Volume 4, Number 3
July/August 1999
The IBM VisualAge Generator Newsletter

VisualAge® Generator

A Powerful New Vision of Programming™

Special e-business issue

Contents

e-business

Special e-business Edition

by Barry Stevenson, VisualAge Generator Development Manager

VisualAge Generator and e-business

We're excited to bring you this special newsletter issue featuring articles on developing e-business solutions using VisualAge Generator technology.

A growing number of customers are beginning to utilize the strength of VisualAge Generator to deploy real enterprise e-business applications. These web solutions are having a significant impact on businesses and provide increased value for companies.

With the accelerating demand for new electronic commerce applications, VisualAge Generator provides unique application solutions capable of leveraging and integrating complex enterprise systems. These systems typically represent core I/T business processes.

What is e-business anyway?

Well, it's the transformation of key business processes through Internet technologies.

The Web is changing numerous aspects of society and nothing is changing as rapidly and significantly as the way businesses operate.

To get the most out of their core business systems, companies of all sizes are now using the Web to communicate with customers by connecting to legacy back-end data-systems that drive day-to-day business workloads. This is e-business—where the strength and reliability of traditional information technology meets the Internet.

The new Web + I/T model merges the simplicity and connectivity of the Internet with the core processes that are the foundation of a business. These new applications are interactive, transaction and data intensive, and they let people do business in more meaningful ways.

Enterprise e-business solutions: the ingredients of success

An e-business company is one that effectively manages constant and continual change. Successful e-business deployments do two things well:

1. Use end-to-end AD tools that lower the complexity of developing e-business apps by quickly extending the reach of web interfaces to core business systems.
2. Build on the existing investment of I/T systems and traditional application development skillsets.

Whether you're just taking your first steps or are already engaged in e-business systems development, exploiting the opportunities that e-business solutions provide requires progressive AD tools. This is where VisualAge Generator can help you succeed.

Cool stuff in this issue:

- Why VisualAge Generator is the right tool for building scalable e-business application solutions: added support from VisualAge for Java and the WebSphere product families
- VisualAge Generator plans for supporting e-business application development using the Enterprise JavaBeans standard
- An in-depth look at the unique e-business architecture used in the MDL project at the State of California
- A technique for moving a VisualAge Generator application to the Internet using Java Servlets ■

VisualAge® Generator

e-business

Building scalable e-business applications using VisualAge Generator*

by John Casey, VisualAge Generator Sales Support

"Few business applications grow as quickly - and unpredictably - as Internet applications. Usage patterns can change overnight, with spikes in demand occurring from inscrutable causes. Because it's so hard to gauge traffic, Web applications must be designed first and foremost with scalability in mind."[1] But how can you design scalable e-business applications, such that they can survive periods of rapid and unpredictable growth? VisualAge Generator, with VisualAge for Java and the WebSphere family of tools, makes designing and building scalable e-business solutions easier.

E-business applications: 3-tier architecture

E-business applications typically divide the processing load among tiers - presentation, logic, and data servers. This architecture for an e-business application is depicted in Figure 1. In a logical 3-tier structure, the web browser uses HTML to present the end-user interface. If Java applets are used, then they will also run on the web browser.

On tier 2, a web application server services the request from the web browser by using business logic in the form of Java servlets and special Java beans called connectors, which access tier 3. Tier 3 consists of the enterprise data and the transaction environment, which is the source for the information displayed via a web browser. The main business applications reside on tier 3.

Two typical solutions

One way businesses choose to react to increased demand on the web application server is to add physical servers until the demand is satisfied. Initially, this solution is simple and effective. The configuration remains constant and workload is distributed across the application servers. But as the number of web application servers grows, so do the costs to provide concurrency, maintenance, and support. These costs might eventually drive businesses to reduce the number of web application servers, add more powerful technology, and upgrade the system software.

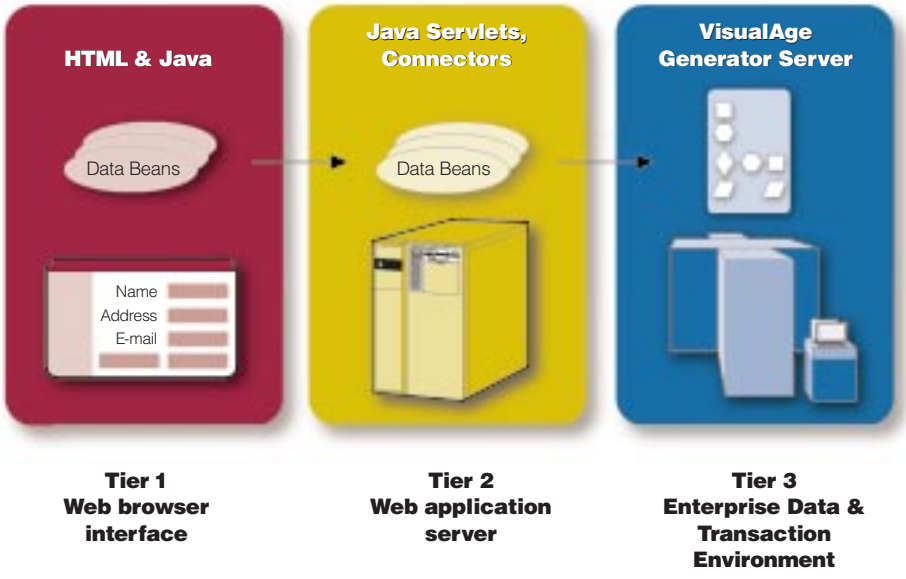


Figure 1. E-business 3 tier architecture

VisualAge Generator, with VisualAge for Java and the WebSphere family of tools, makes designing and building scalable e-business solutions easier.

Using more powerful technology, which is itself a second way to address the scalability issue, means introducing new system software to support the technology. With a change in supporting software, a rewrite of the business's web application will likely be required for this new environment. Again scalability comes at a cost when using traditional development languages because they must be adapted to the new technology.[2]

Design and build for scalability

Each of these scalability alternatives assumes that the web application server will perform all of the business logic. As we have learned from implementing client/server applications where fat clients ran all of the business logic to access remote database servers, application performance suffered from the larger amounts of data moving across the network. So, we partitioned the application's business logic to run on the server. This architecture provides significant benefits, such as reduced network traffic, which results in significantly better application response time. Today e-business applications can also benefit from this architecture—partitioning the business logic to run on the server where data is accessed.[3]

But partitioning the application is only one of the best practices for building an e-business application. To implement best practices, such as those listed in Figure 2, is to design and build a scalable e-business application. But implementing these practices is best accomplished if aided by the right tool. VisualAge Generator, which continues to be a leader in building scalable applications, is a tool that can aid you in implementing these practices.

1. Choose a development environment that lets the I/T department partition application logic and distribute processing across multiple platforms
2. Pick a middleware platform that addresses the application integration needs of most of the Web, database, legacy, and strategic business application platforms you have in place
3. Choose a middleware product that shields developers from low-level coding
4. Institute a testing program for new Web applications prior to deployment
5. Design the application logic first, then select the appropriate hardware infrastructure

Figure 2. Best practices for Web Scalability[4]

The right tool: VisualAge Generator

VisualAge Generator is a powerful, integrated development workbench used by programmers to fully define, test, build and deploy enterprise-level systems on a variety of platforms in record time. VisualAge Generator enables developers to build systems and programs for new computing technologies, such as e-business and network computing, as well as for traditional host support.

VisualAge Generator enables I/T managers to deliver scalable solutions, notably high-end e-business applications, that run on many different operating system environments, that leverage existing programs and data, and that use the existing skill of their staff to create these applications. With VisualAge Generator, I/T managers can rapidly deliver new function that works with their existing environments.

VisualAge Generator: supporting these practices

Building e-business applications with VisualAge Generator will automatically provide scalable servers that do not have to be rewritten when the technology changes. From the same 4GL source, VisualAge Generator can generate e-business application servers that will run on MVS, VSE, VM, AS/400, AIX, HP-UX, OS/2, Windows/NT, and in a future release, Solaris. In addition, by using VisualAge Generator, developers can easily create their web applications with additional scalability features. Let's examine how VisualAge Generator supports implementation of the best practices.

- VisualAge Generator enables application logic partitioning and distributing of processing across multiple platforms. Dynamic Application Partitioning, a VisualAge Generator feature, provides visual and numeric feedback to aid in optimal placement of partitioned logic.
- VisualAge Generator has built-in middleware that addresses the application integration complexity of most of the Web, database, legacy, and strategic business application platforms you have in place.
- VisualAge Generator's built-in middleware shields developers from low-level coding and the complexities of building web-based applications.
- VisualAge Generator enables testing prior to deployment of both the client and server for an e-business application on the developer's desktop.
- VisualAge Generator enables developers to design the application logic first. When the application is defined and tested, then you generate the applications for the appropriate hardware infrastructure. Retargeting the application for a different platform is easy, just regenerate.

Delivering networked solutions

Building e-business applications using VisualAge Generator is possible today through two of its features.

The VisualAge WebConnection feature

The VisualAge WebConnection feature, a part of the VisualAge Generator Developer V3 product, provides a set of parts that enable a programmer to visually define web pages without having to know languages like HTML or PERL. This feature, although not Java based, provides a much simpler approach to creating dynamic web pages than CGI programming. These web applications can dynamically build web pages and invoke server programs. It provides an elegant and easy way to include tier-3 transactions in a web-based system all from within the VisualAge Generator development environment.

VisualAge Generator middleware and JavaBean support

VisualAge Generator provides the Java programming required to connect a Java client to a VisualAge Generator server program. This support is provided through VisualAge Generator middleware and through Java beans created for a server program. For a programmer using VisualAge for Java, WebSphere Studio, or other Java tool, these VisualAge Generator created Java beans simplify and speed up the development of e-business applications where the web client needs to access a tier-3 server. During execution of the e-business application these Java beans and the VisualAge Generator

middleware automatically perform all the data marshaling and data conversion necessary to connect the Java client to the traditional back-end transactions.

