UNIVERSIDADE DO VALE DO RIO DOS SINOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE INFORMÁTICA

JLearningServices: Um *Framework* para Serviços Síncronos em Ambientes para EAD

Letícia Rafaela Rheinheimer

Prof. Sérgio Crespo Coelho da Silva Pinto
Orientador

Monografia submetida como requisito parcial para a obtenção do título de Bacharel em Informática.

São Leopoldo, junho de 2002

Resumo

A Internet popularizou a utilização de serviços de comunicação síncrona, sendo que muitos destes estão sendo utilizados como ferramentas de comunicação em ambientes para educação baseada na *Web*. Em virtude de o campo educacional ser muito dinâmico, torna-se necessária a utilização de tecnologias de Engenharia de *Software* para a prototipação e geração rápida de sistemas computacionais.

Este trabalho apresenta o desenvolvimento de um *Framework* educacional que busca contribuir no sentido de diminuir a carência de ferramentas específicas para EAD. JLearningServices utiliza a tecnologia de *Frameworks* para a geração de serviços síncronos para comunicação um-para-um, um-para-muitos e muitos-para-muitos, dirigidos a ambientes virtuais de aprendizagem baseados na *Web*. JLearningServices também utiliza um conjunto de *Design Patterns* em seu modelo, de forma a proporcionar uma melhor representação dos seus vários *hot-spots* e *frozen-spots*.

Palavras-chave: *Frameworks*, *Design Patterns*, informática e educação.

Abstract

Internet popularized the use of synchronous communication services. Many of these services are being used in Web-based education environments. Education area is very dynamic, so the use of Software Engineering technologies becomes necessary for fast prototype development and computer systems generation.

This work presents the development of an educational Framework that must be a contribution to provide specific tools for distance learning. JLearningServices uses Frameworks technology to generate synchronous services for one-to-one, one-to-many and many-to-many communication for Web-based education environments. JlearningServices also uses a set of Design Patterns in the prototype design to provide better representation of its hot-spots and frozen-spots.

Keywords: *Frameworks, Design Patterns, Web-based education.*

Sumário

Índice de Tabelas

Índice de Figuras

1 Introdução

A educação/treinamento baseada na *Web* usa a WWW como meio para publicação do material didático, aplicação de tutoriais, aplicação de provas e testes e comunicação com estudantes. Também compreende o processo de uso da *Web* como veículo de comunicação para a apresentação de aulas a distância (conferência multimídia). As tecnologias de educação/treinamento baseada na *Web* estão em pleno uso, utilizando uma série de ferramentas que auxiliam os processos de cooperação, coordenação e comunicação [9][39].

Ferramentas que possibilitem a cooperação entre os aprendizes tornam-se cada vez mais utilizadas nos ambientes virtuais. Estas podem ser tanto síncronas (*chats*, ICQ, dentre outras) como assíncronas (*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* (Ensino a Distância - EAD). A maioria dos ambientes virtuais utiliza, para possibilitar a cooperação e comunicação entre os aprendizes, artefatos de *software* que não foram projetados com enfoque educacional. Pôde-se perceber, ao conhecer alguns ambientes no decorrer deste trabalho, que ferramentas como *chat*, *e-mail*, etc, são oferecidas como recursos integrados aos ambientes, mas sem grande relevância no contexto educacional: são apenas auxiliares, quando poderiam contribuir de maneira mais específica no processo de ensino/aprendizagem; existe a integração/adaptação de ferramentas já prontas (*chat*, por exemplo), em lugar de se fazer um estudo na área e desenvolver algo novo, com características específicas para o ensino.

Existe uma grande preocupação com a qualidade na área de EAD, em diversos sentidos, inclusive no que diz respeito a ferramentas que auxiliem o professor e que propiciem uma melhor interação entre os alunos participantes desse processo de ensino/aprendizagem [35][39][40][54]. Assim, vêm-se buscando novas formas de suprir essas necessidades e dar ao EAD maior qualidade, segurança e confiabilidade.

Este trabalho vem trazer uma contribuição neste sentido, oferecendo ferramentas de serviços síncronos em seus diferentes tipos de comunicação (um-para-um, um-para-muitos e muitos-para-muitos) voltadas para a área do ensino.

A dissertação está organizada da seguinte forma: o capítulo 2 apresenta os principais conceitos e a evolução do EAD, especialmente o EAD baseado na *Web*.

No capítulo 3, são descritas as tecnologias de *Frameworks* e *Design Patterns*, utilizadas para o desenvolvimento do *Framework* JLearningServices, bem como sua aplicação na área de EAD .

O capítulo 4 descreve o *Framework* JLearningServices. É apresentado um estudo sobre sistemas de comunicação síncrona em diversos ambientes na *Web*, de forma a permitir elicitar os principais requisitos para a definição do *Framework*. Também é ilustrado e explicado o seu modelo de classes e objetos, destacando os *hot-spots* e o *kernel*. Um exemplo de instanciação do *Framework* é encontrado no capítulo 5.

No capítulo 6, é apresentada a conclusão do trabalho e, por fim, seguem as referências bibliográficas.

2 Ensino a Distância

Neste capítulo, são apresentados os principais conceitos relacionados ao EAD, bem como um pouco sobre sua história, as tecnologias utilizadas e um pouco mais sobre EAD baseado na Web.

Nos últimos anos, o EAD tem se tornado um tópico de destaque em educação. Vem crescendo o número de conferências, simpósios, projetos e publicações relacionadas ao assunto. Com o advento da *Web*, a prática do EAD mudou dramaticamente nas últimas décadas. Tem se buscado, além de inovações tecnológicas, inovações pedagógicas que tornem este processo de ensino/aprendizagem algo mais do que uma simples cópia do modelo tradicional [39][40][54].

2.1 Definição de EAD

A palavra "distância" tem diversos sentidos (pode ser distância geográfica, distância no tempo, etc). O termo "ensino a distância" tem sido aplicado a diversos programas que atendem a numerosas audiências através de uma variedade de meios: material impresso, telecomunicações e outros [54].

Foram definidas quatro categorias para o ensino, levando-se em conta as variáveis tempo e espaço: mesmo tempo e mesmo lugar, tempos diferentes e mesmo lugar, mesmo tempo e lugares diferentes, tempos diferentes e lugares diferentes [54].

O ensino tradicional acontece em um mesmo lugar e mesmo tempo. Tempos diferentes e mesmo lugar significa que uma mesma aula é oferecida em diferentes horários para os estudantes, mas no mesmo lugar (um laboratório de informática, por exemplo).

As duas últimas categorias dizem respeito ao ensino ocorrendo em lugares diferentes. A instrução pode ocorrer ao mesmo tempo e em lugares diferentes quando é usado um sistema de telecomunicação como a TV, por exemplo, sendo chamada de EAD síncrono. Mas também pode ocorrer em lugares e tempos diferentes, sendo chamada de EAD assíncrono. É o caso de alunos acessando um curso na *Web*, onde eles mesmos definem o local e o horário para realizar este acesso.

O EAD tem sido definido através de várias perspectivas ao longo dos anos. Eis algumas definições encontradas na literatura:

- "Ensino por meio de mídia impressa ou eletrônica para pessoas engajadas em um processo de aprendizado em tempo e local diferente dos instrutores e dos outros aprendizes" – Lucena e Fuks [39];
- "Uma atividade planejada e sistemática que envolve a escolha, preparação didática e apresentação de materiais de ensino bem como a supervisão e suporte ao aprendizado do aluno e que é alcançado transpondo-se a distância física entre aluno

e professor através de pelo menos um meio técnico apropriado" – Rudolf Manfred Delling [13];

- "Uma forma de ensino que possibilita a auto-aprendizagem com a mediação de recursos didáticos sistematicamente organizados, apresentados em diferentes suportes de informação, utilizados isoladamente ou combinados e veiculados pelos meios de comunicação, dando destaque a elementos de abertura à democratização do ensino e autonomia do indivíduo" – Decreto nº 2494 de 10/02/1998, que regulamenta o artigo 80 da Lei nº 9394 de 20/12/1996 (Lei de Diretrizes e Bases da Educação Brasileira) [35].

Desmond Keegan [33] identificou cinco elementos principais que compõem uma definição de EAD:

- A separação quase permanente de professor e aprendiz (o que aumenta o processo de aprendizagem);
- A influência de uma organização educacional tanto no planejamento quanto na preparação dos materiais didáticos e na provisão de serviços de suporte aos alunos;
- O uso de meios técnicos (material impresso, áudio, vídeo ou computador) para unir professor e aprendiz e manter o conteúdo do curso;
- A provisão de comunicação em duas vias, assim o estudante pode se beneficiar de ou mesmo iniciar um diálogo;
- A ausência quase permanente de um grupo (o que aumenta o processo de aprendizagem). As pessoas geralmente são ensinadas individualmente e não em grupos, mas há a possibilidade de eventuais encontros com propósitos tanto didáticos como de socialização.

Garrison e Shale [23] argumentaram que a definição de Keegan não era adequada à realidade e nem a possibilidades futuras, e definiram três critérios considerados essenciais para o processo de EAD:

- EAD implica que a maior parte da comunicação entre professor e aluno(s) ocorra de forma não contígua;
- EAD deve envolver comunicação em duas vias entre professor e aluno(s) com a finalidade de facilitar e dar suporte ao processo de ensino/aprendizagem;
- EAD usa a tecnologia para mediar a comunicação em duas vias.

Todas estas definições correspondem a uma visão tradicional do EAD, que estão sujeitas a alterações causadas por mudanças tecnológicas e sociais [54].

2.2 Histórico do EAD

O EAD parece uma idéia nova, mas os conceitos que formam sua base têm mais de um século. Ele iniciou com o ensino por correspondência em fins do século XVIII, meados do século XIX e início do século XX, e tinha por objetivo a formação profissional e capacitação das pessoas para o exercício de certas atividades e para desenvolver determinadas habilidades [35][54].

Otto Peters [47] diz que, de certa forma, o EAD não é algo novo porque tem base em formas de estudo presenciais; porém, a ênfase com que essas formas de estudo são combinadas e integradas é diferente.

A seguir serão apresentadas algumas datas e iniciativas importantes, que ajudarão a delinear a evolução histórica do EAD no Brasil, na América Latina e em alguns outros lugares do mundo.

No ***Brasil*** [35][39], até o princípio do século XX, as escolas por correspondência tinham caráter profissionalizante. Através de revistas e jornais, ofereciam vários cursos como datilografia e radiotécnica.

Nas décadas de 60 e 70, foi utilizado o rádio na educação, quando se formaram núcleos de recepção (grupos de pessoas, principalmente na área rural, que se reuniam para acompanhar o curso em volta do rádio, sob a orientação de um monitor).

A partir da década de 70, a Televisão Educativa popularizou o EAD com a preparação dos exames supletivos, formando a Rede de Televisões Educativas.

Na década de 80, o uso das tecnologias educacionais voltou a ser valorizado, agora tendo o computador como um dos seus instrumentos centrais. A Secretaria Especial de Informática (órgão ligado à Presidência da República) criou a Comissão Especial de Educação com o objetivo de definir normas e diretrizes para a área da informação. Nos anos de 1981 e 1982, foram realizados o I e II Seminário Nacional de Informática na Educação – envolvendo, pela primeira vez, pessoas ligadas diretamente ao processo educacional. Em 1983, foi criado o Projeto EDUCOM (Educação com Computadores); com ele, foram criados cinco projetos piloto que visavam o desenvolvimento de pesquisas e a disseminação do uso de computadores no processo de ensino.

No final da década de 80, houve uma disseminação de aparelhos receptores de televisão em todo o país (cerca de 80% da população possuía uma TV). Começava o desenvolvimento do Sistema Nacional de Telecomunicação, via satélite, bem como a crescente utilização da informática.

Na década de 90, a utilização de transmissões via satélite em canal aberto permitiu um processo interativo nacional entre professores e especialistas, segundo propostas de um grupo interministerial para atualização de docentes do ensino fundamental. Em dezembro de 1995, foi criada a Secretaria de Educação a Distância – SEED/MEC.

No fim dos anos 90, a Internet começou a abrir espaço na educação, encurtando distâncias para os cursos e programas de EAD. O Ministério da Educação vem apoiando o uso de computadores nas escolas com mais de 100 alunos, através do Programa Nacional de Informática na Educação.

Algumas das principais iniciativas em EAD no Brasil [35] foram:

- ***Instituto Universal Brasileiro*** e ***Instituto Radiotécnico Monitor*** (1941);
- ***Movimento Educação Base*** – convênio entre a CNBB e o Ministério da educação.

ampliou o circuito e estruturou o Serviço de Supervisão Pedagógica como apoio aos trabalhos das tele-salas;

- ***Fundação Educacional e Cultural Padre Landell de Moura (FEPLAM)*** – desde 1967 vem realizando, no Rio Grande do Sul, atividades de formação profissional, educação rural e educação para o trânsito através de programas e cursos por correspondência, emissoras de rádio, televisão e telepostos;
- ***Instituto de Radiofusão Educativa da Bahia (IRDEB)*** – no âmbito nacional, promove e programa a utilização de teleducação; através de acordo com o Canadá e a Holanda, foram desenvolvidas diversas atividades de aprimoramento do pessoal em rádio. O IRDEB vem utilizando-se de tecnologias na área de supletivo para a zona rural, construção civil, treinamento de professores alfabetizadores e educação da comunidade;
- ***Telecurso 2º Grau*** – lançado em 1978, através da Fundação Roberto Marinho, associada ao Sistema Globo de Televisão, preparava alunos para exames de 2º Grau por meio de programas de televisão. Hoje, com o nome TELECURSO 2000, procura atingir os mesmos objetivos. Pode ser acompanhado por fascículos vendidos em bancas de jornal, que são elaborados, impressos e distribuídos pela Fundação Roberto Marinho com apoio da FIESP, entre outras instituições;
- ***SENAI, SENAC e SENAR*** - desenvolvem um dos melhores modelos de tecnologia educacional na área de treinamento profissional.Além de preparar para a vida profissional, utilizam como tecnologias básicas as séries metódicas, o ensino individualizado, os meios audiovisuais, o videocassete e a instrução programada;
- ***TV Escola*** – programa do Ministério da Educação e do Desporto, voltado para a formação, aperfeiçoamento e valorização dos professores da rede pública por meio de um canal de televisão dedicado exclusivamente à educação. Exibe programas para formação de professores, bem como vídeos pedagógicos para uso em sala de aula, documentários e ficções produzidos no Brasil e no exterior. Complementam o programa: Revista da TV Escola e cadernos do professor (duas coletâneas de 6 e 5 cadernos, respectivamente), além do Guia da TV Escola;
- ***Canal Futura***, emissora do sistema NET – a parceria entre Alternativa Consultoria, RBS e Canal Futura resultou na produção e veiculação de 24 cursos e 485 programas (informação válida para o ano de 1999).

Hoje, existem diversas universidades que desenvolvem atividades acadêmicas em nível de graduação e pós-graduação a distância. Entre outras, temos: Universidade Federal de Santa Catarina (UFSC), Universidade Federal do Rio Grande do Sul (UFRGS), Universidade Federal de Pernambuco (UFPE), Universidade Federal da Bahia (UFBA), Universidade de São Paulo (USP), Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS) e também do Rio de Janeiro (PUC-RJ), Universidade Anhembi Morumbi de São Paulo e Universidade do Vale do Rio dos Sinos (UNISINOS). Elas desenvolvem atividades de pesquisa e/ou cursos de EAD [26][35][41].

Na ***América Latina*** [35], a primeira experiência em Televisão Educativa aconteceu em El Salvador, na década de 60. A televisão foi usada para uma reforma global do sistema, não sendo imposta uma estrutura tradicional, o que resultou num rendimento maior dos alunos que participavam da TV Educativa (superando os demais alunos).

Foi criada, em 1977, na Costa Rica, a Universidade Estatal a Distância (UNED), que tem caráter autônomo e atende alunos em idade superior a 25 anos. Também em 1977, começou a funcionar a Universidade Nacional Aberta (UMA) da Venezuela, que oferece cursos, em sua maioria, profissionalizantes.

Na Argentina, diversas universidades desenvolvem cursos de graduação a distância.

O México possui algumas instituições onde são desenvolvidos programas de Educação Superior Aberta: Universidade Pedagógica Nacional (Sistema de Educação a Distância), Universidade Nacional Autônoma do México (Sistema Universitário Aberto), Institutos Tecnológicos (Sistemas Tecnológicos Abertos) e Instituto Politécnico Nacional (Sistema Aberto de Ensino). Neste país, o EAD é aplicado de maneira formal através da *Radioprimaria* e *Telesecundaria*.

No Chile, existem as Escolas Radiofônicas. O Equador também possui um sistema desse tipo.

Na Bolívia, existem instituições que fazem educação usando rádio, materiais impressos e promotores, trabalhando com as línguas indígenas quéchua, aymara e guarani.

Os maiores problemas que estes e outros países da América Latina que trabalham com EAD enfrentam são a falta de recursos e a falta de comunicação entre as instituições dentro de cada país e a nível internacional.

Em ***âmbito internacional*** [35][47][54], os meios de EAD mais utilizados são a correspondência, o rádio e a televisão. Várias universidades a distância foram criadas, muitas delas investindo em novas tecnologias para promoção do EAD.

Na Suécia, o estudo por correspondência começou em 1833. Na Inglaterra, começou em 1840 e, três anos depois, este tipo de instrução foi formalizado. Nos Estados Unidos, em 1883, já havia cursos por correspondência em nível acadêmico e, em 1905, a *Calvet School* adotou o ensino por correspondência para as crianças de Baltimore (seguida pela Áustria, em 1914, e pela Austrália, em 1915). A Austrália notabilizou-se pelos cursos por correio, juntamente com o ensino por meio das estações de rádio e pelo telefone (cursos ministrados por universidades ou centros especializados em EAD).

