BUILDING A HIGH PERFORMANCE JAVA SERVLET E-BUSINESS APPLICATION USING JSERVER AND ORACLE WORKFLOW

Derek Mathieson, CERN
James Purvis, CERN

INTRODUCTION

It is estimated that business-to-business E-commerce will reach US$2.8 trillion by 2003. Attracted by dramatic returns on investment, E-commerce is a key component of nine out of ten companies' business plans. At CERN, in the heart of Europe, we believe that the first step and key to successful business-to-business E-Commerce is the implementation of organization-wide E-commerce. This is precisely what has been achieved and built using Java Servlet technology and Oracle Workflow.

SETTING THE SCENE

Imagine you are an employee of an organization and you need to purchase some goods. You open up an electronic catalogue, select your items and make the purchase. You are told that all of the goods will be delivered to your office within 24 hours, except one item that will be delivered to you direct from the Swiss supplier within 48 hours. While your browser is open you decide to also apply for the 'Oracle 8*i*' training course that is currently being run, and to register your holidays that you will be taking next month. All this is achieved in the matter of a few mouse-clicks. Minutes later you receive confirmation that a place has been reserved for you on the training course and that your holiday request has been approved by your supervisor's deputy (since your direct supervisor was absent today). Gone are the days of paper purchase requisitions and other forms circulating around the organization, now all you need to purchase, apply for training, obtain manpower, import/export goods, go on vacation and much more is a browser. All of these tools are electronically available on the Web. If you can paint this picture in your mind then you have just painted a picture of the current working environment and the E-commerce tools available in CERN.

'E-commerce is a **key** component of nine out of ten companies' business

CERN

CERN is the world's leading particle physics research laboratory. Some 6,500 scientists, half of the world's particle physicists, use CERN's facilities. CERN's business is pure science, exploring nature's most fundamental questions. The laboratory's tools, particle accelerators and detectors, are amongst the world's largest and most complex scientific instruments. CERN is currently working on the construction of a new 27-kilometre accelerator, the Large Hadron Collider (LHC) which will not complete until 2005. When CERN was established in the 1950s it set the standard for European Collaboration in science, with the LHC it is set to become the world's first truly global laboratory. However like many businesses in the current economic climate, CERN is expected to continue growing while staff numbers are planned to fall in the coming years, in essence achieving higher productivity with less resources. One way of achieving this is with the use of fast, efficient and streamlined organization-wide electronic workflow.

WORLD WIDE WORKFLOW

Amongst computer scientists, CERN is more often remembered as being the birthplace of the World Wide Web (Tim Berners-Lee & Robert Cailliau), than an organization where several Nobel Prizes of Physics have been achieved. In this context, it is important to remember that the World Wide Web was developed in order to meet the needs of CERN physicists, collaborating and exchanging information in a global physics community. Similarly, an organization-wide E-commerce system at CERN must be available and meet the requirements of the physicists and engineers working or collaborating with CERN, whether they are on the CERN site in Geneva or working from their home institutes in California, Moscow or Delhi. Currently, CERN collaborates with over 500 such institutes around

the world. To provide a concrete example, an engineer working on a design in Finland may order some material from a British supplier, to be delivered to CERN in Switzerland, but to be paid for by a US – European collaboration. This requires the purchase requisition to be initiated in Finland, authorized simultaneously in Europe and in the US, and the goods delivered in Switzerland. With the worldwide workflow system that has been implemented, this can easily be accomplished in less than 24 hours from goods request to delivery.

MULTI-LINGUAL

In the previous example, we omitted to mention that while the Finnish engineer working in Helsinki would more than likely look through the catalogue and complete his information in an English version of the system, when the request came to say France for authorization, the French counterpart would like to see the component details, the form and everything in their web browser in French. Thus the system must not only be worldwide, but it must be multi-lingual.

EDH

EDH (Electronic Document Handling) is the three-letter acronym given to CERN's internal E-business application. EDH currently has over 5000 active users with over 1000 different users a day. An electronic document is processed every 20 seconds. The system is multilingual, web-based using a Java Servlet architecture, and runs Oracle Workflow as the routing engine. Using EDH one may:

- Purchase any of 16,000 standardized items from the CERN Stores Catalogue.
- Make a purchase requisition that will be processed and transmitted to any of CERN's 20,000 suppliers.
- Make a purchase requisition for any new supplier in the world.
- Request for the importing or exporting of goods.
- Request to attend a course from our on-site training catalogue.
- Request to attend external training, conference or other event.
- Request and plan your vacations.
- Request overtime compensation.
- Request additional human resources for a project or activity.

The above procedures cover most of the large administrative areas resulting in electronic-forms only (i.e. no paper forms are used any longer for the above procedures). This equals approximately 100,000 electronic forms a year. Other smaller, low-volume procedures are also under consideration for integration into EDH.

EDH understands the organization's structure, roles and responsibilities and may hence use this knowledge to streamline procedures even further. For example, EDH will never send a document to somebody who is absent. If a person does not sign a document within a given timeframe, then the document may automatically be routed to a deputy or somebody else with equivalent responsibilities in that domain. The net result is a streamlining and standardization of procedures across the organization meaning that the average processing time of an E-document is less than a few hours compared with days if not weeks for the previous paper version.

E-COMMERCE: PAST, PRESENT AND FUTURE.

EDH was developed initially as a Client-Server system (running for PCs, Macintosh and X-windows) in 1991. Even with the client-server version, the benefits were almost instantaneous: reduced delays, streamlining of procedures and no duplicate data entry. However, many of our customers were off-site and could not easily run the client-server version from other countries. By 1995, a Web read-only client was available and it was possible to authorize requests using the Web anywhere in the World. By 1996, the first E-documents became available on the Web, and in 1998 the decision was taken to phase out the Client-Server version and replace it with a fully functional Web version built upon Servlets, Enterprise Java Beans and Oracle Workflow.

Currently much more functionality is available on the Web than was ever available on the client server version. For example it is possible to ask the Web version to 'probe' our materials management software (Baan) in order to

provide the user with estimated delivery delays based upon the product / supplier selected. Additionally certain purchase requisitions are processed 100% electronically, transmitted automatically to the supplier without the intervention of a Purchase Officer.

Despite the above achievements we see that there is still a significant step to be taken and this is to move from internal E-commerce to true business-to-business (B2B) E-commerce. To see what is the right direction in order to take this step, it is perhaps important for us to define what we think is B2B E-Commerce is ***not***.