Development scenario using VisualAge Generator

To build a web application using VisualAge Generator you need to use VisualAge Generator and VisualAge for Java or WebSphere Studio.

Using VisualAge Generator:

1. Develop and test the tier 3 transaction
2. Generate the tier 3 transaction for the server platform and compile into its executables
3. Generate the Java connector beans for the tier 3 transaction

Using VisualAge for Java or WebSphere Studio

1. Import the Java connector beans for the tier 3 transaction.
2. Create the client as a servlet, applet, or application and use the imported Java connector beans to define the tier 3 server access.
3. Test the client together with the tier 3 transaction all on the same workstation.
4. Publish or extend with your favorite HTML tools.

Working together: VisualAge Generator, VisualAge for Java, and WebSphere

The State of California Department of Health Services has recently implemented their Microbial Diseases Laboratory System using web browsers and Java servlets that run on IBM's WebSphere Application Server and connect to a new MVS back-end transaction using VisualAge Generator provided middleware (Figure 3). The Java servlets were developed with VisualAge for Java and the back-end

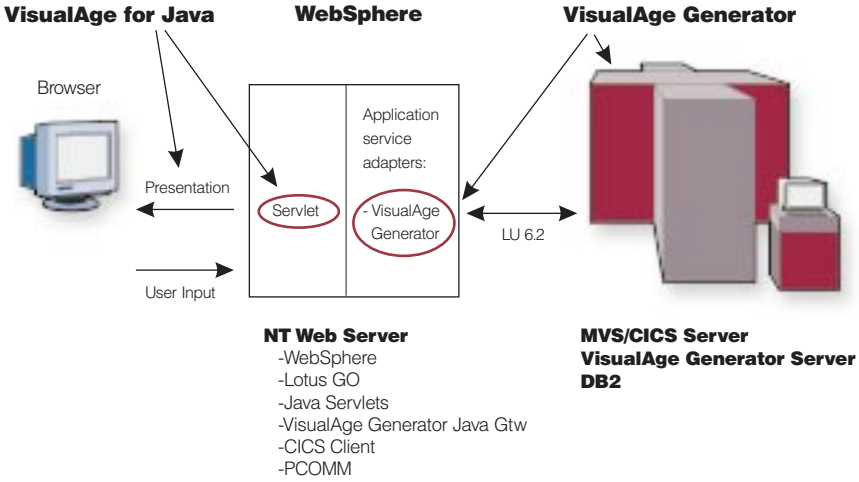


Figure 3. An integrated Web VAGen solution

server transaction was created with VisualAge Generator. With education and mentoring provided by the IBM Services team, the State of California developed, tested, and deployed their new e-business MDL application in just four months (see the article in this issue, "VisualAge Generator on the Web").

Future directions

In a future release, IBM plans to make all the productivity, scalability, and networking features of VisualAge Generator V3 available in the VisualAge for Java development environment. In addition, planned WebSphere integration enhancements will enable traditional host programmers to build e-business applications that run on a WebSphere application server. As the architecture of e-business solutions continues to evolve towards thin clients and component-based servers, the Enterprise JavaBean (EJB) open industry standards will become the key technology for deploying scalable and flexible solutions. VisualAge Generator plans to enable generation of Java-based servers that support the EJB standard (see the article in this issue, "VisualAge Generator Plans for e-Business and Enterprise Java Server").

E-Business applications must be designed and built with scalability in mind. VisualAge Generator, combined with VisualAge for Java and WebSphere, provides a powerful solution for building and running e-business applications: applications that meet the most stringent reliability, scalability, and availability requirements of today's—and tomorrow's—networked business world. ■

****This article discusses plans which are subject to change.***

[1] *'Scalable Web Apps' by David Baum, InformationWeek, February 2, 1992*

[2] *Java's portability can avoid this initially; but support for transaction rates that are higher than what Java can provide today and support for non-relational data access methods will require the use of traditional languages.*

[3] *This shifting of business logic to the data server should also have some additional benefits. Because each tier 2 application now executes less business logic and has less interactions with the network, the web application server will have additional capacity that can support a larger number of concurrent users.*

[4] *'Scalable Web Apps' by David Baum, InformationWeek, February 2, 1992*

VisualAge Generator on the Web— The State of California MDL Project

by Denise A. Hendriks, VisualAge Services

Introduction

Wouldn't you want to know if there was a disease outbreak in your community? Like many other states, the Department of Health Services for the State of California is responsible for protecting the public. They track disease outbreaks within the state and the systems they rely on track samples and specimens, testing, and test results. The Microbial Disease Laboratory (MDL) system is the mechanism the State of California Health Services uses for tracking microbial diseases. This article describes how a small team of developers moved this vitally important system, written in Natural and Adabas, from a proprietary implementation to an open, web-based solution and simultaneously brought it into Y2K compliance.

To provide an open, web-based solution, the deployed MDL application is a Java servlet-based system consisting of approximately 40 servlets, 100 MVS CICS transactions and 30 DB2 tables. This implementation provides a thin-client solution, leveraging their enterprise system. The new web-based implementation at State of California was developed using VisualAge Generator, VisualAge for Java Enterprise Edition, Domino Go Web Server, WebSphere Application Server 2.0, CICS Client, MVS CICS and DB2.

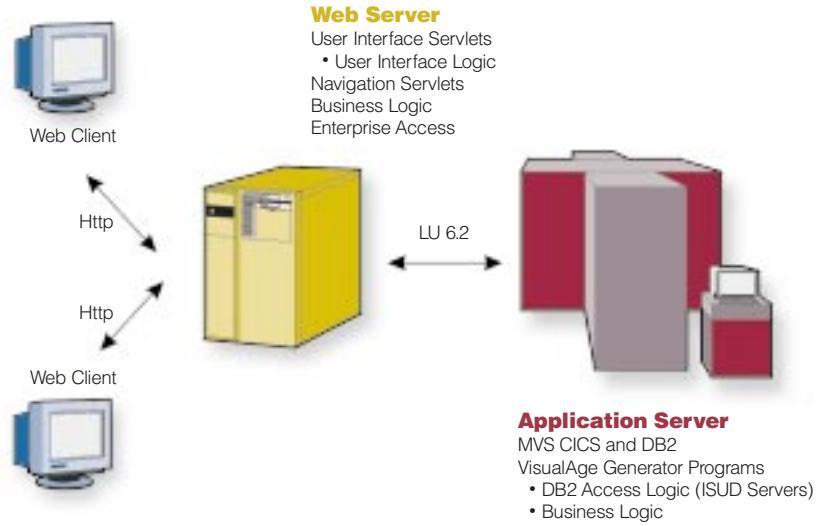


Figure 1. MDL Logical Implementation

The article "Health Dept. uses Java to Spot disease faster; leading-edge, server-side technology speeds access, enables detailed reports," published in the May17th issue of Computer World, describes the MDL project and the details of this leading-edge technology. Tim Sloane, an analyst at Boston-based Aberdeen Group Inc. is quoted in the article as saying, "There are a lot of companies that wish they were in this position now." He estimated that only 5% of companies have deployed such an application.

The MDL System: why change the original system?

The MDL system was originally a combination of software written in VSAM, Natural and Adabas. The State of California wanted to upgrade and enhance their existing system by:

- Making the system Y2K compliant
- Improving data collection quality and timeliness
- Providing a state-of-the-art application for users
- Increasing availability of MDL data to Public Health professionals through intranet and Internet access
- Minimizing their use of proprietary hardware and software
- Defining an extensible architecture that can be built on and reused for future projects.

Before the project began, the State of California moved the Natural/Adabas system, to a 3270-based application using VisualAge Generator. The VisualAge Generator application was targeted for MVS CICS and DB2. This application was implemented by a developer with no previous experience in VisualAge Generator, CICS or DB2. The VisualAge Generator 3270-based solution took approximately six months to complete. This application satisfied the Y2K requirement and removed the dependency on proprietary software.

The State of California Department of Health Services still wanted a state-of-

the-art system for their end users. As part of their long term strategy, they also wanted to make the system available to other users outside of the MDL testing labs via the World Wide Web (WWW). With this in mind, developers started the State of California MDL project in September of 1998 and completed it three months later in December.

The investment: people, skills, and education

The MDL application was developed by a team of six programmers from the State of California Department of Health services assisted by two technical mentors. Four of the six team members had no previous experience with VisualAge Generator. None of the team members had previous experience with Java or web technologies. With strong technical leadership and mentoring from myself (of IBM VisualAge Consulting Services) and Martin Rybczynski of Compete, Inc., the State of California team developed the MDL web-based application in approximately 12 weeks. The effort began with a week of Java education and a VisualAge Generator/ VisualAge for Java mentored workshop provided by Pat McCarthy of the IBM ITSO organization. As the project progressed, other informal education was provided including GUI programming techniques, ENVY training, as well as servlet and web concepts.

The overall architecture: a servlet-based system

The first decision made in implementing the MDL system on the WWW was to use a servlet-based architecture. We did not use an applet-based architecture because the invocation of an applet from a web page requires that the Java code for the applet and all its supporting code be downloaded to the web browser. This architecture has performance implications and all web clients accessing the applet must support the level of Java used to develop the applet. To free the system from compatibility restrictions and ensure good performance, we chose a servlet-based architecture to implement the MDL System on the Web.

Servlets provide the thinnest client, they run on the web server, and they may or may not produce user interfaces. Servlets produce user interface code in HTML, and only HTML is downloaded to the web client in a servlet-based architecture. The generated HTML is a small footprint and therefore downloads very quickly. All Java code in a servlet stays on the web server, removing Java compatibility issues across web clients.

The servlet-based MDL implementation consists of approximately 40 servlets and over 100 CICS/DB2 transactions. Figure 1 illustrates the logical pieces of the application and their deployment. The servlets running on the web server are responsible for delivering user interfaces to the web client and handling navigation through the system. Java beans are used for user interface validation, business logic, and enterprise data access.

