Blois, M., Choren, R., Laufer, C., Ferraz, F. & Fuks, H. (1999) Desenvolvendo Aplicativos para a Web com o Scriba, Anais do XXVI SEMISH - Seminário Integrado de Software e Hardware, Sociedade Brasileira de Computação (SBC), Rio de Janeiro, Julho 1999, pp. 119-133

Desenvolvendo Aplicativos para a Web com o Scriba

Marcelo Blois, Ricardo Choren, Carlos Laufer, Fabio Ferraz e Hugo Fuks

{blois, choren, laufer, ferraz, hugo}@inf.puc-rio.br

Departamento de Informática

PUC-Rio

Resumo

Este trabalho apresenta uma ferramenta que facilita o desenvolvimento de aplicativos para a Web em Java. A ferramenta permite a criação de linguagens de script que podem ser embutidas em páginas HTML para montagem dinâmica de seus dados. Além das vantagens de utilização da tecnologia Java, a estrutura em componentes de software da ferramenta permite que os seus usuários redefinam suas funcionalidades, expandindo ou recriando a sua linguagem básica. Esta ferramenta está sendo utilizada no desenvolvimento de uma nova versão para o ambiente AulaNet de criação e consumo de cursos para a Web.

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.

Introdução

A evolução da tecnologia Java proporcionou o surgimento de novas abordagens para o desenvolvimento de aplicações para a Web. Estas aplicações possuem diversas partes que podem ser organizadas na forma de componentes de software, aproveitando o potencial oferecido pela aplicação dos conceitos de orientação a objetos. O software pode ser visto como um conjunto de componentes interconectados que cooperam para promover o funcionamento de todo o ambiente.

A linguagem Java incorporou nas suas últimas versões um mecanismo de criação de classes Java que funcionam como programas CGI. Este mecanismo permite a ampliação das funcionalidades do servidores Web que possuem a API dos servlets. Com esta tecnologia, aplicações para a Web escritas em linguagens como o C e Perl podem ser portadas para a linguagem Java.

Este trabalho apresenta o Scriba, uma ferramenta de interpretação de comandos para a elaboração rápida de ambientes em Java para a Web. O Scriba proporciona uma forma rápida de construção de aplicações para a Web que utilizem Java e banco de dados. Através de marcações específicas de sua linguagem, o Scriba permite que os dados sejam montados nas páginas em tempo de carregamento. Além de possuir montagem dinâmica de páginas a partir de templates HTML, o Scriba viabiliza a utilização de linguagens de script definidas pelo usuário através da expansão de sua linguagem ou substituição de seu interpretador de comandos.

Outras tecnologias existentes como o CGILua [Hester, Borges e Ierusalimschy 1998], o ASPy [Schatz 1998] e o PHP [PHP3 1999] apresentam a capacidade de inclusão de comandos de suas linguagens em templates HTML que serão interpretados pelo servidor Web antes do fornecimento da página a máquina que fez a requisição. O ASPy, assim como o Scriba, é implementado utilizando-se a API de servlets Java, mas não possui a sua arquitetura organizada em componentes que permitam a configuração de sua linguagem básica. Já o CGILua e o PHP utilizam a linguagem C para a criação de seus interpretadores, também não aproveitando o potencial oferecido pela utilização dos conceitos de orientação a objetos em sua arquitetura.

O Scriba está sendo utilizado no desenvolvimento da nova versão do ambiente AulaNet [Lucena et. al. 1999] de criação e consumo de cursos pela Web. A utilização do Scriba no AulaNet facilita a sua implementação, já que todos os seus elementos de interface tem montagem em tempo de carregamento, permitindo a sua configuração pelo cliente. A seguir, são apresentados dois principais mecanismos Java utilizados na construção da ferramenta.

Servlets e JDBC

As últimas versões do Java incorporam dois novos mecanismos da linguagem utilizados no desenvolvimento de aplicações de rede e banco de dados denominados servlets e JDBC. O novo aspecto lançado como pacote de extensão para o Java Development Kit 1.1 (JDK 1.1) para ser uma subtituição em Java dos programas CGI (Common Gateway Interface) denomina-se servlet [Siyan and Weaver 1997]. Com os servlets, os usuários podem adicionar novas funções aos servidores HTTP.

