Blois, M., Choren, R., Laufer, C., Ferraz, F. & Fuks, H. (1999) Scriba - A Tool for Developing Java Based Web Applications, Proceedings da WebNet'99 World Conference of the WWW, Internet & Intranet, vol 1, Honolulu, EUA, Novembro 1999, pp. 112-118

Scriba — A Tool for Developing Java Based Web Applications

Marcelo Blois
Computer Science Departament, PUC-Rio, Brasil, blois@inf.puc-rio.br

Ricardo Choren
Computer Science Departament, PUC-Rio, Brasil, choren@inf.puc-rio.br

Carlos Laufer
Computer Science Departament, PUC-Rio, Brasil, laufer@inf.puc-rio.br

Fabio Ferraz
Computer Science Departament, PUC-Rio, Brasil, ferraz@les.inf.puc-rio.br

Hugo Fuks
Computer Science Departament, PUC-Rio, Brasil, hugo@inf.puc-rio.br

Abstract: This paper presents Scriba — a tool that facilities the development of Java based Web applications. The tool allows the creation of script languages that can be embedded in HTML pages in order to process data dynamically. As well as the advantages of the use of Java technology, the structure in the tool's software components allows its users to redefine its functions, expanding or recreating its basic language. The AulaNet environment is an example of application developed with Scriba.

Introduction

The evolution of Java technology has made it possible for new approaches for the development of Web applications to appear. These applications possess diverse parts that may be organized in the form of software components, making the most of the potential offered by object oriented languages like Java. The software can be seen as a group of interconnected components that cooperate in order to promote the operation of the whole environment.

This paper presents Scriba, a tool that interprets commands for the fast development of Java based Web environments. Scriba makes the construction of Java based Web applications easier and quicker. Through specific markings in its language, Scriba enables data to be put on the pages in loading time. As well as possessing dynamic creation of pages with HTML templates, Scriba also makes the use of user defined script languages viable through expanding its language or substituting its command interpreter.

Scriba is being used in the development of a new version of AulaNet [Lucena et. al. 1999] an environment for creating and attending courses on the Web. The following sections give details of Scriba's own language and the way it operates.

Servlets and JDBC

The latest versions of Java incorporate two new language mechanisms used in the development of network and database applications known as servlets and JDBC. The new extension package for the Java Development Kit 1.1

(JDK 1.1) intended to be a Java substitute for CGI programs (Common Gateway Interface) is known as a servlet [Siyan and Weaver 1997]. With servlets, users can add new functions to HTTP servers.

HTTP servers that possess a servlet API provide an environment with all the resources that the servlets need to operate, and a model of security that maintains the integrity of the system. Servlets can contain any Java function or package that does not have graphic interface. They can read or write in files, process calculations, communicate with databases and create HTML files, functioning like a traditional CGI program, with the advantages of the Java language in terms of structuring in components and performance.

The latest versions of Java also incorporate JDBC [Hamilton, Cattell and Fisher 1997]. JDBC is an API for the execution of SQL statements on any relational database. The API of the JDBC allows users to write database applications using a purely Java interface. The JDBC carries out the following main operations: it establishes a connection with a database, sends SQL statements and processes the result. This solution is the one adopted by Scriba to communicate with users' databases.

Scriba

Scriba is a tool for creating dynamic HTML pages and defining Java classes to implement the processing of Web applications. Scriba offers a basic language that allows one to consult ODBC databases, exchange values stored and invoke user defined classes for the realization of functions specific to the application's domain.

Users who wish to implement Web applications using servlets and Java find the necessary elements in Scriba for the implementation of the whole application using HTML pages and Java classes. HTML pages can have the Scriba code embedded in them that carries out, among other things, data searches in any ODBC database. The Java user defined classes bring together the specific functions of the application in development. Scriba deals with all the control operations and creation of objects for the functioning of the servlets, communication with ODBC databases and operations referring to interpretation and execution of the commands of its language.

One of the main characteristics of Scriba is the possibility of configuring its functions through the user redefining its components. The structuring of Scriba in software components interrelated with well defined interfaces allows its users to rewrite most of its components. They can adapt Scriba to interpret any other language. There are two types of configuration of the Scriba functions: users can extend the basic language supplied through creating new commands in the interpreter or they can define a completely new interpreter for Scriba.