O rádio foi utilizado nos Estados Unidos na década de 20 (pelo menos 176 estações foram construídas em instituições educacionais). Em 1940, o rádio tornou-se veículo de EAD no Canadá. Bélgica e Holanda têm tradição histórica em educação radiofônica.

Em 1930, nos Estados Unidos, programas experimentais de televisão foram produzidos na Universidade de Iowa, sendo oferecidos via *broadcast* apenas na década de 1950. No fim dos anos 1980 e início dos 1990, o desenvolvimento de sistemas de comunicação de fibra ótica permitiu a expansão de sistemas de áudio e vídeo de alta qualidade na educação.

Nos Estados Unidos, em meados de 1980, cursos por computador começaram a ser oferecidos. No início de 1998, ¾ de todas as escolas primárias dos Estados Unidos estavam conectadas à Internet.

A tendência de desenvolvimento do EAD no Canadá é semelhante à dos Estados Unidos.

A Universidade da África do Sul é a mais antiga universidade a distância. É uma universidade por correspondência que aperfeiçoou os métodos das escolas por correspondência comerciais e que hoje passa por um processo de transformação e renovação, visando sanar falhas e adequar seus sistemas de ensino e aprendizagem às novas circunstâncias. Hoje é a maior escola superior do país e está entre as aproximadamente dez maiores universidades a distância do mundo.

Em 1969, em Londres, foi fundada a primeira universidade aberta. Ela iniciou suas atividades em 1971 como escola superior autônoma e financiada essencialmente pelo governo, que decidiu instalar a universidade exclusivamente como escola superior para adultos, fazendo uso continuado da televisão e do rádio, realizando o desenvolvimento profissional de cursos, enfatizando o atendimento aos telestudantes, buscando criar cursos de extensão e desenvolvendo estratégia para o uso da comunicação digital. Hoje ela está entre as maiores universidades do Reino Unido, da Europa e do mundo, e serviu como incentivo para o surgimento de instituições similares na Alemanha, Japão, Canadá e, até mesmo, Sri Lanka e Paquistão.

A Universidade de Educação (a distância) da Espanha funciona desde 1973. Possui Centros Associados Regionais em número suficiente para evitar o deslocamento dos alunos. O EAD é complementado com ensino presencial (individualmente ou em grupos), além de proporcionar a convivência entre professores/tutores e os alunos, servindo também para a avaliação das provas a distância e desenvolvimento de atividades artísticas, culturais e recreativas.

Criada em 1974, a Universidade da Alemanha iniciou suas atividades em 1975, sendo financiada pelo Estado. Teve como primeiro objetivo diminuir a sobrecarga das universidades tradicionais (estavam sempre superlotadas, obrigando os candidatos a esperarem durante anos até que sua admissão pudesse ser efetuada). Após essa fase, buscou-se atender donas de casa, deficientes e aqueles que exercem atividades profissionais. Num terceiro momento, a preocupação foi o atendimento científico a pessoas já graduadas. Seu modelo de ensino teve grande aceitação. A universidade mantém uma rede de 49 centros de estudo Alemanha e 3 centros de estudo na Áustria, na Suíça e na Hungria.

A Universidade Central por Rádio e Televisão da China iniciou suas atividades em 1979. Constitui-se de uma teleuniversidade em Pequim e 28 teleuniversidades nas províncias. Esse sistema é organizado em subprefeituras, centrais pedagógicas de distritos e classes televisivas locais. Sua criação teve por objetivo melhorar o nível cultural e científico geral de toda a nação, formando mais pessoas mais rapidamente e a custos menores. Hoje essa universidade atende a alunos classificados em três grupos: pessoas que exercem uma atividade profissional e querem qualificar-se, formandos de escolas secundárias que receberam a indicação de uma vaga na universidade a distância e egressos das escolas que estão desempregados.

Em 1982, ocorreu a união de 66 universidades dos Estados Unidos para fazer uso comum da TV a cabo, do satélite e da teleconferência. A Universidade Estadual de Oklahoma foi escolhida como sede desse consórcio, cabendo a ela as funções de coordenação, assessoria, marketing, treinamento, ajuda na produção de programas e venda de fitas de vídeo (o que não impedia as demais universidades de atuarem isoladamente). Hoje, mais de 250 universidades pertencem ao quadro de associados pagantes.

Existem várias outras universidades a distância. Em [47] pode ser encontrado um estudo sobre algumas delas.

2.3 Tecnologias para EAD

As tecnologias apontadas pela literatura [27][35][39][54] como as principais utilizadas em EAD através dos tempos são:

- ***Material impresso*** – historicamente, é o meio instrucional básico. Seu uso pode ser como meio mestre, meio complementar ou meio suplementar. Possui alto valor para a auto-instrução. O meio impresso que um professor bem sucedido costuma desenvolver apresenta o conteúdo e orienta a aprendizagem: levanta questões para despertar interesse, seleciona linguagem e abordagens adequadas ao nível do aprendiz, é construído para um público específico no contexto de um curso ou disciplina, apresenta os conteúdos com leveza e em linguagem simples e direta, encaminha para a consulta a outras fontes, entre outros;
- ***Correspondência*** – com o auxílio do correio, são enviadas ao aluno as lições junto com formulários de avaliação do tema estudado. Após estudar, o aluno preenche o formulário de auto-avaliação e o envia, pelo correio, para o professor/tutor ou centro educativo. Após corrigidas, as provas são devolvidas ao aluno com as correções e comentários para recuperação do conteúdo. Em alguns casos, podem ser formados grupos locais para reuniões presenciais (com auxílio de um tutor), de forma a exercitar a participação em grupo, o senso crítico e apresentação de problemas encontrados na aprendizagem;
- ***Rádio*** – assume papel ativo no processo da instrução e pode ser usado em combinação com outros meios. Seu potencial é transmitir a informação como estímulo auditivo. Permite relação unidirecional entre o emissor (rádio) e o receptor (ouvinte). É um meio de baixo custo e pode ser utilizado de forma coletiva. Os alunos podem ser reunidos em um local onde recebem programas de rádio didáticos, a orientação de um tutor e complementação das transmissões através de material impresso. Pode também ser desenvolvida a reflexão grupal. Este meio pode ser substituído por fitas gravadas, que também permitem que debates e suas conclusões sejam gravados e enviados para outros grupos receptores;
- ***Televisão*** – como recurso instrucional, tem muitas vantagens: fornece recursos visuais e movimento em um único formato; não intimida os alunos; aumenta a motivação, pois a visualização é mais interessante; é eficaz na introdução, revisão e sumarização de conceitos; possibilita melhoria da visão da realidade, transpondo visualmente a outros ambientes, e permite a visualização do fato ou objetivo sob diversos ângulos. A transmissão pode ser realizada em circuito aberto (canais habituais, públicos ou privados) ou fechado (que apresenta altos custos e dificuldades de produção específicas). Não apresenta grandes possibilidades de controle, sendo necessário incluí-lo no planejamento da utilização. É um meio unidirecional, devendo ser utilizado com outros meios complementares;
- ***Audiocassete*** – é utilizado como meio complementar para o ensino. Permite que o aluno controle o tipo e a duração das mensagens, assim como pode repeti-las quantas vezes desejar e for necessário. Se comparado a outros meios, tem custos

mensagem, fundamentada na fala dos receptores, abrindo possibilidades de interatividade;

- ***Videocassete*** – permite utilização individual ou coletiva, e os custos de produção são relativamente altos. Algumas vantagens são: facilidade no manuseio do equipamento, qualidade da imagem e do som, possibilidade de repetição instantânea da imagem, possibilidade de uso individual ou em grupos e adaptação às mais diversas clientelas. Mas ainda é necessária a elaboração de um projeto pedagógico coerente que considere o vídeo como objeto de estudo e como recurso para ensino. O grande desafio da equipe produtora de programas de videocassete é levar os alunos a pensar;
- ***Telefone*** – permite comunicação bilateral interativa, proporcionando resposta instantânea às colocações do interlocutor. Pode ser utilizado como meio privado ou semiprivado de comunicação a distância, como recurso de instrução para levar informação a um receptor. Apesar de sua limitada utilização nos processos educativos, ele constitui-se num recurso complementar ou suplementar. Possui alto grau de controle tanto do emissor quanto do receptor;
- ***Fax*** – possibilita a transmissão de mensagens escritas, e sua grande vantagem é fornecer registro escrito do que é transmitido, com custos mais reduzidos do que o uso do telefone;
- ***Computador*** – várias são as suas formas de utilização, desde a avaliação formativa de alunos até a administração do ensino. Antes do surgimento da *Web*, seu uso em EAD estava muito ligado à produção de multimídia na forma de CD-ROM. Nas últimas três décadas, o uso da Informática no ensino seguiu dois modelos bem sedimentados: Sistemas de Instrução assistida por Computador (SIAC) e Sistemas Tutores Inteligentes (STI). Esses modelos definem um paradigma na interação entre o estudante e o sistema que tem sido criticado por não possibilitar o trabalho em equipe através de redes de computadores, o que limita a exploração de teorias pedagógicas que defendem a interação social como facilitador do processo de aprendizagem;
- ***Internet*** – a conhecida WWW permite, nos dias de hoje, a transmissão de todo o tipo de mídia de ponto a ponto do planeta. Com seu uso surgiu o que é chamado de "multimídia interativa". E mais recentemente, com os ambientes virtuais de estudo, surgiu um novo paradigma de Informática na Educação em que a relação entre sujeitos atinge um novo patamar, baseado na troca de informações "plena", que possibilita a criação de comunidades virtuais que interagem através de redes em debates sincronizados e/ou assíncronos. Essa nova abordagem aponta em direção às novas diretrizes pedagógicas sócio-construtivistas-interacionistas. Esta e outras questões relativas ao uso da Internet no EAD serão vistas mais detalhadamente na seção 2.4;
- ***Satélite*** – é uma tecnologia de comunicação que permite a utilização de televisores para fins educativos, viabilizando inúmeros canais para veiculação de filmes, cursos, programas, etc dirigidos a públicos especialmente interessados. Estes são canais dedicados, por assinatura. Outra aplicação do satélite é feita através de serviços oferecidos por empresas como a EMBRATEL, por exemplo, que possibilita a realização de videoconferências, distribuição de vídeos promocionais, educativos ou de treinamento, acesso à Internet, entre outros.

2.4 EAD Baseado na *Web*

Segundo Lucena e Fuks [39], a questão chave da implantação de novas tecnologias de suporte à educação é transformar o paradigma tradicional da educação como fábrica, com horários pré-determinados e funções que devem ser exercidas rotineiramente, para a educação como entretenimento, onde são promovidos o interesse e a motivação para buscar a informação desejada. De todas as tecnologias para EAD descritas, nenhuma tem maior potencial do que a Internet. Seu uso oferece algumas ***vantagens*** [40][54], devido a certas características:

- ***Flexível*** – a qualquer hora e a partir de qualquer lugar pode-se acessar o curso, desde que existam os recursos mínimos, como computador conectado à rede e programa de navegação na Internet; os conteúdos do curso e a maior parte das ferramentas para comunicação podem ser acessados em diferentes plataformas (não importa se o aluno está utilizando um PC/Windows ou um computador Macintosh);
- ***Dinâmica*** – é facilmente atualizável e possibilita o contato direto, a qualquer momento e por razões imediatas, à troca com professores/tutores, equipe de apoio do curso e outros colegas;
- ***Aberta*** – além do ambiente virtual criado para o curso, é possível realizar pesquisa em diversos *sites* na Internet, ampliando conceitos e informações oferecidas na estrutura do curso e possibilitando que os alunos percorram bibliotecas e outros *sites* internacionais, sem custos adicionais, desde que não existam barreiras lingüísticas;
- ***Sem fronteiras internacionais*** – pode-se atingir pessoas presentes em qualquer lugar do mundo, desde que não haja obstáculos da língua, para colaborar na solução de dúvidas, participação em fóruns e debates, etc;
- ***Amigável*** – requer do aluno mínimos conhecimentos de navegação, como a manipulação do *browser* e familiaridade com os recursos comunicativos da Internet; promove o aprendizado ativo e facilita o envolvimento intelectual com o conteúdo do curso;
- ***Adaptável às necessidades do aluno*** – o EAD com uso da WWW é adequado à formação continuada de profissionais que não podem interromper suas atividades de trabalho e também não podem se deslocar para participar de cursos presenciais, além de poupar tempo e dinheiro em eventuais viagens que seriam necessárias em alguns cursos.

Contudo, existem algumas ***limitações*** [54] que devem ser consideradas:

- ***Problemas de acesso*** – em áreas rurais e de baixo poder socioeconômico, muitos alunos podem não ter acesso à Internet, mesmo que ela esteja disponível de alguma forma; limitações de banda podem tornar difícil o acesso a tecnologias mais avançadas de vídeo, multimídia e gráficos mais pesados;
- ***Congestão de tráfego*** – este pode ser um problema comum em certos períodos do dia e para estudantes que usam modems mais lentos;
- ***Foco em tecnologia***, em lugar do conteúdo;
- ***Novo paradigma*** – o instrutor terá de aceitar um novo paradigma de educação: o de facilitador e guia do aprendizado, em vez de um simples fornecedor de informações; por outro lado, deve lidar com as dificuldades dos alunos que têm medo da tecnologia;

- ***Tópicos que não se adaptem bem ao uso do computador*** – outro problema relacionado seria a falta de recursos/treinamento/assistência para desenvolver certos conteúdos ou mesmo a falta de tempo devido a estas tarefas serem muito trabalhosas;
- ***Responsabilidade*** – os estudantes devem sentir-se motivados e manter organização e disciplina para participar, o que é difícil para alguns;
- ***Demora do feedback*** – enquanto respostas a questões costumam ser instantâneas em cursos presenciais, nesta modalidade de ensino o *feedback* pode levar horas ou até mesmo dias para ser recebido.

De acordo com Lucena e Fuks [39], o uso apropriado da *Web* no EAD deve observar, entre outros ***aspectos***:

- Aspectos ***pedagógicos*** – referem-se ao ensino e aprendizado. Alguns pontos importantes são: a incorporação de metodologias e estratégias pedagógicas no ambiente de EAD para reduzir as restrições impostas pelo meio; o desenvolvimento de estratégias para dar "poder" ao aprendiz, encorajando o trabalho independente e em grupo, e também a interação; providenciar soluções para dar apoio ao aprendiz contra o fenômeno de "perdido no ciberespaço"; dar acesso a conteúdos e atividades ao longo do tempo, para evitar que alguns alunos fiquem muito atrasados ou muito adiantados, o que pode causar uma sensação de falta de coerência;
- Aspectos ***tecnológicos*** – devem levar em conta o fenômeno exponencial crescente da velocidade dos computadores e da redução de seu custo. A velocidade da rede ainda será um obstáculo por mais alguns anos (por exemplo, para transmissão de imagens pesadas). O medo de certos alunos diante dos equipamentos e suas frustrações devido às dificuldades também devem ser considerados;
- Aspectos ***organizacionais*** – deve-se levar em conta que a preparação de um curso via *Web* demanda muito mais tempo do que a preparação de um curso convencional, amplificando também os desafios típicos de um ambiente de aprendizado convencional. Outro ponto importante é o suporte permanente ao curso: um *site* que não seja animado e atualizado constantemente se torna desinteressante e pouco atrativo; se, além disso, não for oferecido suporte tecnológico e humano no momento do curso, haverá dificuldade em obter sucesso;
- Aspectos ***institucionais*** – é preciso o reconhecimento, por parte das instituições de ensino, do trabalho realizado pelo corpo docente para preparar e manter seus cursos atualizados. Isto demanda muito tempo e esforço e, se não for valorizado, ninguém vai querer fazer este trabalho.

A *Web* proporciona novas formas de interatividade entre professores e alunos, e alunos entre si: horário de atendimento virtual (horário pré-determinado de atendimento pela Internet), horário de atendimento arbitrário (qualquer horário), interação aluno-aluno, teste e avaliação de desempenho baseados na *Web*, possibilidade de retorno constante do aluno, assim como do aperfeiçoamento do relacionamento interinstitucional entre professores e alunos e da administração e marketing do programa (divulgação de amplo material sobre o curso, para ajudar na tomada de decisão do aluno de fazê-lo) [39].

Mas existem alguns ***desafios*** [3][40] na aprendizagem *on-line*: as diferentes mídias que apóiam o EAD, anteriores à Internet, tinham um alcance limitado, relacionado ora à qualidade das relações interativas entre aprendizes e professores, ora

aos resultados alcançados e mesmo a abrangência do público, entre outros aspectos. Com o surgimento da Internet, viu-se crescer também o interesse por cursos e novas opções educacionais a distância, ampliando a oferta, interesses e formando novos públicos.

O aluno de EAD mudou bastante seu perfil, adequado à nova mídia utilizada. Os que não se estimulavam a fazer um curso a distância por correspondência, mesmo percebendo a necessidade de atualização profissional e cultural, estimulam-se agora pelo fato de estas possibilidades estarem na Internet. O aspecto da ampliação de ofertas de cursos adequada a este público diferenciado é um fato que desafia o desenho instrucional dentro de uma abrangência dificilmente controlável.

Sob outra perspectiva, é importante observar o lado do professor/tutor/autor virtual, que tem que se adequar e reformular posturas didático-pedagógicas, de forma a garantir qualidade educacional na sua tarefa de desenvolver cursos on-line em um ambiente pouco familiar a ele. Segundo Maria Luiza Belloni [3], a formação do professor deve torná-lo apto a atuar em três grandes dimensões: pedagógica, tecnológica e didática, preparando-o para a inovação tecnológica e suas conseqüências pedagógicas, bem como para sua formação continuada numa perspectiva de formação ao longo da vida.