Os servidores HTTP que possuem a API de servlets provêem um ambiente com todos os recursos que os servlets precisam para rodar e um modelo de segurança que mantém a integridade do sistema. Os servlets podem conter qualquer função ou pacote Java que não tenha interface gráfica. Eles podem ler ou escrever em arquivos, processar cálculos e comunicação com bancos de dados e gerar arquivos HTML, funcionando como um programa CGI tradicional, com as vantagens fornecidas pela linguagem Java em termos de estruturação em componentes e desempenho.

Para estender a funcionalidade do servidor HTTP, o servlet deve herdar as características da classe **HttpServlet** do pacote **javax.servlet.http**. Para poder processar as requisições feitas ao servidor, o usuário precisa implementar o método **service** que possui parâmetros com referências para todos os dados vindos com a requisição do cliente HTTP. Após o processamento deste método, o usuário poderá gerar uma página HTML como resposta a requisição.

As últimas versões da linguagem Java também incorporam o JDBC [Hamilton, Cattell and Fisher 1997]. O JDBC é uma API para a execução de sentenças SQL sobre um banco de dados qualquer. A API JDBC permite que os usuários escrevam aplicações de banco de dados usando uma interface puramente Java. O JDBC realiza as seguintes operações principais: estabelece uma conexão com um banco de dados, envia uma sentença SQL e processa seu resultado.

O JDBC pode ser utilizado para ter acesso direto a diversos bancos de dados de mercado, mas os drivers JDBC para eles são desenvolvidos por empresas independentes e não são gratuitos. A forma de acesso gratuita a banco de dados a partir do Java oferecida pela Sun é a ponte JDBC-ODBC. Através desta ponte, a API JDBC usa o ODBC para ter acesso ao banco de dados. Esta alternativa é a melhor em termos de custo, mas é um pouco mais lenta que as interfaces diretas com o banco de dados. Esta solução é a adotada pelo Scriba para a comunicação com bancos de dados de usuário.

O Scriba

O Scriba é uma ferramenta que permite a criação de páginas HTML dinâmicas e a definição de classes em Java para a implementação do processamento de aplicações WWW. Usuários que desejam implementar aplicativos para a Web utilizando servlets e linguagem Java encontram no Scriba os elementos necessários para a implementação de todo o aplicativo utilizando páginas HTML e classes Java. As páginas HTML podem conter código Scriba embutido que realiza, entre outras coisas, busca de dados em um banco de dados ODBC qualquer. As classes Java definidas pelo usuário agrupam as funções específicas da aplicação em desenvolvimento. O Scriba trata todas as operações de controle e criação de objetos para o funcionamento dos servlets, comunicação com banco de dados ODBC e operações referentes a interpretação e execução dos comandos de sua linguagem.

Uma das principais características da ferramenta Scriba é a possibilidade de configuração de suas funções através da redefinição de seus componentes pelo usuário. A estruturação do Scriba em componentes de software interrelacionados com interfaces bem definidas permite que seus usuários reescrevam qualquer um de seus componentes. Eles podem adaptar o Scriba a interpretação de uma outra linguagem qualquer. Existem dois tipos de configuração das funções do Scriba: os usuários podem estender a linguagem básica fornecida através da criação de novos comandos no interpretador ou podem definir um interpretador completamente novo para o Scriba.

Como a ferramenta foi criada como tecnologia de base para a reestruturação do ambiente AulaNet, a sua linguagem inicial conta com os comandos necessários a implementação do AulaNet. Os comandos existentes permitem basicamente a montagem de páginas dinâmicas a partir de valores armazenados em bancos de dados, a definição de variáveis locais e globais e a definição de classes de usuário para processamento de valores de formulários.

O Scriba é formado por três componentes básicos: um divisor de tokens, um interpretador de comandos e um receptor de requisições HTTP. A figura 1 mostra a arquitetura do Scriba e as classes que compõem cada um de seus componentes.

