Usando o *Framework* JLearningServices para Instanciar Serviços Síncronos para Ambientes de EAD

Letícia Rafaela Rheinheimer, Sérgio Crespo C. S. Pinto

PIPCA – Mestrado em Computação Aplicada – UNISINOS
93.022-000 – São Leopoldo – RS – Brasil

{leticiar,crespo}@exatas.unisinos.br

Resumo. *JLearningServices é um Framework educacional que busca contribuir no sentido de diminuir a carência de ferramentas específicas para EAD utilizando a tecnologia de Frameworks para a geração de serviços síncronos dirigidos a ambientes virtuais de aprendizagem baseados na Web. Em seu modelo são utilizados Design Patterns para proporcionar uma melhor representação dos seus vários hot-spots e frozen-spots. Um modelo inicial do JLearningServices foi apresentado em artigo publicado no XII SBIE, em 2001. Este novo artigo apresenta a evolução do modelo do Framework e mostra como proceder para realizar sua instanciação. O artigo traz um exemplo de instanciação para criação de um chat.*

Palavras-chave: *Frameworks*, *Design Patterns*, EAD, comunicação síncrona.

Abstract. *JLearningServices is an educational Framework that aims to contribute for reducing the need of specific distance learning tools using Frameworks to generate synchronous services applied to virtual learning environments based on the Web. Framework's model uses Design Patterns technology to provide a better representation of its many hot-spots and frozen-spots. An initial model of JLearningServices was presented in the Brazilian Symposium on Education and Computer Science, in 2001. This new paper presents Framework's model evolution and shows how instantiate the Framework. This paper brings an instantiation example for creating a chat application.*

Key words: Frameworks, Design Patterns, distance learning, synchronous communication

1. Introdução

Em ambientes virtuais de aprendizagem, ferramentas síncronas e assíncronas são cada vez mais utilizadas para auxiliar o processo de cooperação e colaboração entre os aprendizes [Crespo, Fontoura e Lucena 1998] [Lucena e Fuks 2000]. Como ferramentas síncronas temos *chats* e ICQ, dentre outras, e como ferramentas assíncronas temos *e-mail*, listas, fórum, etc. Estas ferramentas, em sua maioria, já existem bem antes do surgimento dos ambientes que oferecem suporte à educação baseada na *Web* e não foram projetadas com enfoque educacional; são oferecidas como recursos auxiliares integrados aos ambientes, apenas para possibilitar a comunicação entre os usuários.

O *Framework* JLearningServices apresenta uma proposta para suprir a necessidade de ferramentas de qualidade voltadas ao ensino. Este *Framework* busca trazer, para ferramentas de diferentes tipos de comunicação, características e recursos que contribuam de maneira mais específica no processo de ensino/aprendizagem. O objetivo não é desenvolver novamente ferramentas que já existem há muito tempo, como o *chat*, mas sim dotar este tipo de ferramenta de características de EAD para que possa ser utilizada com maior proveito em sistemas da área. Também busca-se desenvolver um *software* que permita um alto nível de reutilização, o que pode ser conseguido de maneira eficiente combinando o uso das tecnologias de *Frameworks* e *Design Patterns*.

Um modelo inicial do *Framework* [Rheinheimer e Crespo 2001] contempla os requisitos funcionais e não funcionais de aspectos pedagógicos para ferramentas síncronas em ambientes de EAD. Este modelo descreve os diferentes tipos de comunicação síncrona (um-para-um, um-para-muitos e muitos-para-muitos) bem como características e recursos que poderiam ser utilizados pelos diversos tipos de aplicação. Estes recursos e características foram definidos através de estudo feito sobre algumas ferramentas já existentes, através de um processo de engenharia reversa.

Este artigo apresenta um modelo atualizado do *Framework* e a forma de instanciá-lo, trazendo um exemplo prático de instanciação.