Scriba is made up of three basic components: a token divider, a command interpreter and a HTTP request receiver. Figure 1 shows the Scriba architecture and the classes that compose each of its components.

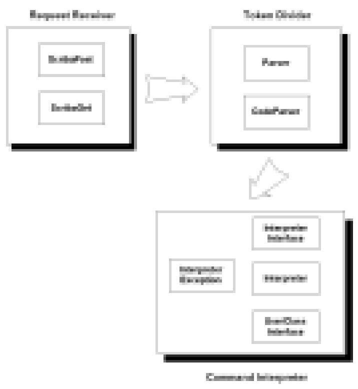

Figure 1: Scriba components and classes.

The token divider is made up of two classes: *Parser* and *CodeParser*. This component is responsible for reading HTML templates and for checking the existence of the boundaries of its language. When a language token is found within the boundaries $* and *$ the divider passes this token to the interpreter so that the actions referring to the language command contained in the token may be taken.

The command interpreter is composed of the following classes: *InterpreterInterface*, *InterpreterException*, *Interpreter*, and *UserClassInterface*. So that the users write specific interpreters for their languages, the *InterpreterInterface* class supplies a patttern that a new class should follow in order to be defined as an interpreter in Scriba. The Interpreter class obeys the *InterpreterInterface* specifications through inheritance and implements the actions corresponding to each of the language commands. The *InterpreterException* class deals with all the errors created in the interpreter, standardizing the messages and explanations about the problems found in the interpretation of the commands. The last class of this component, *UserClassInterface*, defines the format that all the user defined classes for processing form values should present.

The HTTP request receiver receives the parameters passed by the HTTP server when a page is requested using the GET and POST methods. This component carries out all the necessary operations to implement the Java servlets and trigger the token divider to search the HTML template indicated looking for language tokens. This component is made up of the following classes: *ScribaGet* that deals with GET type of requests, and *ScribaPost* that deals with POST type of requests.

Figure 2 presents the execution flow provoked by the *ScribaGet* and *ScribaPost* classes for each request made to Scriba.

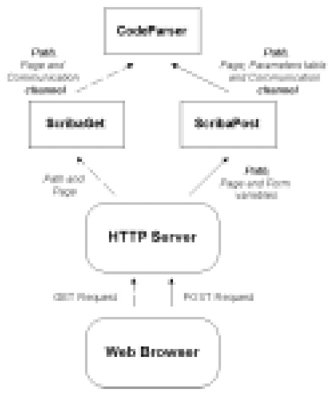
CodeParser
Path, Page and Communication channel
Path, Page, Parameters table and Communication channel
ScribaGet
ScribaPost
Path and Page
Path, Page and Form variables
HTTP Server
GET Request
POST Request
Web Browser

Figure 2: *ScribaGet* and *ScribaPost* execution model.

After being created by the *ScribaGet* and *ScribaPost* classes, objects of the *CodeParser* class inspect the HTML template supplied and pass the tokens found to an object of the Interpreter class that will take the necessary action acccording to the language commands. The basic language commands defined to implement AulaNet will be presented in the next section.

Operational Details

In order to implement Web environments with dynamic pages, Scriba has a basic language conceived from the necessities found in the development of AulaNet. The Scriba language has commands that search for values in ODBC databases and create user defined classes. There is also a syntax so that the user can expand the language without having to redefine the interpreter completely.

The language embedded in the HTML files obeys the following general rules:
1. So that the code divider recognizes the existence of the commands embedded, it is necessary for them to be delimited by a marker at the beginning of commands (**$***) and one at the end of commands (***$**).
2. Scriba recognizes commands separated by **;;** inside a HTML template. If it is necessary to define more than one command in the same place in a file, the user can use this construction: **$*** <command1>**;;** <command2>**;;**…**;;**<commandN>***$**.

3. The command lines may be situated in different places in the file, however the argument and the command name can not be divided among various lines of text.

In order to construct Web applications Scriba has 8 commands as follows: *database.open*, *database.close*, *database.select*, *database.select.set*, *database.select.set.pattern*, *class.new*, *form.parameter*, *language.expand*.

The *database.open* command is responsible for creating a reference to an ODBC database. Its argument is made up of the name of the database registered in the ODBC (DSN). This command does not return values, it just prepares the database to receive SQL requests.