The development: tools and technology

The MDL application was developed in VisualAge for Java and VisualAge Generator and deployed using WebSphere Application Server and Lotus Domino Go HTTP server. VisualAge for Java and VisualAge Generator work together to provide developers with a powerful end-to-end development platform. The developers are able to use the debugging facilities in both VisualAge for Java and VisualAge Generator. Figure 2 shows the software and configuration details of the development platform for the MDL application.

The team did all development and unit testing for the MDL project on NT based machines. The developer's machines had VisualAge Generator 3.1, VisualAge for Java Enterprise 2.0, and Internet Explorer installed for code development and unit testing. The data used during development resided on the enterprise system in DB2 on MVS CICS. Access to the data during development was provided by the DB2 Client Application Enabler V2.1.

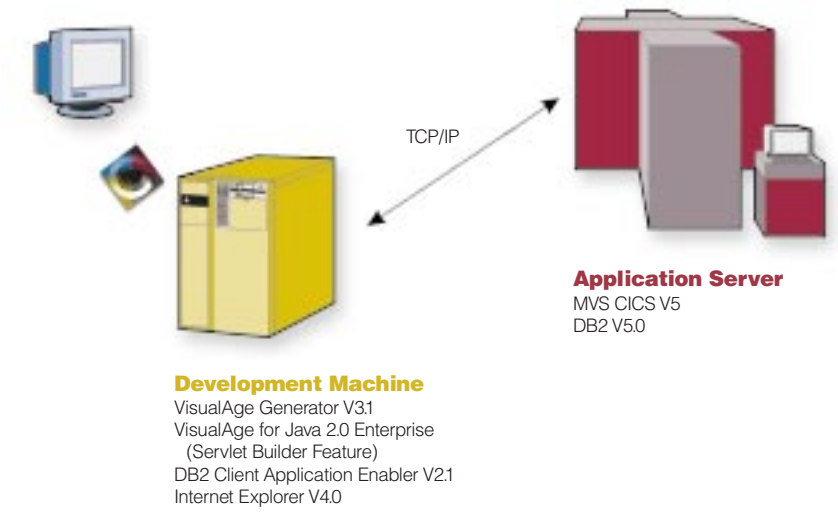


Figure 2. MDL Development Environment

Since the State of California had already migrated their system from Natural and Adabas to VisualAge Generator and DB2 before this project began, the important goal was to leverage as much of that legacy work as possible. By using the VisualAge Generator Java wrapper functionality, we were able to leverage the existing development investment and reuse all of the existing VisualAge Generator data access modules without modifying them.

VisualAge Generator

The VisualAge Generator Java wrapper functionality enables developers to generate Java wrappers for VisualAge Generator programs. These wrappers can be used from within Java to call VisualAge Generator server programs and pass data. This functionality allows developers to easily put a Java front end on VisualAge Generator applications, bringing the enterprise to the Web. For more information on the VisualAge Generator Java Wrapper functionality, see Martin Rybczynski's white paper, "Accessing the Enterprise from a Java Applet" in the May 1998 issue of VisualAge Magazine.

VisualAge for Java Enterprise Edition

The MDL Development team used the Servlet Builder feature of VisualAge for Java to construct the servlets of their application. The Servlet Builder feature provides a visual programming environment for building and testing servlets.

By using the VisualAge Generator Java Wrapper functionality, we were able to leverage the existing development investment and reuse all of the existing VisualAge Generator data access modules without modifying them.

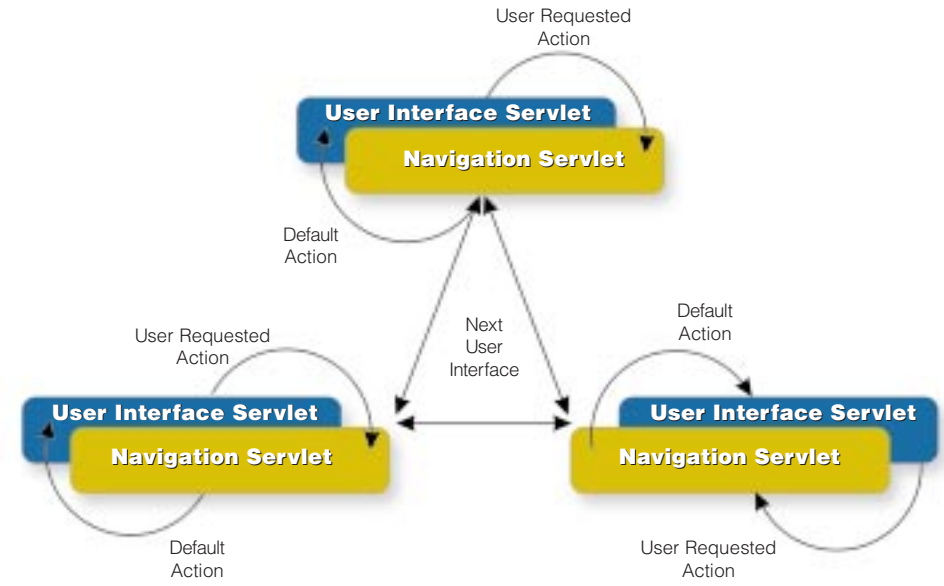


Figure 3. User Interface Servlets and Navigation Servlets

MDL on the web: the implementation

One of the main keys to the successful implementation of an enterprise application on the Web is the separation of the user interface from the business logic and enterprise data access. This provides not only the best performance but also the most flexible and scaleable solution. In deploying an enterprise application on the Web there are several areas that need to be addressed: the user interface, user interface validation, navigation, business logic, enterprise access, and message handling. This section examines how these issues are addressed in the State of California MDL application.

Basic processing

Servlets are coded in Java and deployed on the server, so all processing done in a servlet takes place on the web server. The servlet-based architecture is defined by a request/response interaction. A request is submitted to the web server, and the specified servlet produces a response. In the case of a servlet, the response is HTML presented to the user in a web browser on the client system. In the case of a non-visual servlet, the response is the invocation of another servlet. Upon the completion of this response, the servlet is not active until another request is received by that servlet. This is the underlying architecture in the MDL application.

The user interface

The MDL application is a series of non-visual servlets, called navigation servlets, and visual servlets, called user interface servlets. The MDL user interface servlets and navigation servlets were developed using the Servlet Builder feature of VisualAge for Java Enterprise Edition. The user interface servlet, implemented as a visual servlet, produces an HTML interface for the application. Each user interface servlet is coupled with a navigation servlet to control navigation through the application as shown in Figure 3.

The navigation servlets are non-visual servlets and therefore do not produce HTML. Their role is to process requests from the user interface servlets and

determine the next servlet to be invoked—either a user interface servlet or another navigation servlet. The combination of user interface servlets and navigation servlets form the user interface portion of the MDL application.

The basic flow through the MDL application is via the backbone of navigation servlets. Figure 4 details the flow through the navigation and user interface servlets. The basic processing steps are:

1. A request is submitted to the web server from the web client.
2. A navigation servlet is invoked to handle the submitted request.
3. If the request contains no user action, the navigation servlet invokes a user interface servlet.
   a. The navigation servlet's response is to produce HTML.

      The user sees HTML in the web browser and completes input of data.

   b. A request containing the user action is submitted to the web server.
4. If the request contains a user action, the navigation servlet processes the action.
   a. If the action was processed without error:

      A request with no user action is submitted to the next navigation servlet.

   b. If the an error occurred while processing the user action:

      A request containing no user action and the error message is submitted to the web server.

Business logic

The user interface servlet's primary role is to define and present the user interface for the application. The user interface in a servlet-based architecture is defined as HTML. The domain data for the application is accessed via data beans. Data beans are Java beans that contain data accessed from the enterprise system or data derived from business logic accessing the enterprise system.

If the navigation servlet is invoked by a user-requested action, such as pressing a button, the navigation servlet is given control and proceeds in processing the user requested action.

Processing the user requested action in the MDL system requires user interface data beans, validation beans, user action service beans, resource beans, and data beans as shown in Figure 5.

Processing of the user request action starts with validating the user input. Validating the user input in the MDL system is accomplished via validation beans. The role of the validation bean is to take the data submitted by the user via the user interface data bean and perform validation based on the desired action. In a servlet, the user-submitted data is available to the servlet through form data in the submitted request. The VisualAge for Java Servlet Builder feature, used to develop the user interface servlets, provides a form data bean to be used by the servlet. The form data bean is the user interface data bean for the MDL system. The form data bean is validated by a validation bean. The validation beans are non-visual beans written in Java and are specific to the user interface they validate.

One of the main keys to the successful implementation of an enterprise application on the Web is the separation of the user interface from the business logic and enterprise data access.

Once the user data has been validated by the validation bean, the navigation servlet processes the requested action by performing the necessary business logic in the action service bean. The business logic of the MDL application is written in Java as a part of the action service bean.

The Java code that executes the business logic in the action service bean uses resource beans. Like validation beans and action service beans, the resource beans are written in Java. The resource beans encapsulate the access to the enterprise system.