O artigo está organizado como segue: na seção 2 é feita uma breve descrição das tecnologias de *Frameworks* e *Design Patterns*; a seção 3 apresenta o modelo do *Framework* JLearningServices; a forma de instanciação do *Framework* é descrita na seção 4; a seção 5 traz um exemplo de instanciação para construção de uma aplicação de *chat* e a conclusão é apresentada na seção 6.

2. *Frameworks* e *Design Patterns*

Design Patterns são soluções genéricas para problemas recorrentes em Engenharia de *Software* [Gamma, Helm e Vlissides 1995]. Cada padrão identifica classes e instâncias participantes com seus papéis, colaborações e a distribuição de responsabilidades, sendo que estes elementos podem ser customizados para resolver um problema num contexto particular. O uso de padrões deve contribuir para reutilização e flexibilização no desenvolvimento de *software* Orientado a Objetos.

Segundo Pree [Pree 1995], *Frameworks* unificam sistemas de *software* para um domínio específico e constituem uma construção semi-acabada de blocos de códigos prontos para uso, em associação com uma arquitetura global, obtida pela composição e interação entre os blocos. Os aspectos de um *Framework* que não são projetados para adaptação (*frozen-spots*) são representados por meio de métodos *template* e constituem o *kernel* do *Framework*. Um *hot-spot* é uma classe abstrata que não possui implementação e deve ser especializada (customizada) para necessidades específicas da aplicação a ser gerada. A especialização pode ser realizada tanto por herança como por delegação, dependendo da forma como os *hot-spots* foram planejados [Crespo 2000].

Uma vez que, tanto *Frameworks* quanto *Design Patterns* promovem flexibilização e reutilização, pesquisas reconhecem a necessidade da integração dessas duas tecnologias [Crespo 2000] [Fontoura, Haeulser e Lucena 1998][Rheinheimer e Crespo 2001][Wirfs-Brock, Wilkerson e Wiener 1990] [Pree 1999].

3. Modelo do JLearningServices

Diversos sistemas de comunicação síncrona foram observados com o objetivo de se fazer uma engenharia reversa e, a partir disto, definir as características pertinentes ao *kernel* do *Framework* e aquelas que deveriam constituir os *hot-spots*.

O processo de engenharia reversa realizado resultou no desenvolvimento de um primeiro modelo, que serviu de base para a modelagem do *Framework*. Em seguida, com base nas informações obtidas, foi possível definir as classes do *kernel* e dos *hot-spots* do *Framework* [Rheinheimer e Crespo 2001].

Ao longo do tempo, o modelo criado recebeu algumas revisões e alterações. Uma última versão do modelo pode ser encontrada em [Rheinheimer 2002].

Nas subseções a seguir, serão descritas as partes do modelo referentes à estrutura servidor, à estrutura cliente e aos *hot-spots* do *Framework*. Essas partes são ilustradas através de Diagramas de Classes simplificados (diagramas mais detalhados encontram-se em [Rheinheimer 2002]).

3.1 Servidor

A classe *Server* (Figura 1) assume o papel de aplicação servidora. Esta classe está relacionada com as conexões, representadas pela classe *Connection*. O servidor pode receber e controlar diversas conexões.

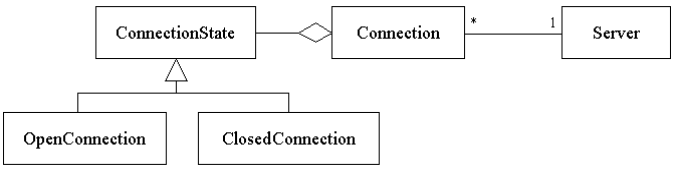

Figura 1 – Servidor e conexões

O *Design Pattern State* [Gamma, Helm e Vlissides 1995] foi utilizado para definir os possíveis estados de uma conexão: aberta ou fechada. Este padrão comportamental permite que, de acordo com o estado em que um objeto se encontra, o comportamento desse objeto seja modificado. Isso pode permitir que se remova conexões "mortas", fazendo com que o servidor trabalhe de maneira mais eficiente. Este padrão também foi utilizado para definir os possíveis estados do servidor (em funcionamento, esperando o recebimento de conexões e fechado), de modo a garantir um maior controle sobre o seu funcionamento.