The *database.close* command closes a database previously opened by *database.open*. This command does not have arguments and does not return any value. Scriba uses a *DataManager* denominated class to provide the communication services with the database. This class brings together the calls to JDBC API that implement the JDBC-ODBC bridge.

The *database.select* command allows the execution of a SQL SELECT statement about the database previously opened by *database.open*. Its argument is the SELECT statement and its return value is composed of one unique field returned by this statement. If the return of the SELECT statement implied more than one field or line, the *database.select* command would only return the first field of the first line, ignoring the rest of the values returned. For example, if the command found were *database.select:=SELECT Name, Account FROM Users*, the return of the SELECT clause would possibly be made up of more than one register containing the name and number of each user's account. Even if the SELECT clause executed returned more than one register, Scriba would only inform the value of the Name of the first user register. In order to carry out clauses like the latter example, Scriba has a special syntax of the *database.select* command which is the *database.select.set* command.

The *database.select.set* command creates a table with the values returned by a SQL query statement internally in Scriba. This command does not have a return value and its argument is composed of the SELECT statement. In order to have access to the fields stored in the table the user should indicate which pattern is to be repeated changing the fields indicated by a value in the table. The syntax used to define the repetition pattern is the *database.select.set.pattern* command. Figure 4 shows how Scriba executes the *database.select.set* command.

The *database.select.set.pattern* command has as its argument any text that will be repeated for each line in the table of returned values. Within the argument of this command, the user can indicate to Scriba one of the fields to be changed dynamically by the values in the table through the *set.name_of_field* syntax. Returning to the previous example, the *database.select.set:=SELECT Name, Account FROM Users* command would be executed, creating a table with the returned values by the SELECT clause. The user could declare, for example, the command *database.select.set.pattern:=<tr><td>set.Name</td><td>set.Account</td></tr>* to create an HTML table with the returned values of the select clause.

The *class.new* command creates a class object whose name is indicated in its argument. The object is created dynamically by Scriba to carry out Java based user defined operations. With this command Scriba allow users to program sections of Java code to make specific processes with its application. The whole class created has access to the HTML form variables, allowing the implementation of programs that deal with data like the majority of languages do. For a user to create a class that can be declared in the *class.new* command he should define the class as child of the *UserClassInterface* class of the command interpreter component. When inheriting the characteristics of *UserClassInterface* through the IMPLEMENTS declaration in Java, the user will know which methods he must create and the parameters that he will receive in his class.

If a user wants to have access from his page to some variable declared on a previous form, he can use the *form.parameter* command. This command possesses as a argument the name of the variable that he wants to have access to and returns its value. With this command the user can pass values between different templates without having to retrieve these values on all the pages.

If it is necessary to expand the Scriba language, the user can use the *language.expand* command. This command allows users to create a class that implements the *InterpreterInterface* interface to interpret a proprietary language expanded in relation to the tool's original language. This class should have an action method that will receive a token and process it, identifying the command involved and carrying out the corresponding action. There is no limit to creating classes of this type.

Scriba allow users to rewrite the main parts of two of its components: the token divider and the command interpreter. In order to guarantee compatibility between the Scriba component parts and the parts created by the user, two classes are supplied: Parser and *InterpreterInterface*. The *Parser* class should be inherited by the user defined class that will substitute the *CodeParser* class. The *CodeParser* class is useful to break the HTML template tokens. If the type of template or the language markers of the user need to be changed, all that has to be done is to rewrite the *CodeParser* class according to its new definition, inheriting the characteristics of the Parser class.

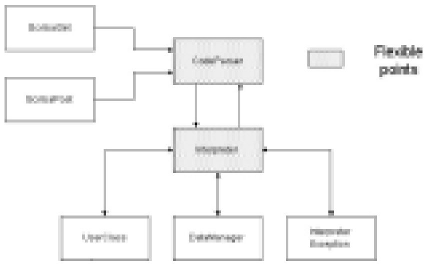

Figure 3: Scriba architecture and flexible points.

If the language to be interpreted is completely new, the user can define a new *Interpreter* class with the language commands and their respective actions. This new class should implement the *InterpreterInterface* class, inheriting its characteristics. The new interpreter should receive tokens from the *CodeParser* class and identify the statements of its language within these tokens, creating a convenient return value for each type of command interpreted. Figure 3 shows a view of the Scriba architecture, identifying its flexible points.

