

INTEGRATED JAVA TECHNOLOGIES IN ON-LINE STORE DEVELOPMENT

Ciprian Băcilă, PhD Candidate, Ovidiu Vereş, PhD Candidate

Babeş Bolyai University of Cluj Napoca,
Theodor Mihaly, 58-60, 400591

Abstract: In developing an on-line store it is vital that we use an application which can offer maximum efficiency regardless of the platform it uses. The applications developed for this purpose have to be as robust as possible, providing the clients with access anywhere and the site administrators with the possibility to monitor continuously the changes in the data base. This article presents the means of obtaining such an application which can be easily adapted to all needs.

Keywords: business layer, servlet, Java beans, JavaServer pages (JSP), Java database connectivity (JDBC)

1. INTRODUCTION

Workflow management systems (WfMSs) are increasingly being used to manage business processes associated with distributed global enterprises. Some of the benefits of using a WfMS are: ability to visualize the overall process and interdependencies between various tasks, automation of the processes, and automated coordination and collaboration between various business entities [1].

Existing, commercially available WfMS do not offer sufficient flexibility for distributed organizations that will be participating in the global market. These systems have rigid, centralized architectures that do not operate across multiple platforms.

Improvements can be made by employing a distributed network of autonomous software agents that can adapt to changing circumstances. In the past, WfMSs were used in more well-defined activities, such as manufacturing, where the processes tend to be more established and stable. But in the current climate WfMSs may be used for more fluid business processes, such as e-commerce, or in processes involving human interactions, such as the software development process. In such situations, at times, it is not always possible to predict in advance all the parameters that may be important for the overall processes. In particular some of the reasons for wanting an adaptive WfMS are as follows:

- It may not be possible to specify all the process details associated with a complex process at the outset. The initial model may represent a high-level view of the process, which includes some of the sub-processes. Gradually some of these sub-processes may be refined as the stakeholders obtain more experience and knowledge of a particular process.
- Due to changes in the market, new requirements may be imposed which can impact the process definition. This change in the market may also include the availability of some new technologies, which may require the modification of the process as well, [2].

In this article, we describe an agent-based framework that provides a flexible infrastructure for incorporating dynamic changes to the currently executing process model. This architecture provides a mechanism for communication of distributed components in order to support interorganizational WfMS. The focus of the paper will be to show how WfMSs may be applied in the field of on-line commerce.

Workflow management systems (WfMSs) based on agent technology can cope with the rapidly evolving business environment better than most other systems as they are more flexible and open. The combination of collaborating agents and the Coloured Petri Net (CPN)-formalism in JBees enables a flexible and adaptive system with the possibility of simulation, analysis, and monitoring of the process execution in

order to identify potential inconsistencies and to provide appropriate information to the workflow administrator for the purpose of the process improvement [1], [2].

2. USING JAVA AS DEVELOPING ENVIRONMENT

The first and the most important thing in building an application for on-line commerce is choosing the programming language in which the application should be developed. We chose Java programming language for several reasons. Java language is powerful, but, at the same time, overly complex features that have bogged down other object-oriented languages like C++, are avoided. By using a simple language it's easier for programmers to write robust, bug-free code. As a result of its elegant design and next-generation features, the Java language has proved popular with programmers, who typically find it a pleasure to work with Java after struggling with more difficult, less powerful languages, [3].

Another reason would be the *Java platform*. The Java platform is just as important as the Java programming language. All programs written in the Java language rely on the set of predefined classes that comprise the Java platform. Java classes are organized into related groups known as packages. The Java platform defines packages for functionality such as input/output, networking, graphics, user-interface creation, security, and much more [3].

The Java platform is not an operating system, but for programmers, it is an alternative development target and a very popular one at that. The Java platform reduces programmers' reliance on the underlying operating system, and, by allowing programs to run on top of any operating system, it increases end users' freedom to choose an operating system.

Another key benefit of Java is its security features. Both the language and the platform were designed from the ground up with security in mind. The Java platform allows users to download un-trusted code over a network and run it in a secure environment in which it cannot do any harm: un-trusted code cannot infect the host system with a virus, cannot read or write files from the hard drive, and so forth. This capability alone makes the Java platform unique [4].