O divisor de tokens é composto pelas classes **Parser** e **CodeParser**. Este componente é responsável pela leitura dos templates HTML e pela verificação da existência dos delimitadores de sua linguagem. Quando um token da linguagem é encontrado entre os delimitadores **$*** e ***$**, o divisor passa este token para o interpretador para que as ações referentes ao comando da linguagem contido no token sejam tomadas.

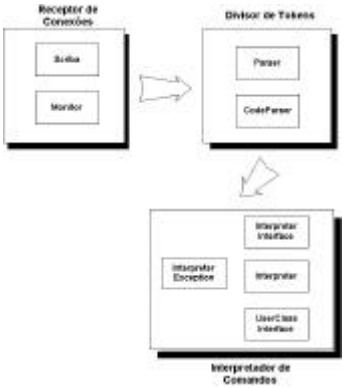

Figura 1: Componentes e classes do Scriba.

O componente interpretador de comandos é composto pelas classes **InterpreterException** e **Interpreter** e pelas interfaces **InterpreterInterface** e **UserClassInterface**. Para permitir que os usuários escrevam interpretadores específicos para suas linguagens, a interface **InterpreterInterface** fornece o padrão que uma nova classe deve seguir para ser definida como interpretador no Scriba. A classe **Interpreter** obedece as especificações de **InterpreterInterface** implementando as ações correspondentes a cada um dos comandos da linguagem. A classe **InterpreterException** trata todos os casos de erro gerados no interpretador, padronizando as mensagens e explicações sobre os problemas encontrados na interpretação dos comandos. A interface **UserClassInterface** define o formato para a criação de todas as classes definidas pelo usuário para processamento sobre valores de formulários.

O receptor de requisições HTTP recebe os parâmetros passados pelo servidor HTTP quando uma página é requisitada usando os métodos GET e POST. Este componente realiza todas as operações necessárias a implementação dos servlets Java e aciona o divisor de tokens para vasculhar o template HTML indicado em busca de tokens da linguagem. Este componente é formado pela classe **Scriba**, que trata requisições do tipo GET e POST, e pela classe **Monitor**, que realiza operações de gerenciamento da memória global da ferramenta.

Quando uma requisição é recebida, a classe **Scriba** armazena os valores passados pelo browser em sua memória interna para posterior consulta no interpretador de comandos. O Scriba procura sempre a declaração dos parâmetros **scribapath** e **scribapage** para saber qual a página a ser rastreada em busca de tokens. Caso não exista indicação para a página a ser ratreada, o Scriba identifica a requisição como sendo feita com método POST e tenta localizar o parâmetro **scribaclass** com o nome da classe de usuário que tratará a requisição. Após a identificação destes parâmetros, a classe **Scriba** cria um objeto da classe **CodeParser** para rastrear o arquivo indicado nos parâmetros ou cria um novo objeto da classe de usuário indicada em **scribaclass**, acionando o seu método **action** para tratar a requisição recebida.

A figura 2 apresenta de forma esquemática o modelo de execução do receptor de requisições HTTP.

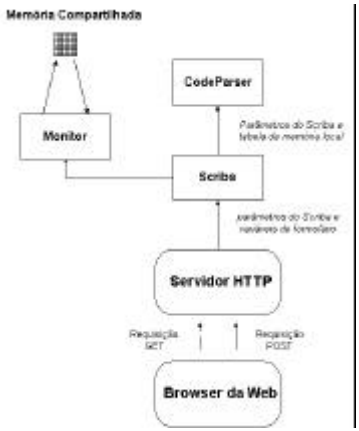

Figure 2: Modelo de execução do componente receptor de requisições HTTP.

Após serem criados pelo objeto da classe **Scriba**, objetos da classe **CodeParser** inspecionam o template HTML fornecido e passam os tokens encontrados para um objeto da classe **Interpreter** que tomará as devidas ações de acordo com os comandos da linguagem. Os comandos da linguagem básica definida para implementação do AulaNet serão apresentados na próxima seção. Esta seção apresenta também o fluxo de execução que cada comando provoca no Scriba e como um usuário deve proceder para estender a linguagem do Scriba, configurando-a de acordo com suas necessidades.