2.4.1 Ensinando Através da WWW

São inúmeras as escolas, universidades e centros de formação que oferecem cursos a distância e que usam os recurso tecnológicos para "entregar" a informação ao aluno. Esta solução inova no sentido de manter os alunos em suas casas, conectados a alguma forma de comunicação, minimizando, inclusive a falta de espaço físico para atender a todos. No entanto, é apresentado um lado conservador, pois a abordagem educacional é baseada no sistema tradicional de ensino (em vez de se transmitir a informação via giz e quadro negro ou via livro, a informação agora é entregue via rede de computadores) . Por outro lado, tanto os computadores quanto as redes de computadores oferecem ótimos recursos para a criação de novas abordagens educacionais [39][40].

Do mesmo modo que existe uma grande variedade de abordagens presenciais, existem diversas ***abordagens de EAD*** [39][40], que podem ser classificadas basicamente em três modalidades:

- ***Broadcast*** – usa meios tecnológicos para passar a informação aos aprendizes e não pressupões nenhum tipo de interação do aluno com os meios que transmitem a informação;
- ***Virtualização da escola tradicional*** – reproduz, via telemática, a sala de aula tradicional;
- ***Promoção da construção do conhecimento*** – cursos que usam os recursos da *Web* como um sistema completo de apresentação e discussão do conteúdo.

Neste contexto, pode-se distinguir dois enfoques que diferem na forma de representação do conhecimento e criação do significado e, portanto, na forma como se dá o aprendizado: o objetivismo e o construtivismo.

A filosofia do objetivismo é a de que a informação no mundo exterior é independente da mente e pode ser caracterizada em termos objetivos, para ser transmitida e comunicada do professor para o aluno. O conteúdo do curso é eminentemente organizado pelo professor para ser apresentado ao aluno.

No construtivismo, cada aluno constrói individualmente uma representação de conhecimento própria, interna e pessoal, que é indexada por sua experiência particular. É proposta uma nova estratégia para aprender, onde é necessário projetar-se várias maneiras de o aluno sintetizar, organizar e reestruturar a informação. A última modalidade educacional citada segue este enfoque. Ela possui um alto custo, se comparada às outras; o professor não consegue atender mais do que vinte alunos; ela implica em mudanças no processo educacional que mesmo a educação presencial ainda não conseguiu implementar. Por outro lado, esta abordagem de EAD utiliza a rede de forma mais eficiente, explorando as verdadeiras potencialidades dessa nova tecnologia e apresenta-se como um recurso que pode facilitar o processo de mudança, rompendo restrições de tempo, espaço e seqüência. Ela implementa uma solução educacional de alta qualidade, permitindo a preparação de cidadãos aptos a participarem da sociedade do conhecimento, e nos permite entender como propiciar as condições para o aprendiz construir conhecimento contextualizando sua realidade de maneira contínua.

Existem também diversos ***métodos de ensino*** [39] aplicáveis ao EAD. Estes podem ser gerais ou específicos.

Alguns métodos gerais são:

- ***Disseminação*** - repete o ensino tradicional, pois trata-se apenas de distribuir a informação (informação sobre o curso, organização de *home pages* com *links* para outras *home pages* recomendadas, transcrição de comunicação entre alunos, etc);
- ***Facilitação*** – tem o propósito de dar apoio e assistência ao estudante. *Chats* e grupos/listas de discussão desempenham um papel fundamental, e recaem sobre o professor as funções, entre outras, de fornecimento de diretrizes, condução de discussões, sugestão de recursos e formulação de perguntas estimulantes;
- ***Trabalho cooperativo*** entre os alunos, através dos mecanismos de comunicação disponíveis;
- ***Colaboração externa*** – interação com agentes e com comunidades do conhecimento, externos ao curso. Pode-se usar professores "visitantes" ao longo do curso inteiro, por exemplo, com uma facilidade que não seria possível de implementar de maneira tradicional;
- ***Desenvolvimento gerativo*** – geração de conteúdo; elaboração, pelos alunos, de *home pages* originais sobre determinados pontos que passam a ser incorporados ao conteúdo do curso;
- ***Desempenho de papéis*** – trata-se de atribuir responsabilidades simuladas a um grupo de pessoas e, a partir destes papéis, simular uma situação onde eles são desempenhados.

Entre os métodos específicos, encontram-se:

- ***Publicação de informações*** com o objetivo de fornecer informação sobre o curso (programa, textos associados às aulas, como matricular-se, entre outros);
- ***Fornecimento de referências*** sob a forma de endereços de *home pages*;
- ***Utilização de recursos para comunicação*** do grupo (listas de discussão, conferência e *newsgroups*);
- ***Complementação de conteúdo*** com a realização de atividades específicas como: capturar diálogos (por exemplo, gravar um *chat*) para reprodução e discussão

posteriores; disponibilizar recursos como texto digital, em HTML ou para *download*; promover o desenvolvimento de páginas *Web* pelos alunos;

- ***Publicação de projetos***, viabilizando a modelagem, discussão e revisão dos projetos que são publicados.

Através de algumas técnicas, o EAD baseado na *Web* pode estimular o pensamento criativo, o pensamento crítico e o aprendizado cooperativo [39].

Por exemplo, a criatividade pode ser estimulada através de atividades como: técnica do *brainstorm*, onde se lança uma idéia e segue elaborando sobre ela; *Role Playing Games*, onde é promovida uma discussão sobre determinado assunto e tarefas/papéis são atribuídos aos seus participantes, para que eles vivenciem situações difíceis de serem entendidas de outras formas; redação criativa (contando e completando histórias, resumindo debates); simulação; uso de analogia e associações forçadas (coisas do tipo "um bom professor é como...", associando a outras profissões, buscando que o aluno justifique a analogia que foi estabelecida), entre outras.

Pode-se estimular o pensamento crítico, entre outras formas, através de: uso de organizadores gráficos, pedindo que o aluno faça um diagrama representando, por exemplo, o ensino de aula convencional; votação e métodos de classificação e estabelecimento de ordenamentos, pedindo que o aluno caracterize e classifique idéias geradas durante o *brainstorm*, por exemplo; solicitação da manifestação do aluno sobre o que é mais/menos interessante; elaboração de resumos e sumários; julgamentos simulados.

A aprendizagem cooperativa, por sua vez, pode ser estimulada através de técnicas como: atividade entre parceiros (compartilhar informações, rever trabalhos e discuti-los juntos); mesas redondas; conferência assíncrona (cafés eletrônicos, grupos de discussão); conferência síncrona (entrevistas a partir de um *chat*); competição entre equipes; painéis, simpósios e debates, entre outras.

Isto tudo envolve motivação. O uso da *Web* para estimular o aprendizado ainda está sendo estudado, mas em [39] são apresentados dois modelos que mostram porque o EAD baseado na *Web* pode ser intrinsecamente interessante.

O primeiro modelo considera quatro fatores de motivação para o aprendizado: atenção, relevância, confiança e satisfação. A atenção é relativamente fácil de ser obtida na *Web*, devido à riqueza de informação disponível e da variedade de estratégias de *design* de recursos multimídia; mantê-la, porém, não é fácil, devido à concorrência. A relevância é muito mais uma questão de alinhamento com o interesse em determinado tópico e a percepção de sua utilidade para os objetivos a longo prazo; pode estar relacionada a fatores extrínsecos (como, por exemplo, conseguir um trabalho) ou intrínsecos (vontade de aprender sobre determinado assunto). Confiança e satisfação se relacionam à percepção do estudante quanto a ser capaz de ter sucesso a partir dos resultados obtidos e, em geral, atuam mais sobre a persistência da realização das tarefas ao longo do tempo do que na interação com tarefas momento a momento; ainda não está muito claro que técnicas da *Web* ajudariam a suprir estes dois fatores motivacionais.

O segundo modelo inclui os seguintes fatores: desafio, fantasia e curiosidade. O desafio é proporcionado por objetivos explícitos e por resultados difíceis de prever, enquanto a fantasia envolve a imersão do aprendiz num ambiente interessante que o convida ao envolvimento; por enquanto, estes dois fatores não estão diretamente relacionados com a *Web*. A curiosidade está mais relacionada à motivação intrínseca e é

diretamente relevante para o EAD baseado na *Web*, em função da riqueza de recursos que ela concentra e possui.

O ***papel do professor*** [3][39][40], no uso desta nova tecnologia, é desenvolver o conteúdo de forma a guiar o aluno na construção e elaboração de novos conhecimentos, ao mesmo tempo em que deve demonstrar explicitamente seu interesse no desenvolvimento do aluno, procurando esclarecer suas dúvidas, indicar caminhos, ultrapassar as dificuldades. É possível, até, estabelecer um grau de "intimidade" com o aluno, desenvolvendo um acompanhamento personalizado do percurso do mesmo durante o curso. Ele deve: moderar o grupo, coordenando e tentando dar ritmo e organização à interação existente dentro dele, bem como compondo as mensagens e garantindo a netiqueta; facilitar a auto-avaliação; atender aos alunos, seja de forma presencial, pela Internet ou por telefone; evitar dar aulas e ser muito claro quanto às suas expectativas, principalmente no que se refere à avaliação; guiar a conversação, mas jamais dominá-la; encorajar a comunicação e a metacomunicação (comunicação a respeito do curso), bem como o respeito entre os aprendizes.

Já o ***papel do aluno*** [39] é: acessar e usar o programa de computador regularmente; participar ativamente e interagir muito com os seus colegas; ter iniciativa na exploração da tecnologia e dos recursos que ela oferece; colaborar ao máximo possível com o instrutor; saber organizar os conteúdos do curso; entender muito bem o processo de auto-avaliação e avaliação do grupo; ser cortês e evitar o uso de pseudônimos; trocar mensagens sempre levando em conta o bom uso do campo *"subject"* (título e resumo do assunto da mensagem), o tamanho da mensagem (não deve ser muito longa), o português e a maneira de escrever corretos, a facilitação do entendimento e o foco da mensagem (não deve ser genérica). O sucesso do EAD baseado na *Web* depende muito da iniciativa do aprendiz.

No estabelecimento destes papéis, surgem alguns desafios: induzir a participação do aluno e equilibrar a quantidade de contatos síncronos e assíncronos. Em [39] são apontadas algumas indicações. Para induzir a participação, pode-se publicar um conteúdo que enfatize o envolvimento do aprendiz, com o objetivo de promover a interatividade. Pode-se, também, atribuir determinados papéis aos aprendizes (como apresentador, debatedor, moderador) como meio de animação ou usar pequenos grupos para realizar tarefas. Quanto ao equilíbrio na comunicação, pode-se escolher assuntos para serem discutidos de forma assíncrona em grupos de discussão e de maneira síncrona em *chats*, mas a forma de fazer isso de maneira adequada ainda está em elaboração. Os princípios básicos para se criar uma comunidade dinâmica de aprendizagem são responder sempre às perguntas apresentadas, ser competente *on-line* e organizar a interação.

2.4.2 Desenvolvimento de Ambientes para EAD

O que diferencia um EAD via *Web* medíocre de um EAD altamente efetivo é o fato seus recursos/capacidades da *Web* serem utilizados para melhorar o ensino. Para tirar partido da *Web* como um meio educacional, alguns ***requisitos básicos*** [39] devem ser satisfeitos:

- ***Fácil de aprender*** – o usuário deve poder rapidamente executar trabalhos no sistema. No caso da *Web*, ela tem uma estrutura transparente e uma interface padronizada, usada em várias aplicações. Uma certa redundância assegura o acesso fácil a todos os elementos de um *site*;
- ***Eficiente na utilização*** – uma vez que o usuário (autor) aprende o sistema, um alto nível de produtividade é possível; no caso da *Web*, o projeto da estrutura e da interface devem permitir que os usuários focalizem no conteúdo;

- ***Fácil de lembrar*** – o usuário casual pode voltar a usar o sistema depois de um tempo sem usá-lo, sem precisar aprender tudo de novo; no caso da *Web*, o projeto do sistema é claro e, sobretudo, consistente. A consistência reduz a sobrecarga cognitiva e auxilia a memorização e a orientação;
- ***Poucos erros*** – os usuários não cometem muitos erros no sistema e, se cometem um erro, a recuperação é simples; no caso da *Web*, o conteúdo e a estrutura são atualizados regularmente. As referências (*links*) são verificadas e também atualizadas regularmente;
- ***Agradável de usar*** – os usuários devem estar subjetivamente satisfeitos ao usar o sistema; no caso da *Web*, o projeto do sistema permite aos usuários concentrar-se no essencial, sem preocupação com a sobrecarga visual, e é o suficientemente flexível para acomodar iniciantes e veteranos.

Uma função dos que organizam o curso é perceber os diferentes tipos de aprendizagem: como saber mais; como memorização; como aquisição de fatos habilidades e métodos; como aprender a dar sentido/abstrair significado, relacionar partes de um tema entre si e com o mundo real; como interpretação/compreensão da realidade de diferentes maneiras. Eles devem observar se os cursos e métodos que estão sendo aplicados estimulam todas as formas conhecidas de aprendizagem [39].

Em [39] são apontados alguns ***critérios básicos de projeto***:

- ***Facilidade de acesso*** – o conteúdo tem que se apresentar de forma rica, sem sobrecarregar a banda passante disponível. Se os arquivos começarem a ficar grandes demais, começam a se tornar inacessíveis;
- ***Clareza*** – linguagem, estrutura de informação e apresentação visual devem prover uma orientação explícita no *site* educacional;
- ***Eficiência*** – o foco na aplicação de hipertexto em educação deve ser colocado no aprendizado do conteúdo e não nas idiossincrasias do *design*;
- ***Foco*** – o hipertexto é muito sedutor em termos de escolhas (muitas referências a outros *sites*), então a questão é oferecer a quantidade adequada de ligações. O objetivo é alcançar a profundidade sem perder o foco;
- ***Consistência*** – deve haver uma identidade visual para o *site* educacional, com as funções aparecendo de forma uniforme;
- ***Flexibilidade*** – o projeto do *layout* e da estrutura da *Web* devem ser adaptáveis a mudanças, com os ambientes projetados com uma grande quantidade de pontos de flexibilização.

Questões como segurança (uso de senhas, entre outros) e direitos autorais (se são propriedade do autor ou da instituição) também são pontos importantes a considerar [54].

Em [39] são apontadas ***ferramentas*** desejáveis para aperfeiçoamento de aulas baseadas na *Web*:

- Ferramenta ***para anotações*** – as notas de aula ou transparências do instrutor aparecem em um lado da tela do computador e, do outro lado, um bloco de anotações. Os aprendizes podem fazer anotações em tempo real diretamente no bloco. As anotações podem ser estudadas mais tarde em associação com as transparências e podem ser compartilhadas com outros aprendizes;

- Ferramenta ***para apresentação de perguntas*** – o aprendiz entra com sua pergunta através da ferramenta e seu nome, juntamente com a pergunta formulada, são apresentadas ao instrutor que pode decidir quando respondê-la;
- Ferramenta ***para preenchimento de lacunas em transparências*** – os aprendizes recebem versões parcialmente preenchidas das transparências do instrutor no momento em que ele está apresentando a transparência. O instrutor pede ao aluno para preencher as lacunas na transparência como forma de testar sua compreensão;
- Ferramenta ***de teste*** – o instrutor pode oferecer questões de múltipla escolha durante a aula e obter *feedback* direto através das respostas dos aprendizes;
- Ferramenta ***de monitoramento*** – o instrutor pode mudar para a tela de qualquer aluno no grupo para monitorar diretamente o que o aprendiz está fazendo, talvez durante o uso da ferramenta de teste ou da ferramenta de preenchimento de lacunas;
- Ferramenta ***para simular o contato face a face*** – o instrutor pode decidir enviar o conteúdo da sua tela para a tela de um aprendiz em particular ou mandar a tela de um aprendiz para todos os demais.

Em [54] afirma-se que a oferta de cursos via Internet deveria ser vista dentro do contexto de um sistema efetivo de EAD. Anthony Kaye [32] identificou quatro subsistemas que o compõem:

- Subsistema regulador – consiste da gerência, tomada de decisões, planejamento, custos e avaliação de processos, geralmente responsabilidade dos administradores do programa e da instituição;
- Subsistema curso – inclui *design*, produção, distribuição e recebimento do material de ensino do curso, juntamente com o gerenciamento do sistema de comunicação mantido para interação no curso;
- Subsistema aluno – é responsável por admitir alunos e controlar seu progresso dentro do curso;
- Subsistema logístico – inclui aquisição e manutenção de equipamentos e *software*, e a admissão e treinamento de pessoal.

Como em qualquer sistema, cada um destes subsistemas precisa funcionar efetivamente e estar sincronizado com outros subsistemas, para o bom funcionamento do sistema como um todo.

Segundo Lucena e Fuks [39], há um ***modelo geral*** emergente de ambientes de desenvolvimento de cursos baseados na *Web*, organizados nas seguintes etapas:

- ***Criação de conteúdo*** – existem duas opções: ou o professor é especialista em ferramentas para *Web*, faz o curso em HTML e depois o transfere para o ambiente de EAD, ou o professor usa um ambiente para desenvolvimento no qual estão ocultos os detalhes técnicos. Em muitas situações vale mais a pena usar a segunda opção, pois reduz o trabalho de produção;
- ***Criação de um curso no servidor*** – o administrador de cursos pode criar um "gabarito", segundo a solicitação do professor. O professor se conecta ao servidor e personaliza a interface com o aluno usando menus, cria uma *home page* do curso, define os componentes desejados e recursos de navegação, inscreve as assistentes e os alunos;

- ***Envio de conteúdo para o servidor*** do curso;
- ***Interação do administrador com o sistema*** – através de senha, o administrador liga-se ao sistema, a partir de qualquer máquina com conexão Internet, para adicionar e remover cursos, autorizar a inscrição dos alunos, etc;
- ***Interação do professor com o sistema*** – o professor pode se interligar ao sistema através de uma senha para se comunicar eletronicamente, atualizar conteúdo, atribuir tarefas ao estudante e monitorar o progresso dos alunos, através de exames, estatísticas sobre o *site*, etc;
- ***Interação do estudante com o curso*** – ele se liga ao sistema através de nome e senha, usa os recursos de comunicação eletrônica. Espera-se que ele assimile o conteúdo do curso e faça progressos em ritmo próprio, usando referências na *Web*, acompanhando tutoriais, assistindo aulas gravadas e vendo transparências. Espera-se que ele elabore sua própria *home page* através de um gabarito pré-estabelecido;
- ***Interação do auxiliar do professor com o curso*** – ele se liga ao sistema através de nome e senha. O auxiliar tem os recursos de comunicação iguais aos do professor e dos alunos, podendo corrigir exames e atribuir notas.