3. BUSINESS LAYER – AN IMPORTANT STEPS IN DEVELOPING

An important part in the design process is the task of creating a business layer. This portion of the application is wedged between presentation (what the user sees) and data (what the application depends on). The business layer, then, does just what it implies: it performs business (logic). Data on its own is rarely relevant, and often makes no sense without some context applied to it. In the same fashion, the presentation layer must have something to present. In this business layer, then, data is manipulated, transformed, and converted into content suitable for presentation [5].

The core of this layer is the code that actually executes business logic. This code maps to a company's business processes; in the best case, a single module of code represents a single business process. These modules can then be called to obtain a client's outstanding balance, for example. This figure is rarely stored in the database, but instead is calculated from the client's purchases subtracted from his or her assets. This allows the raw data to be masked from the presentation layer of an application; instead of asking for data and performing calculations, the application needs to request only the business process that results in a client's account balance, and format the result [5].

With entity beans in place for handling data access to most of our application, it makes sense to continue to leverage EJB for the business logic in our application. In this case, EJB session beans are a good fit. Session beans are complementary to entity beans and are designed specifically for handling business tasks. Additionally, they can be easily modularized, allowing mapping from a single business task to a

session bean (as mentioned earlier, this is optional). In this way, session beans can be "strung together" in logical ways, creating complete business processes. In this case, we used the term *business process* to refer to a series of individual tasks.

For example, obtaining a client's account balance or checking for availability of a new stock might be a business task; however, the complete business process of checking the balance, ensuring that enough is left to make a purchase, and then buying the stock comprises multiple individual tasks. Because each session bean can access the entity beans for the data it needs, one session bean may be used in multiple business processes without having to modify the single business task [6].

The biggest decision to be made regarding session beans is the type of bean to use for each task; session beans come in two flavors, stateful and stateless. *Stateful* session beans reside in memory once they are created and maintain information across requests. *Stateless* session beans, on the other hand, are "fire-and-forget" beans, which execute a request and are then disposed of until requested again. All information in the bean is trashed between requests [5].

While stateful beans can be helpful for processes or tasks that span multiple requests, they are often slower and obviously require more memory in the virtual machine. Stateless session beans, though, are very fast, and often only a few instances are needed to serve hundreds of requests. These qualities make them ideal for most business tasks, and preferable for better application performance [5].

JSP files don't require compilation. These files can enclose Java code in scriptlet, declaration, and expression elements. The most general of these are scriptlets, which can enclose multiline Java code. Scriptlets are enclosed with the markup `<%` and `%>`. You can use the out object's `println` method to send text back to the browser. JavaServer Pages (commonly, but not officially, referred to as JSP) have the functionality and syntax with a bear remarkable resemblance to Active Server Pages (ASP) [7].

JSP operates in many ways like server-side includes. The main difference is that instead of embedding a `<SERVLET>` tag in an HTML page, JSP embeds actual snippets of servlet code. This is an attempt to separate content from presentation, more convenient than server-side includes for pages that have chunks of dynamic content intermingled with static content in several different places [8].

Just like server-side includes and servlet chaining, JSP doesn't require any changes to the Servlet API. But it does require special support in your web server. Behind the scenes, the server automatically creates, compiles, loads, and runs a special servlet to generate the page's content. We can think of this special servlet as a background, workhorse servlet. The static portions of the HTML page are generated by the workhorse servlet using the equivalent of `out.println()` calls, while the dynamic portions are included directly [7], [8].

The first time you access a JSP page, you may notice that it takes a short time to respond. This is the time necessary for the server to create and compile the background servlet. Subsequent requests should be as fast as ever because the server can reuse the servlet. The one exception is when the *.jsp* file changes, in which case the server notices and recompiles a new background servlet. If there's ever an error in compiling, you can expect the server to somehow report the problem, usually in the page returned to the client [8].

JSPs are actually based on Java servlets (they're translated into servlets before being run), which are pure Java code. Most servlets are based on the `javax.servlet.http.HttpServlet` class, and they often simply override the `doGet` method, which is passed a request object that holds data from the browser (including the browser type and the data from any HTML controls) and a response object that lets you tailor your response to the client browser [7].