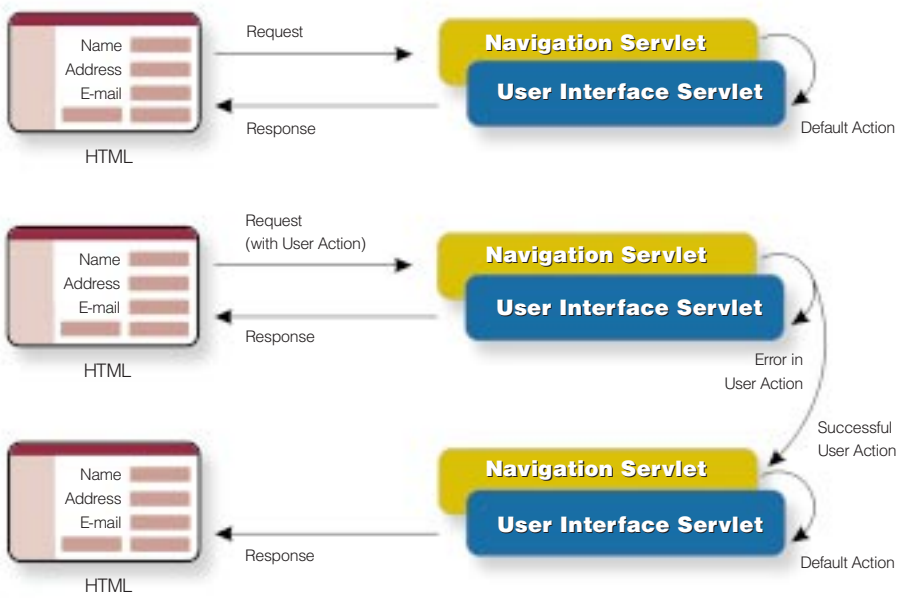


Figure 4. MDL Application Flow

They handle translation of user data (in this case from Unicode to EBCDIC), transport of user data to the enterprise system, invocation of the server program, and the return of data from the server program.

The server application used to access the enterprise data in the MDL application was developed using VisualAge Generator and targeted for MVS CICS and DB2. Access to these server programs by the resource beans makes use of generated Java wrappers to call the VisualAge Generator Server programs. The wrappers handle all communication with the enterprise system, both in transporting and translating the data submitted from the user interface and invoking the server program on the enterprise system.

Upon completion of the business logic via the resource beans, control returns to the navigation servlet, which determines the next action. In the case of successful completion of the requested action, control is transferred to the next navigation servlet, which invokes its user interface servlet. This presents the next portion of user interface to the end user and the process is repeated.

Message handling

If an error occurs in processing the user-requested action, an exception is thrown. Throwing an exception stops execution. The exception returns control to the navigation servlet so that the originating user interface servlet can be re-invoked with a message indicating the source of the error. The exception itself contains the error message text. The message text is obtained through a message bean. The message bean encapsulates a message table and the processing necessary to retrieve a message based on a key. The message table used for the MDL application is a hash table keyed by message number. When an error occurs while the user-requested action is being processed, message text is requested from the message bean. The message key and any inserts to the message are supplied to the message bean. The message bean responds to this request by returning a message string including inserts. The message is placed in the exception before it is thrown. The exception is caught by the navigation servlet and passed to the user interface servlet so it can be displayed to the user for an appropriate response.

Deployment—The run-time environment

Figure 6 shows details for the software used in the deployment of the MDL system. The MDL web server is Lotus Domino GO with the WebSphere Application Server Standard Edition. Lotus Domino GO provides the HTTP server and WebSphere Application Server Standard Edition provides the servlet engine necessary to run the user interface servlets and navigation servlets along with the validation, message, action service, and resource beans. The resource beans also make use of VisualAge Generator NT server for access to the enterprise server. The enterprise system is DB2 on an MVS CICS system.

Communication in the MDL web system is a combination of HTTP and LU6.2. The web clients communicate through HTTP to the web server. The MDL resource beans communicate through CICS Client over LU 6.2 to the MVS CICS enterprise system.

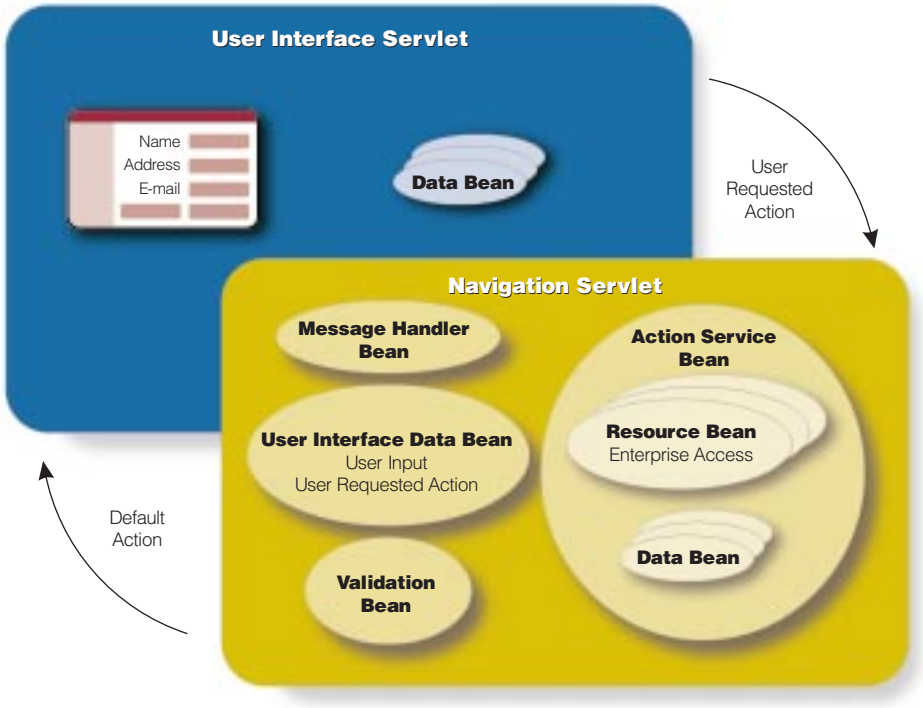


Figure 5. Navigation Servlet and User Interface Servlet Implementation

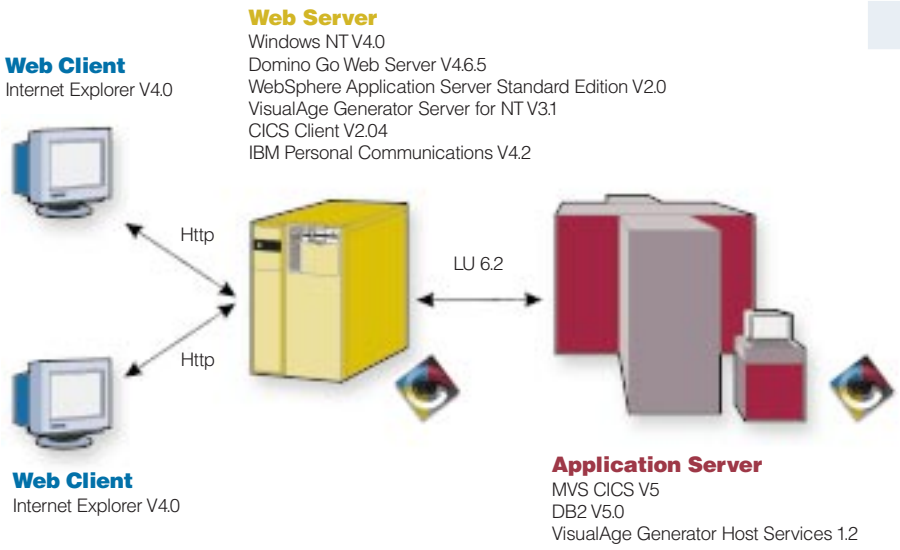


Figure 6. MDL Run-time System

Conclusion

The MDL project exemplifies the power of VisualAge Generator and VisualAge for Java in deploying a web-based enterprise application. In a very short period of time, a small team of developers brand new to the technology were able to successfully develop and deploy a key business application. The State of California MDL system has been successfully deployed and the end-user feedback is extremely positive with respect to both performance and usability.

A special thanks to the dedicated support from the RTP Development Lab, especially Larry Smith of the VisualAge for Java Servlet Builder team, Henry Koch and John Snyder of the VisualAge Generator Development team and Jeff Tennenbaum of the CICS Client support team. Also of invaluable help was Nik Teshima of the VisualAge for Java Support team in Toronto Development Lab and Andrew Cornwall of OTI. ■

e-business

VisualAge Generator Plans for e-business and Enterprise Java Server*

by Tim W. Wilson, VisualAge Generator Architect

As the topology of e-business solutions continues to evolve towards thin clients and component-based servers, the Enterprise JavaBeans (EJB) open industry standard will establish itself as the key technology for deploying scalable and flexible solutions.

VisualAge Generator is designed to enable enterprise developers to create application systems through very productive high level abstractions that simplify the programming complexity of a wide range of run-time platforms and transactional environments. As the EJB platforms become available, the need for high productivity tooling will extend to such new environments, and VisualAge Generator intends to support the emerging Enterprise Java Server (EJS) as new run-time environments within the context of web application development.

Several enhancements to the VisualAge Generator environment, such as the use of VisualAge for Java as the base development platform, the addition of high level constructs that simplify the development of thin client systems and the automatic generation of EJB, are planned to be developed by IBM in the near future. This will enable customers to continue to rapidly build and deploy e-business systems that meet the most stringent availability, scalability, and performance requirements of today's and tomorrow's networked business world.

This article was written by Tim Wilson, one of the lead architects of VisualAge Generator. Tim describes how VisualAge Generator is uniquely positioned to provide a highly productive solution for creating web-enabled systems. Although all these planned enhancements are very exciting, there is no need to wait. Read the other articles in this newsletter that describe how you can build e-business solutions with VisualAge Generator TODAY.

Introduction

VisualAge Generator has earned a reputation for being more than just a highly productive rapid application development (RAD) environment. Customers have embraced VisualAge Generator for the creation of scalable enterprise systems for a wide range of run-time platforms. But what happens when the paradigm shifts to web-enablement, and new metaphors for programming evolve? The answer is still VisualAge Generator, which allows our customers to leverage their skills and investments while keeping pace with the latest trends in technology. This article describes how VisualAge Generator plans to support the development of web applications, using a programming model that incorporates the emerging standard of Enterprise JavaBeans (EJB).