Em [40] são apresentadas algumas questões importantes que as ***instituições*** deveriam considerar antes de implantar cursos via *Web*:

- ***Questões básicas***
  - Por que queremos desenvolver cursos *online*?
  - Por que não oferecer cursos a distância em mídias mais baratas e que atingem um maior número e pessoas?
  - Por que construir um ambiente próprio?
  - Que tipos de cursos serão oferecidos? Para público interno ou externo?
  - Quais os objetivos dos cursos on-line? Promoção da instituição? Criar novos mercados? Pesquisa? Melhorar o ensino tradicional?
- ***Questões de mercado e infra-estrutura tecnológica***
  - Temos laboratórios de acesso à rede disponíveis para atender a uma grande demanda de alunos?
  - Qual a porcentagem de alunos conectados à rede?
  - Qual a porcentagem de docentes conectados à rede? Destes, que porcentagem conhece recursos e ferramentas de aprendizagem disponíveis ou fazem uso de *e-mail*, *chats*, listas de discussão, etc?
- ***Questões de desenvolvimento acadêmico pedagógico do ambiente***
  - Os recursos e tecnologias utilizados promovem mesmo o aprendizado?
  - Existe interatividade no ambiente?
  - Como motivar alunos e professores para trabalhar em ambientes mediados por computador?
  - Até que ponto pode-se incorporar recursos gráficos e multimídia no ambiente?
  - Como promover a interação e aprendizagem colaborativa entre aluno e professor durante todo o curso?

- Como prender a atenção do aluno?
- Como deve ser pensado o conteúdo do ambiente virtual?
- Qual o tamanho ideal de um texto par ser exibido na tela?
- Como avaliar? De forma presencial? A distância?

- ***Questões administrativas***
  - Como medir "horas-aula"?
  - Como remunerar o professor?
  - Quanto cobrar do aluno?
  - Direitos autorais pertencem ao professor ou à instituição?
  - Existe legislação específica sobre cursos *online*?
  - Como medir a freqüência?
  - Qual o limite de alunos em uma turma na *Web*?

É importante ter certeza do motivo pelo qual se pretende fazer cursos virtuais. Deve fazer parte da missão da instituição o desejo de mudanças de cultura, estabelecer novos papéis para professores e alunos e criar novas metodologias e formatos de ensino/aprendizagem.

Segundo Guarezzi e Camilotti [27], a problemática da qualidade educacional não é algo novo; no entanto, antes não havia o grau de centralidade que a mesma atinge hoje. Quanto ao uso de recursos tecnológicos na educação, não é suficiente que se saiba dominar a técnica trabalhando com textos, sons, imagens, *software*, e sim que se tenha em mente qual o propósito de sua utilização, em que proposta de educação pode ser inserida. É necessário ir além do entendimento das tecnologias, compreendendo o processo de construção do conhecimento e como acontece o processo de ensino/aprendizagem nesta sociedade onde a comunicação permite que um número cada vez maior de pessoas tenha acesso a informações que se atualizam rapidamente.

Ao final deste capítulo, pode-se dizer que EAD ocorre não somente com a separação da distância geográfica, mas envolve tempo (EAD síncrono e assíncrono) e alguns outros fatores, sendo alvo de tentativas de definição até os dias de hoje. Tendo origem há mais de um século, várias foram as tecnologias utilizadas nesta modalidade de ensino. Hoje, a Web é o meio mais moderno para comunicação entre professores e aprendizes a distância; mas a mudança não é apenas tecnológica, envolve também aspectos pedagógicos, com uma grande preocupação quanto à qualidade do ensino. Não se quer reproduzir o ensino presencial através deste novo meio, e sim criar todo um contexto próprio de EAD com técnicas, recursos e metodologias que favoreçam o processo de ensino/aprendizagem e o tornem melhor do que tem sido até os dias de hoje. No próximo capítulo serão vistas as tecnologias de Design Patterns e Frameworks, bem como sua contribuição para o desenvolvimento de sistemas de EAD.

3 *Frameworks* e *Design Patterns*

Este capítulo apresenta uma descrição das tecnologias de Frameworks e Design Patterns e explica sua utilidade para a área de EAD.

Com o advento da *Web*, uma nova plataforma se tornou disponível, introduzindo uma nova forma de disponibilizar a informação para consulta. A velocidade com que surgem novas necessidades provoca um constante repensar sobre a forma de se desenvolver sistemas de *software*. Diversas metodologias são projetadas de forma a atender a estas necessidades; porém, dois aspectos chamam mais atenção: a tecnologia a ser utilizada para o desenvolvimento de *software* e a possibilidade da reutilização de rotinas, bibliotecas e serviços já desenvolvidos.

Segundo Amjad Umar [57], as novas tecnologias para o desenvolvimento de aplicações têm-se centrado em: uso da *Web* e aplicações Cliente/Servidor. Mattson [46] e Basset [2] colocam questões importantes sobre o processo de desenvolvimento de *software* como:

- Como construir novas aplicações usufruindo das vantagens deste novo paradigma?
- Que tipo de estrutura é necessária para dar suporte à reutilização e ao desenvolvimento rápido e confiável de sistemas de *software*?

Pesquisas têm mostrado que altos níveis de reutilização de *software* podem ser alcançados através do uso de *Frameworks* Orientados a Objetos (OO) e *Design Patterns* [8][11][22][48].

3.1 *Design Patterns*

Design Patterns (Padrões de Projeto) são soluções genéricas para problemas recorrentes em Engenharia de *Software*. Diz Christopher Alexander [1]: "cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira". 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 [22]. Assim, o uso de padrões deve contribuir para reutilização e flexibilização no desenvolvimento de *software* OO [10][11][17].

De modo a facilitar seu entendimento e utilização, os *Design Patterns* são descritos de forma estruturada e catalogados, contendo quatro elementos essenciais [22]:

- ***Nome*** do padrão;
- ***Problema*** – descrição do contexto no qual pode-se utilizar o padrão, podendo ser a descrição de problemas de projeto específicos, ou de estruturas de classe/objeto, ou

ainda uma lista de condições que devam ser satisfeitas para que faça sentido utilizá-lo;

- ***Solução*** – descrição abstrata de um problema de projeto e como um arranjo geral de elementos (classes e objetos) resolve o mesmo;
- ***Conseqüências*** – resultados e análises das vantagens e desvantagens da aplicação do padrão.

3.1.1 Descrição de *Design Patterns*

Em [22] é apresentado um gabarito para descrição de padrões, que possui a seguinte estrutura:

- ***Nome e classificação*** do padrão – um bom nome, pois se tornará parte do vocabulário de projeto, e a categoria na qual o padrão é classificado (ver seção 3.1.2);
- ***Intenção e objetivo*** – curta declaração sobre o que faz o padrão, quais seus princípios e sua intenção e que tópico ou problema particular de projeto ele trata;
- ***Também conhecido como*** – outros nomes conhecidos para o padrão, se existirem;
- ***Motivação*** – um cenário que ilustra um problema de projeto e como as estruturas de classes e objetos no padrão solucionam o problema, para ajudar a entender suas descrições mais abstratas;
- ***Aplicabilidade*** – quais as situações nas quais o padrão pode ser aplicado, que exemplos de mau projeto ele pode tratar e como essas situações podem ser reconhecidas;
- ***Estrutura*** – representação gráfica das classes do padrão baseada na Object Modeling Technique (OMT) e diagramas de interação para ilustrar seqüências de solicitações e colaborações entre objetos;
- ***Participantes*** – classes e/ou objetos que participam do padrão e suas responsabilidades;
- ***Colaborações*** – como os participantes colaboram para executar suas responsabilidades;
- ***Conseqüências*** – como o padrão suporta a realização de seus objetivos, quais seus custos e benefícios e o resultado de sua utilização, e que aspecto da estrutura do sistema ele permite variar independentemente;
- ***Implementação*** – considerações específicas de linguagem, se existirem, e quais armadilhas, sugestões ou técnicas é preciso conhecer quando da implementação do padrão;
- ***Exemplo de código*** – trechos de código que ilustram como o padrão pode ser implementado em C++ ou *Smalltalk*;
- ***Usos conhecidos*** – exemplos do padrão encontrados em sistemas reais;
- ***Padrões relacionados*** – padrões que estão intimamente relacionados com este, as diferenças importantes e com quais outros padrões este deveria ser utilizado.

Este gabarito foi definido por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, conhecidos como *The Gang of Four* (GOF) [22]. As linguagens citadas são aquelas que os autores acharam mais adequadas, mas nada impede que se utilizem outras linguagens ou se defina um roteiro diferenciado.

Como exemplo da utilização deste gabarito, a seguir são mostrados alguns itens da descrição do *Design Pattern State* [22] (**Tabela 1**).

STATE	Comportamental de objetos

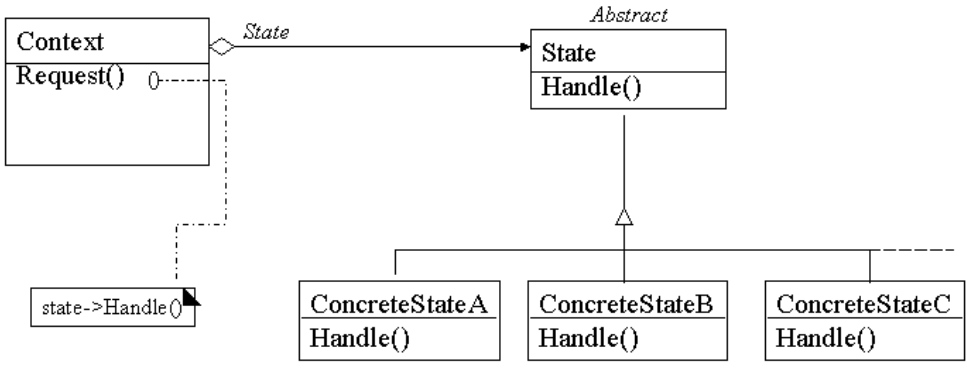
Intenção
Permite a um objeto alterar seu comportamento quando seu estado interno muda. O objeto parecerá ter mudado de classe.

Também conhecido como
Object for States

Aplicabilidade
Use o padrão *State* em um dos casos seguintes:

- O comportamento de um objeto depende do seu estado e ele pode mudar seu comportamento em tempo de execução, dependendo desse estado;
- Operações têm comandos condicionais grandes, de várias alternativas, que dependem do estado do objeto. Este estado é normalmente representado por uma ou mais constantes enumeradas. Freqüentemente, várias operações conterão esta mesma estrutura condicional. O padrão *State* coloca cada ramo do comando adicional em uma classe separada. Isto lhe permite tratar o estado do objeto como um objeto propriamente dito, que pode variar independentemente de outros objetos.

Estrutura



Participantes

- *Context*
  - Define a interface de interesse para os clientes.
  - Mantém uma instância de uma subclasse *ConcreteState* que define o estado corrente.
- *State*
  - Define uma interface para encapsulamento associado com um determinado estado ao *Context*.
- Subclasses *ConcreteState*
  - Cada subclasse implementa um comportamento associado com um estado do *Context*.

Padrões relacionados
O padrão *Flyweight* explica quando e como objetos *State* podem ser compartilhados. Objetos *State* são freqüentemente *Singletons*.

Tabela 1 — Exemplo parcial de descrição do *Design Pattern State*

3.1.2 Classificação de *Design Patterns*

Design Patterns podem ser classificados segundo as seguintes categorias [22]:

- Padrões ***de criação*** - abstraem o processo de instanciação. Ajudam a tornar o sistema independente de como os objetos são criados, compostos e representados. Um padrão de criação de classes usa a herança para variar a classe que é instanciada, enquanto que um padrão de criação de objetos delegará a instanciação para outro objeto. Os padrões de criação dão muita flexibilidade no que é criado, quem cria, como e quando é criado. Permitem configurar um sistema com objetos "produto" que variam amplamente em estrutura e funcionalidade, sendo que esta configuração pode ser estática (especificada em tempo de compilação) ou dinâmica (especificada em tempo de execução);
- Padrões ***estruturais*** – preocupam-se com a forma como classes e objetos são compostos para formar estruturas maiores. Os padrões estruturais de classes utilizam a herança para compor interfaces ou implementações. Este tipo de padrão é particularmente útil para fazer bibliotecas de classes desenvolvidas independentemente trabalharem juntas. Já os padrões estruturais de objetos, em lugar de compor interfaces ou implementações, descrevem maneiras de compor objetos para obter novas funcionalidades. A flexibilidade provém da capacidade de mudar a composição em tempo de execução, o que é impossível com a composição estática de classes;
- Padrões ***comportamentais*** – preocupam-se com algoritmos e a atribuição de responsabilidades entre objetos. Não descrevem apenas padrões de objetos ou classes, mas também os padrões de comunicação entre eles. Estes padrões caracterizam fluxos de controle difíceis de seguir em tempo de execução; eles afastam o foco do fluxo de controle para que seja possível concentrar-se somente na maneira como os objetos são interconectados. Padrões comportamentais de classes utilizam herança para distribuir o comportamento entre classes, enquanto padrões comportamentais de objetos utilizam a composição de objetos. Alguns descrevem como um grupo de objetos pares (*peer objects*) cooperam para a execução de uma tarefa que nenhum objeto sozinho poderia executar por si mesmo.

3.1.3 Utilização de *Design Patterns*

Decompor um sistema em objetos não é uma tarefa fácil: eles nem sempre são encontrados na fase de análise e nos estágios iniciais de projeto. Existem diversos fatores que contribuem para dificultar esse processo: encapsulamento, granularidade, dependência, flexibilidade, desempenho, evolução e reutilização, entre outros. Como vencer estes desafios? *Design Patterns* podem ajudar [22].

Design Patterns facilitam a identificação de abstrações menos óbvias, bem como os objetos que podem capturá-las. Por exemplo: como decidir o que deve ser um objeto? Existem padrões que descrevem maneiras de representar subsistemas completos como objetos (*Design Pattern Façade*), como suportar enormes quantidades de objetos nos níveis de granularidade mais finos (*Design Pattern Flyweight*), como decompor objetos em objetos menores (*Design Patterns Builder* e *Command*), etc [22].

Também existem padrões que ajudam a definir interfaces de objetos, através da identificação de seus elementos-chave e tipos de dados que são enviados através da interface. Um exemplo é o *Design Pattern Memento*. Há ainda outros padrões que especificam relacionamentos entre as interfaces. Tudo isso é muito importante, pois em

um sistema OO só é possível conhecer um objeto, solicitar-lhe algo, através de sua interface [22].

Padrões de criação, como *Abstract Factory*, *Builder* e *Singleton*, permitem abstrair o processo de criação de objetos, propiciando várias maneiras de associar uma interface com sua implementação de forma transparente no momento da instanciação. Isto reduz dependências de implementação entre subsistemas, favorecendo a reutilização. Em outras palavras, se algum aspecto da implementação herdada de uma classe não for apropriada para novos domínios de problemas, a classe mãe deve ser reescrita ou substituída por algo mais apropriado. Esta dependência limita a flexibilidade, o que pode ser resolvido com herança de classes abstratas, que fornecem pouca ou nenhuma implementação (comum aos *Design Patterns*) [22].

Para que se possa ter maior reutilização, é importante que o sistema possa evoluir de acordo com as mudanças nos requisitos existentes. Um *Design Pattern* permite que o sistema mude de maneira específica, com a variação de algum aspecto da estrutura do sistema independentemente de outros aspectos [22].

Mas como decidir qual o padrão correto para tratar certo problema? Algumas abordagens são sugeridas [22]:

- Considerar como os padrões solucionam problemas de projeto;
- Examinar o item *Intenção* da descrição de cada padrão, podendo-se focalizar a busca em sua classificação, a fim de selecionar os que pareçam relevantes para o problema em questão;
- Estudar como os padrões se inter-relacionam;
- Estudar padrões de finalidades semelhantes;
- Examinar causas que possam resultar na reformulação do projeto;
- Considerar o que deveria ser variável no projeto, o que se gostaria de poder mudar sem precisar reformular o projeto.

Outra questão é: como utilizar o *Design Pattern* escolhido? Em [22] é sugerida uma abordagem passo a passo:

- Ler o padrão por inteiro uma vez, para obter sua visão geral;
- Voltar e estudar as seções *Estrutura*, *Participantes* e *Colaborações*, para compreender as classes e objetos e como se relacionam entre si;
- Olhar a seção *Exemplo de Código*, para ver um exemplo concreto do padrão;
- Escolher nomes para os participantes do padrão que tenham sentido no contexto;
- Definir as classes;
- Definir nomes específicos da aplicação para as operações no padrão;
- Implementar as operações para suportar as responsabilidades e colaborações presentes no padrão.

Design Patterns não devem ser usados indiscriminadamente. Devem apenas ser aplicados quando a flexibilidade que oferecem é realmente necessária, sob pena de haver perdas em relação a custo/benefício de sua utilização (deve-se avaliar a seção *Conseqüências* na descrição dos padrões) [22].

3.2 *Frameworks*

De acordo com alguns autores, um *Framework* é:

- "Um conjunto de classes que constitui um *design* abstrato para soluções de uma família de problemas" – Johnson e Foote [31];
- "Um conjunto de objetos que colaboram com o objetivo de cumprir um conjunto de responsabilidades para uma aplicação ou um domínio de um subsistema" – Johnson [30];
- "Uma arquitetura desenvolvida com o objetivo de se obter a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com grande potencial de especialização" – Mattson [44].