Modo de Operação

Para permitir a implementação de ambientes para a Web através da criação de páginas dinâmicas, o Scriba possui uma linguagem básica inspirada nas necessidades encontradas no desenvolvimento do ambiente AulaNet. A linguagem Scriba conta com comandos de busca de valores em bancos de dados ODBC, definição de variáveis locais ou compartilhadas e criação de classes definidas pelo usuário. Existe também uma sintaxe utilizada para que o usuário possa expandir a linguagem sem precisar redefinir completamente o seu interpretador.

A linguagem embutida nos arquivos HTML obedece as seguintes regras gerais:

1. Para que o divisor de tokens reconheça a existência de comandos embutidos, é necessário que estes estejam delimitados pelo marcador de início de comandos (**$***) e pelo marcador de fim de comando (***$**).

2. O Scriba reconhece comandos separados por **;;** dentro de um template HTML. Caso seja necessária a definição de mais de um comando no mesmo local do arquivo, o usuário poderá usar a construção: **$***<comando1>**;;** <comando2>**;;**…**;;**<comandoN>***$**.

3. Uma linha de comando da linguagem é sempre composta de uma classe de comando, de um nome de comando específico da classe e de um ou mais argumentos delimitados por parêntesis. Por exemplo, a linha de comando **database.select ("aulanet", "select * from tabela")** indica a execução de um comando **select** da classe **database** com dois argumentos: **"aulanet"** e **"select * from tabela"**.

4. O Scriba entende o operador + para concatenação de cadeias de caracteres dentro dos argumentos de um comando.

Os comandos existentes na primeira versão do Scriba são: **database.user**, **database.select**, **database.set.define**, **database.set.repeat**, **database.set.get**, **variable.define**, **variable.get**, **variable.set**, **variable.define.shared**, **variable.get.shared**, **variable.set.shared, class.new**, **language.define**.

O comando **database.user** é responsável pela definição de um nome de usuário e uma senha para serem utilizados na conexão com uma base de dados ODBC. Seu argumento é composto pelo nome do banco de dados cadastrado no ODBC (DSN), o nome do usuário e a senha para conexão. Este comando não retorna valores, criando apenas uma associação entre o banco de dados e um usuário para conexão. Caso o banco de dados a ser utilizado não possua usuário ou senha, não é necessário utilizar este comando antes de realizar qualquer consulta ao banco.

O comando **database.select** permite a execução de uma sentença SQL SELECT sobre um banco de dados cadastrado no ODBC. Seu argumento é composto pelo nome do banco de dados e pela sentença SELECT a ser executada. Este comando retorna um único campo dentre os valores do resultado da sentença SELECT. Por exemplo, o Scriba executaria sem problemas o comando **database.select (“aulanet”, ”SELECT Name FROM Users WHERE password=’xxxx’”)**. O nome do usuário recuperado seria então colocado no lugar da declaração do comando em tempo de execução. Se o retorno da sentença SELECT implicar em mais de um campo ou mais de uma linha, o comando **database.select** só retornará o primeiro campo da primeira linha, ignorando todo o resto dos valores retornados. Por exemplo, caso o comando encontrado fosse **database.select (“aulanet”, “SELECT Name, Account FROM Users)**, possivelmente o retorno da cláusula SELECT seria composto de mais de um registro contendo o nome e o número da conta de cada usuário. Mesmo que a sentença SELECT executada retorne mais de um registro, o Scriba informaria apenas o valor do nome do primeiro registro de usuário. A figura 3 apresenta o comportamento do Scriba ao receber um comando **database.select**.

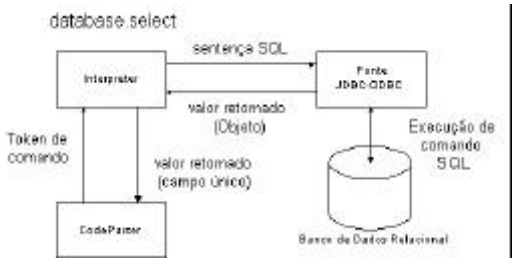

Figura 3: Esquema de execução do comando **database.select**.