A solid foundation

VisualAge Generator has a long history of providing programmers with a highly productive environment for developing application systems that execute across a wide range of run-time platforms: most notably, CICS and IMS. There are several basic reasons why VisualAge Generator is an excellent tool for programmers developing for complex environments like these. VisualAge Generator provides:

- A high-level abstraction for writing psuedoconversational applications. In this type of application each display to a 3270 terminal is the end of a transaction. A complete application is written by stringing a set of these transactions together and caching the session state between them. VisualAge Generator simplifies this task by letting programmers work as if they were developing a simple single-threaded program. At run time, each "serving" of a text screen causes the program to terminate. However, the program state for this session is saved by VisualAge Generator, so when the user submits a request (PF key) back to the server, this program is reinvoked and picks up where it left off. All of this is part of the code that VisualAge Generator generates; the programmer does not need any special knowledge.
- A set of high-level and polymorphic I/O, Unit of Work (UOW) and remote procedure call (RPC) abstractions in the 4GL that hide the complexity of underlying system services.
- A simulated execution environment and a powerful interpretive debugger that is linked with VisualAge Generator's specification facilities. This environment facilitates rapid iteration between specification and verification, freeing the developers from the costly process of repeatedly generating and deploying applications to the real run-time environment until all logic errors have been uncovered and fixed.

Now, let's compare this to a conversational web application. An important point to make here is that a conversational web application looks, in form, like a CICS or IMS psuedoconversational application, i.e. the same processes are carried out in both environments:

- Users fill in forms and submit requests to the server.
- Server programs run business logic and serve the results to users.
- The processes are repeated until the program ends.

The logical question to ask, therefore, is can the 4GL programming model that worked so well for CICS also work for e-business applications that are conversational in nature? We think the answer is . . . a resounding yes!

Consider the skills

So, what skills do you need to build a robust e-business application? The flood of new technologies from IBM, Microsoft,

and other vendors are either client-centric in nature or simply make the mid-tier an extended client. So programmers envision having to learn new languages, programming models, and environments to make it all work together. Gaining these skills is time-consuming and expensive and it is these up-front costs that keep many customers from quickly adopting these new technologies. Many customers have huge install bases of IBM systems, but the programmers who write applications for these systems already have most of the critical skills necessary for building robust e-business applications: transactions, data access, security, recovery, etc.

So is now a good time to go to the Web? We think the answer is . . . and how!

In order to make e-business work, it's critical to make the cost of entry into this technology as low as possible. The easier IBM makes it for your company to use the skills and systems you already have, the lower the cost of entry. The support that VisualAge Generator and WebSphere Studio plan to provide will allow programmers to build e-business applications using their current skills and existing platforms with all the benefits of reliability and scalability that this implies. This environment also allows for the adoption of new platforms, such as Enterprise Java Server, without requiring developers to learn Java or OO programming. In addition, VisualAge Generator will be open, so that the adoption of OO, Java, etc. can be done at a pace set by you that makes sense for your business.

VisualAge Generator support for web application development

The basic idea for how VisualAge Generator will support web applications is to use the programming model that already exists for building CICS applications. In this model, the programmer codes what appears to be a single-threaded, structured 4GL program with a series of screens that take input data from and show output data to the user. When it is generated for CICS, this program becomes a psuedoconversational COBOL program that handles saving session state between screens, commit processing, and remembering the execution stack so that when this program is restarted as the result of a submit request, it can get back to where it left off. For a web application, the programmer codes the same application; however, instead of sending out a character-based screen to a terminal, the program will send and receive the defined I/O data to and from a web browser. The result is an e-business application, whose business logic can run on any of the platforms that VisualAge Generator targets (MVS, VSE, NT, OS/2, AIX, HP-UX, OS/400, VM, and more to come).

The new User Interface Record

A key difference in the way developers code their programs is in the definition of the screen I/O data. With the classic text user interface (TUI), programmers defined a 4GL construct called a Map: a definition of a text screen including high-level definitions of input and output fields. Developers can define input/output edit routines (system or user-defined) and various other useful attributes on these fields. The fields are accessible by the 4GL language as data items to be manipulated. There is another construct called a Record that contains a set of data items. Depending on the type of record created (Working Storage, SQL, DL/I Segment, etc.), developers can define certain higher-level attributes for each type. For instance, with an SQL record developers can define which database Table or View the record is associated with, the columns to which the data items correspond, the SQL data type for each item, etc. Another way to think of a Map is as a Record with an associated presentation (3270 screen layout). With these principles in mind, the VisualAge Generator team is planning to define a new Record type - User Interface Record. It will define the input/output data formats and the validation edits in much the same way they are defined for the fields in a Map, but there will be no definition of screen layout anywhere in the program.

Default Java Server Pages created for you

Once the UI Record is defined, VisualAge Generator will generate JavaBeans and Java Server Pages, which contain all the code defined by the programmer for input validation and output formatting but nothing about how this data is to be displayed. The definition of how data is to be displayed becomes the job of a web UI developer. An HTML page designer using WebSphere Studio tools enhances the default JSPs, which use the generated beans as the source of business data that should be displayed to the end user. A key point here is that UI developers don't have to code anything about the actual business processing. They are only responsible for displaying the relevant data and providing whatever services in the presentation are deemed necessary to make the program, as a whole, easily understood by the end user. This clear separation of concerns between the programmers who code the business logic and those who code user interfaces has many benefits for the process of developing and maintaining applications.

The definition of the presentation is a completely separate step to be carried out by someone conversant in the UI technology of choice (JSP, GUI, HTML, XML, etc). Clearly, this can be the same programmer. Still, separating the tasks makes the project easier for even a single programmer to manage.

Same program—any user interface

Why is it important to the business as a whole to separate business processing from presentation? Business processing includes these basic elements:

- Input/output data edits and validation.
- Navigation logic between steps of the business process; there can be other screens that have no intrinsic part to play in the steps of the business process, but instead act as aids to the current screen or gather extra information. These screens in our model would not "submit" back to the server. Navigation to these screens is seen as a natural part of the presentation.
- Data access and results processing.
- Error conditions

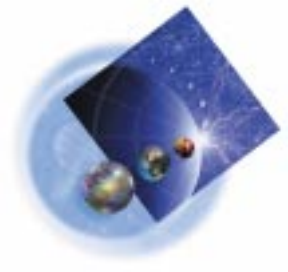
Most of the consequences of linking these elements into the presentation logic have to do with the problems of change management. For instance, if developers associate a validation rule with a particular entry field of a GUI, any change to this rule requires changes in the GUI. If navigation between the steps of the business process are embedded in the GUI code, then changes in the business process require changes to the GUIs. Clearly, if developers surface the layout of actual data to presentation logic, then again, changes to that data cause widespread havoc. So hard coding business process concerns into the presentation logic makes even small changes expensive.

VisualAge Generator solves most of these problems through the use of the UI Record for data validation and formatting, and the server program for doing all the navigation and data access logic. Changes to the server have no effect on the presentation code and vice-versa. Most changes to the UI Record would not cause changes to the presentation. However, changes such as adding/removing elements to/from the UI Record would cause changes, as expected.

This separation of business and presentation processing has another key benefit: it is possible to use different UI technologies with the same server, even at the same time. Programmers can choose a user interface format such as HTML without locking the whole development organization into everything that goes with these technologies. Developers have this kind of flexibility because most of the processing that is important to the business has been moved to the server platform (CICS, IMS, etc.), while only the presentation logic is left on the client.

A basic problem for customers who are coming from traditional IBM systems is that most of the AD tools that exist on the workstation come from the perspective of the client. They force business processing to be distributed to the clients, which then forces all of the developers writing business logic to learn these new tools and programming models. IBM is in a unique position to provide AD tools that put the highly reliable, scalable, well understood IBM systems into the driver's seat rather than under utilizing them as back-end servers that access legacy data. VisualAge Generator provides an environment that can achieve this goal.

Can the 4GL programming model that worked so well for CICS also work for e-business applications that are conversational in nature? We think the answer is . . . a resounding yes!



Bridging to Enterprise Java Server (EJS)

What we have described is an approach to using CICS, IMS, or other systems that VisualAge Generator targets to drive e-business applications. This is a very good thing for the near term. However, because our customers have invested in various types of systems, we need to provide them with an environment in which their applications can access many different systems. The need is very similar to the situation that existed before CICS or IMS. The emerging standard of Enterprise JavaBeans (and the implied server on which they exist - Enterprise Java Server) is the industry's answer to this problem. In fact, the short description for EJB/EJS is "CICS for Objects." Over time, more and more businesses will want to move into this new environment. A key question is how can VisualAge Generator most easily bring our customers forward into this new environment.

Here again, the skills required to build applications that run on an EJS are an issue to consider. This knowledge is in addition to what is already required to build an e-business application today. VisualAge Generator addresses this by taking advantage of the solution described previously; a web application still looks essentially the same running under EJS as it does running under CICS. In the future, VisualAge Generator

plans to allow a program, developed as described in previous sections, to be targeted at generation time to an EJS environment. Then any data accessed by the program is controlled as a resource by the Resource Manager(s) of the EJS. With this approach, programmers will be able to move towards these new systems and become productive very quickly.

What about OO?

So far, nothing has been said about OO. After all, EJB is "CICS for Objects." No matter how much OO developers use, there is still a "business process" that describes the steps and controls flow from screen to screen; Program is still a very good way to describe this process to a computer. Thus, the next step for VisualAge Generator is to allow an OO model first to be accessed by the Program code and then be created using the VisualAge Generator 4GL.

To this end, VisualAge Generator will add extensions to the 4GL that allow it to find and send messages to these objects. These objects can obviously be Enterprise Java Entity Beans. Enterprise Java Entity Beans allow the creation of high-level abstractions such as "Customer" or "Account" that hide the implementation details of data access and layout and group together the business processes relevant to that data.

A future release of VisualAge Generator will also enable developers to generate a session bean for any server program. When deployed in an EJB server, the session bean provides a call method that accepts parameters of the server program and calls the server program using the VisualAge Generator middleware. The session bean essentially acts as a gateway to the server program.