O servidor controla as aplicações clientes através da implementação de dois *Design Patterns*: *Observer* [Gamma, Helm e Vlissides 1995] e *Chain of Resposibility* [Gamma, Helm e Vlissides 1995].

O *Design Pattern Observer* (Figura 2) é utilizado para notificar os clientes envolvidos em uma sessão de *chat*, por exemplo, a respeito de alterações que ocorram em um algum cliente específico. Este padrão comportamental permite que, tendo ocorrido a modificação de um objeto, os demais sejam notificados e atualizados em tempo de execução.

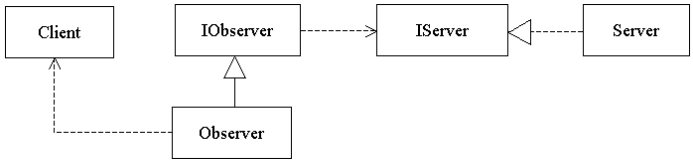

Figura 2 – *Design Pattern Observer*

Já o *Design Pattern Chain of Responsibility* (Figura 3) é utilizado para controle das requisições de *login*, necessárias para que um sistema seja acessado. Este é um modelo comportamental que permite que várias requisições sejam feitas e acumuladas até que possam ser atendidas.

Figura 3 – *Design Pattern Chain of Responsibility*

Neste contexto, existe ainda a classe *UserInformation*, que está ligada ao servidor e é responsável pelo armazenamento dos dados dos usuários, que precisam ser consultados pelo servidor no momento de efetuar o *login*.

As classes referentes às aplicações clientes são descritas a seguir.

3.2 Cliente

Os tipos de aplicações clientes (Figura 4) que podem ser geradas com o uso do *Framework* foram definidos utilizando o *Design Pattern Application Types* [Fayad, Schmidt e Johnson 1999], que separa aplicações em categorias de acordo com seu comportamento e características.

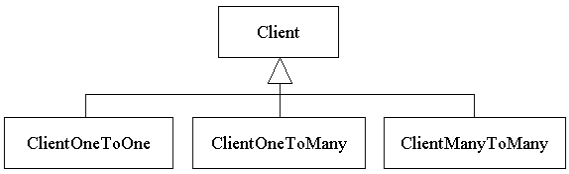

Figura 4 – Tipos de aplicações clientes

A classe *Client* possui especializações de acordo com o tipo de comunicação utilizado pela aplicação que será gerada: um-para-um (semelhante ao *Instant Messenger*, encontrado em http://americaonline.com.br/aim/), um-para-muitos (semelhante ao ICQ, disponível em http://www.mirabilis.com/) ou muitos-para-muitos (mesmo tipo de comunicação de um *chat*).

Aplicações muitos-para-muitos (representadas pela classe *ClientManyToMany*) podem ter uma ou mais salas, definidas pela classe *Room* (Figura 5) para interação entre diversos participantes (classe *Participant*). As salas podem ser pré-definidas ou criadas por algum participante. O número máximo de participantes em uma sala também pode ser configurado. Uma sala pode ser bloqueada ou liberada para interação. Existe também a possibilidade de manter conversa privada entre participantes, o que é tratado pela classe *PrivateRoom*.