For us, the following is ***not*** B2B E-commerce:

- Suppliers providing us with portals to type our purchase requests into their catalogue. Is an employee in your organization who orders a book from Amazon.com performing a B2B transaction? For us the answer is NO.
- Manual transferring of purchase requests from our system to a suppliers Web based system (we do this already)
- Sending of purchase requests by FAX or EMAIL (we do both of these already)
- Any unidirectional or bi-directional transfer of data which requires manual intervention at either point in order to transmit or receive the data (we do this already)
- Making available the purchase order to the suppliers via an https Web interface (we do this already)

We are looking at bi-directional, 100% automatic, error-free exchange of information between businesses or communities with no manual intervention in order to provide a net value-added benefit (e.g. reduced delivery delay, price reduction etc). For example, when a user makes a purchase requisition for a product at Oracle, this would be automatically transmitted to their purchasing system, and the delivery delay would automatically be returned to our system so that the user knows when the goods ordered will be delivered. This for us is the beginning of Business-to-Business E-commerce. EDH provides the first step in this direction, so let us examine this system and its requirements in more detail.

THE EDH WEB SERVER

When describing a Web Site as 'high performance' we need to quantify what we mean, the EDH Web Server processes around 50,000 hits on an average day. Although to some readers this may seem relatively low, the EDH server already suffers from most of the problems faced by sites with much higher hit rates. The techniques that we used to support our 5,000 on-line users are applicable to almost any web site.

SERVER PERFORMANCE

The Hit Rate of a Web Site is an important measure of the sites performance. Unfortunately like many similar measurements in the computing industry it is subject to some interpretation. A *hit* on a Web Site can mean many things to different people. On some sites the downloading of a file counts as a hit. That means that a web page with 10 graphic images on it would count as 11 'hits'. Ten images per page is a low figure for many web sites; at the time of writing the Oracle home page included 24 different images (many of which were repeated, making 159 in total). On our web site a typical EDH document page contains no more that 10 images, all of which are re-used throughout the site, meaning that a users browser will usually cache the image and not re-request it after the first time it is needed.

On the EDH web site we can make use of the fact that all of our users need to identify themselves to the web server before they can access the site. Currently, on an average day around 1000 users visit the EDH site.

SESSION TRACKING

As with most e-business web sites, the EDH application is interested in setting up a dialog with our clients, presenting them with a series of linked web forms until they have completed their task. This naturally implies that we need to store some information about the progress of their task on our server. Since the HTTP protocol is 'state-less' we have no way of determining if the user will ever complete their task. If we do not manage this correctly, we potentially have to store information about an infinite number of partially completed tasks. One of the most

important parts of any servlet environment is its ability to track a particular users interaction with the server; this is termed *session tracking*.

The first challenge in session tracking is overcoming the 'state-less' HTTP protocol. There are two common ways of achieving this.

1. URL Rewriting

With URL rewriting every URL on every page sent to the user is specially modified to contain a unique 'key' that identifies the session. You have probably already used sites like this yourself, they are easy to identify since the URL that you are using often appears as a long hexadecimal string.

This is the most reliable form of session tracking, as it does not make use of any special features (such as Cookies) on the client browser. The primary disadvantage of this approach, is the cost of re-writing every URL on each page.

2. Cookies

A Cookie is a small piece of information that is sent to your browser from a Web Server. This data is accepted by your browser, checked for length, expiration date, path and domain then saved. When a user clicks and visits a page the browser checks the URL of the page against its cookie database, if it has a cookie that matches the domain and path of the link it will send the cookie to the server along with the request for the page. When used for session tracking the cookie is set to contain a unique ID number that identified the users session on the server. Since the same cookie is sent every time they access a page, it is relatively easy for the web server software to identify the user.

Memory Usage

Critical to any multi-user application is the issue of memory usage. Many developers feel that by introducing an automatic *garbage collector* (see sidebar) into the Java Virtual Machine, the designers of Java solved all of Java applications memory management problems. In some respects this is true, since many of the old 'memory-leaks' common in older applications written in C or C++ have now been eliminated, although Java has replaced these with its own type of memory leak.

One of the most common memory leaks in Java is caused by the use of Java's collection classes (Vector, ArrayList, Hashtable, HashMap, etc.). When collections have a lifetime that is the same as the class (static members), it is vital that it is possible for the collection to be emptied at some point. It was a common problem in early Java Swing applications that some of its internal collections were impossible to clear, and could grow without bound in long running applications. In fact Oracle's own thin (all Java) JDBC driver exhibits a similar (very slow) memory leak. An excellent tool that we have used to highlight memory consumption problems in Java is a product called OptimizeIt from Intuitive Systems, Inc. (http://www.optimizeit.com).

Automatic Garbage Collector

The automatic garbage collector in Java is implemented as a separate low-priority *thread*[†]. It is responsible for recycling (i.e. marking it as available) memory within the JVM. Memory becomes suitable for recycling, as soon as no object has a reference to it. Normally when a method returns to its caller, all of the temporary objects that it created are ready to be recycled. Unfortunately although the memory occupied by these objects is no longer needed, it cannot be reused until after the garbage collector has marked it as available.

In busy JVM (serving many users simultaneously) the garbage collector, being a low-priority thread, runs very infrequently, and in fact it can happen that the JVM can appear to run out of memory when in fact there is still a lot of memory that could be reused.

Performance

In the final analysis, the performance of a web site (in terms of the users experience) is the most important measure of a sites future. If a visitor to the site experiences poor response times, loss of information, or even worse loss of service during the completion of a transaction,

[†] A *Thread* another name for a lightweight process. In Java many threads run at the same time, and in the same memory space. Your application runs as one or more threads, along with other specialized system *threads* that are responsible for managing the resources on you Java Virtual Machine

they are unlikely to feel inclined to visit again. In our application we have the relative luxury of having a user base that has no alternative other than to use our site, but that in no way makes us complacent. In fact we are very aware that the service we provide needs to be as high a quality as possible.

There are many factors that can affect the users experience of a web site, some of which are out with our control, such as the quality of the network between our network and the users (CERN operates one of Europe's Internet eXchange Points meaning it is rare that CERN's connectivity is a problem).