As your programmers become more OO knowledgeable and an EJB component market blossoms, they can take advantage of these technologies with VisualAge Generator, but do so at whatever pace makes sense for your business.

Conclusion

To put it simply, we believe that it is possible to allow your company to move to e-business through a route that is already secure and well-traveled. VisualAge Generator enables businesses to use their existing systems, skills, and programming models to build e-business applications. With the maps already drawn and the road already in place, your organization will be able to move to e-business at a quick pace. ■

This article discusses plans which are subject to change.

So is now a good time to go to the Web?

We think the answer is . . . and how!

Moving your application to the Internet using Java

by Reginaldo W. Barosa, Certified AD Specialist—IBM Brasil

Editor's note: In his article "Moving Your Application to the Internet" (April/May 1998 issue of the newsletter), Reginaldo showed us step-by-step how to web enable a VisualAge Generator application system using Smalltalk Web parts. To get a full understanding of the purpose of the system he discusses here, glance back at the previous article. If you don't have it handy, you can look at it in PDF on our web site at: http://www.software. ibm.com/ad/visgen/library/#newsletters.

One easy way to move VisualAge Generator application systems to the Internet is by using Java Servlets. In this article we will walk through building the same application system we built in a previous article, but this time we'll use Java Servlets instead of Smalltalk Web parts. It is a good idea to use the Smalltalk Web parts to prototype your solution. Then rewriting the presentation portion using VisualAge for Java is relatively simple.

For this example, we will use VisualAge for Java Enterprise Version 2 and VisualAge Generator Version 3.1. The application system will access a DB2 database using VisualAge for Java Data Base Access beans and then read the details using the JavaBeans and code generated by VisualAge Generator. We could also use Java applets, but for this example, we'll use Java Servlets.

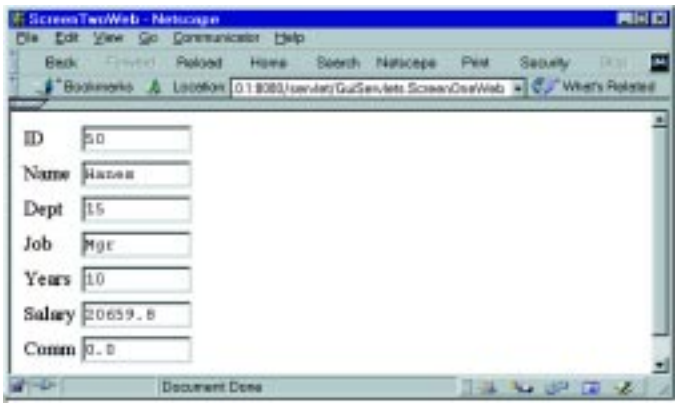
The application we will build has two windows:

- The first screen shown in the figure queries the STAFF table when the **List DB using Java beans** button is selected. It is built using VisualAge for Java and Database Access beans.

  Note that the query to the DB2 is done on the client side (that runs on the Web Server).

- On the first screen, we selected the id 50 and pressed the **Call VisualAge Generator v3.1** button.

The second screen queries the details of the selected ID. This second screen was built using VisualAge for Java using Servlet beans and the JavaBeans generated by VisualAge Generator. The VisualAge Generator program runs on the Server, where the query is done (this could be on MVS/CICS).



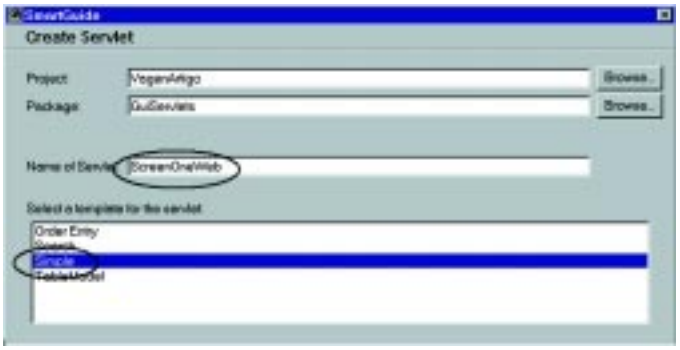
To develop these two windows:

1. Create the first screen using VisualAge for Java with Servlet Builder and Data Access Beans.
2. Create a VisualAge Generator program, prepare to pass the selected ID to the second screen and generate the JavaBeans to be used in VisualAge for Java.
3. Import into VisualAge for Java the classes provided by VisualAge Generator and the beans generated by VisualAge Generator.
4. Create the second screen using VisualAge for Java with Servlet Builder and the beans generated by VisualAge Generator.
5. Test the programs.

1 - Create the first screen using VisualAge for Java

The activities here are:

1. Using VisualAge for Java, create a class using the SmartGuide, options File/Quick Start/ Servlet/Create Servlet and name it **ScreenOneWeb**, as shown in the figure.



2. Using the category **Servlet**, add the beans: **HtmlImage**, **HtmlForm**, **1 HtmlPush Button**, **HtmlList** and **HtmlLineBreakers**.
   The settings of the image should point to the gif to be shown. In our example, code:

```
- localFileName =
c:\IBMVjava\ide\project_resources\SunJFCSwingSet
\Images\duke2.gif
- source = http://127.0.0.1/duke2.gif.
```

   Change the names and text of the buttons. Name them **pbListaDB** and **pbCallVG**. Change the name of the List to **efList**. See below:

   Save what you have done so far using options **Bean/Save Bean**.

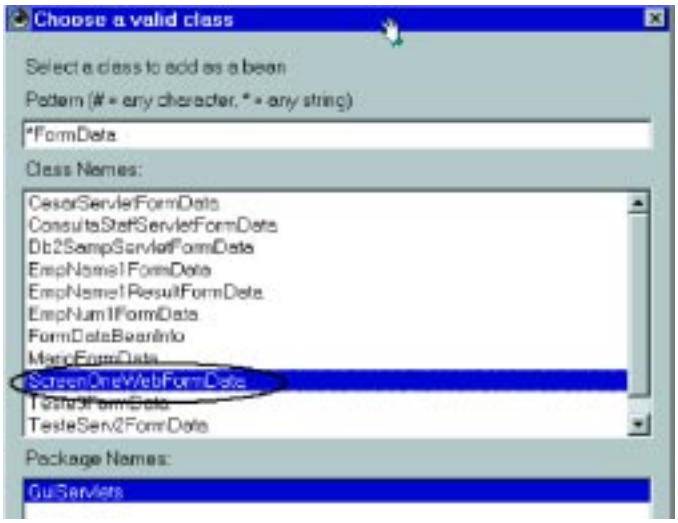
3. From the category **Database**, add the **Select** bean to the free form surface (white area). Alter its properties to query the STAFF table. In our example, the query will be:

```
SELECT STAFF.ID FROM STAFF ORDER BY STAFF.ID ASC
```

4. Since the efList needs an array of Strings to show the query and the Select does not have this property, we need to create a Java method that receives the query Select result and creates that array. This method is called **loadList** and is shown here:

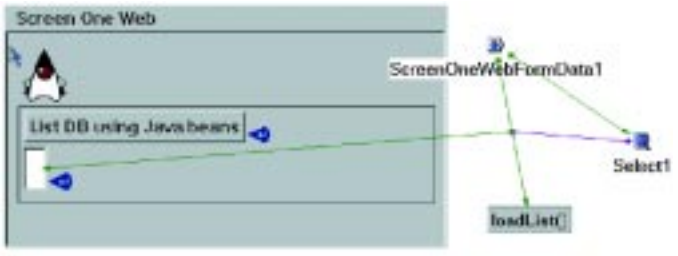
```
/* Receive the instance of the Select with the results.
   * create an array of Strings with the query result
     and load the column ID
   * return the array of Strings */
   public java.lang.String[]
     loadList(com.ibm.ivj.db.uibeans.Select queryResult) {
   int i; // loop counter
   int numberRows = queryResult.getNumRows();
     //# of rows
   String array[] = new String[numberRows];
     //create array
   for (i = 0; i < array.length; i++)//
     loop until read all rows
         {
         array[i] =queryResult.getColumnValueToString
         ("STAFF.ID");
         try {
                 getSelect1().nextRow();
         } catch (java.lang.Throwable ivjExc) {
                 handleException(ivjExc);
         }
     }
     return array;
}
```

5. From the category Servlet, add the bean **FormData** to the free form surface and select **ScreenOneWebFormData** as the class name:



6. Make the connections as listed in the table.

From bean/connect	Property/Event	To bean/connection	Method/Parameter
ScreeOneWebForm Data1	pbListaDBPressed	Select 1	execute()
ScreeONeWebForm Data1	pbListaDBPressed	free form surface (event to Code)	loadList (Select)
Select1	this	connection above	queryResult
connection above	normalResult	efList	items



For details and instructions on how to move the code to the production environment, see the redbook,* Unlimited Enterprise Access with Java and VisualAge Generator—SG24-5246, *published by ITSO.

7. Now we can test the application. Be sure you are using a browser that supports JDK 1.1. When you select the **List DB** button, the list field will have the IDs from the STAFF table.

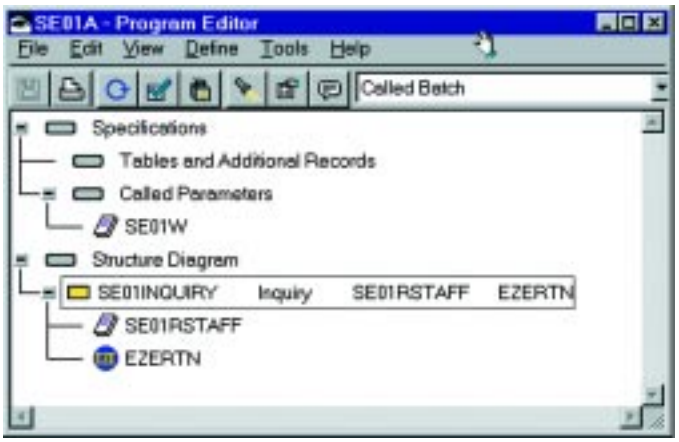
Note that we do not need to start a Web Server to test Servlets with VisualAge for Java, since VisualAge for Java generates the HTML and emulates a Web Server.