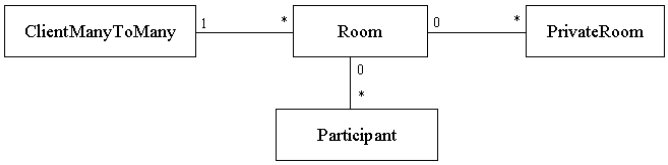

Figura 5 – Sala de *chat* e participantes

Uma sessão de *chat* (ou de qualquer outro tipo de aplicação) pode ser gravada em um arquivo de *log* (classe *SessionLog*) e posteriormente manipulado de acordo com o interesse do usuário - manter o conteúdo parcial/completo do *log* ou deletá-lo. A classe *SessionLog* (Figura 6) foi especializada para contemplar a gravação desses arquivos por qualquer um dos tipos de aplicação: as subclasses de *SessionLog* devem implementar outros recursos para o trabalho com os arquivos de *log*, cada uma de acordo com suas particularidades.

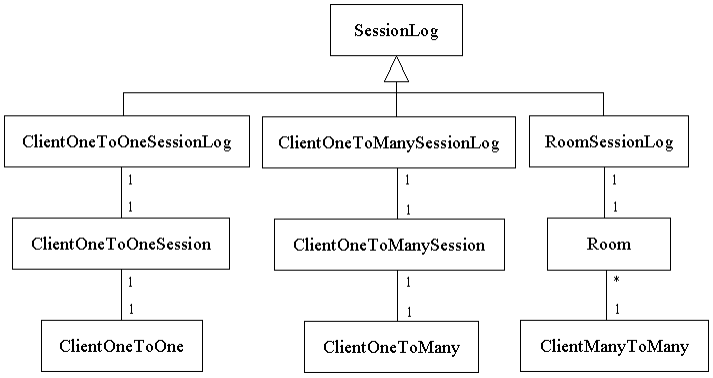

Figura 6 – *Logs* dos diferentes tipos de aplicações

As classes que formam a estrutura servidor/cliente descritas até o momento formam o *kernel* do JLearningServices. Também foram definidos seus *hot-spots*, sendo que alguns podem ser utilizados para todos os tipos de cliente e outros são referentes apenas a aplicações muitos-para-muitos.

3.3 Hot-spots existentes no *Framework*

Terminado o processo engenharia reversa, foram identificados pontos de um sistema de comunicação síncrona que podem ser flexibilizados, bem como alguns recursos que poderiam ser interessantes no contexto de EAD, e que passaram a constituir os *hot-spots* do JLearningServices:

- **Definição de papéis para os participantes** – criação de papéis que permitam aos participantes realizar certas operações tais como:
  - Mediador – possibilidade de definir o papel de um mediador em uma sala de *chat*, que teria controle sobre os demais participantes, ou não ter esta figura presente;
  - Aluno – papel que pode ser definido para os demais participantes de uma sala de *chat*, coordenados ou não por um mediador, e que poderiam interagir segundo as permissões definidas por funcionalidades associadas a este papel.
- **Definição de funcionalidades** – relacionamento de uma série de funcionalidades a cada papel criado, como, por exemplo:
  - Permissão para retirar um participante de uma sala;
  - Execução de aplicações em outros clientes – um professor pode, por exemplo, mandar abrir uma URL na máquina dos alunos;
  - Possibilidade de conversa reservada – permitir que dois usuários troquem mensagens em uma seção de *chat* sem que outros possam ler.
  - Uso de recursos gráficos – uma aplicação pode ter ícones para identificar os usuários ou utilizar botões com ações definidas; o usuário pode fazer desenhos e enviá-los.
  - Formatação de mensagem – definição, pelo usuário, de tamanho da fonte, cor e estilo do texto de uma mensagem a ser enviada.
  - Uso de som – possibilidade de oferecer, se desejado, algum recurso para uso de som nos diferentes tipos de aplicações.

  A partir disso, foi definido um amplo conjunto de *hot-spots* para aplicações clientes.

Para uso de todos os tipos de clientes foi criada a classe *Resource* (Figura 7), que corresponde a recursos que podem ser utilizados nos diferentes tipos de aplicações: sons, imagens, ícones, botões de ação, texto, aplicações e elementos gráficos (classes *Sound*, *Image*, *Icon*, *ActionButton*, *Text*, *Application* e *Graphic*, respectivamente).