What we can control, however, is the quality of the hardware and software that we use to implement the application. For our site we use a Sun Enterprise 450 running SunOS 5.6 with four 300Mhz CPUs, 2Gb of physical memory and 165Gb of disk. Network connectivity is provided by a 100-Mb Fast Ethernet connection.

Our web server is a Netscape Enterprise Server version 3.5.1, accesses to the EDH Servlets are load balanced across 20 Java Virtual Machines each with an allocation of 64Mb of RAM (more on this later).

WHY JAVA?

In the past EDH had been built using a variety of technologies. Many of the programmers that worked on the project were students or postgraduates with limited experience and short contracts. The result was a system made by many different people and using many different technologies. At one point EDH had components written in C, C++, Java, PL/SQL, PRO*C, Python, Perl, and prolog (we don't know why we had a tendency towards languages beginning with the letter P!). In the end it left us with a system that was almost impossible to maintain. Two years ago we began a project of re-engineering the EDH application. An important goal in this exercise was to rationalize the number of technologies down to one.

Java's background of being a safe language for running inside clients web browsers also makes it very suitable for writing very stable applications within the server environment. The architecture of the Java Virtual Machine is such that it is continually checking the operation performed by the application, and prevents errors in one part of the system from affecting others.

Sun identifies 'Write once, run anywhere' as the 'core value proposition' of the Java platform. What this means is that is possible for us to develop and test our application using inexpensive PCs, and when it is ready, the *binary* can be copied to and run on our Sun Server without modification.

Java's portability and stability comes with a cost however, Java programs run somewhat slower than programs written in more traditional languages (such as C or C++). Although, since EDH is an interactive application, it spends most of its time waiting for user input, so speed is less of a concern.

Finally, we chose Java because it is an object oriented language, thus enabling us to make use of modern design techniques, and developers are between 30% and 60% more productive compared with C or C++. Since October 1998 there have been around a quarter of a million lines of Java Code written for the EDH application, by an average of around 5 full-time programmers.

CODING STANDARDS

With sufficient financial resources it is possible to build very high availability hardware systems, with 'hot-swappable' components, built in redundancy, and transparent 'fail-over', but all of this would be pointless if the underlying application software was of poor quality.

Along with the adoption of a single language for the implementation of EDH, we also adopted strict coding standards, with design and code reviews. We based most of our standards on the Sun Java coding standards, although we have also borrowed some ideas from C++ styles. Although introducing these practices took some effort in the beginning, we can now easily see the improved quality and reliability of our system. We are firm believers that these practices are essential to running a service such as the EDH application, owing to its large user base, and their demand for very high availability.

JAVA VIRTUAL MACHINES

The Java language achieves its hardware independence by operating on a standardized, virtual computer system. All *real* computer systems must *emulate* this virtual computer system using a program called the Java Virtual machine.

The choice of Java Virtual machine can have a large influence on the overall performance and stability of a Java web site. Currently there are three commercial Java Virtual Machine implementations for Sun Hardware.

SUN JVM (CLASSICAL)

The original Java Virtual Machine provided by Sun operated around a simple interpreter that translated each Java instruction into equivalent instructions for the local hardware, each time they were encountered. Java threads were also simulated in software (so called *green threads*) in order to simplify the design. Although this technique worked well, the resulting performance was quite poor. More recent implementations from Sun make use of *native* operating system threads (allowing the JVM to exploit multiple CPU hardware platforms) and are based around *Just-In-Time* (JIT) compiler technology. With a JIT compiler, the JVM software is able to compile large groups of Java instructions into blocks of equivalent native instructions. The JIT compiler runs within the JVM at the same time as the application. Although the performance increase can be dramatic (around a factor of 4), a JIT compiler is limited by the fact that it has to complete the translation very quickly, and can perform very few optimizations of the resulting native code. If a JIT compiler took a long time to compile a part of the application, the JVM would effectively pause.

Another disadvantage of JIT technology is the fact that it has no way of knowing how many times a particular piece of recompiled code will run. The JIT compiler may take some time compiling a section of code that will run only once, effectively wasting time and memory. Also, a JIT compiler has to be very conservative about what it *can* compile. A Java system is a very dynamic environment where new classes can be loaded at any time, possibly modifying the behavior of existing classes, in this case if the JIT had compiled the code into native code, the result of the newly loaded class may not be seen.

It was because of these (and other reasons) that Sun developed the HotSpot JVM.

SUN HOTSPOT PERFORMANCE ENGINE

The HotSpot Performance Engine (now known as the HotSpot Server VM) was designed to maximize the performance of Java code. It achieves this by addressing some of the limitations of the classical, JIT-based JVM:

- ### *DYNAMIC PROFILING*

  The name *HotSpot* refers to the *dynamic profiling* performed by the JVM, one of the most significant improvements to the design of a JVM. With dynamic profiling, the JVM monitors how often areas of the application code run and then when it has identified a hotspot (a frequently used area), it will then expend effort in natively compiling the Java bytecode. The HotSpot JVM also supports dynamic 'de-optimization' a technique that allows the native compiler to be less conservative when compiling by automatically removing compiled code if a newly loaded class modifies the behavior of a class.

- ### *IMPROVED GARBAGE COLLECTION*

  The HotSpot JVM employs a completely new *generational* garbage collection algorithm. This identifies and separately manages short-lived objects (which typically live for the life of one call) from long-lived objects, resulting in significantly better memory usage.

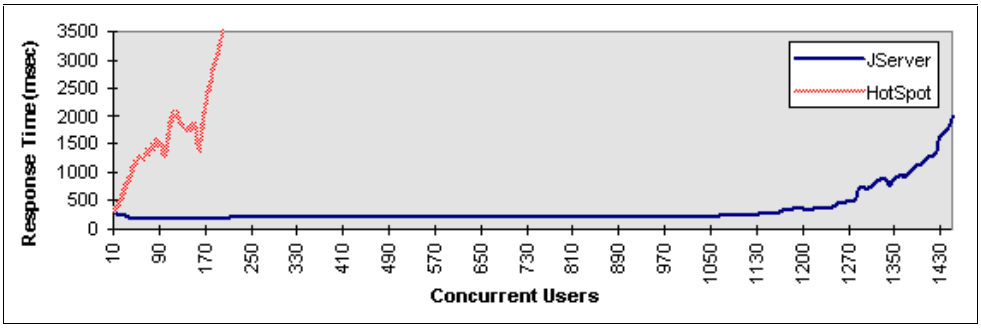
- ### *FAST SYNCHRONIZATION*

  In multi-threaded Java applications coordinating access to shared resources using the `synchronized` keyword is typically a major bottleneck. In the HotSpot JVM the common case (where the resource is not contended) is now much faster.