O comando **database.set.define** cria uma associação entre um nome definido pelo usuário e uma sentença SELECT que deverá ser executada sobre um determinado banco de dados. Um **set** para o Scriba é um conjunto de valores retornados a partir da execução de uma sentença SQL SELECT. Os argumentos do comando **database.set.define** são: o nome pelo qual o usuário passará a referenciar o **set**, o banco de dados aonde o **set** será gerado e a sentença SQL SELECT que dará origem ao **set**. Este comando não retorna valores, apenas gerando uma associação para um **set** dentro do Scriba.

O comando **database.set.repeat** executa a sentença SQL do tipo SELECT definida para um **set** anteriormente declarado com o comando **database.set.define**. Após a execução da sentença, o comando realiza as operações descritas em seu segundo argumento, caso algum registro seja retornado. Se não forem retornados registros,

o comando executa o que está descrito em seu terceiro argumento. Os argumentos do comando são: o nome do **set**, uma cadeia de caracteres que será repetida para cada registro retornado pela cláusula SELECT e uma cadeia de caracteres que será mostrada caso nenhum registro seja encontrado. Este comando é muito útil para apresentação de dados armazenados em tabelas de forma automática. As cadeias de caracteres do segundo e terceiro argumentos podem conter qualquer tipo de marcação HTML ou concatenação de várias cadeias, incluindo o retorno dos comandos **variable.get** e **database.set.get**.

O comando **database.set.get** possui como argumento o nome de um campo indicado em uma sentença SQL SELECT de um **set** dentro do Scriba. Este comando só pode ser utilizado dentro dos argumentos do comando **database.set.repeat** para recuperar o valor dos campos retornados pela sentença SELECT associada ao **set** definido. A figura 4 mostra como o Scriba executa o comando **database.set.repeat**, que possui uma ou mais chamadas ao comando **database.set.get** em seus argumentos.

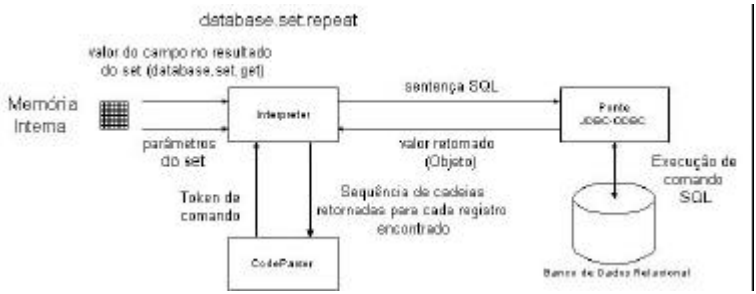

Figura 4: Esquema de execução do comando database.set.repeat e database.set.get.

Para definir variáveis no Scriba o usuário conta com os comandos **variable.define**, no caso de variáveis locais cuja definição só permanece durante a interpretação de uma página, e **variable.define.shared**, no caso de variáveis compartilhadas ou globais, cuja definição permanece mesmo após o término da interpretação da página. Os argumentos destes dois comandos são os mesmos: o nome da variável, o tipo da variável, que pode ser integer, float, double, string ou char, e opcionalmente o seu valor inicial.

Para recuperar os valores das variáveis definidas, o Scriba possui os comandos **variable.get** e **variable.get.shared**, utilizados na recuperação de variáveis locais e globais respectivamente. Estes comandos possuem como argumento somente o nome da variável que se quer saber o valor. Um usuário também pode atualizar o valor de uma variável através dos comandos **variable.set** e **variable.set.shared**, atribuindo novos valores a variáveis locais e globais respectivamente.

Caso o usuário queira ter acesso a algum campo de formulário a partir de sua página, poderá usar o comando **variable.get**. Desta forma, o usuário poderá passar valores entre diferentes templates sem precisar recuperar estes valores em todas as páginas.

O comando **class.new** cria um objeto da classe cujo nome é indicado em seu argumento. O objeto é criado dinamicamente pelo Scriba para realizar operações definidas em Java pelo usuário. Com este comando, o Scriba permite que os usuários programem trechos de código Java para fazer processamentos específicos de sua aplicação. Toda classe criada tem acesso as variáveis de formulários HTML e variáveis definidas localmente, permitindo a implementação de programas para tratamento de dados como acontece com a maioria das linguagens.