Segundo Wolfgang Pree [49], *Frameworks* unificam sistemas de *software* para um domínio específico e constituem em 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 [11]. *Hot-spots* são representados por métodos *hook*. Cada *hook* tem como foco um

números 2 e 4, conforme seu posicionamento na tela) e a barra de cores (indicada pelo números 3). O editor pode ter todos ou apenas alguns destes elementos. Por exemplo: em (A), temos um editor gerado com o menu, a barra de cores e a barra de desenho alinhada à direita da tela (*hot-spots* 1, 3 e 4); em (B), temos um editor gerado com o menu e a barra de desenho alinhada à esquerda da tela (*hot-spots* 1 e 2); em (C), temos um editor gerado com o menu, a barra de desenho alinhada à esquerda e a barra de cores (*hot-spots* 1, 2 e 3).

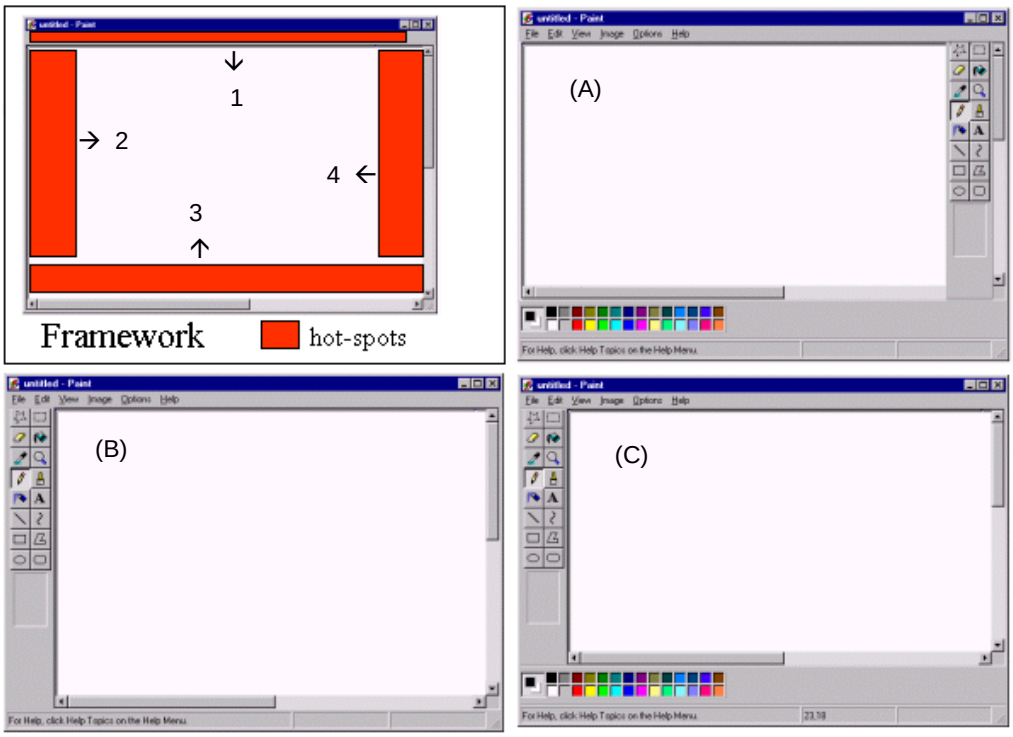


Figura 2 — Exemplo de aplicações obtidas a partir da instanciação de um *Framework*

Pesquisas reconhecem a necessidade da integração de *Frameworks* e *Design Patterns*, uma vez que os mesmos também promovem flexibilização e reutilização. Porém, não está claro como identificar os *hot-spots* em *Frameworks*, e como fazer a conexão entre a seleção do *Design Pattern* e a implementação destas variações [11][19][22][48][58].

3.2.1 Tipos de *Frameworks*

Há vários tipos de *Frameworks*, nem todos OO, para os quais muitas formas de classificação têm sido propostas [21][59]. Estes tipos podem variar de *Frameworks* de aplicação (que ajudam, por exemplo, a desenvolver interfaces de usuário) a *Frameworks* de mais baixo nível (que fornecem serviços básicos para comunicação, impressão e sistemas de arquivos). Existem também *Frameworks* de domínio específico, que solucionam problemas em áreas determinadas (como, por exemplo, acesso a dados, segurança e multimídia) [17][21][59].

Frameworks de aplicação, por sua vez, são classificados com base em seu escopo. Temos *Frameworks* de infraestrutura (simplificam o desenvolvimento de uma infraestrutura de sistema portável e eficiente: sistema operacional, comunicação, interfaces e processamento de linguagens), integração *middleware* (comumente

utilizados para integrar aplicações distribuídas e componentes) e negócios (relativos a domínios como telecomunicações, aviação, manufatura e finanças) [16][17].

Um *Framework* também pode ser classificado quanto ao uso [11][16][17][21][42]:

- ***Architecture-driven***, ou ***inheritance-focused***, ou ***white box*** – utilizado através da derivação de suas classes;
- ***Data-driven***, ou ***composition-focused***, ou ***black box*** – utilizado por meio da combinação de suas classes.

O tipo *whitebox* é mais difícil de usar, pois exige que o programador conheça a estrutura do *Framework*, mas também é considerado mais poderoso na mão de *experts*. Possui classes incompletas, ou seja, que contém métodos sem codificação significativa. Para utilizar este tipo de *Framework*, programadores devem modificar seu comportamento utilizando herança para sobrescrever métodos de classes [16][17][42].

Já o tipo *blackbox* exige apenas que se saiba combinar as classes necessárias para realizar a instanciação. Possui componentes prontos para adaptação. Modificações são feitas através da composição de suas classes. É considerado o tipo ideal, pois é mais fácil utilizar componentes já existentes do que construir um novo [16][17][42].

Outra classificação diz respeito ao modo como o *Framework* interage com outras aplicações [16][21]:

- ***Called*** *Framework* – é uma entidade passiva que pode ser chamada por outras aplicações. Corresponde a uma biblioteca de código e é utilizada através da chamada de funções ou métodos desta biblioteca;
- ***Calling*** *Framework* – é uma entidade ativa que incorpora o controle dentro do próprio *Framework*. Aplicações fornecem métodos customizados ou componentes que são chamados pelo *Framework*.

Em geral, não se tem um *Framework* puramente *whitebox* ou *blackbox*, ou puramente *called* ou *calling*, e sim uma combinação dos dois tipos. Algumas abstrações comuns entre vários componentes *blackbox*, dentro de um mesmo domínio de problema, também podem ser modeladas e implementadas utilizando herança e sobreposição de alguns métodos. Virtualmente, cada *Framework* é chamado por alguma parte da aplicação e também chama alguma outra parte [16][17].

3.2.2 Processo de Desenvolvimento

Um *Framework* OO captura os aspectos comuns a uma família de aplicações. Ele também captura a experiência do projetista durante o processo de construção da aplicação. Isto possibilita reutilização tanto em *design* como em programação [53], o que não se consegue num processo de desenvolvimento OO tradicional.

No processo tradicional [45][46], a fase de Análise fornece subsídios para a elaboração de um *Design* que resultará na implementação de uma única aplicação.

No processo baseado em *Frameworks* [45][46], a fase de Análise do Domínio define requisitos que resultarão no *Design* de um *Framework* que pode ser instanciado diversas vezes, gerando diferentes aplicações pertencentes ao domínio analisado. Assim, não é necessário um novo trabalho de Análise e *Design* para construir uma nova aplicação, apenas deve-se customizar os *hot-spots* que constituirão as características desejadas para o *software*. Assim, *Frameworks* proporcionam reuso em três níveis: Análise, Projeto e implementação [16][42].

A **Figura 3** ilustra as diferenças entre os processos abordados.

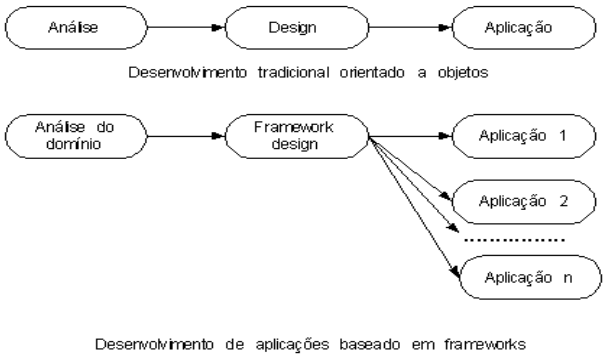


Figura 3 — Desenvolvimento OO tradicional *versus* baseado em *Frameworks*

O processo de desenvolvimento de um *Framework* é mais difícil e trabalhoso, pois são estudados problemas de um determinado domínio e observadas características de diversas aplicações pertencentes a este domínio [10][16]. O conjunto de requisitos definidos deve ser o mais completo possível, para garantir que o *Framework* funcione corretamente. Além disso, toma mais tempo devido às diversas iterações de modelagem, codificação, testes e correções, necessárias para garantir o funcionamento adequado do *Framework* [16].

Em geral, a primeira versão de um *Framework* é *whitebox*. A experiência leva a melhoramentos no *Framework* que, quase sempre, vai se tornando mais e mais *blackbox* (para isto, é fundamental que ele seja utilizado, testado). Finalmente, chega o momento em que o *Framework* é considerado pronto [16].

O desenvolvimento de um *Framework* pode ser realizado de algumas formas, dependendo daquilo em que se baseia: experiência, Análise de Domínio ou *Design Patterns*. Existem diversos elementos em comum entre estes processos.

O ***processo baseado em experiência*** [45][46] (**Figura 4**) inicia-se com o desenvolvimento de *n* aplicações (mínimo de duas) baseadas no domínio do problema. Quando prontas, observa-se as características em comum das aplicações, e as colocam no *Framework*. Para verificar se as características extraídas estão corretas refaz-se as aplicações com base no *Framework* desenvolvido. No caso de demandar muito esforço reescreve-se o *Framework* quando necessário e utiliza-se essa experiência no desenvolvimento de sua nova versão. Desenvolve-se outras aplicações com o *Framework* e repetem-se as interações quantas vezes forem necessárias [10].

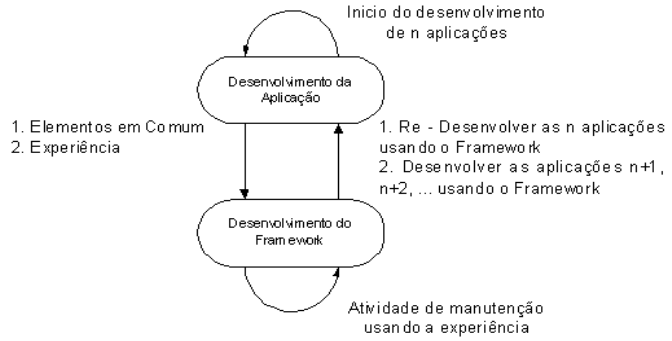


Figura 4 — Desenvolvimento de *Frameworks* baseado em experiência

A **Figura 5** nos permite visualizar o ***processo baseado na Análise do Domínio*** [45][46], onde a primeira etapa é analisar o domínio do problema para identificar e entender possíveis abstrações [10][16]. Analisar o domínio requer analisar aplicações existentes e, segundo consta em [16], consiste de três atividades maiores:

- ***Análise do contexto*** *do domínio* – identificar o escopo do domínio;
- ***Modelagem das abstrações*** *do domínio* – identificar os componentes certos do domínio e seus dados, comportamento e interação;
- ***Arquitetura*** *do domínio* – gerar o *backbone* do *Framework*.

Após identificar as abstrações, desenvolve-se o *Framework* juntamente com uma aplicação de teste, modificando o *Framework* quando necessário, revisando as aplicações anteriores para ver se continuam funcionando [10][16].

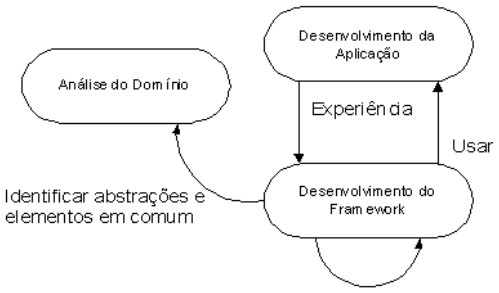


Figura 5 — Desenvolvimento de *Frameworks* baseado na Análise de Domínio

O primeiro passo do ***processo com utilização de Design Patterns*** [45][46] consiste em desenvolver uma aplicação e em seguida aplicar sistematicamente um conjunto de padrões para criar o *Framework*. A partir daí ocorrem interações entre aplicações e *Framework* [10][11].

Este processo é mostrado pela **Figura 6**.

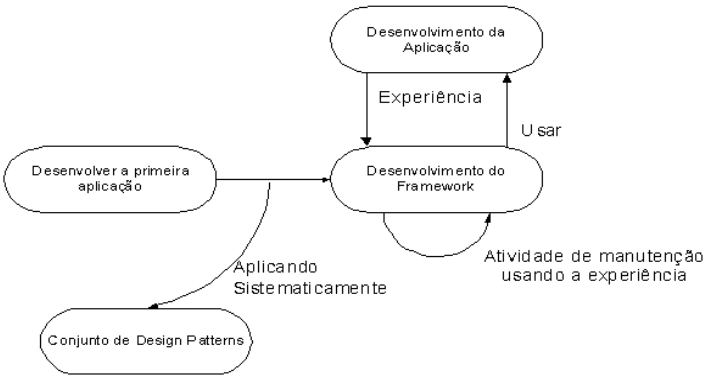


Figura 6 — Desenvolvimento de *Frameworks* utilizando *Design Patterns*

Design Patterns mapeiam características estruturais de um *Framework* e podem, em adição, proporcionar inspiração na busca dos *hot-spots*. Também documentam aspectos mais técnicos do *design*: a solução que oferecem está ligada a um contexto e problema, proporcionando documentação para "o que" e "por que" do *design*.

Proporcionam flexibilidade, facilitando o reuso e as modificações, contribuindo para a diminuição das iterações necessárias até que o *Framework* seja considerado pronto [16][42].

Apesar dos aspectos diferenciados, os processos de desenvolvimento descritos possuem ***elementos em comum*** [10]:

- É feita a Análise de Domínio do problema;
- A primeira versão do *Framework* é desenvolvida utilizando as abstrações encontradas;
- Uma ou mais aplicações são desenvolvidas baseadas no *Framework*. O teste é importante para verificar se o *Framework* é realmente reutilizável;
- Problemas encontrados no desenvolvimento da aplicação baseado no *Framework* são capturados e resolvidos na próxima versão;

Na **Figura 7** temos o que seria um ***processo geral*** de desenvolvimento [45][46], segundo os aspectos citados.

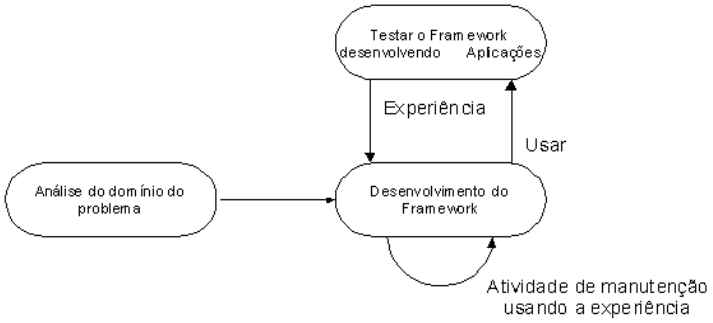


Figura 7 — Processo geral de desenvolvimento de *Frameworks*

Após repetir o ciclo várias vezes, o *Framework* vai atingindo um nível de maturidade aceitável, possibilitando desta forma a sua reutilização por vários usuários.

3.2.3 Diretrizes de Desenvolvimento

Existe um pequeno número de diretrizes para o desenvolvimento de *Framework*. Uma das mais importantes que se deve ter em mente quando se está desenvolvendo um *Framework* é como criar a interação entre ele e o seu usuário, de maneira que a sua instanciação seja feita sem muito esforço [10][60]. Para o usuário, é mais difícil lidar com abstrações mal definidas do que criar suas próprias [60].

O foco deverá estar em como o usuário interage com o *Framework*. Quando o usuário for desenvolver uma aplicação instanciando o *Framework*, é necessário determinar quais os passos que serão realizados pelo *Framework* e quais serão realizados pelo usuário [10].

É importante identificar quais classes e operações do *Framework* o usuário terá que utilizar. A quantidade de código escrita deverá ser reduzida a um mínimo, e isto é possível pelos seguintes motivos [10]:

- Implementações concretas do *Framework* que poderão ser utilizadas sem nenhuma modificação;
- Redução do número de classes que devem ser derivadas ao mínimo possível.

É importante considerar também questões como [10][60]:

- Utilizar convenções para nomes de classes, operações e como as operações serão utilizadas;
- Ter um bom controle quanto ao uso de herança múltipla;
- Garantir que o *Framework* atenda a todos os requisitos necessários (para que isto ocorra, a Análise de Requisitos deve ser bem feita);
- Manter a modelagem do domínio do problema independente da interface com o usuário e plataforma;
- Permitir que funcionalidades possam ser facilmente adicionadas ou modificadas;
- Documentar bem as interações entre as classes do *Framework* e as classes do usuário e código (utilizar um padrão de *design* e codificação, bem como fornecer exemplos que demonstrem o uso do *Framework*);
- Realizar correções e melhoramentos a partir de testes, *feedback* dos usuários e observações próprias, tomando cuidado para não modificar o *Framework* com muita freqüência (corrigir erros imediatamente, adicionar novos recursos ocasionalmente e evitar ao máximo realizar modificações em interfaces).