So far, we have portrayed servlets as an alternative to other dynamic web content technologies, but we haven't really explained why we think you should use them. What makes servlets a viable choice for web development? We believe that servlets offer a number of advantages over other approaches, including: portability, power, efficiency, endurance, safety, elegance, integration, extensibility, and flexibility [9].

Servlets support safe programming practices on a number of levels. Because they are written in Java, servlets inherit the strong type safety of the Java language. In addition, the Servlet API is implemented to be type-safe. While most values in a CGI program, including a numeric item like a server port number, are treated as strings, values are manipulated by the Servlet API using their native types, so a server port number is represented as an integer. Java's automatic garbage collection and lack of pointers mean that servlets are generally safe from memory management problems like dangling pointers, invalid pointer references, and memory leaks. Servlets can handle errors safely, due to Java's exception-handling mechanism. If a servlet divides by zero or performs some other illegal operation, it throws an exception that can be safely caught and handled by the server, which can politely log the error and apologize to the user. If a C++-based server extension were to make the same mistake, it could potentially crash the server. A server can further protect itself from servlets through the use of a Java security manager. A server can execute its servlets under the watch of a strict security manager that, for example, enforces a security policy designed to prevent a malicious or poorly written servlet from damaging the server file system.

One of the most interesting and powerful ways to use JavaServer Pages is in cooperation with JavaBeans components. JavaBeans are reusable Java classes whose methods and variables follow specific naming conventions to give them added abilities. They can be embedded directly in a JSP page using `<BEAN>` tags. A JavaBean component can perform a well-defined task (execute database queries, connect to a mail server, maintain information about the client, etc.) and make its resulting information available to the JSP page through simple accessor methods [7].

The difference between a JavaBeans component embedded in a JSP page and a normal third-party class used by the generated servlet is that the web server can give JavaBeans special treatment. For example, a server can automatically set a bean's properties (instance variables) using the parameter values in the client's request. If the request includes a name parameter and the server detects through introspection that the bean has a name property and a `setName(String name)` method, the server can automatically call `setName()` with the value of the name parameter. There's no need for `getParameter()`.

A bean can also have its scope managed automatically by the server and can be assigned to a specific request (where it is used once and destroyed or recycled) or to a client session (where it's automatically made available every time the same client reconnects) and can even be implemented as a servlet. If the server detects that a bean implements the `javax.servlet.Servlet` interface (either directly or by extending `GenericServlet` or `HttpServlet`), it will call the bean's `service()` method once for each request and the bean's `init()` method when the bean is first created. The utility of this functionality is debatable, but it can be used by beans that need to prepare somehow before handling requests.

Using a JavaBeans component with JavaServer Pages greatly reduces the amount of code necessary in the page by a clean separation of content (the functionality the bean provides) from presentation (the HTML structure of the page). By using a well-defined API to interact with the bean, even nonprogrammers can write JSP pages. JSP files are able to connect to compiled Java code using JavaBeans, and developing those applications are no problem now that you know how to use linked folders. In JSP, we can connect to the bean code in a variety of ways, including instantiating an object using Java in a JSP scriptlet. The recommended way of doing things, however, is to use the JSP `jsp:useBean` element to create a JavaBean

object. Then use the `jsp:getProperty` element to get the value of a bean property, and use the `jsp:setProperty` element to set a bean property's value.

Once the on-line shop is created, it's very important to have an easy way of keeping track of the transactions. For this, an easy method would be to implement a module that generates graphic reports. This problem may be easily solved using **JFreeChart** library. JFreeChart is a free chart library for the Java(tm) platform. It is designed for use in applications, applets, servlets and JSP. JFreeChart is distributed with complete source code subject to the terms of the GNU Lesser General Public Licence, which permits JFreeChart to be used in proprietary or free software applications. JFreeChart can generate pie charts, bar charts (regular and stacked, with an optional 3D-effect), line charts, scatter plots, time series charts (including moving averages, high-low-open-close charts and candlestick plots), Gantt charts, meter charts (dial, compass and thermometer), symbol charts, wind plots, combination charts and more [10].

4. CHOOSING A DATABASE

It's hard to find a professional web site today that doesn't have some sort of database connectivity. Webmasters have hooked online front ends to all manner of legacy systems, including package tracking and directory databases, as well as many newer systems like online messaging, storefronts, and search engines. But webdatabase interaction comes with a price: database-backed web sites can be difficult to develop and can often exact heavy performance penalties. Still, for many web sites, especially intranet applications, database connectivity is just too useful to let go. More and more, databases are driving the Web [5], [7].