Unfortunately, the actual speed improvement when using the HotSpot JVM is highly dependant on your application code, varying from no improvement at all over a classic JIT up to as much as a factor of 15. When we first tried HotSpot 1.0 with EDH we actually found it to be less performant than a classical JVM.

ORACLE JSERVER

Oracle JServer is Oracle's own implementation of the Java Virtual Machine specification that runs within an Oracle 8*i* database. By running *within* the database Java applications can leverage all of the scalability and reliability of the database server. It is the only JVM implementation that uses the scalability of a RDBMS in order to be able to support thousands of concurrent users. The graph below (taken from Oracle's site) demonstrates the scalability of JServer in comparison to Sun's HotSpot 1.0.



Scalability of Oracles JServer: (test platform: Sun Enterprise 450 with 4 CPUs)

The developers of Oracles JServer have been able to make significant improvements compared with traditional JVM designs, for example:

> 'By running *within* the database Java applications can leverage all of the **scalability and reliability** of the database server'

- *JACCELERATOR*[†]

  An innovative feature of JServer is its use of native compilation. In a server environment the portability offered by interpreted Java bytecode is not necessary. Server applications are typically longer lived than those of a client, which means it is worthwhile investing more CPU cycles to produce highly optimized native code than could be done by JIT (or HotSpot) compilers. Oracle calls this 'way-ahead-of-time-compilation' and it can significantly improve server performance.

- *GENERATIONAL SCAVENGING GARBAGE COLLECTOR*

  Similar to HotSpot JServer also uses an advanced garbage collection strategy (built upon Oracle existing Multithreaded Server technology) that greatly improves the memory utilization of Java Applications.

- *SCALABILITY*

  Oracle has made significant advances in reducing the amount of memory required for a typical, stateful Java session. On JServer each session appears to be running on its own private JVM, yet the memory overhead is typically under 50kb.

- *DATABASE INTERGRATION*

  Since most e-Business web sites typically involve extensive use of relational databases, it is vital that Java applications can efficiently access the database. By integrating the JVM within the database the communications overhead imposed by running in an external JVM is eliminated. Oracle has also provided a special 'Server' JDBC driver that can be used to directly access the database's SQL engine through the same high-performance APIs that were previously only available to PL/SQL stored procedures.

---

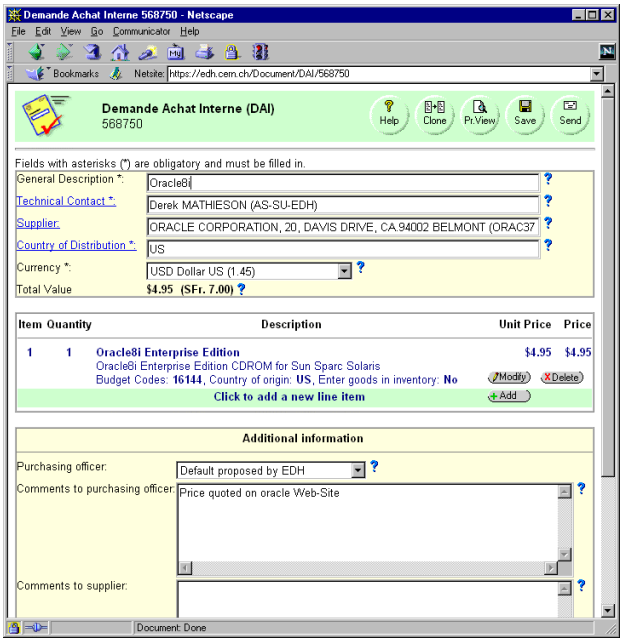
† JAccelerator has been used to natively compile all of the standard Java classes provided with the JVM, but it only becomes available for application developers to compile their own classes in Oracle 8i Release 3 (version 8.1.7).

JServer is a relatively new product, yet we have been using it successfully in production with EDH for the past two years, and have found it to be a very stable implementation.

ARCHITECTURE

USER INTERFACE

The EDH User interface is based around simple HTML forms. Due to our large and varied user base, we aimed to support a very low minimum standard. Initially we only required our users to have Netscape Navigator 3.0 or Internet Explorer 3.0. This meant that we have to be very conservative in the use of HTML and JavaScript. Recently we have raised the minimum standard to Netscape Navigator 4.0 or Internet Explorer 4.0, which has allowed us to make use of cascading style sheets.



An Example of the EDH User Interface

SERVLETS

All of the form processing in EDH is performed using Java Servlets. The choice of Java technology had already been made (see earlier section on Why Java?). We chose Java Servlets because of their simplicity, stability, and simple session tracking model.

COMMON BUSINESS OBJECTS

A core feature of the EDH application architecture was the development of the Common Business Objects (CBOs). The objects were designed to represent common types of object in out business domain, such as People, Account Codes, Purchase Orders, and Delivery Addresses. These Objects were modeled on the Enterprise Java Beans (EJB) architecture, although at the time that we began the project the EJB standard had only recently been released and there were no stable implementations available that we could use. Instead we created our own simple component model, using the same principals as EJB. We intend, once stable and proven implementations exist, to migrate our components into true EJB's in order to benefit from the support of commercial design and development tools.

COMMON INPUT OBJECTS

Coupled to most of our CBO's are what we term Common Input Objects (CIO's). These objects implement the user interface for a particular CBO. Currently all of our CIO's implement their user interface as an HTML form component, although the design is abstracted in such a way as to make it possible for us to replace the user interface at a later stage with an alternative technology (for example Java Applets) if necessary.

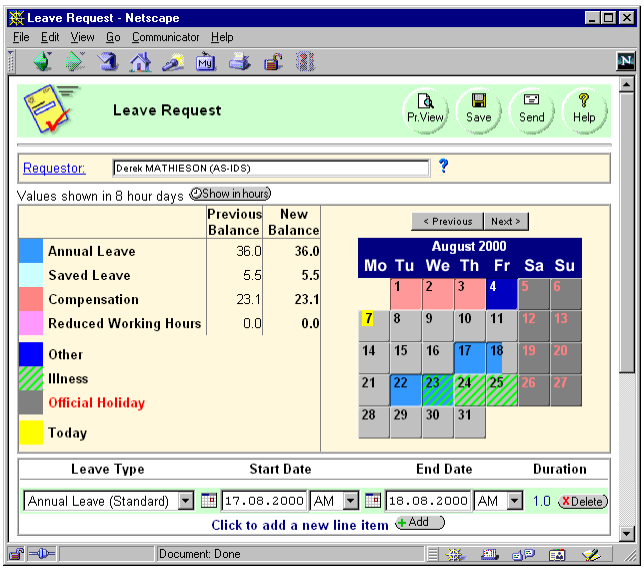
TEMPLATES

Like many Java Servlet applications the actual HTML form is generated from a series of template files.