Um bom *Framework* tem algumas propriedades que podem torná-lo mais reutilizável, como [21][42]:

- ***Facilidade de uso*** – deve ser fácil de entender e facilitar o desenvolvimento de aplicações;
- ***Extensibilidade*** – deve permitir a adição de novos componentes ou propriedades facilmente;
- ***Flexibilidade*** – é a habilidade de utilizar o *Framework* em mais de um contexto. Em geral, isto se aplica ao domínio de problema. *Frameworks* que podem ser utilizados em múltiplos domínios, como *Frameworks* de interface gráfica, são especialmente flexíveis. Porém, deve-se tomar cuidado com o uso indiscriminado de *hooks*, que pode tornar o *Framework* muito complexo sem adicionar nada em termos de funcionalidade, levando a problemas de complexidade e performance;
- ***Completude*** – ainda que *Frameworks* sejam incompletos, já que não podem contemplar todas as possíveis variações dentro de um domínio, uma relativa completude é desejada. Implementações padrão podem ser fornecidas para as abstrações dentro do *Framework*, assim elas não precisam ser reimplementadas dentro de cada aplicação e os desenvolvedores podem executar o *Framework* para ter um melhor entendimento de como ele funciona. Podem ser disponibilizadas implementações de operações comuns, que o desenvolvedor possa escolher, tornando o *Framework* mais fácil de usar e mais completo.
- ***Consistência*** – deve-se utilizar uma nomenclatura consistente, bem como convenções de interface ou estrutura de classes. Isto pode ajudar a acelerar a compreensão do *Framework* por parte dos desenvolvedores e reduzir erros em seu uso.

Um *Framework* não precisa, necessariamente, atender a todos os requisitos citados para ser bom; mas, se o fizer, será utilizado mais vezes e com maior facilidade.

3.2.4 Documentação de *Frameworks*

Comentários no código e documentação constituem uma parte necessária no desenvolvimento de qualquer *software*, mas são especialmente importantes quando se

está desenvolvendo um *Framework*. Se outros desenvolvedores não entenderem o *Framework*, não irão utilizá-lo.

A documentação tem o objetivo de introduzir rapidamente como o *Framework* pode ser utilizado de um modo geral. Deve-se deixar claro quais as classes que podem ser utilizadas diretamente, as devem ser instanciadas e aquelas que devem ter métodos sobrescritos. Detalhes da implementação do *Framework* não são importantes [10][60].

Em geral, é bom disponibilizar [10][60]:

- Descrição do *Framework*;
- Exemplos de utilização, onde cada exemplo é apresentado passo a passo (não se descreve todas as possibilidades de utilização do *Framework*, mas é interessante mostrar seu uso em diversos contextos);
- Diagramas da arquitetura do *Framework*;
- Apresentação de onde e como o *Framework* pode ser adaptado ou estendido.

Existem diferentes ***tipos de pessoas que podem reutilizar um Framework***. Isto requer que diversos tipos de informação sejam disponibilizados, com diferentes níveis de abstração ou detalhe [16]:

- *Reuso por um* ***desenvolvedor de aplicações*** – a prioridade para um desenvolvedor é saber como customizar o *Framework*. Ele precisa conhecer os *hot-spots* relevantes, quais classes especializar, quais métodos sobrescrever, como manter um protocolo de colaboração entre as classes. A documentação deve ser prescritiva. O desenvolvedor pode não ser um grande conhecedor do domínio de problema ou ter experiência em desenvolvimento de *software*.
- *Reuso por um* ***desenvolvedor responsável pela manutenção do Framework*** – o responsável pela manutenção e evolução do *Framework* deve entender seu *design*. Também deve conhecer diversos aspectos como: o domínio ao qual o *Framework* está relacionado, as razões que influenciam a escolha dos *hot-spots*, quais são os *Design Patterns* utilizados e o motivo pelo qual foram selecionados, a responsabilidade de cada classe, as interfaces, a colaboração entre as classes, entre outros. A documentação deve ser descritiva. O desenvolvedor é, ao mesmo tempo, *expert* em *software* e no domínio do problema.
- *Reuso por um* ***desenvolvedor de outro Framework*** – desenvolvedores de *Frameworks* buscam idéias em *Frameworks* existentes, mesmo que pertençam a outros domínios. O interesse principal é pelos *Design Patterns* que proporcionam flexibilidade aos *hot-spots*. O tipo de informação requerida é similar àquela utilizada pelos responsáveis pela manutenção. Os desenvolvedores são *experts* em *design* de *software* mas não são, necessariamente, *experts* no domínio do *Framework* que estão reutilizando.
- *Reuso por um* ***verificador*** – alguns desenvolvedores podem ter a necessidade de verificar certas propriedades do sistema de modo a satisfazer requisitos que envolvem algum tipo de rigidez. Isto exige o uso de métodos formais de especificação e verificação. A especificação para reuso costuma ser mais descritiva do que prescritiva: cabe ao reutilizador perceber as implicações da especificação em termos da customização desejada. Deve-se especificar claramente aspectos como: protocolos que o desenvolvedor pode customizar, colaborações que devem ser suportadas por novas subclasses, entre outros.

A **Tabela 2** exemplifica o que foi dito categorizando o público usuário de *Frameworks* segundo atribuições (decisão de uso, utilização e manutenção) e relaciona aspectos que devem constar na documentação do *Framework* para cada um deles.

Os estilos de documentação descritos têm o objetivo de facilitar o entendimento de um *Framework* para que possa ser reutilizado. Pode-se combinar estilos ou utilizar algum em particular, sempre visando facilitar o aprendizado do *Framework*.

3.2.5 Utilização de *Frameworks*

Conforme foi visto na subseção 3.2.1, um *Framework* pode ser utilizado (instanciado) de duas maneiras:

- ***Conectando componentes já existentes*** – é feita reutilização da interface do *Framework* e das regras para conexão dos componentes (tipo *blackbox*);
- ***Criando novas subclasses concretas*** – as subclasses são bem acopladas à superclasse; é necessário um maior conhecimento das classes abstratas (tipo *whitebox*).

E, como descrito em 3.2.4, um *Framework* pode ter vários tipos de usuários: desenvolvedores de aplicações, responsáveis pela manutenção do *Framework*, usuários de outros *Frameworks*, etc, cada qual necessitando de um certo conjunto de informações para fazer uso do *Framework* escolhido.

Basicamente, um usuário deve utilizar a documentação existente sobre um *Framework* para aprender os aspectos que lhe são pertinentes e então partir para a instanciação (ou manutenção, ou outro tipo de uso, conforme o tipo de usuário).

3.2.6 Vantagens e Desvantagens do Uso de *Frameworks*

O desenvolvimento de qualquer tipo de *Framework* requer a consideração de alguns pontos importantes, como a necessidade real de se ter um *Framework*. Embora seu uso proporcione benefícios como reuso de *design*, também apresenta pontos negativos como o custo elevado de desenvolvimento se comparado a uma única aplicação [21].

Como vantagens [21], podemos citar:

- ***Reuso***;
- ***Facilidade de manutenção*** – devido às partes em comum existentes nas aplicações desenvolvidas;
- ***Qualidade*** – devido a reusabilidade, *design* testado e funcionamento comprovado.

Algumas desvantagens [21] são:

- ***Alto custo*** – são necessários mais tempo e dinheiro do que é preciso para desenvolver uma única aplicação;
- ***Mudanças no domínio de problema*** – o domínio deve ser relativamente estável pois, se mudar, serão perdidos os esforços de desenvolvimento a não ser que o *Framework* possa ser facilmente modificado;
- ***Evolução*** – qualquer mudança no *Framework* irá afetar as aplicações desenvolvidas a partir dele. Esta atualização das aplicações pode envolver mais custos e, se não for feita, a vantagem de se ter um código comum é perdida.

3.3 Aplicação de *Design Patterns* e *Frameworks* em EAD

O uso de *Design Patterns* e *Frameworks* se mostra adequado para o desenvolvimento de aplicações onde os requisitos mudam rapidamente, como é o caso da área de EAD.

Apesar de todas as facilidades tecnológicas da *Web*, os programas de EAD, sem exceção, começam com um planejamento cuidadoso do instrutor e devem enfocar as necessidades dos estudantes. Portanto, a tecnologia apropriada só pode ser selecionada uma vez que estes elementos são compreendidos. Eles não acontecem espontaneamente, evoluem com o trabalho árduo e os esforços dedicados de educadores e instituições e só prosperam através dos esforços consistentes e integrados de estudantes, universidades, instrutores, pessoal de apoio e administradores. Os recursos pedagógicos, até então ao alcance de educadores para aulas presenciais, estão sendo adaptados em sistemas de *software* que suportam a elaboração de cursos via *Web* com vários recursos e serviços didáticos e de comunicação, como ferramentas interativas para o processo de ensino/aprendizagem [39][52].

O uso de *Frameworks* permite que todo o trabalho de obtenção dos requisitos, bem como o *design* e a implementação de sistemas sejam reaproveitados. Aliada ao uso de *Design Patterns*, a tecnologia de *Frameworks* proporciona redução de tempo e recursos necessários para o desenvolvimento, além do reuso de *software*; isto é muito importante, pois na área do ensino nem sempre se dispõe de tempo, e os recursos muitas vezes são escassos. A adição/modificação de recursos/funcionalidades aos sistemas é facilitada por essas tecnologias e os sistemas desenvolvidos possuem maior confiabilidade, uma vez que o *Framework* já foi utilizado e testado diversas vezes. Isto também é muito importante, pois professores/alunos devem poder confiar nas aplicações que utilizam, sabendo que funcionam corretamente e que não causarão prejuízos ao seu trabalho.

Pode-se concluir, após a leitura deste capítulo, que Design Patterns são de grande ajuda na modelagem de sistemas, já que abordam pequenos problemas que se repetem e cada um apresenta uma solução já testada, adequada para certas situações. O uso de Frameworks evita o retrabalho de análise, design e implementação no desenvolvimento de sistemas; como se referem a um domínio específico, estas etapas são realizadas para o desenvolvimento do Framework e, a partir dele, podem ser geradas diversas aplicações (apenas se customizam os aspectos desejados para determinada aplicação). A união destas tecnologias resulta em economia de tempo, custo e recursos, mostrando-se ideal para áreas onde os requisitos mudem com certa rapidez, como é o caso do EAD. O próximo capítulo trará uma aplicação prática destas tecnologias na área de EAD: será apresentado o desenvolvimento do Framework JLearningServices.

4 JLearningServices

Este capítulo apresenta o desenvolvimento do Framework JLearningServices. Um artigo sobre o Framework foi publicado ao XII Simpósio Brasileiro de Informática na Educação (SBIE'2001), servindo de base para a escrita do capítulo.

A geração de ferramentas para suporte a EAD tem-se mostrado promissora, embora alguma destas não incorporem os requisitos necessários para o enfoque educacional. Em virtude disto, faz-se necessário a obtenção de maior flexibilidade no processo de geração de artefatos de *software* que promovam suporte a EAD. O *Framework* JLearningServices [51] oferece solução para este problema, na medida em que novas aplicações síncronas poderão ser rapidamente geradas com alto grau de confiabilidade.

Não se trata de desenvolver novamente o que já existe há muito tempo, como o *chat*, mas sim de dotar este tipo de ferramenta de características de EAD para que possam ser utilizadas com maior proveito em sistemas desta área. Por exemplo, em certo ambiente de EAD pode não ser desejado o uso de conversa reservada, o que poderia ser excluído na geração do *Framework*.

O processo de desenvolvimento de um *Framework* depende do grau de experiência da organização no domínio do problema. Uma organização com maior experiência poderá adotar um processo mais avançado no desenvolvimento [11]. Neste trabalho, optou-se por utilizar um processo baseado na análise do domínio em virtude da experiência já adquirida no domínio do problema.

Este processo caracteriza-se por:

- Analisar o domínio do problema para identificar e entender possíveis abstrações a partir do estudo de aplicações existentes;
- Identificar-se as abstrações;
- Desenvolver o *Framework* juntamente com uma aplicação de teste, modificando o *Framework* quando necessário, revisando as aplicações anteriores para verificar se continuam funcionando.

As seções 4.1, 4.2 e 4.3 descrevem os passos acima, respectivamente.

4.1 Análise de Sistemas de Comunicação Síncrona em Ambientes de EAD

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*.

Engenharia reversa [50][55] é o processo de se analisar *software* com o objetivo de entender e reconstruir seu *design* e especificação. Isso pode ser feito através da observação de código fonte e/ou programa em execução, o que estiver disponível para análise.

A seguir, será feita uma breve descrição dos principais sistemas analisados. Também será mostrado um pequeno modelo feito para cada sistema, contendo as principais características verificadas durante sua observação. Os modelos foram desenhados segundo a notação da *Unified Modeling Language* (UML) [4], utilizando uma versão da ferramenta *TogetherJ*. Esta ferramenta permite o desenvolvimento de sistemas utilizando a linguagem Java, e encontra-se disponível para *download* gratuito em *www.oi.com*.

Um dos sistemas estudados foi o ***Chat do Colégio Santa Úrsula*** (*www.ursula.com.br/colegio/chat*). Nele, foram observadas as seguintes características: o sistema possui um servidor que permite o acesso de diversos clientes. Cada cliente possui apenas uma sala de bate-papo, que permite a interação entre diversos participantes. Para entrar na sala é necessário *login* (assim como em todos os sistemas descritos a seguir) e permite-se conversa reservada. Não existe a possibilidade de se formatar o texto das mensagens a serem enviadas. Seu modelo pode ser visto na **Figura 8**.

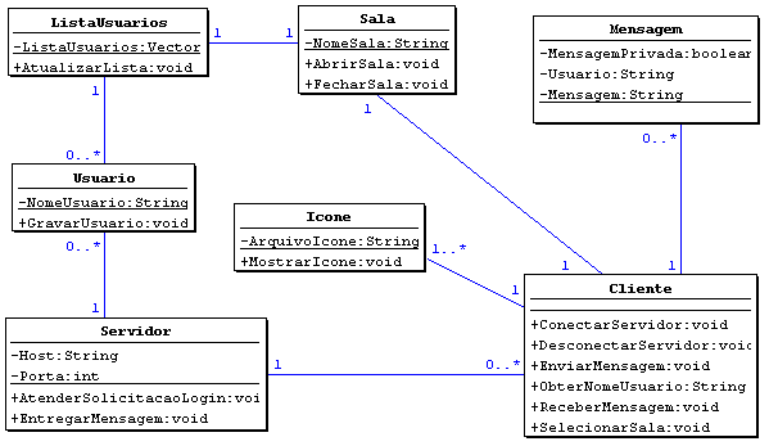


Figura 8 — Modelo simplificado do *chat* do colégio Santa Úrsula

No ***Instant Messenger*** (*www.aol.com/aim/home.html*), mostrado na **Figura 9**, o usuário possui uma lista de contatos e indica, através de texto, o seu *status* (*online* ou *offline*).

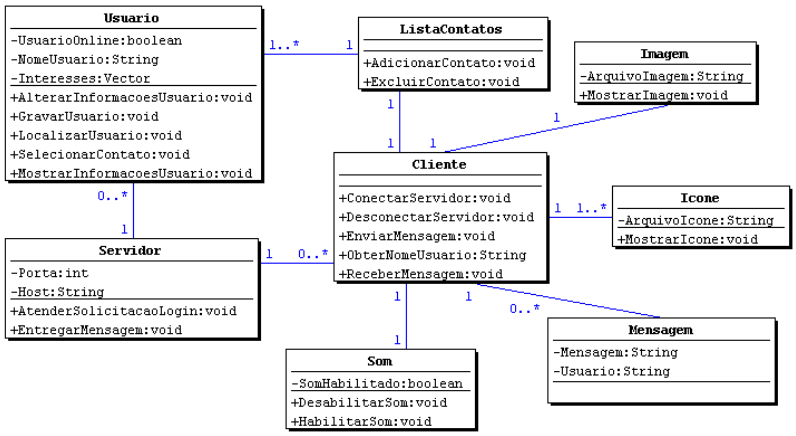


Figura 9 — Modelo simplificado do *Instant Messenger*

O *Instant Messenger* permite que se localize outros usuários com interesses em comum. Da mesma forma que para o ICQ, são mostradas apenas as características de maior interesse no modelo.

O ***ICQ*** (*www.mirabilis.com*), no momento da instalação, permite que o usuário se conecte a um dos servidores disponíveis na Internet, define para o usuário um número de identificação e permite que informe outros dados como o nome, por exemplo. Pode-se escolher o tipo de comunicação desejado: *chat*, voz, *message board*, conferência (múltiplos usuários), transferência de arquivos, jogos. Serve de plataforma para aplicações *peer-to-peer*, como *Netscape CoolTalk* e *Microsoft NetMeeting*. Devido à complexidade do ICQ, o modelo aqui exibido contempla características de uma forma mais global, apenas para dar idéia do tipo de característica que é de maior interesse. A **Figura 10** mostra o modelo do *ICQ*.

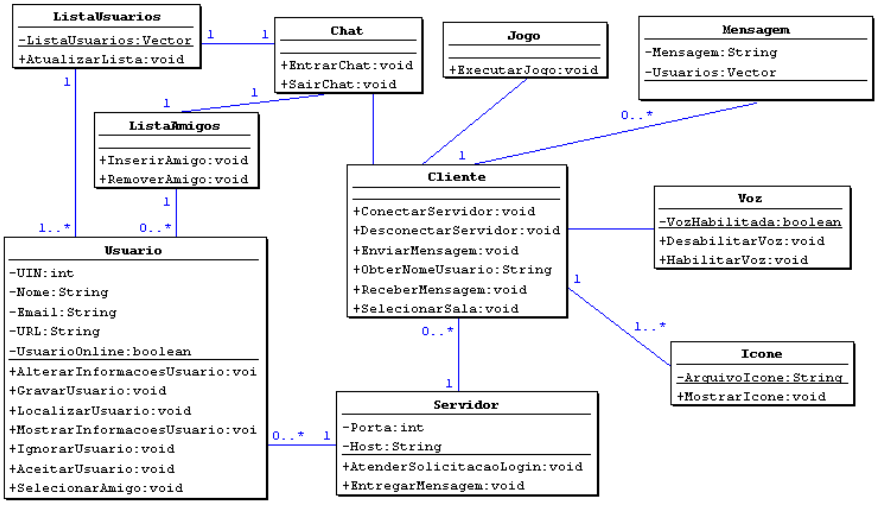


Figura 10 — Modelo simplificado do ICQ

A ***Sala de Chat do Laboratório de Estudos Cognitivos*** (LEC) da UFRGS (*www.psicologia.ufrgs.br*) possui um servidor que permite acesso de diversos clientes e várias salas são oferecidas. Há uma lista de ações e uma lista de sons que podem ser utilizados. A mensagem a ser enviada pode ser formatada. É permitida a realização de conversa reservada bem como obter o *profile* de um usuário. O modelo deste *chat* pode ser visto na **Figura 11**.