The language of choice for databases is the Structured Query Language (SQL). Most databases now come with tools to make the creation of data structures simple; these are usually graphical and present a visual means of creating data structures. Additionally, a number of third- party tools are good for this sort of task. I'll focus on using pure SQL in this section, so the code will work on any database, on any platform, without you having to learn or buy a specific vendor's tool. This part introduces relational databases, the SQL used to manipulate those databases, and the Java database connectivity (JDBC) API itself. Servlets, with their enduring life cycle, and JDBC, a well-defined database-independent database connectivity API, are an elegant and efficient solution for webmasters who need to hook their web sites to back-end databases [7].

The biggest advantage for servlets with regard to database connectivity is that the servlet life cycle allows servlets to maintain open database connections. An existing connection can trim several seconds from a response time, compared to a CGI script that has to reestablish its connection for every invocation. The maintenance of the database connection depends only on the task at hand. Another advantage of servlets over CGI and many other technologies is that JDBC is database-independent. A servlet written to access a Sybase database can, with at two-line modification or a change in a properties file, begin accessing an Oracle database.

We have chosen to list a code

example of the application that manages the user table in the data base. The same code may be used to create the others tables changing only the name of the tables and the name of the attributes in the other tables. For this application we have chosen a "Derby" data base developed by IBM and Apache Software Foundation. This is a free version taken from the "<http://java.sun.com>." web site.

```
import java.sql.*;
import java.util.*;
import java.io.*;
```

```

public class Aplicatie {
    public int contor=0;
    public String driver = "org.apache.derby.jdbc.EmbeddedDriver";
    public String protocol = "jdbc:derby:";
    public Connection conex = null;
    public Statement st = null;
    public ResultSet rs = null;
    private String[] s1= {"Nume", "Prenume", "Telefon", "Adresa", "UserName"};
    private String[] s2 = new String[5];
    private BufferedReader br;
    private String sir;
    public String s=new String();
    public String inter=new String();

    public Aplicatie() {
        try {
            //jdbc:derby:jar:("C:/)databasePathWithinArchive
            Class.forName(driver);
            conex=DriverManager.getConnection(protocol+"derbyDB;create=true", "administrat
                or", "parola");
            /* Driver registration. The data base is on the same machine as the aplicatie.
            * So I used "embedded driver".*/
            //The connection creates the data base in the "derbyDB" directory in current directory.
            st = conex.createStatement();
            st.execute("DROP TABLE Utilizatori");
        }
        ....
        public void adaug() {
            /* Adds a new user in the users table. If the user already exists in the table,
            * returns an error message.*/

            br=new BufferedReader(new InputStreamReader(System.in));
            try {
                for(int i=0;i<5;i++) {
                    System.out.print(s1[i] + " : ");
                    s2[i] = br.readLine();
                }
                if(utilizatorExistent(s2[0])) System.out.println("Nu puteti adauga pe "+s2[0]+"
deoarece este deja prezent in baza de date.");
                else {
                    st.executeUpdate("INSERT INTO Utilizatori VALUES (" +
                        """" + s2[0] + """, """" + s2[1] + """, """" + s2[2] + """, """" + s2[3] + """, """" + s2[4] + """)");
                    System.out.println("Adaugare efectuata!");
                }
                contor++;
                inter="Persoana";
                s=inter.concat(Integer.toString(contor));
                s=s.concat(".csv");
                RandomAccessFile ta=new RandomAccessFile(s, "rw");
                ta.writeBytes(" " + s2[1] + " " + s2[2] + " " + s2[3] + " " + s2[4]);
            }
        }
    }
}

```

```

        catch(SQLException e){System.out.println("Eroare SQL la adaugare de utilizator");}
        catch(IOException s){}
    }
}

```

5. SELECTING A VIABLE GRAPHIC INTERFACE

The graphic interface of an on-line shop should be kept as simple as possible. We design that a page divided in three areas should be good enough: a rectangular layer for banners in the top of the page, a column with links for navigating on the site on the left side and the main area in which the information selected from the navigation column is displayed. Banners posted on the top layer should be links to other sites that have a complementary activity, and those sites should post links to your site. This is a very efficient method of advertising the content of your on-line shop.