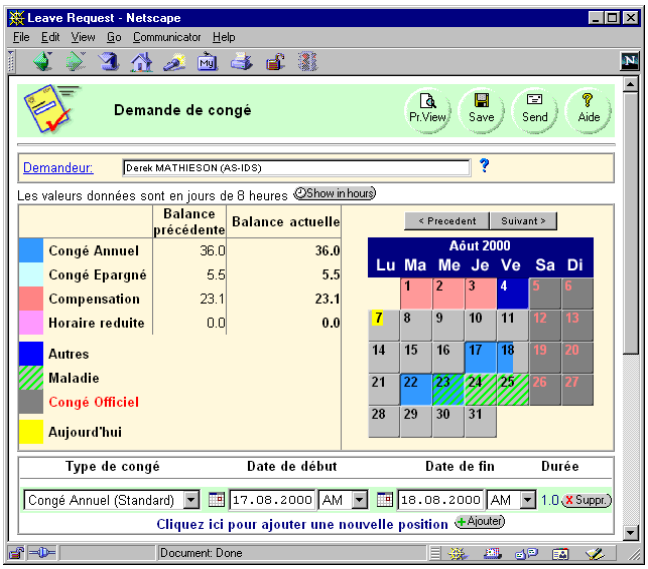
```
<table>
  <tr>
    <td>Total Price:</td><td>#VALUE#</td>
  </tr>
</table>
```

A snippet from a typical template file

The text between the # characters is replaced by the servlet with the correct value just before the page is displayed on the users browser. As a matter if principal, we have no HTML code within the Java Servlets, by using several templates it makes it possible for the application to present its user interface in the users preferred language (currently EDH fully supports both English and French, CERN's official working languages).



EDH Vacation Request in English

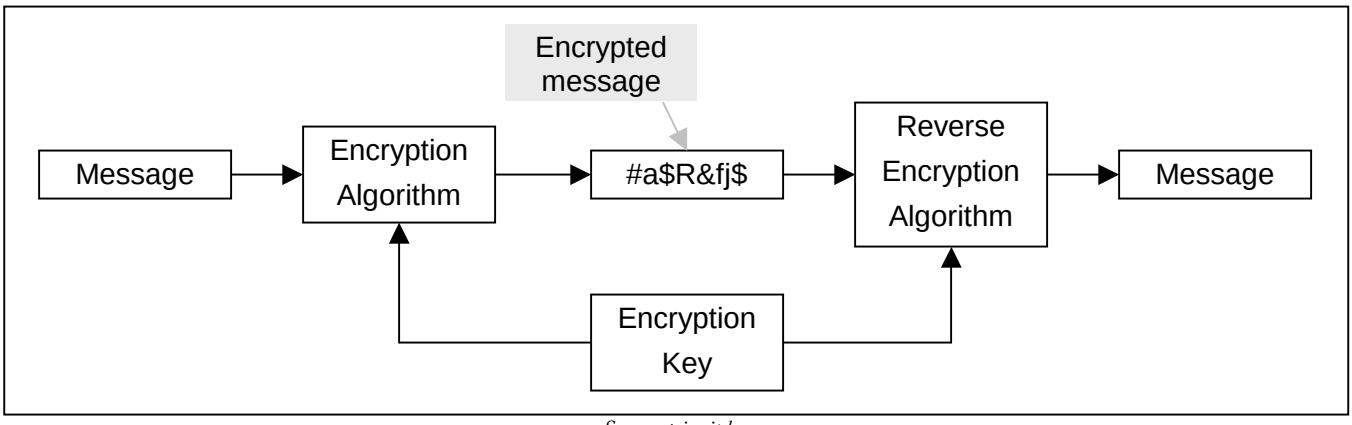


EDH Vacation Request in French

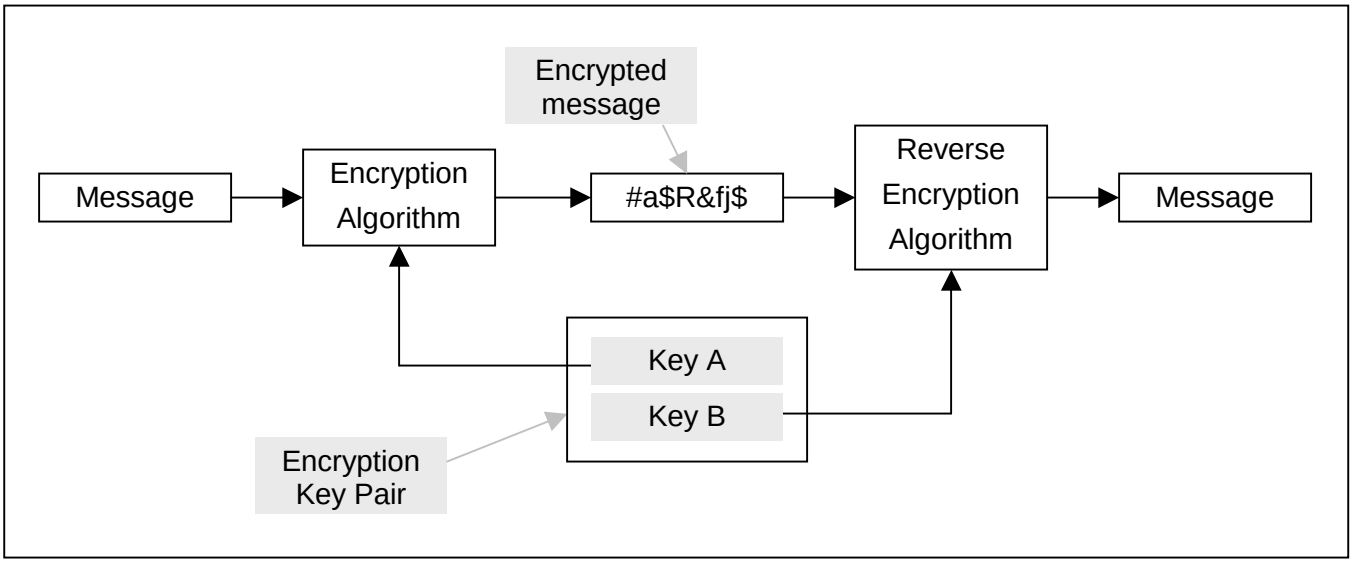
COMMON LOGIN

Authenticating the identity of users is fundamental to any e-business applications, and in the EDH application this was one of the first areas that we addressed. The EDH User Login that we developed proved to be so successful that it was adopted by all of CERN's other web based administration applications and was re-christened '*The Common Login*'.