Scriba for AulaNet

Scriba was invented within the philosophy of the creation of Web based cooperative software environments that use databases to store their information. AulaNet is a case of the application of this technology. There are plans to modify the AulaNet architecture, using Scriba and Java, to restructure it in object oriented components that integrate to offer the services supplied by the environment.

AulaNet is an environment for creating and attending courses through the Web. Teachers in AulaNet can create courses by selecting a set of communication, cooperation and coordination mechanisms. After choosing the mechanisms for a course, teachers can input the content that will be seen by the students. The students can communicate and cooperate to construct knowledge and carry out activities in groups. As it facilitates the creation of courses through selecting mechanisms and inserting materials AulaNet is considered a high-level solution for low-end users.

AulaNet is made up of a group of HTML pages, CGILua scripts [Hester, Borges and Ierusalimschy 1998] and an MS Access database. The CGILua tool allows the creation of dynamic HTML pages and the processing of HTTP requests. CGILua possesses a library for communication with ODBC databases. AulaNet uses Microsoft Access to store its data. Approximately 90% of the functionality of AulaNet is implemented using these components. The other 10% is made up of freeware software and two commercial softwares, offering the main part of the communication functions supplied by the environment.

The strategy adopted to substitute CGILua with Scriba will be guided to reduce the number of files in the software. The programs that would be executed in the templates to compose the page dynamically will be transformed into one or more Scriba commands, as they only use database functions. The Scriba functions map out the majority of the actions performed to search for data and exchange parameters in the templates.

The new version of AulaNet should benefit from some improvements brought about by the use of Java technology, like the portability of the programs developed. Another improvement brought about by the use of Scriba is the access to databases. As AulaNet uses the Access database, some concurrency problems were identified when various simultaneous accesses were performed in the database. The use of the JDBC to access the database reduces the problems of concurrency, as all the access methods used are synchronized by the monitor mechanism implemented by the Java language.

There are plans to create a software documentation pattern, benefiting from the reduction of files brought about by Scriba. The number of files in the Scriba based version of AulaNet should be approximately 50% less than the number of files in the current version, based on the experience of implementing 25% of the software already made until now.

Conclusion

Scriba, a tool for creating Web based applications, aims to simplify the creation and maintenance of Java based applications. Scriba adapts well to the types of applications that store information in ODBC databases. In order to implement its capabilities Scriba possesses a component structured architecture.

To facilitate creating dynamic pages Scriba has a language that executes SQL statements and creates user classes. The Scriba language also has the capability to recognize extended commands, allowing users to broaden the capabilities of its language without having to change the Scriba components. Another interesting characteristic of Scriba is the possibility to redefine the tool’s components. Users can redefine the language to be interpreted and its markers through the redefinition of the classes responsible for the identification of language tokens and command interpretation.

Some improvements can already been identified for a future version of Scriba, according to the results obtained in its use:

- extension of the basic language;
- capacity to insert code directly in the script;
- a new approach for developing TCP/IP generic servers and clients by the definition of their communication protocol.

Finally we aim to develop Scriba so that it can possess a form of graphic representation of applications that allows the creation of a tool for the semi-automatic generation of programs. This will enable users to specify the application with graphics and create pages that compose it visually, avoiding contact with the code prematurely.

References

Hamilton, G., Cattell, R., & Fisher, M. (1997). *JDBC Database Access with Java – A Tutorial and Annotated Reference*. Addison-Wesley.

Hester, A., Borges, R., & Ierusalimschy, R. (1998). Building Flexible and Extensible Web Applications with Lua. *WebNet'98-World Conference of the WWW, Internet & Intranet, 1998,* Association for the Advancement of Computing in Education, Charlottesville, VA.

Lucena, C. J. P., Fuks, H., Milidiú, R., Laufer, C., Blois, M., Choren, R., Torres, V., & Daflon, L. (1999). AulaNet: Helping Teachers to Do Their Homework. *Multimedia Computer Techniques in Engineering Education Workshop,* CMES (Computer Method in Engineering Science) Network, Graz, Austria. 16-30.

Siyan, K. S. & Weaver, J. L. (1997). *Inside Java*. New Riders Publishing.