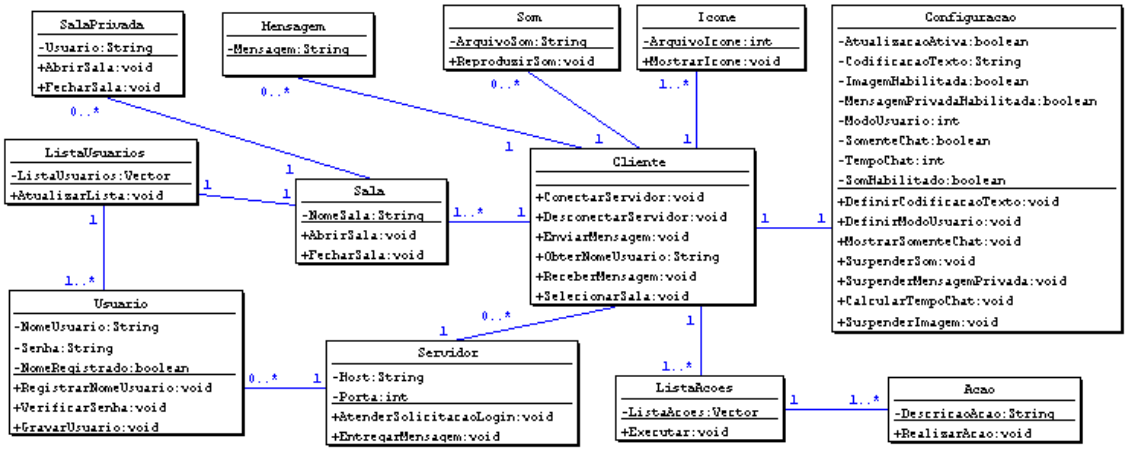


Figura 11 — Modelo simplificado do *chat* do LEC - UFRGS

The Palace (*www.humanas.unisinos.br/ambiente/oficinas/the_palace/index.htm*) possui diversos provedores. Cada servidor permite o acesso de diversos clientes, onde vários participantes interagem. A conversação pode ser feita através de texto, imagens, som. Existem ferramentas para desenhar na tela. É possível enviar mensagens para todos ou para algum participante específico. O modelo apresentado na **Figura 12** segue as mesmas observações do *ICQ* e do *Instant Messenger*.

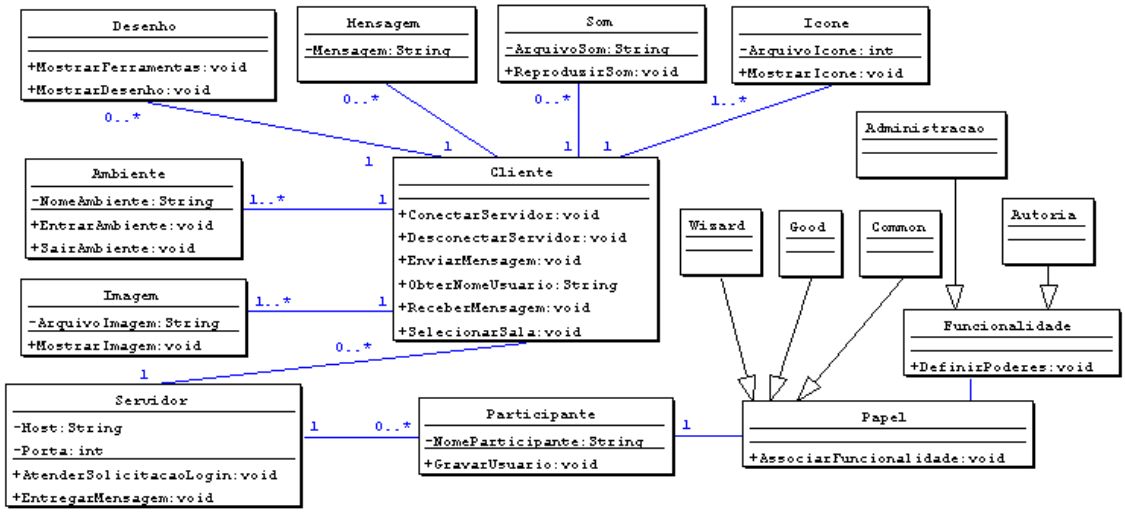


Figura 12 — Modelo simplificado do *The Palace*

No ***Book Chat*** (modelo exibido na **Figura 13**) foi observada a sala *Science Fiction* (*www.4-lane.com/bookchat/pages/sciencefictionchat.html*). Existe um provedor que permite o acesso de vários clientes a diversas salas. O sistema faz uso de ícones, imagens e texto (é possível formatar o tamanho da fonte). Para cada sala é informada a lista dos usuários que estão participando. É permitida a configuração de alguns parâmetros, como uso ou não de imagens, atualização ou não da tela e taxa de *refresh*. Existe a figura de um administrador que pode, por exemplo, alterar a taxa de atualização da tela e alertar os usuários a esse respeito.

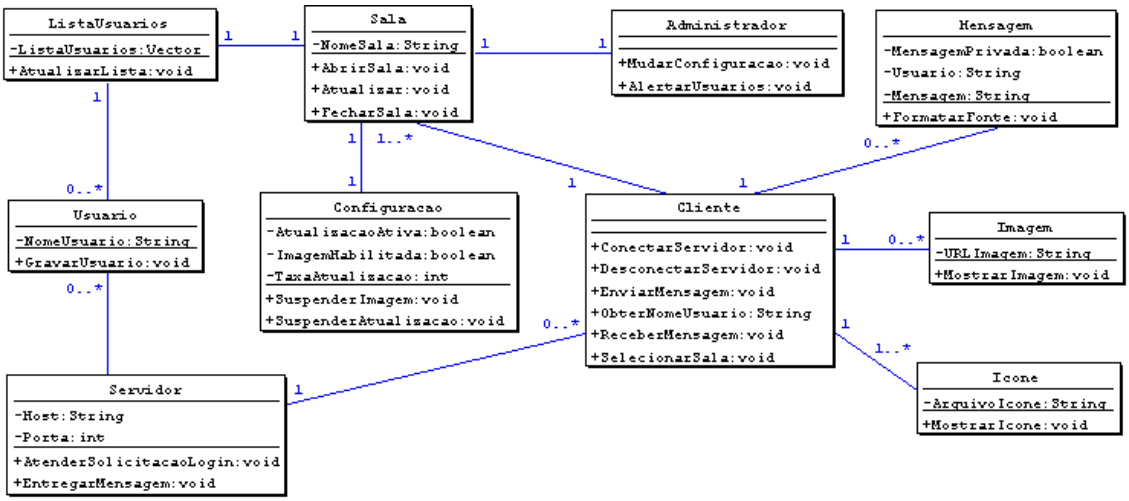


Figura 13 — Modelo simplificado do *Book Chat*

No ***Chat Machine*** (*www.chatmachine.com*) é possível que vários clientes se conectem a um servidor e utilizem várias salas. O sistema faz uso de ícones, som e permite formatação da fonte da mensagem. Pode-se criar ou simplesmente entrar em salas públicas ou privadas. Cada sala mostra a lista dos seus participantes. Pode-se configurar a "mensagem do dia". Seu modelo é mostrado na **Figura 14**.

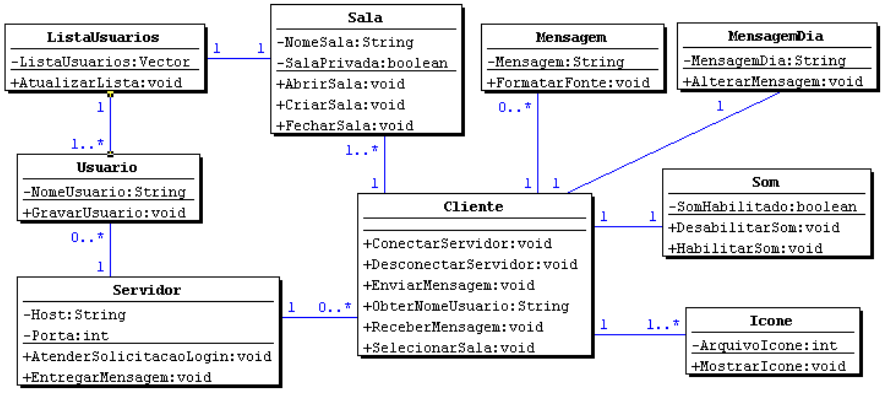


Figura 14 — Modelo simplificado do *Chat Machine*

No ***Student Center Chat – Java Light*** (*www.studentcenter.org/network.php*) vários clientes podem se conectar a um servidor e entrar em qualquer sala. Pode-se criar salas públicas ou privadas. É controlado se a sala foi ou não criada pelo usuário. É exibida a lista de usuários de cada sala e pode-se classificar um usuário como amigo (ele então faz parte de uma lista de amigos). Pode-se ver as informações cadastradas para um usuário ou localizá-lo no sistema. As mensagens podem ter cor e tamanho da fonte definidos pelo usuário e o sistema faz uso de som e ícones. A **Figura 15** traz o modelo deste *chat*.

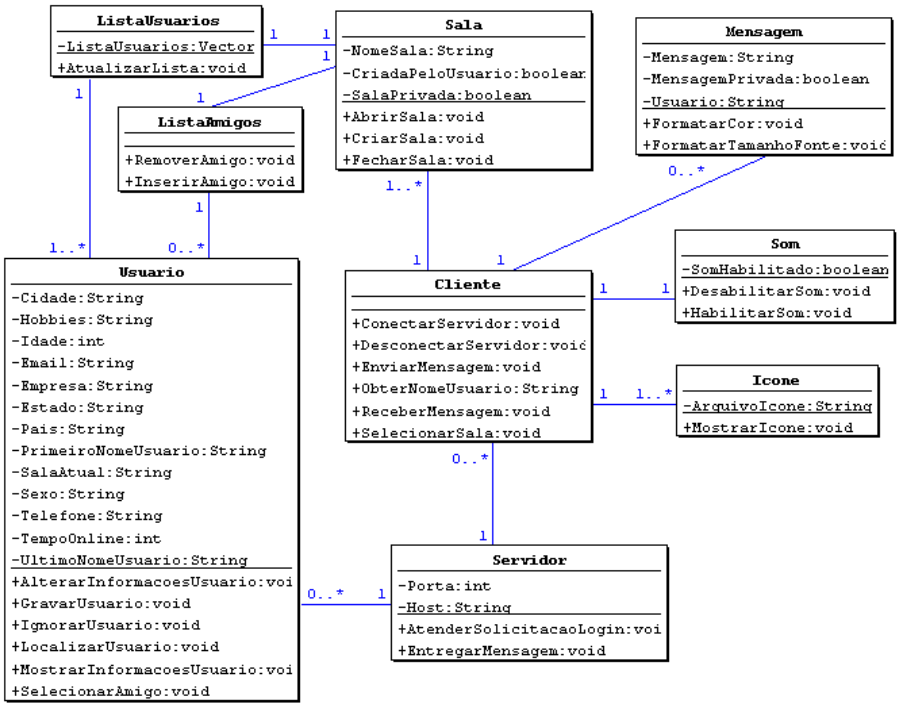


Figura 15 — Modelo simplificado do *Student Center Chat* Java *Light*

O ***Student Center Chat – Java*** (*www.studentcenter.org/network.php*) é semelhante ao anterior. A principal diferença entre os *chats* é a exibição na tela, além da lista de ususários, da lista de amigos, de uma lista de ícones disponíveis e de uma lista de outras salas disponíveis. Há outras pequenas diferenças de interface e no modo como as informações do sistema são exibidas. Seu modelo pode ser visto na **Figura 16**.

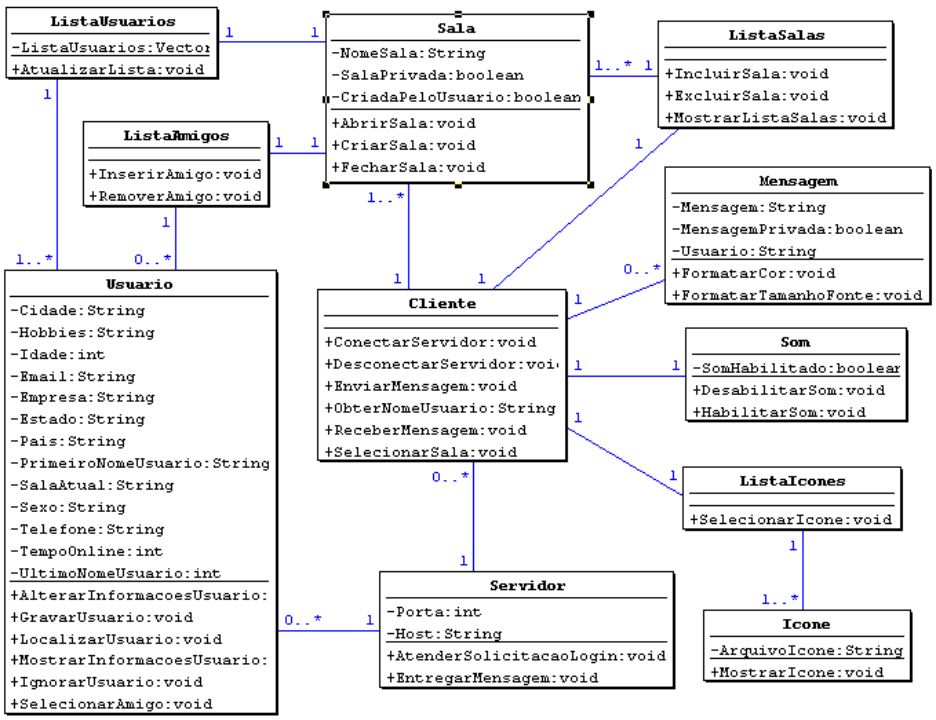


Figura 16 — Modelo simplificado do *Student Center Chat* Java

O ***Arab Chat*** (*www.arabchat.com*) possui um servidor que permite o acesso a várias salas por diversos clientes. Os usuários podem utilizar *tags* HTML para formatar a cor e o estilo das mensagens, assim como enviar pequenos ícones. O sistema também faz uso de imagens. Seu modelo pode ser visto na **Figura 17**.

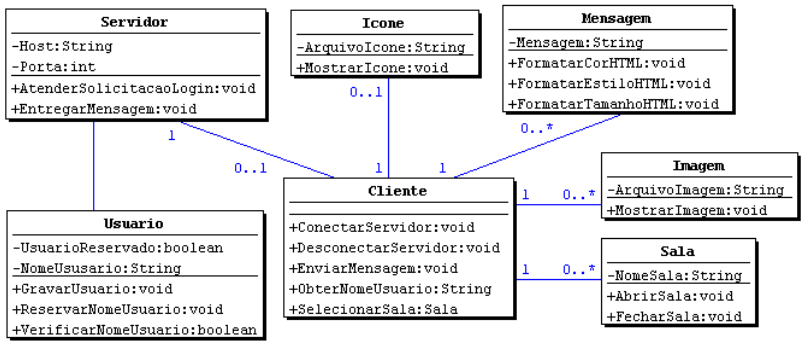


Figura 17 — Modelo simplificado do *Arab Chat*

Também o ***Funky Chat*** (*www.funkytimes.com/chat.htm*) possui um servidor ao qual diversos clientes se conectam e fazem uso de várias salas, públicas ou privadas. O sistema faz uso de ícones, som e de uma lista de ações (numa lista de ações, por

exemplo, poderia ter-se a participante Ana que escolheria a ação "sorrir" e a mensagem "Ana sorri para todos na sala" seria enviada aos demais participantes). A lista dos usuários de cada sala é mostrada na tela. Há uma série de itens que podem ser configurados, como a suspensão das mensagens privadas, a suspensão do uso de imagens e o cálculo do tempo no *chat*, entre outros. A **Figura 18** mostra o modelo do *Funky Chat*.

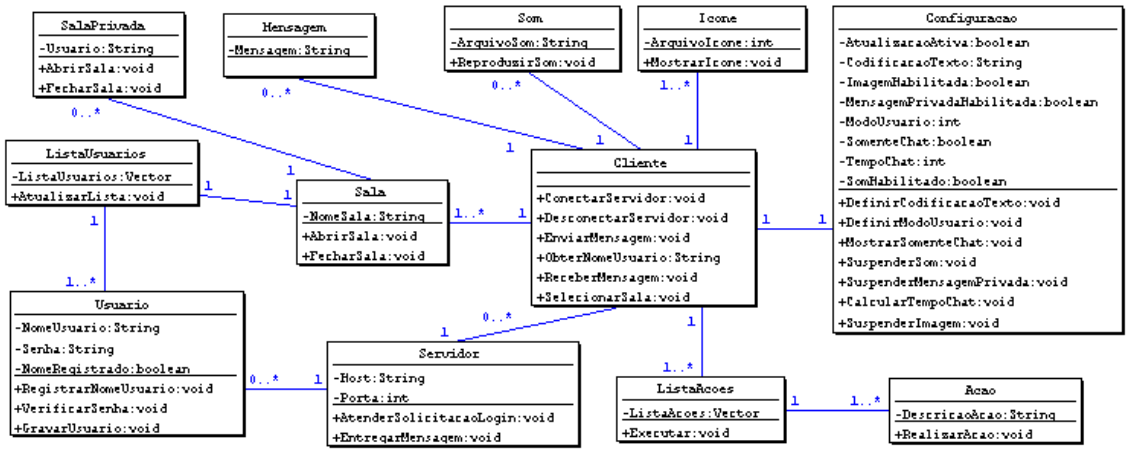


Figura 18 — Modelo simplificado do *Funky Chat*

O ***Excite Chat*** (*http://talk.excite.com/*) proporciona a vários clientes a possibilidade de se conectarem a diversas salas através de seu servidor. As salas podem ser públicas ou privadas e exibem sua lista de usuários. Pode-se localizar um usuário no *chat*, ignorá-lo ou simplesmente ver informações cadastradas a seu respeito. Seu modelo é exibido na **Figura 19**.