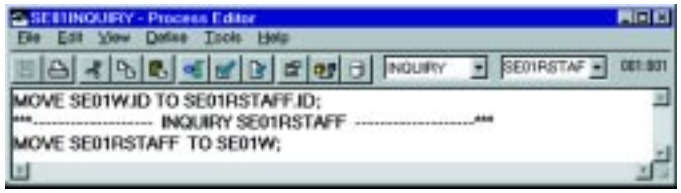
The next step is to select one ID (example 50) and select a second button (not implemented yet) that will call a VisualAge Generator program. This VisualAge Generator program will run on the server (could be an MVS server) and send the details of the selected ID.

2 - Create a VisualAge Generator program and generate JavaBeans.

Now let's play with the generator. Assume that the server will be an NT machine, the same machine where the client is running.

1. Create a VisualAge Generator program that will access the details of the ID received from the first screen. Name this program **SE01A**. It is a called program and the logic is straightforward. (See the graphical representation.)

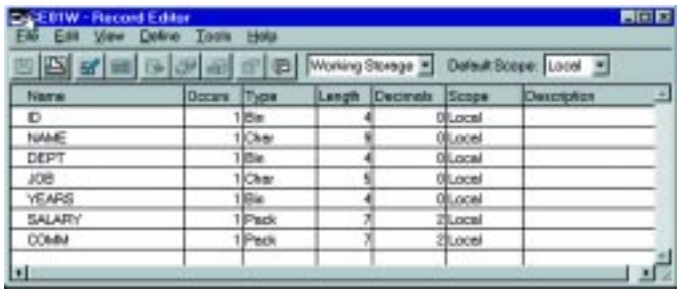




The SQL statements are:

```
SELECT
   ID, NAME, DEPT, JOB, YEARS, SALARY, COMM
   INTO
   :ID, :NAME, :DEPT, :JOB, :YEARS, :SALARY, :COMM
   FROM STAFF T1
   WHERE ID = :ID
```

This program will receive a record called **SE01W**. This record will be the linkage between the VisualAge for Java and the code generated by VisualAge Generator. The field ID will receive the parameter from the first screen we built (Screen One Web screen).



2. Test the execution of this called program. Use the test facility to be sure that the program is working and all data is returned in the SE01W working area.

We are now ready to generate the code. We will need to perform the generation twice. The first generation will create the server program (to run on MVS, for example) and the second generation will create the JavaBeans that we will later add to VisualAge for Java to build the second screen.

To better understand the next steps, refer to the redbook *Unlimited Enterprise Access with Java and VisualAge Generator (SG24-5246)* published by ITSO Santa Teresa.

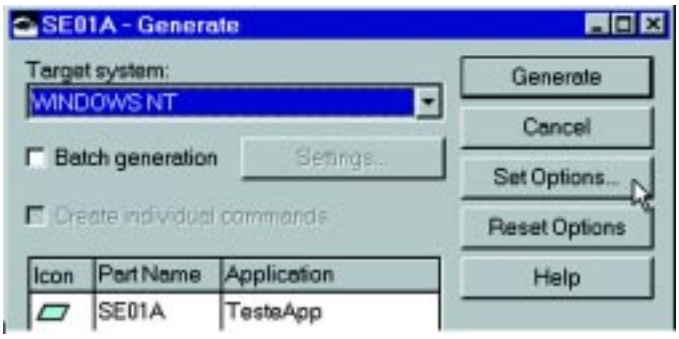
3. Create the Generation Options (JAVAVGGEN):

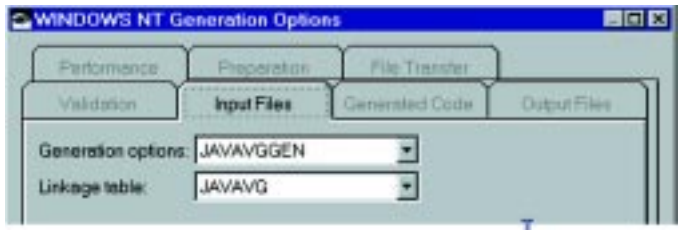
```
/system=winnt /GENOUT=c:\javavg /LINKAGE=JAVAVG.LKG
   /sqldb=sample

/dbuser=userid /dbpassword=password /prep
```

4. Create the Linkage Table (JAVAVG):

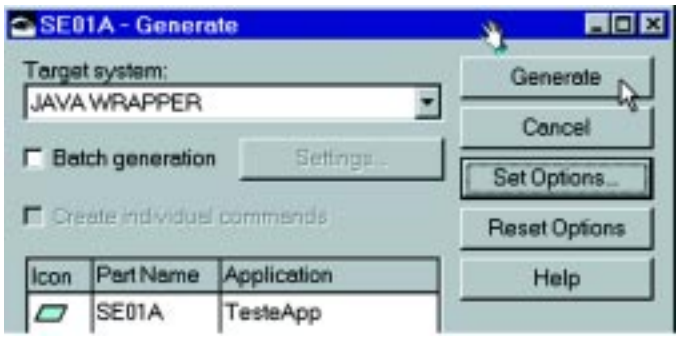
```
:calllink applname=* remotebind=runtime
   linktype=remote remotecomtype=tcpip
   luwcontrol=server location=localhost
   serverid=VAGenerator

contable=CSOI1252.
```

5. Generate the server program. Since I do not have MVS on hand to test, we will generate for NT, that is, we will generate and compile using C++ (you will need in this case the C++ compiler and the VisualAge Generator Server). See the generation options:





6. Generate the JavaBeans, using the same options as above:



Generation creates the files Se01a.java and Se01w.java. Those are the beans that we will import into VisualAge for Java.

7. Since the linkage table has the option *remotebind=runtime*, we need to create an external file using this option. We could just use cut/paste from the linkage table and create the file **`c:\javavg\JAVAVG.LKG`**. The file contains the linkage table defined before.

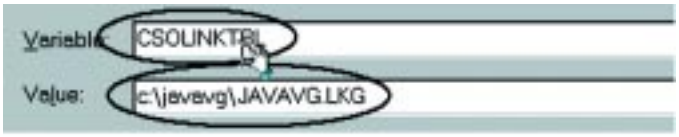
8. Enable TCP/IP client/server communication support and add an entry at the TCP/IP services file. Edit the file located at **`c:\winnt\system32\drivers\etc\services.tc`** and add the line below at any place:

```
VAGenerator 4200/tcp # VG Link to server
```

9. Create a command (like *startTCP.cmd*) that will start the TCP/IP server before executing the application. This server program is provided by VisualAge Generator. Example I used:

```
/* Start TCP/IP listener */

start csotcps
```

10. Using the NT System Properties, go to the Environment page and add the variable **CSOLINKTBL**, pointing to where the VisualAge Generator *linkage table* is present. That could be defined for the *user variable*, so you do not need to reboot the system.



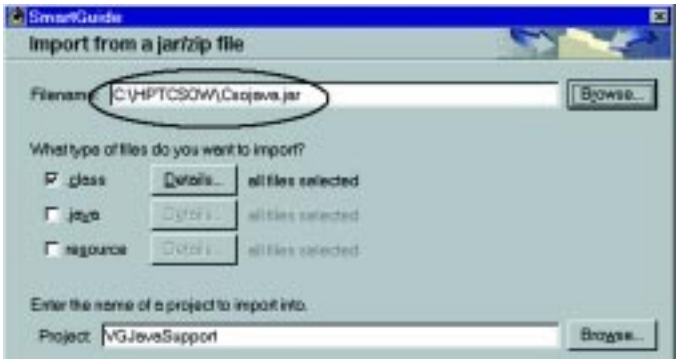
11. Using the same dialog, add to the *system variable* the directory where the *dll* generated by VisualAge Generator was created. This change will require you to reboot the system.

Reboot your system to NT.

3 - Import into VisualAge for Java the classes and beans

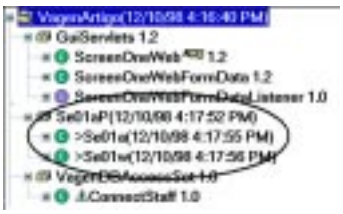
Now, before you start coding the second page screen, ensure that VisualAge for Java has all necessary classes and beans. To do this:

1. Import the **Csojava.jar** classes shipped with VisualAge Generator Developer and VisualAge Generator Server. Using VisualAge for Java, create a project (example **VGJavaSupport**) and import it.



2. Import the beans generated by VisualAge Generator:

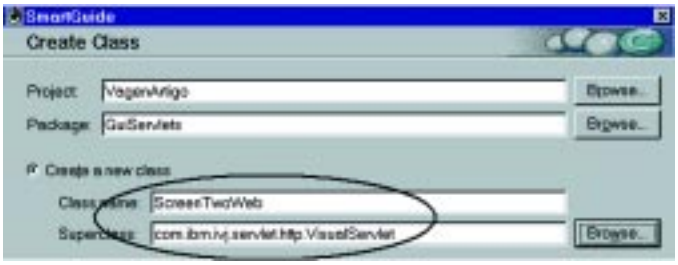
3. The classes available will be:



4 - Create the second screen using VisualAge for Java and beans generated by VisualAge Generator

Now we can create the second web page. We will name it **ScreenTwoWeb**.