Para um usuário criar uma classe que possa ser declarada no comando **class.new**, ele deverá definí-la segundo a interface **UserClassInterface** do componente interpretador de comandos. Ao implementar as características de **UserClassInterface** através da declaração IMPLEMENTS em Java, o usuário saberá quais os métodos que deve criar. Todo processamento do usuário deverá ocorrer dentro de um método denominado **action** que é acionado automaticamente pelo Scriba depois que um objeto da classe é criado. O usuário poderá realizar qualquer tipo de processamento em Java, gerando inclusive trechos de código HTML. A figura 5 mostra o comportamento do Scriba durante a execução do comando **class.new**.

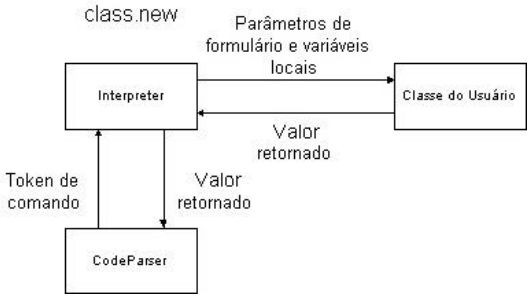

Figura 5: Esquema de execução do comando class.new.

Se for necessário expandir a linguagem do Scriba, o usuário poderá usar o comando **language.define**. Este comando permite que o usuário crie uma classe que implemente a interface **InterpreterInterface** para interpretar uma linguagem proprietária expandida em relação a linguagem original da ferramenta. Esta classe deverá ter um método **action** que receberá um token e o processará, identificando o comando envolvido e executando a sua ação correspondente. Não há limite para criação de classes deste tipo. O comando **language.define** tem como argumentos

um nome para representar o interpretador do usuário internamente no Scriba e o nome da classe que implementa o interpretador.

Para que o Scriba identifique que um comando deve ser interpretado por uma classe externa, o usuário deverá utilizar a sintaxe **language.nome_do_interpretador.comando(parâmetros)**, onde **nome_do_interpretador** é o nome definido para o interpretador do usuário internamente no Scriba, **comando** é o comando a ser passado para a classe do usuário interpretar e **parâmetros** são os parâmetros do comando entendido pela classe do usuário.

A figura 6 mostra como o esquema de expansão da linguagem do Scriba funciona.

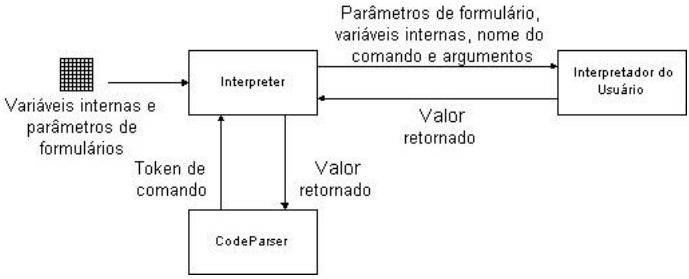

Figura 6: Esquema de expansão de linguagem no Scriba.

O Scriba permite que seus usuários reescrevam as partes principais de dois de seus componentes: o divisor de tokens e o interpretador de comandos. Para garantir a compatibilidade entre as partes dos componentes do Scriba e as partes geradas pelo usuário, são fornecidos os tipos: **Parser** e **InterpreterInterface**. A classe **Parser** deve ser herdada pela classe definida pelo usuário que substituirá a classe **CodeParser**. A classe **CodeParser** serve para quebrar os tokens do template HTML. Caso o tipo de template ou os marcadores da linguagem do usuário necessitem de alteração, basta que ele reescreva a classe **CodeParser** de acordo com a sua nova definição, herdando as características da classe **Parser**.

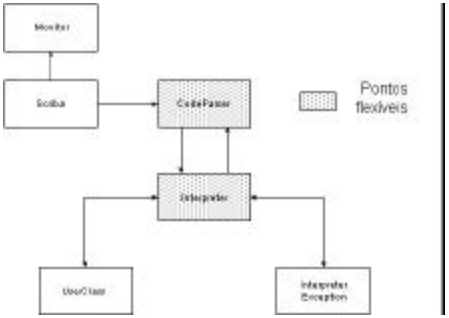

Figura 7: Arquitetura do Scriba e seus pontos flexíveis.