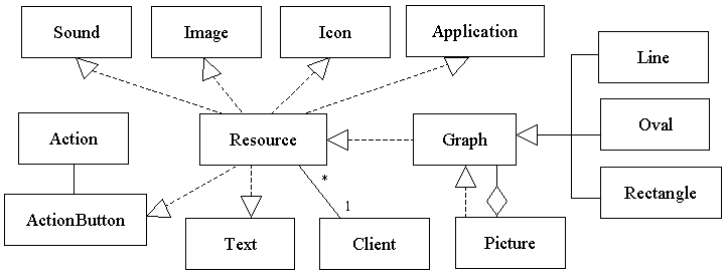

Figura 7 – *Hot-spots* de todos os tipos de aplicações

Existe também a classe *Action*, associada à classe *ActionButton* para permitir a definição de uma ação para um botão. Para a definição dos elementos gráficos foi utilizado o *Design Pattern Composite* [Gamma 1995]. Assim, pode ser feito um desenho (classe *Picture)* composto por qualquer combinação de elementos que especializam a classe *Graphic*.

Especificamente para aplicações muitos-para-muitos, foi definida a classe *Role* (Figura 8), que diz respeito aos papéis que podem ser assumidos por um participante. Dois papéis já existentes são mediador e aluno, mas outros podem ser definidos.

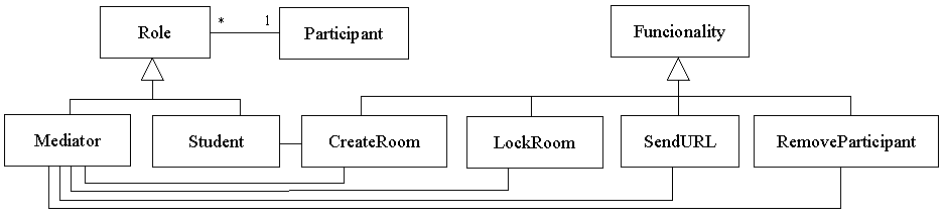

Figura 8 – *Hot-spots* das aplicações muitos-para-muitos

Foram também modeladas algumas das diversas funcionalidades (classe *Funcionality*) que podem estar associadas aos papéis (classe *Role*): *RemoveParticipant*, *SendURL*, *CreateRoom*, *LockRoom* e *DefineMaxParticipants*, que correspondem respectivamente, às possibilidades de remover um participante, enviar uma URL para que seja aberta nas máquinas dos demais participantes, criar uma nova sala, bloquear uma sala para que ninguém mais possa entrar nela e definir o número máximo de participantes para uma sala). Assim como no caso dos papéis, outras funcionalidades podem ser criadas e associadas a um ou mais papéis.

A próxima seção mostra como utilizar o *Framework*.

4. Como instanciar o *Framework*

Java foi a linguagem escolhida para a implementação do *Framework*, devido à sua independência de plataforma e por oferecer recursos para comunicação e interfaces gráficas.

As classes referentes ao *kernel* do JLearningServices possuem os métodos básicos necessários para o funcionamento dos serviços síncronos definidos. Já as classes referentes aos *hot-spots* são interfaces que devem ser implementadas para definir os comportamentos desejados para a aplicação que está sendo criada.

Para a instanciação do *Framework*, são compiladas as classes referentes ao *kernel* em conjunto com aquelas classes referentes aos *hot-spots* que foram escolhidos e implementados pelo usuário do *Framework*. A subseção a seguir descreve como realizar a implementação de *hot-spots*.

4.1 Como implementar *hot-spots*

Os *hot-spots* são classes abstratas ou de interface que possuem métodos sem implementação. Quando queremos utilizar um *hot-spot* na criação de uma aplicação, devemos estender sua classe abstrata ou implementar sua interface, conforme for o caso.