The categories of products are listed on the column on the left part of the page. When clicking on a category displayed on the left column, in the main layer on the center-right part of the webpage the products that belong to that particular category are listed.

The login area should be on the top of the left column. If a person does not have an account on the site one should be given the possibility of creating one. When a person tries to log in on the application, one should be searched for in the data base and then checked if the user name and the password are correct. If the user name or the password is incorrect the application should display warning messages where the users are asked to reintroduce their personal data, until it matches with the one in the data base. After a user logs-in he has the possibility of buying different product. When a user orders a certain quantity from a product, that quantity will automatically be deducted from the one available on stock.

The administrator of the web application should have exclusive access to parts of the application that helps him control the data base. The application should have a page special built for introducing new products in the data base, a page where the site administrator has access to different charts that allows him to see the sales volume of a product over time, the existing stocks of different products categories. The administrator of the site should validate the orders and should have the option to send the products to the user or to reject the order. If the site administrator decides to reject an order the quantity of products that the user ordered are automatically returned into the data base.

6. CONCLUSIONS AND FUTURE WORK

When creating a web application for an on-line shop the software should be divided into three components: one whose purpose is the graphic design of the site; one that implements the data base support; and one that insures the connection between the site and the data base. The link between the site and the data base is realized on a Java platform.

The Java Servlets are used for accessing the data base. The Java Beans after the login in order to select user when the shopping list is being created and when the order is shipped. Using the WfMSs helps the site administrator by automatically up-dating the data base and by creating at request graphic reports concerning the sales situation or the users' activity.

The product page should be dynamically created based on the information existent in the data base. The shopping cart should introduce into the data base both the user who performs the transaction and the products that the user buys. Thanks to the use of the Java platform, this kind of applications could run on any operating system. The selection of the operating system will be made according to the type of data base chosen.

In this article we have discussed the way in which the application can be developed without taking into account the number of users that access it simultaneously. Moreover, certain issues related to the security that this application offers the users should also be brought into question. Most problems related to the security of the users' personal data are generated by the fact that the web application can be accessed anywhere, and when it is accessed from a public place the confidential data should not remain visible after logout. The design of any web application should ensure the simultaneous access of a high number of users and also the confidentiality of their personal data.

ACKNOWLEDGEMENT

This work was supported by Romanian National Authority for Scientific Research under the grant no. PN2 91-049/2007 SIDE.

REFERENCES

- [1] Lars Ehrler, Martin Fleurke, Maryam Purvis, Bastin Tony Roy Savarimuthu (2005), Agent-based workflow management systems (WfMSs) Published online: 8 November 2005 Springer-Verlag
- [2] Ernesto Montaldo, Roberto Sacile & Antonio Boccalat (2003), Enhancing Workflow Management in the Manufacturing Information System of a Small-Medium Enterprise: An Agent-Based Approach, *Kluwer Academic Publishers. Manufactured in The Netherlands.*
- [3] Justin Gehrtland, Bruce A. Tate (2004), *Better, Faster, Lighter Java*, O'Reilly, Printed in the United States of America.
- [4] Halloway, Stuart Dabbs. "Java Properties Purgatory, Part 1 and 2." InformIT August 23, 2002 (<http://www.informit.com>).
- [5] Brett McLaughlin (2002), *Building Java Enterprise Applications Volume I: Architecture*, O'Reilly & Associates, Printed in the United States of America.
- [6] Retting, Michael J. with Martin Fowler. "Reflection vs. Code Generation." javaworld.com, November 2, 2001 (<http://www.javaworld.com/javaworld/jw-11-2001/jw-1102-codegen.html>).
- [7] O'Reilly & Associates (2003), *Java Network Programming, Second Edition*, O'Reilly & Associates, Printed in the United States of America.
- [8] Bruce W. Perry (2004), *Java Servlet & JSP Cookbook*, O'Reilly, Printed in the United States of America.
- [9] Jason Hunter & William Crawford (1998), *Java Servlet Programming*, O'Reilly & Associates, Printed in the United States of America.
- [10] <http://www.jfree.org/jfreechart/>