Se a linguagem que deve ser interpretada for completamente nova, o usuário poderá definir uma nova classe **Interpreter** com os comandos da nova linguagem e suas respectivas ações. Esta nova classe deverá implementar a interface **InterpreterInterface**. O novo interpretador deverá receber tokens vindos da classe **CodeParser** e identificar as sentenças de sua linguagem dentro destes tokens, gerando uma saída conveniente para cada tipo de comando interpretado. A figura 7 mostra uma visão da arquitetura do Scriba, identificando seus pontos flexíveis.

Aplicação do Scriba

O Scriba foi criado dentro da filosofia de criação de ambientes cooperativos de software para a Web que utilizem bancos de dados para manutenção de suas informações. O AulaNet é um caso de aplicação desta tecnologia. A arquitetura do AulaNet será reorganizada em componentes orientados a objeto que se integram para o oferecimento dos serviços providos pelo ambiente.

O AulaNet [Lucena et. al. 1999] é um ambiente para a criação e consumo de cursos na Web. Cada professor do ambiente AulaNet pode criar um curso através da selecão de mecanismos de comunicação, cooperação e coordenação. Após escolher os mecanismos existentes no curso, o professor pode inserir o conteúdo que será visualizado pelos alunos. Os alunos podem se comunicar e cooperar para a construção de seu conhecimento e a realização de atividades em grupo.

O AulaNet é formado de um conjunto de páginas HTML, scripts CGILua e um banco de dados MS Access. O CGILua permite a criação de páginas HTML dinâmicas, o processamento das requisições HTTP e possui uma biblioteca para a comunicação com bancos de dados ODBC. Cerca de 90% da funcionalidade do AulaNet é implementada utilizando-se estes componentes. Os outros 10% são compostos de software freeware e dois software comerciais, oferecendo a maior parte das funções de comunicação providas pelo ambiente.

A estratégia adotada para a substituição do CGILua pelo Scriba procura diminuir o número de arquivos existentes no software. Os programas que seriam executados

dentro dos templates para compor a página dinamicamente serão transformados em um ou mais comandos do Scriba, já que eles só utilizam funções de banco de dados. As funções do Scriba mapeam a maioria das ações realizadas para a busca de dados e troca de parâmetros nos templates.

Os programas que tratam os dados vindos de formulários HTML serão substituídos por classes Java e algumas de suas funções trazidas para o código embutido nas páginas. Os programas que acionam diferentes templates de acordo com o resultado de um processamento passarão a acionar apenas um template cuja parte principal é adaptável pela classe de usuário.

A nova versão do AulaNet pretende se beneficiar de alguns aprimoramentos trazidos pelo uso da tecnologia Java, como a portabilidade do programas desenvolvidos. Como não são utilizados componentes de interface gráfica Java (AWT), que possui algumas incompatibilidades dependendo da plataforma aonde rodam, o AulaNet poderá ser colocado em qualquer máquina Java independente do sistema operacional que esteja sendo executado.

Outra melhoria proporcionada pelo emprego do Scriba é o acesso ao banco de dados. Como o AulaNet utiliza um banco de dados Access, alguns problemas de concorrência foram identificados quando vários acessos simultâneos são realizados sobre a base de dados. A utilização de JDBC para acesso ao banco de dados diminui os problemas de concorrência, já que todos os métodos de acesso utilizados são sincronizados pelo mecanismo de monitores implementado pela linguagem Java.

O mecanismo de criação de threads utilizado pelo Java proporcionará uma maior velocidade na execução dos scripts. As linguagens tradicionais como o C utilizam um mecanismo de criação de processos para atender as requisições feitas aos programas CGI, criando várias instâncias do mesmo programa na máquina. O mecanismo utilizado na linguagem Java é a criação de diferentes threads de execução sobre o mesmo programa para cada requisição realizada. A mudança de contexto provocada pela criação de um novo thread em um programa é bem menos dispendiosa que a provocada pela geração de um processo totalmente novo, levando a um maior desempenho.