Por exemplo: um *hot-spot* existente no modelo do JLearningServices é *Funcionality*, que representa funcionalidades que podem ser associadas a determinados papéis (classe *Role*) assumidos por participantes (classe *Participant*) de uma sala de *chat*. Para criar uma nova funcionalidade, é preciso construir uma classe que implemente a interface *Funcionality*, codificando os métodos definidos em *Funcionality* e criando outros métodos conforme desejado.

A Figura 9 mostra o código da interface *Funcionality*.

```
public interface Funcionality {

  public void defineFuncionality();

}
```

Figura 9 – Classe *Funcionality*

A Figura 10 descreve a estrutura de uma funcionalidade chamada SendURL e que já faz parte do *Framework.*

```
public class SendURL implements Funcionality {

  public void defineFuncionality() {
    // o código desejado vai aqui
  }

  // podem ser criados outros métodos aqui

}
```

Figura 10 – Estrutura da funcionalidade *SendURL*

O mesmo processo é válido para todos os *hot-spots*.

Na seção seguinte é mostrado um exemplo de aplicação síncrona criada com o JLearningServices.

5. Exemplo de instanciação

Para exemplificar o uso do JLearningServices, foi criada uma pequena aplicação de *chat*, com apenas uma sala sem outros recursos além de texto. Foi estendida a classe *Server* para criar uma nova classe chamada *ChatServer* (Figura 10).

```
public class ChatServer extends Server {

  public ChatServer() {
    super();
    (…)
    while (true) {
      try {
        connect();
        String IP = conn.getConnection().getInetAddress().getHostAddress();
        System.out.println("Accepted from: " + IP);
        serverThread = new Thread() {
        public void run() {
          try {
            (…)
            while (true) {
              String msg = receive();
              broadcast(msg);
            }
          }
          catch(IOException ioe) {
            System.err.println("IOException: " + ioe.getMessage());
            }
            finally {
              (…)
              try {
                disconnect();
              }
              catch(IOException ioe) {
                System.err.println("IOException: " + ioe.getMessage());
              }
            }
          }
        };
        serverThread.start();
      }
      catch(IOException ioe) {
        System.err.println("IOException: " + ioe.getMessage());
      }
      System.out.println("Waiting...");
    }
  }

}
```

Figura 10 – Estrutura de código principal da classe *ChatServer*

A seguir, é mostrada uma tela com a execução da aplicação servidora (Figura 11).

Figura 11 – Exemplo de execução de aplicação servidora

Para a aplicação cliente foi definida a programação das classes *ClientManyToMany* e *Room* a fim de definir a interface gráfica do *chat* e seu comportamento. Esta interface é mostrada na figura a seguir (Figura 12).

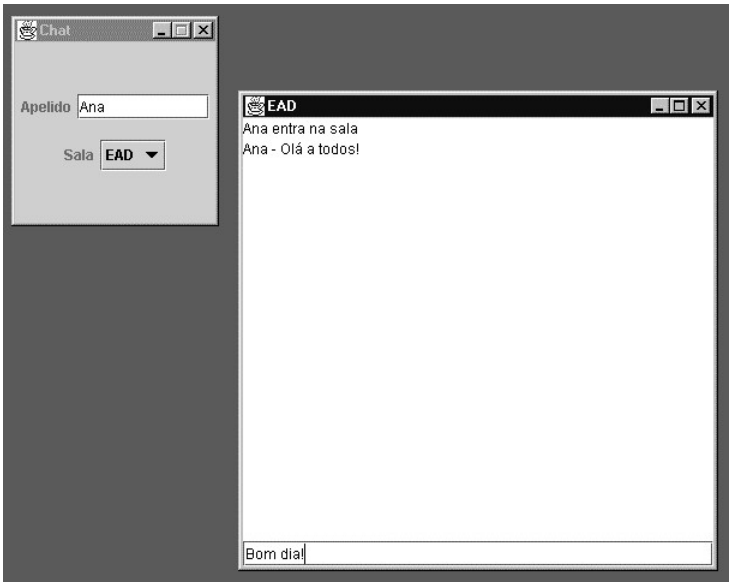

Figura 12 – Exemplo de execução de aplicação cliente

Este foi um exemplo simples de como o JLearningServices pode ser utilizado para a geração de aplicações síncronas. A seguir, será mostrado como estender o *Framework* para que um agente externo possa acessar as mensagens trocadas em sessões de *chat*.