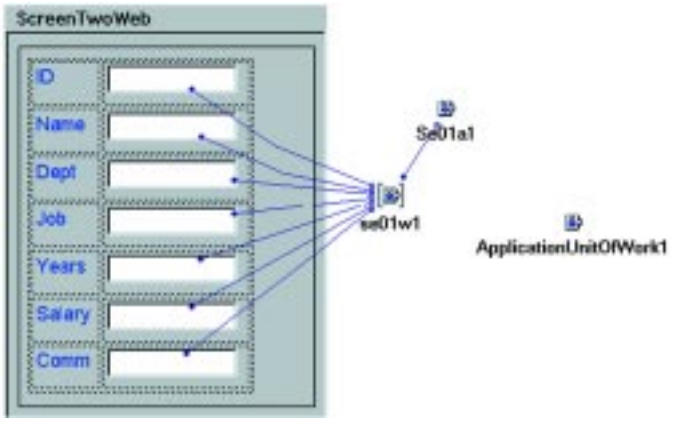
1. Create a new class **ScreenTwoWeb**, that has **VisualServlet** as its superclass:



2. Using the category **Servlet**, add the beans: **HtmlForm**, **HtmTable**, **HtmlText** and **HtmlEntryField**. Change them according to the figure:

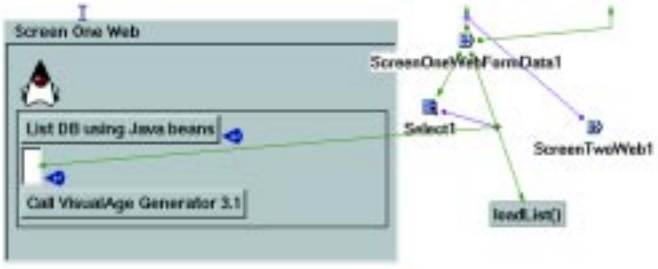
3. Add the classes **ApplicationUnitOfWork** and **Se01a** that we imported previously. Note that *Se01a* and *Se01w* are generated by VisualAge Generator and *ApplicationUnitOfWork* is provided by the product.

From the bean **Se01a**, tear-off the property **Se01w** and build the connections. The connections must be from each attribute of the *Se01w* to the property *string* of *HtmlEntryField*. Save the class to generate the code.



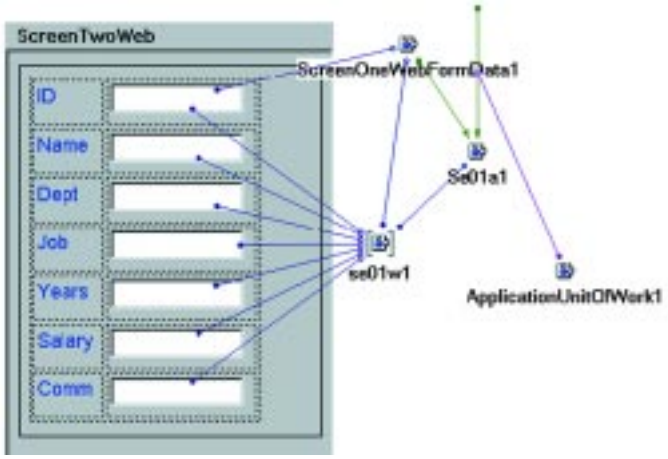
4. Edit and modify the first screen (*ScreenOneWeb*) by adding a new **HtmlPushButton** that will call the second screen and change this bean name to **pbCallVGPressed**. Save the beans. Add the second screen (class *ScreenTwoWeb*) to the free form surface. Build the connections as shown in the table.

#	From bean/connect	Property/Event	To bean/connection	Method/Parameter
1	ScreenOneWebForm Data1	pbCallVGPressed	free form surface	method transferToService Handler
2	ScreenTwoWeb1	this	connection 1 above	value
3	ScreenOneWebForm Data 1	pbCallVGPressed	free form surface	method isTransferring
4	connection 3 above		Set parameters.. to **true**	



5. Add the class **ScreenOneWebFormData** to the free form surface. This object will hold the selected *ID* in the first screen (*efList* of *ScreenOneWeb*). Build the connections as shown in the table.

#	From bean/connect	Property/Event	To bean/connection	Method/Parameter
1	ScreenOneWebForm Data1	efListSelected ItemString	HtmlEntryField1 (ID)	string
2	free form surface	initialize()	Se01a1	unitOfWork
3	ApplicationUnitOf Work1	this	connection 2 above	value
4	ScreenOneWebForm Data1	pbCallVGPressed	Se01a1	execute()
5	ScreenOneWebForm	efListSelected ItemString	Se01w1	id

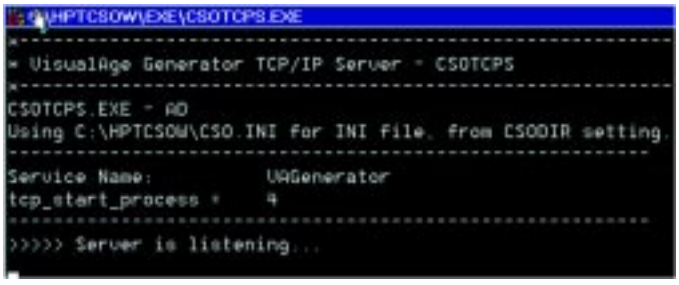


Save your work. We are now ready to test it.

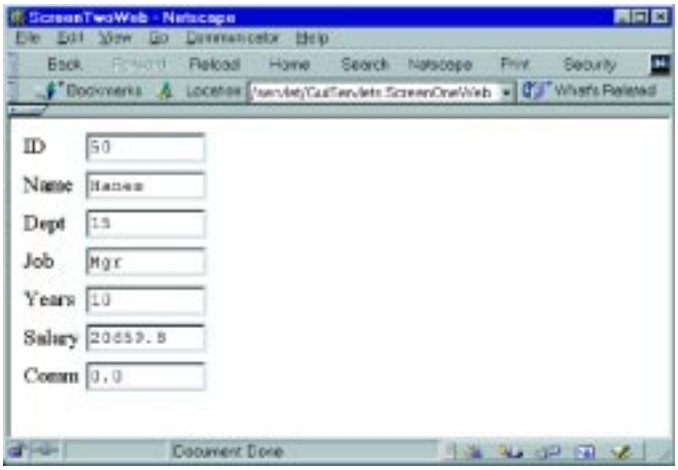
5 - Test the programs

Now we can test what you have done. You can test all the programs on the same machine using NT 4.0 and service pack 4. To do this:

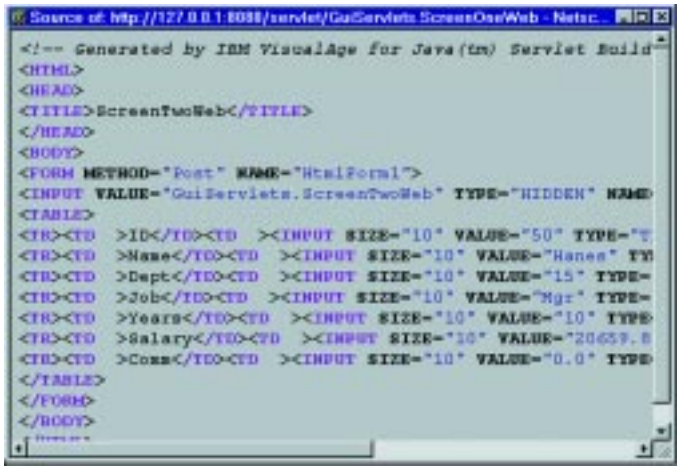
1. Start the TCP/IP listener executing c:\javavg\startTCP.cmd The result of this execution is shown in the figure.



2. Using VisualAge for Java, test the class **ScreenOneWeb**. Ensure that Class Path is assigned. Click the button **List DB using Java beans**, select one ID (like 50) and select the button **Call VisualAge Generator 3.1**. Your screen should look like this.



The HTML code generated by the Java Servlet looks like this.



Conclusion

It is easy to connect existing VisualAge Generator Server applications to Java Servlets. If you want to use Java Applets instead, the process is the same, but in this case the Java *RMI* is used. For details and instructions on how to move the code to the production environment, see the redbook *Unlimited Enterprise Access with Java and VisualAge Generator - SG24-5246*, published by *ITSO*. ■

Acronyms

3GL	third-generation language
4GL	fourth-generation language
AIX	Advanced Interactive Executive
API	Application Programming Interface
AS/400	Application System/400
CAE/2	Client Application Enabler/2
CASE	Computer-aided Software Engineering
CICS	Customer Information Control System
CICS OS2	Customer Information Control System Operating System/2
CPU	central processing unit
CSP	Cross System Product
DB2	Database 2
DBCS	double-byte character set
DBMS	database management system
DCE	distributed computing environment
DRDA	distributed relational database architecture
EMEA	Europe/Middle East/Africa
GUI	graphical user interface
IBM	International Business Machines
IMS	Information Management System
LAN	Local Area Network
MSL	member specifications library
MVS	Multiple Virtual Storage
NT	Microsoft Windows NT
OS/2	Operating System/2
OS/390	Operating System/390
OS/400	Operating System/400
RAD	rapid application development
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
VM	Virtual Machine
VSE	Virtual Storage Extended
WWW	World Wide Web

We hope that you've enjoyed this special edition on e-business. It provides an overview of e-business concepts, investigates tools and architecture and even gives you the nitty gritty of building a web application.

But there's a lot more out there! Visit these sites for the latest:

VisualAge Generator Training

Correction: The next VisualAge Generator class is scheduled for September 20 in Research Triangle Park. For more information see the IBMLink website: http://www.ibm.com/ibmlink. Companies interested in enrolling employees should send e-mail to: websphere_consulting@us.ibm.com.

http://www.ibm.com/software/websphere

http://www.ibm.com/software/vajava

http://www.ibm.com/software/vagen

http://www.ibm.com/e-business

http://www.ibm.com/redbooks

The VisualAge Generator Newsletter

This newsletter is published by the IBM Software Solutions Division, Research Triangle Park Development Laboratory. Letters to the editor are welcome. Please address correspondence to:

The VisualAge Generator Newsletter
Managing Editor
IBM Corporation
Dept. TF6B/062
P.O. Box 12195
3039 Cornwallis Road
RTP, NC 27709-2195
USA
FAX: (919) 254-0206

G242-0315-12