The common login is based on an encrypted Cookie that is stored on the users browser. Fundamental to the design is the use of an asymmetric cipher (also known as *public key encryption*). In regular encryption (symmetric cipher) the information is encrypted using a secret piece of information (known as the *key*), in order to decrypt the information the encryption can be reversed using the same key. With an asymmetric cipher *two* keys are used. If one key is used to encrypt the message, then only the other key can be used to decrypt the message. Also knowing one key does not help you work out what the other key may be.



Symmetric cipher



Asymmetric cipher

Asymmetric ciphers are fundamental to the success of e-commerce on the Internet. The asymmetric cipher known as RSA (after R. Rivest, A. Shamir and L. Adleman, its inventors) is the basis of the secure HTTPS protocol; used to secure electronic transactions worldwide.

In the common login the cookie is encrypted using one key that is kept a secret. The other key is freely distributed and can be used by anyone to decrypt the cookie. That way anyone can decrypt the cookie, and use the information it contains to identify the user. But only our server computer has the secret key that can be used to create new login cookies.

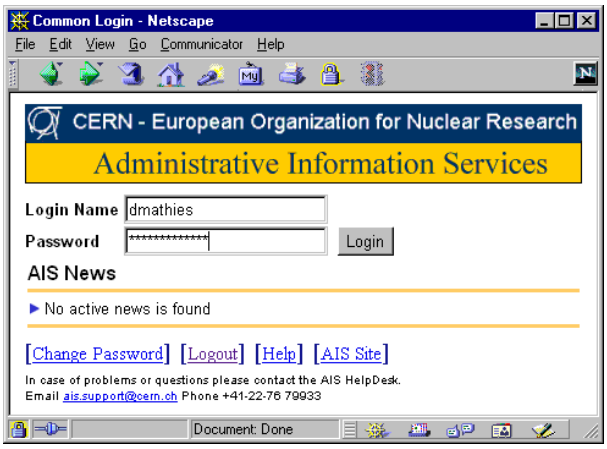
The Common login operates as follows:

The user accesses our site either by typing in the URL, or by clicking on a hyperlink to one of our electronic documents. For example: in e-mail from our workflow engine.

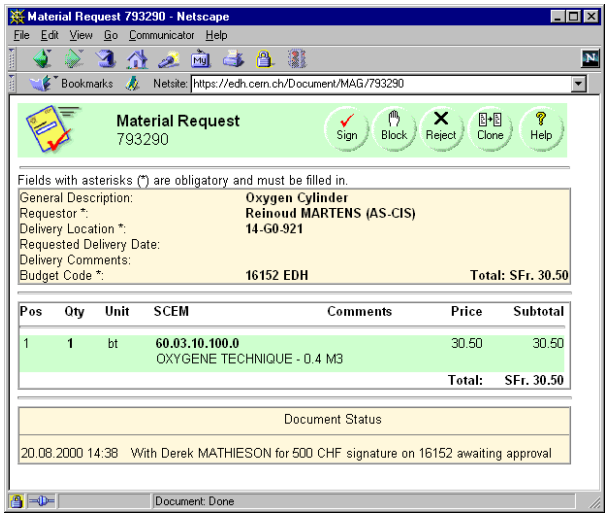
Since the user has not yet logged in, they are redirected to our login server.

The login server presents our user with a secure form, with which they can enter their username and password.

The login server is a Java Servlet, which accesses our User Database using JDBC.



If the username and password match one of our registered users, then the encrypted login cookie is generated and sent back to the users browser, along with another re-direct, back to the URL that they were originally seeking.

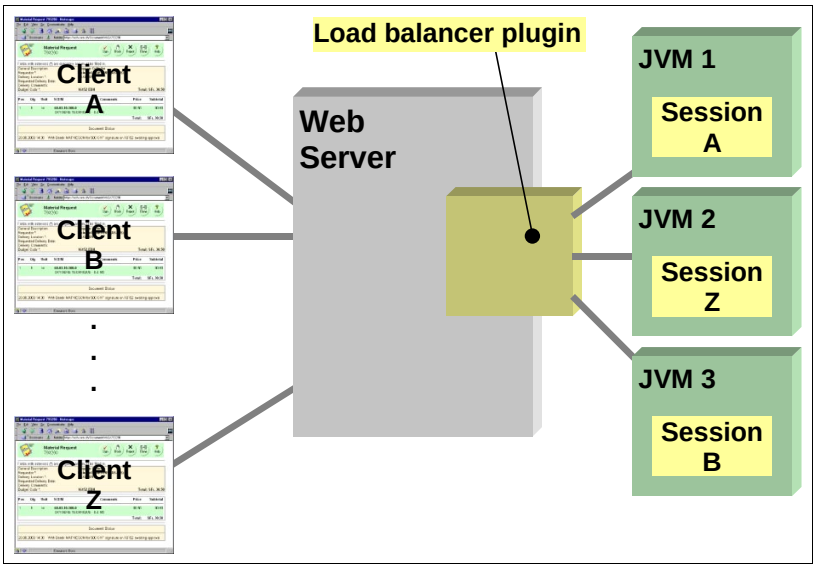


The verification of the cookie and initial re-direct is performed by a plugin that we have written for either Netscape or Apache web servers. By implementing the verification in a Web Server plugin we can use the same authentication mechanism for static HTML pages, along with any web application that is compatible with either Netscape or Apache Servers (including Oracle Application Server applications). Since the web server plugin only contains the public decryption key, we can freely distribute the plugin for use by other web applications that want to make use of the technology.