5.1. Plugando um agente externo na aplicação cliente

Um recurso implementado para o *Framework* foi um editor de arquivos de *log* (classe *RoomSessionLog*), o qual pode ser utilizado como está ou ajustado conforme desejado. Uma visão da interface deste editor é mostrada a seguir (Figura 13).

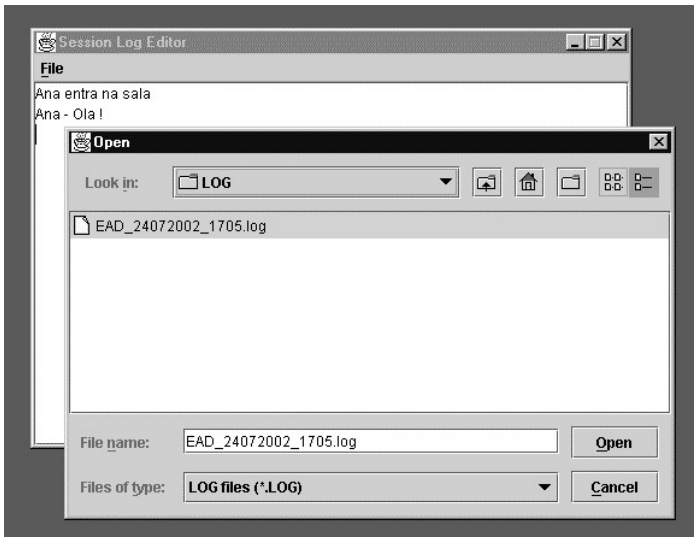

Figura 13 – Editor de arquivos de *log*

Este editor de *log* para aplicações de *chat* pode ser modificado para que se torne um recurso ainda mais interessante: um monitor para as mensagens de *chat*, de forma que possa ser utilizado para verificar/controlar o que acontece nas salas utilizadas.

A classe onde o editor é implementado (*RoomSessionLog*) possui um método chamado *runEditor()*, responsável pela execução da ferramenta. Pode-se estender, na classe *RoomSessionLog*, o método *runEditor()*, de forma que suporte esta nova funcionalidade. Também seria necessário implementar, na aplicação servidora, um método que envie as mensagens para este monitor de mensagens, além de enviá-las às aplicações dos demais participantes. Para isto, pode-se aproveitar os métodos que a classe Server possui com relação ao *Design Pattern Observer*.

Na próxima sessão, são apresentadas algumas conclusões sobre o artigo.

6. Conclusão

A principal contribuição do trabalho é o desenvolvimento de um *Framework* para a área de EAD e que permite a geração de serviços síncronos em seus diferentes tipos de comunicação, promovendo flexibilização e reutilização de *software* através do uso de *Frameworks* e *Design Patterns*. Buscou-se dotar este tipo de ferramenta de recursos/características para EAD, de forma que sua utilização em ambientes de ensino seja mais proveitosa.

