When a new connection is requested, a valid connection is obtained. In practice, during a test carried out on a servlet application, failure did not trigger service discontinuation since each page re-requested the connections from the pool. However, in an application with a Java client, developed with WebLogic BeanBar components, service was not reestablished.
7	Support for multiple data sources within one page or graphical window. Possibility of creating standalone connections.	☆☆☆	By coding/
8	Extended management of connection pools (multi-instances and maximum number of connections).	☆☆☆	

Step 9: Language richness, openness

N°	Description	Rating	Comments
1	Creation of functional classes that support object polymorphism.	☆☆☆	
2	Support for all variable types (dates, multi-dimensional tables, dynamic tables, and structures).	☆☆☆	
3	Creation of features, with transfer of variables and recursive elements.	☆☆☆	
4	Variable management at session and application level.	☆☆☆	To implement a variable at application level, it suffices, for example, to declare it a servlet instance variable. Indeed, a single instance of each servlet is created and shared.
5	Presence of features/methods which define the length of time-out (at session level, dynamically, and at application level for all of the sessions).	☆☆☆	Via the servlet API
6	Presence of complete invalidation management features/methods (possibility to force the end of a session and launch a process when an event "log-out" is received).	☆☆☆	Via the servlet API
7	Server-side generation of print files using values coming from the data source.	☉	
8	Server-side generation of image files (GIF, standard formats) using values coming from the data source.	☉	
9	Capacity to handle File Upload (the tool must allow for the recovery of HTTP flow in a string for example).	☆	Low-level coding.
10	Presence of an object/relational mapping module.	☆	Limited CMP EJB system. Third-party products exist: Object People's Toplink and WaterShed's ROF.
11	Presence of a distributed objects system.	☆☆☆	Yes, RMI is well integrated in both WebLogic and Visual Café.
12	Possibility of interfacing with a standard distributed object model (DCOM, CORBA, and EJB).	☆	EJB/RMI, but be careful: some proprietary aspects remain, which invalidate any real interoperability. DCOM to a certain extent (development too cumbersome). WebLogic Server can be a CORBA client; it has been announced as a CORBA server in its next version via RMI use on IIOP. Furthermore, BEA is getting ready to implement a validation campaign for interoperability solutions (rooted in C++).
13	SMTP, POP3/IMAP4 and LDAP support.	☆	LDAP client via JNDI. Other protocols should be supported in the next version via the JavaMail framework.
14	The processing server is an HTTP and FTP client.	☆	HTTP client via a low-level API. Fastidious programming.
15	Possibility of proposing different levels of security for applications (SSL, SET, use of firewalls...).	☆☆☆	SET is not supported. SSL is integrated. It is possible to use a firewall thanks to the HTTP tunneling technology that WebLogic offers, which makes it possible to transfer T3 connection frames on the HTTP protocol. HTTP tunneling is supported between a client application and the server. For an applet, it is also possible to do tunneling on HTTPS (careful, this results in a very strong decrease in exchange-level performance).
16	Openness to ERP, with "business" modules adapted to ERP (SAP, Peoplesoft...).	☉	No. Nonetheless, we note openness to eLink; an EAI (Enterprise Application Integration) tool, which is edited by BEA and has ERP adapters.
17	Openness to TP monitors (Tuxedo, Encina...), integration middleware (MQ Series...).	☆☆☆	Openness to Tuxedo via a high-level, Jolt-based API. Openness to M3. CICS can be reached via a BEA product called "Java to Host". MQ Series can be reached via its own Java interfaces. It is also possible to use eLink (product sold separately).
18	Consideration of XML (XML application server, reuse of XML components...).	☉	

Step 10: Deployment

N°	Description	Rating	Comments
1	Multi-platform application server: Unix, Linux, and Windows NT.	☆☆	Server is present on a large number of platforms. Be careful, some platforms only support JDK 1.1.x.
2	Support for several types of interface with HTTP servers: CGI, ISAPI, and NSAPI.	☆	Only ISAPI and NSAPI. Integration with the Apache interface has been announced for the next version.
3	Possibility of deployment on other application/object servers (ASP, LiveWire, NCA cartridge, EJB, JSP, servlets...).	☆	EJB to a certain extent, JSP/Servlet with relatively little effort if the code does not use a specific framework.
4	Support for application-level and session-level failover.	☆☆	Redundant session in memory of another WebLogic server or in a database. Objects stored at session level must be able to be serialized. Clustering objects for EJB Home interfaces.
5	Possibility of modifying the application without stopping the services ("on the fly" modifications).	☆☆	Considerable evolution since version 4. Deployment, redeployment and on the fly deletion of EJBs, servlets, connection pooling, JSP. In some cases it is necessary to restart the server, for example, when modifying ACLs, configuring SSL or configuring system properties (number of threads, etc).
6	Possibility of spreading the server processes across several physical machines (distributed objects created with the tool, or relying on the application server), and of separating the HTTP server, the application server and the DBMS server.	☆	WebLogic Server still integrates its own HTTP server, which can possibly interact with a different server found on another machine.
7	Support for load balancing, with the possibility of configuring the balancing algorithm.	☆☆	

A study conducted by
TechMetrix Research
(Groupe SQLI)

Authors

Philippe Mougin
Franck Gonzales

Translation and Editing

Gina Faucher

Contributors

Nicolas Farges
Yannick Bessy

Publication date: March 2000

USA

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

Tel. : 1 781-270-7486
Fax : 1 781-270-7487

<http://www.techmetrix.com>
info@techmetrix.com

EUROPEAN HEADQUARTERS

TechMetrix Research
55/57 Rue Saint Roch
75001 PARIS
FRANCE

Tel. : (33) 01 44 55 40 00
Fax : (33) 01 44 55 40 01

<http://www.techmetrix.net>
info@techmetrix.net

SWITZERLAND

TechMetrix Research
World Trade Center
Avenue Gratta-Paille 2
CH-1000 LAUSANNE 30
PO Box 476

Tel. : (41) 021 641 10 65
Fax : (41) 021 641 13 10

<http://www.sqli.ch>

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.