O número de arquivos da versão do AulaNet baseada no Scriba deve ser da ordem de 50% do número de arquivos da versão atual, baseado na experiência de implementação de 25% do software feita até o momento.

Conclusão

A ferramenta Scriba de criação de aplicativos para a Web visa simplificar a criação e manutenção de aplicativos que utilizem a tecnologia Java. O Scriba se adapta

bem aos tipos de aplicativos que utilizam armazenamento de informações em banco de dados ODBC. Para implementar as suas funcionalidades o Scriba possui uma arquitetura estruturada em componentes.

Para facilitar a criação de páginas dinâmicas o Scriba possui uma linguagem de script para a execução de sentenças SQL e criação de classes de usuário. A linguagem do Scriba possui ainda capacidade de reconhecer comandos estendidos, permitindo que o usuário amplie as suas funcionalidades sem necessidade de alteração dos componentes do Scriba.

Outra característica interessante é a possibilidade de redefinição de componentes da ferramenta. O usuário poderá redefinir toda a linguagem a ser interpretada e os marcadores para a mesma através da redefinição das classes de divisão de arquivos em tokens e interpretação.

O Scriba aproveita as características de desempenho da linguagem Java para execução de programas CGI. O modelo de execução baseado em threads implementado pelo Java permite um maior desempenho dos programas CGI que não necessitam de criação de processos filhos e trocas de contexto dispendiosas.

A próxima versão do AulaNet já está sendo implementada com o Scriba e apresenta uma diminuição da ordem de 50% do número de arquivos utilizados na sua implementação. A implementação do AulaNet em Scriba está possibilitando a estruturação do ambiente na forma de componentes de software interrelacionados, facilitando a identificação das partes que compõem o software e a sua documentação.

Algumas melhorias já podem ser identificadas para uma próxima versão da ferramenta Scriba, de acordo com os resultados obtidos em sua utilização:

- **Extensão da linguagem básica.** A linguagem utilizada no Scriba pode ser ampliada para acomodar maiores funcionalidades que evitem a necessidade de implementação de classes feitas pelo usuário.
- **Inclusão de servidores TCP/IP.** Um mecanismo semelhante baseado em componentes para a criação de clientes e servidores genéricos TCP/IP baseados em Java já está sendo desenvolvido para o Scriba. Esta atualização permitirá que usuários do Scriba possam especificar o protocolo de comunicação entre cliente e servidor. O usuário poderá se concentrar na implementação das funções necessárias a execução dos comandos do protocolo, deixando para o Scriba o controle e manutenção das informações relevantes a conexão e interpretação do protocolo.

Pretende-se ampliar o Scriba para que ele possua uma forma de representação gráfica de aplicativos que permita a criação de uma ferramenta para a geração

semi-automática de programas. Com isso, seus usuários estarão aptos a especificar o aplicativo graficamente e gerar as páginas que o compõem visualmente, evitando o contato com o código prematuramente.

Referências Bibliográficas

HAMILTON, G., CATELL, R., & FISHER, M. JDBC Database Access with Java – A Tutorial and Annotated Reference. Addison-Wesley, 1997.

HESTER, A., BORGES, R., & IERUSALIMSCHY, R. Building Flexible and Extensible Web Applications with Lua. In: *WebNet'98-World Conference of the WWW, Internet & Intranet, 1998*, Association for the Advancement of Computing in Education, Charlottesville, VA, 1998.

LUCENA, C. J. P., FUKS, H., MILIDIÚ, R., LAUFER, C., BLOIS, M., CHOREN, R., TORRES, V., & DAFLON, L. AulaNet: Helping Teachers to Do Their Homework. In: Multimedia Computer Techniques in Engineering Education Workshop, Technische Universitat Graz, Graz, Austria, 16-30, 1998.

PHP3. PHP: Hypertext Preprocessor. Internet WWW page, at URL <http://www.php.net> (versão atual de 22 de Maio de 1999)

SCHATZ, B. ASPy. Internet WWW page, at URL <http:// www.dstc.edu.au/aspy/> (versão atual de 19 de Novembro de 1998)

SIYAN, K. S. & WEAVER, J. L. Inside Java. New Riders Publishing, 1997.