Com o uso das tecnologias de *Frameworks* e *Design Patterns*, obtém-se reutilização em várias fases de desenvolvimento: análise, projeto e codificação. Para desenvolver a parte do *Framework* referente a aplicações muitos-para-muitos, por exemplo, várias ferramentas de *chat* foram analisadas. Desta análise, resultou a definição das características que constituiriam o *kernel* e os *hotspots*. Após algumas melhorias, obteve-se uma ferramenta que, para gerar diferentes aplicações de *chat*, necessita apenas de alguns ajustes. Não é mais necessário o trabalho de análise e projeto, apenas algum trabalho de codificação. Também pode-se acrescentar recursos ao *Framework* sem necessitar de grandes modificações. Se fosse utilizado o processo convencional de desenvolvimento de *software*, para cada *chat* desejado haveria um novo trabalho de análise, projeto e codificação, o que causaria uma grande quantidade de retrabalho. Em resumo, o desenvolvimento do *Framework* gera uma quantidade maior de trabalho quando da sua criação, poupando muito trabalho quando da sua utilização para desenvolver diferentes aplicações.

O ambiente foi desenvolvido segundo uma metodologia de desenvolvimento em espiral. Nas fases de implementação e instanciação do protótipo utilizou-se a linguagem Java. Houve a preocupação de tornar o protótipo apto a ser executado tanto em ambientes *Windows* quanto em ambientes *UNIX* ou *Linux*. Para isso, foram utilizados recursos oferecidos pela linguagem Java para tornar o ambiente independente de plataforma, no que diz respeito ao uso de arquivos e diretórios.

Um trabalho de revisão e complementação tanto do modelo quanto do código está sendo feito para que a ferramenta venha a se tornar mais completa e eficiente.

7. Referências

[Crespo 2000] Crespo, S. "Composição em WebFrameworks", Tese de Doutorado. PUC-Rio, Departamento de Informática, 2000.

[Crespo, Fontoura e Lucena 1998] Crespo, S., Fontoura, M. e Lucena, C. "A Web-Based Educational Environments Comparison Using a Conceptual Model Compatible with the EDUCOM/IMS Platform", Brazilian Symposium on Education and Computer Science (SBIE'98), Fortaleza, Brazil, 1998 (in Portuguese).

[Fayad, Schmidt e Johnson 1999] Fayad, M., Schmidt, D. e Johnson, R. "Building Application Frameworks: Object-Oriented Foundations of Framework Design", New York: John Wiley & Sons, 1999.

[Fontoura, Haeulser e Lucena 1998] Fontoura, M., Haeulser, E. e Lucena, C. "A Framework Design and Instantiation Method Based on Viewpoints", In: Monografias em Ciência da Computação. MCC/98. PUC-RIO, Departamento de Informática, 1998.

[Gamma, Helm e Vlissides 1995] Gamma, E., Helm, R., Johson, R. e Vlissides, J. "Design Patterns: Elements of Reusable Object-Oriented Software", Reading: Addison-Wesley, 1995.

[Lucena e Fuks 2000] Lucena, C. e Fuks, H. "Professores e Aprendizes na Web: A Educação na Era da Internet", Rio de Janeiro: Clube do Futuro, 2000.

[Pree 1995] Pree, W. "Design Patterns for Object-Oriented Software Development", Reading: Addison-Wesley, 1995.

[Pree 1999] Pree, W. "Rearchitecturing Legacy Systems: Concept & Case Study", WICSA'99 (First Working IFIP Conference on Software Architecture), San Antonio, Texas, 22-24 February 1999.

[Rheinheimer e Crespo 2001] Rheinheimer, L. e Crespo, S. "JLearningServices: Um *Framework* para Serviços Síncronos em Ambientes para EAD", XII Simpósio Brasileiro de Informática na Educação (SBIE'2001), Espírito Santo, UFES, Brasil, 2001.

[Rheinheimer 2002] Rheinheimer, L. "JLearning Services: Um *Framework* para Serviços Síncronos em Ambientes para EAD", Trabalho de Conclusão de Curso, Centro de Ciências Exatas e Tecnológicas, UNISINOS, 2002.

[Wirfs-Brock, Wilkerson e Wiener 1990] Wirfs-Brock, R., Wilkerson, B. e Wiener, L. "Designing Object-Oriented Software", Englewood Cliffs: Prentice-Hall, 1990.