Load Balancing

One area that we identified very quickly when developing EDH was the need to spread the load of the application on several Java Virtual Machines at the same time. This was because a single virtual machine rapidly became overloaded with only a relatively small number of users. The problem usually manifests itself with the JVM running out of available memory, even although the JVM had been allocated 64Mb of RAM. This was because as the JVM gets loaded, the garbage collector fails to keep up with the number of objects being freed, this results in new memory allocation calls failing, even when there is plenty of free memory (only all of it is currently waiting to be marked free by the garbage collector).

In some sense Java applications can become a victim of their own success, since as soon as they become popular the garbage collector slows down. In EDH, we solved this problem by writing another Web-Server plug-in which allowed us to automatically spread the users over several JVMs (at the time of writing we are using 20 JVMs all running on the same physical computer).



Load-balancing across multiple Java Virtual machines

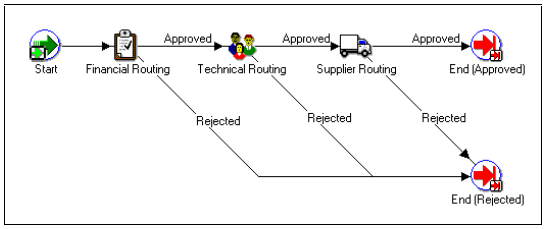
The load-balancing algorithm that we employ is very simple; the user ID that is held in their encrypted cookie determines the choice of JVM. Since the user Ids are effectively randomly distributed across our users, this gives us a reasonably balanced load.

Workflow at CERN

Workflow at CERN involves the routing of an electronic document to the people that are required to approve it. Approval of a document involves the user entering their *authorization password* in a specially encrypted field in the document. We increase the security of the HTTPS connection to our web server by masking their authorization password using a JavaScript implementation of the MD5 algorithm plus a unique 'Authorization Key' generated by our server. Our own auditors and all of the external institutes that use EDH have accepted this authorization password as equivalent to a paper signature.

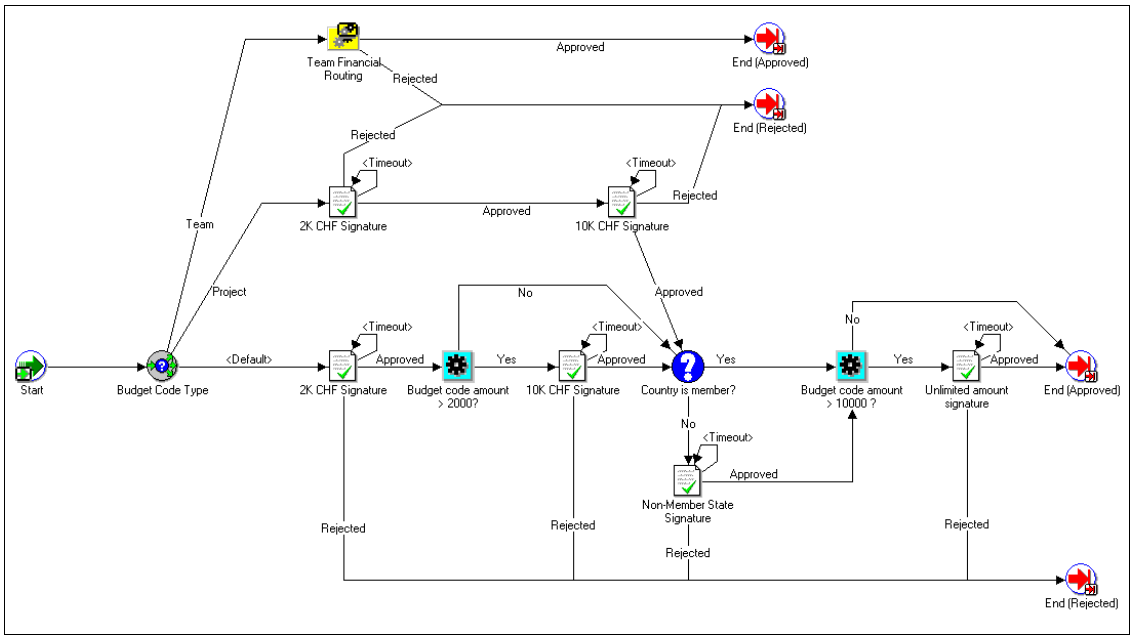
One of the complexities of the situation at CERN is the enormous number of special workflows that we require. The routing of financial documents has to take into consideration not only CERN's own financial rules, but also the requirements of other institutes for whom CERN spends money. Additionally different working practices within the organization are also supported which means that currently there are around 270 different paths that can be chosen depending on the type of document, who created it, and whose money is being spent.

Within Oracle Workflow Builder we have defined how each document should be routed. For example for a Purchase Order the workflow process is defined as shown below.



Example Purchase Order Routing

This may look simple, but each of the steps in the diagram itself expands to another sub-process such as that shown in the diagram below, where the CERN Standard Financial Routing is defined.



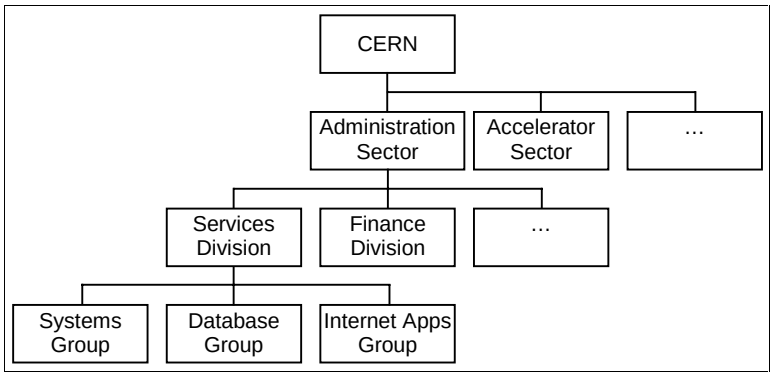
Standard Financial Routing

SIGNATURE RIGHTS DATABASE

In order that the workflow system is able to decide who has the right to sign for a particular step in the workflow process we have a database of signature rights for every workflow action. Most of the data in this signature rights database is derived automatically from our corporate databases. The organization structure from our human resources database is merged with data from our financial system to obtain a database of what accounts someone can spend from and to what limit.

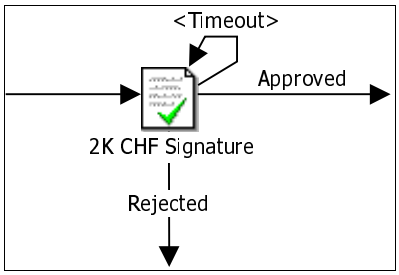
The diagram on the next page shows the basic arrangement of the organization. CERN is divided into four *Sectors*, each of which is split into three or four *Divisions*, which are in turn split into *Groups*, which are further split into *Sections*[†].

[†] Some *Sub-Sections* also exist at CERN.



Simplified Structure of CERN

In the Administration Division we define that a Group Leader can spend up to 10,000 Francs (about US$5,800) on any account in their group, and that a section leader can spend up to 2,000 Francs (about US$1,200) on any account in their section. This means that either the Group Leader or the Section Leader can sign a document for 500 Francs, however the signature rights are automatically *prioritized*, such that the section leader will be selected first. The Group Leader would be asked to sign only if the Section Leader was unavailable (for example, on absent due to training, vacation, etc.).



A typical Signature Action

In a typical routing definition there are many *signature* actions, for example financial documents require at least one signature to authorize payment. These actions appear on the diagram like the example shown above. When the workflow reaches this step in the routing a special stored procedure is called which determines who has the right to sign for the expenditure according to our signature right database. The procedure then uses our human resources database to select the first person that has the right to sign and is not absent.

Content Based Workflow

At many points within the workflow it is necessary to interrogate specific properties of the document (for example the total price of the purchase, or the name of the external supplier). In Oracle Workflow this is achieved by writing a user defined stored procedure that is called by the workflow engine during the routing. Normally this stored procedure would be written in PL/SQL but because all of our electronic documents are written in Java it would be useful to be able to re-use the Java implementation. By running Oracle Workflow within an Oracle 8*i* database it is possible to write stored procedures directly in Java and in this way Oracle workflow can directly call the `getTotal()` method of our Purchase Order CBO, thus eliminating any the additional maintenance caused by duplicated code in Java and PL/SQL.

Conclusions

Probably the most important conclusion that we can make from our experience is that it **is** possible to create a successful multi-lingual high performance E-commerce application using Java Servlets. We have found Java to be an excellent choice for server-side applications; its stability and ease of development make it possible for relatively small programming teams to rapidly produce high quality and reliable systems.

We have found that a well-designed development environment is vital to the success of projects such as these. Introduction of coding standards with formal code-reviews greatly increases the quality of the software. Additionally, by reviewing another developers code other team members are also aware how other parts of the system work, and are then able to track down any bugs when they arise.

SERVER-SIDE VALIDATION

At the beginning of the project we had many debates as to the need for client-side (JavaScript) validation. Some members of the design team felt that it was essential that we perform local validation before we sent any data to the server. Clearly the benefit would be in terms of responsiveness, but at a significant cost in terms of maintainability. JavaScript implementations vary enormously between browsers (even between the *same* version on different platforms) meaning that we would have to test on all of the platform/browser combinations (we have around 110 accessing our site!). Coupled with this is the fact that JavaScript is *not* Java, therefore we could not directly re-use the validation code that we anyway needed to write on the server (we could not *rely* on the client-side validation since it is trivial to bypass it).

Finally we agreed that we would minimize the amount of JavaScript on the client and perform all of the validation on the server. In fact the only JavaScript that we now have is to make the Enter key submit the form and to copy fields between lookup screens.

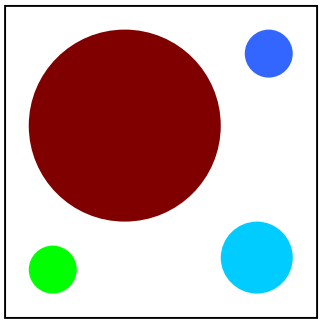
In production server-side validation *is* fast enough, we have personally used EDH over 56kbit modem links and long distance Internet connection without problem. By having all of the business logic located in a single place we have a more maintainable system without any possibility of inconsistent checks being made by different parts of the system.

THE FUTURE

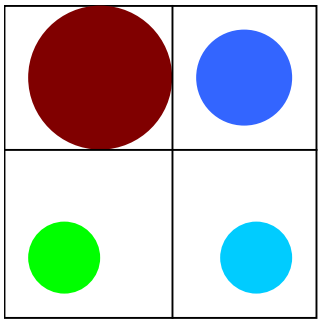
Currently we are using 20 instances of Suns classic JVM with a standard JIT compiler. Gradually the number of JVMs has increased as the site was subject to more and more load (initially we had just 5 JVMs). Although currently this is a stable configuration we cannot continue to increase the number of JVM indefinitely (each JVM consumes around 85Mb of memory with about 30Mb resident). In the future we plan to run all of the Servlets directly within the JServer JVM.

As mentioned earlier, within JServer each session appears to have its own private JVM yet the memory overhead per session is in the order of 50kbytes. What this means is that we can afford to allocate a separate session (and JVM) to each connected user, and by doing so this allows us to make use of existing administration tools to control the activity of the users.

Currently many users share the same JVM and communicate with the database via a shared pool of database connections. By sharing a single JVM is it almost impossible to prevent the activity of one user from affecting the others.



Shared JVM: A single user can dominate



Private JVMs: Each user is bounded by the JVM

Conversely, by using JServer the amount of resources consumed by a session can be strictly controlled using standard Oracle tools. Limits on memory, CPU, etc. can all be set.

We have already run tests on using Oracle 8i release 2 (version 8.1.6), with very encouraging results, during the presentation in October we will present our findings in more detail.

CONTACT INFORMATION

For more information regarding our work on Oracle Workflow we can be contacted via e-mail at: Derek.Mathieson@cern.ch and James.Purvis@cern.ch.

REFERENCES

[CERN] ..... CERN - European Laboratory for Particle Physics, Geneva, Switzerland
http://www.cern.ch

[CODING] ..... EDH Java Coding Standards
http://ais.cern.ch/apps/edh/CodingStandards

[EJB] ..... JavaSoft - *Enterprise Java Beans Specification* -
http://www.javasoft.com/products/ejb/index.html

[JSA] ..... JavaSoft - *Java™ Servlet API* -
http://www.javasoft.com/products/servlet/index.html

[WFMC] ..... Workflow Management Coalition - *Terminology & Glossary* -
http://www.aiim.org/wfmc/standards/docs/glossary.pdf