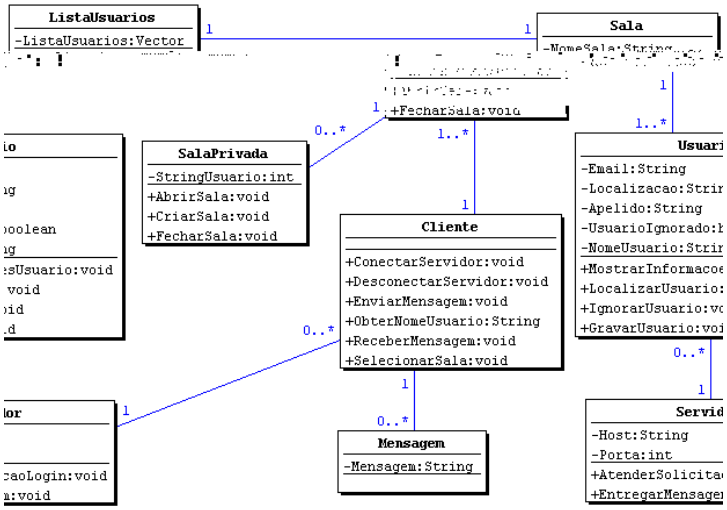


Figura 19 — Modelo simplificado do *chat Excite*

No ***Galaxy Net IRC Network*** (*www.galaxynet.org/chat/*) vários clientes podem se conectar a um servidor e fazer uso de várias salas, públicas ou privadas. A lista dos usuários de uma sala é mostrada na tela. Pode-se formatar a cor da mensagem e configurar alguns itens como, por exemplo, o canal ao qual pertence o *chat*. O modelo é mostrado na **Figura 20**.

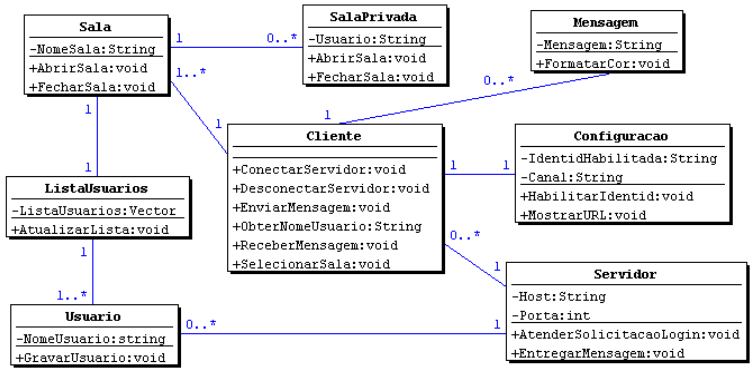


Figura 20 — Modelo simplificado do *Galaxy Net IRC Network*

O ***Super Chat*** (*www.superchat.org/irc/*) é um sistema simples. Um servidor permite que clientes se conectem a ele e façam uso de diversas salas. Uma lista de usuários é exibida para cada sala. Pode ser feito uso de som. O modelo deste *chat* pode ser visto na **Figura 21**.

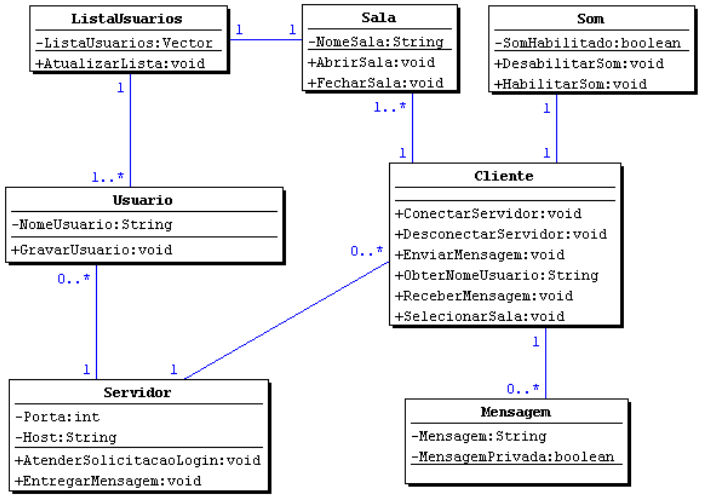


Figura 21 — Modelo simplificado do *Super Chat*

O ***Yahoo!*** possui um *chat* Java que foi possível acessar estando cadastrada. O endereço *chat.yahoo.com/?room=Generation%20X::1600326618&ayb=btu&identity=leticiarafaela&client=Java* foi exibido no *browser*. O sistema possui um servidor que permite conexões de vários clientes e acesso a diversas salas, que são classificadas como públicas, privadas ou seguras. A lista dos participantes de uma sala é exibida na tela. Usuários podem ser colocados na lista de ignorados ou na lista de amigos. É feito uso de som e ícones. Há uma lista de emoções, que funciona de forma semelhante a uma lista de ações. Mensagens podem ter seu texto formatado em cor e estilo, ou ter a formatação ignorada. Vários aspectos podem ser configurados, como ignorar ou não convites para participar de outras salas, habilitar ou não um filtro de palavras, avisar ou não quando um amigo estiver *online*, entre outros. Seu modelo pode ser visto na **Figura 22**.

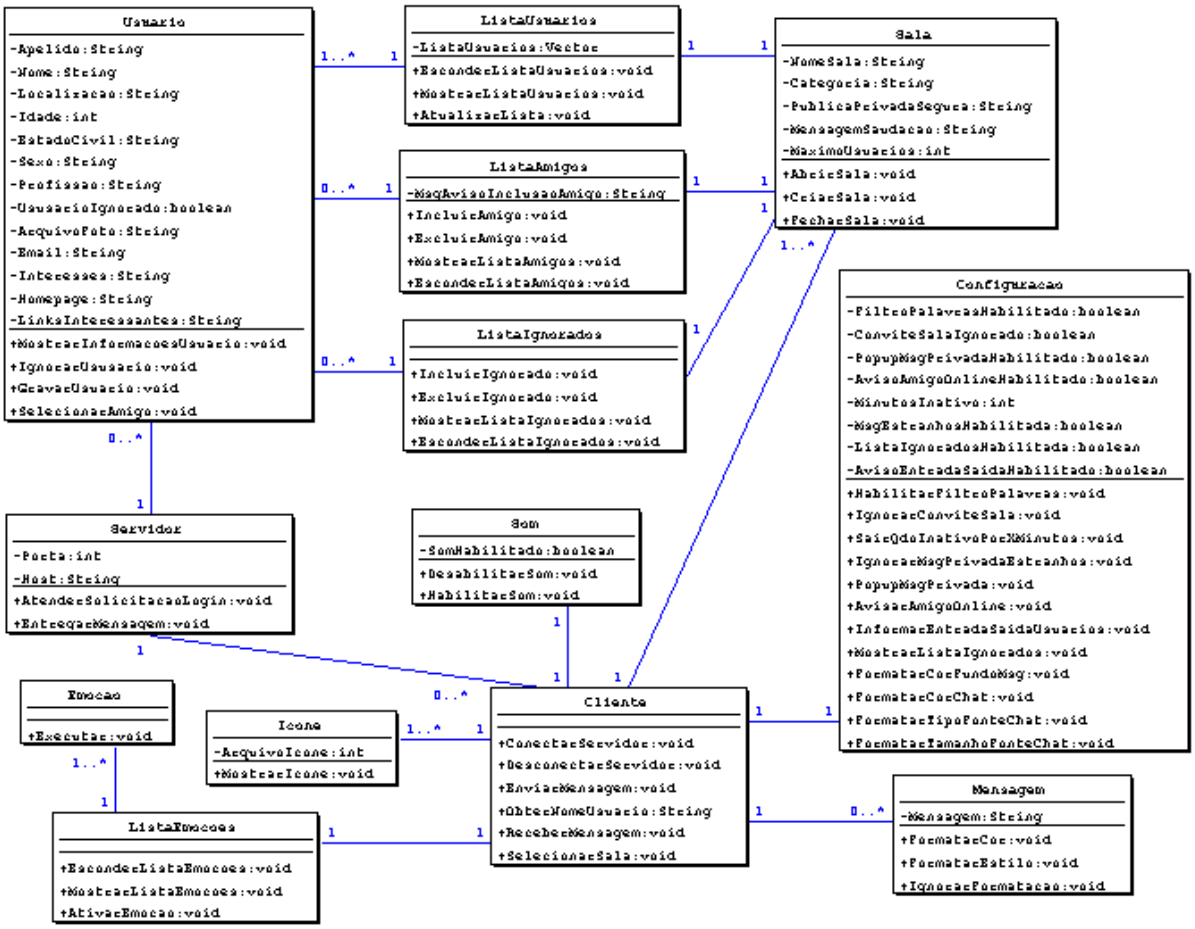


Figura 22 — Modelo simplificado do *chat Yahoo!*

No ***Teen.com*** (*http://chat1.teen.com/eshare/server?action=4*) também é permitido aos clientes conectarem-se a um servidor e acessar as salas de seu interesse. Existem áreas às quais as salas pertencem. Cada sala tem uma lista dos usuários que estão participando. O sistema permite que se envie mensagens privadas, que se localize um usuário e faz uso de ícones. A **Figura 23** mostra o modelo do *chat Teen.com*.

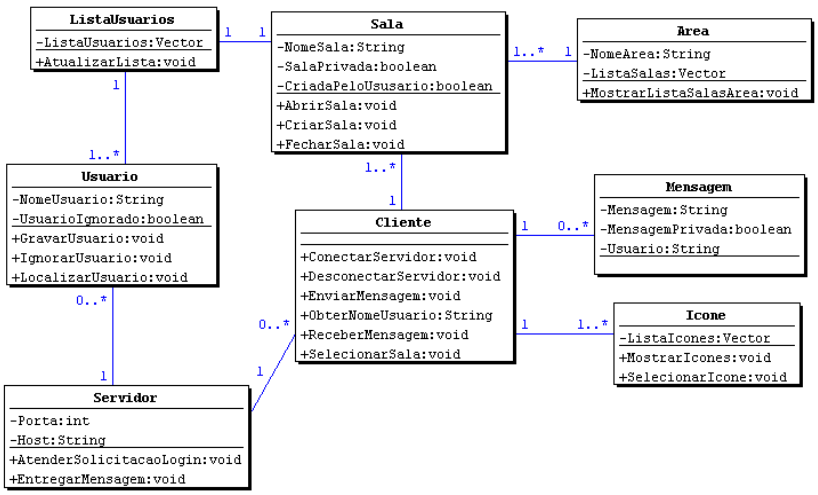


Figura 23 — Modelo simplificado do *Teen.com*

O ***TG Chat*** (*www.iseek.com/cgi-bin/is/ipages?-s+q222a+xxx+yyy+Welcome*) também é um sistema muito simples. Um servidor permite a conexão de vários clientes e uso de diversas salas. É feito uso de ícones. Há uma lista de salas disponíveis e não é possível formatar o texto da mensagem. Pode-se pedir para ver uma lista dos usuários que estão em uma sala. A **Figura 24** traz o modelo deste *chat*.

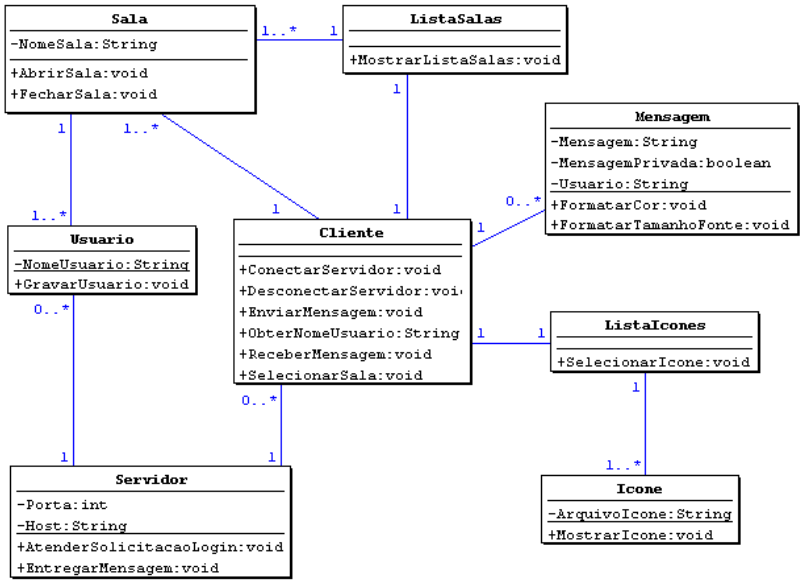


Figura 24 — Modelo simplificado do TG *Chat*

O ***Wrestling Chatbox*** (*www.geocities.com/colosseum/Field/3294/chatbox.html*) possui um servidor ao qual diversos clientes podem conectar-se. Diversas salas podem ser acessadas, públicas ou privadas. Também são mostradas listas de usuários das salas e pode-se utilizar som e ícones. A mensagem pode ter a fonte formatada. Na **Figura 25** é mostrado o modelo deste *chat*.

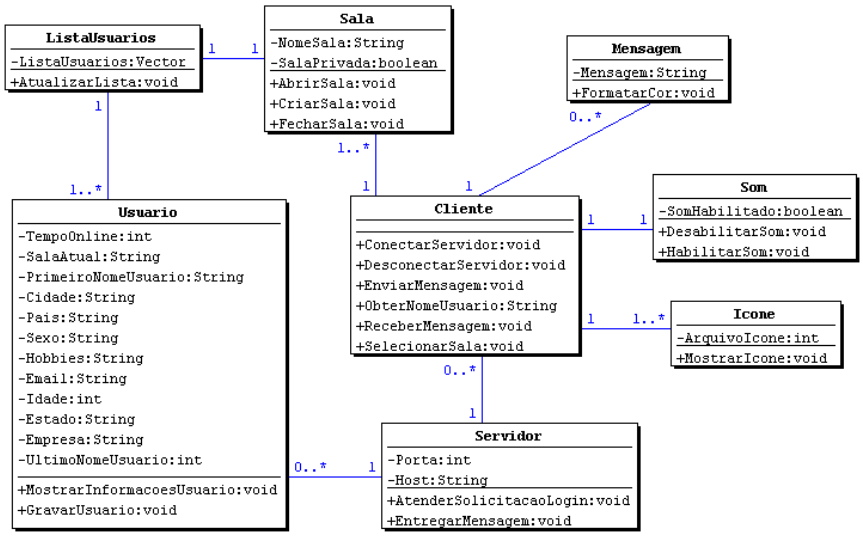


Figura 25 — Modelo simplificado do *Wrestling Chatbox*

É possível perceber pontos em comum nos sistemas síncronos descritos, bem como características que não estão presentes em todos eles, ou que se apresentam em alguns de formas diferentes. Esses pontos em comum devem fazer parte do *kernel* do *Framework*, enquanto os *hot-spots* serão constituídos pelas características diferenciadas. O processo de engenharia reversa que foi realizado resultou no desenvolvimento de um primeiro modelo, que serviu de base para a modelagem do *Framework*. Este modelo pode ser visto na **Figura 26**.

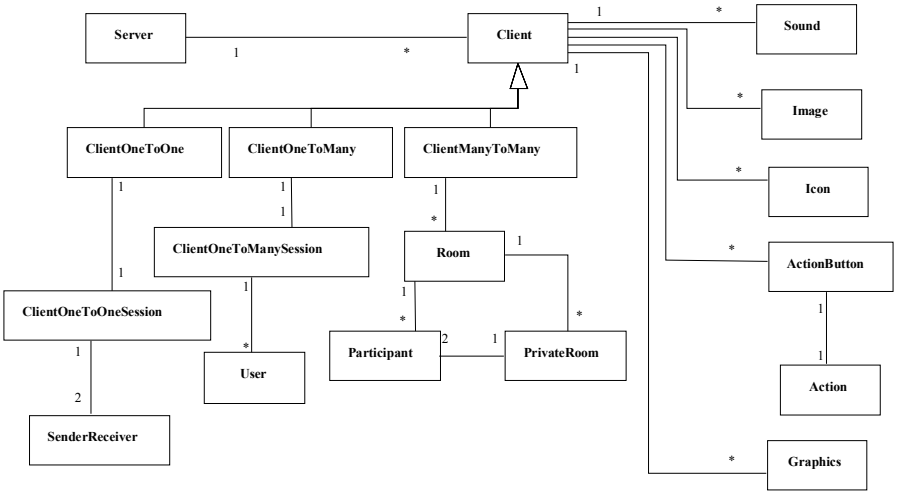


Figura 26 — Modelo genérico dos serviços síncronos analisados

4.2 Pontos de Flexibilização dos Ambientes Analisados

Com base nas informações obtidas, foi possível definir as classes do *kernel* e dos *hot-spots* do *Framework*. Foram definidos alguns *hot-spots* próprios de uma aplicação muitos-para-muitos e outros que são comuns a todos os tipos de aplicação.

Como pontos a serem flexibilizados de um sistema de comunicação síncrona, temos:

- ***Definição de papéis para os participantes*** – para cada participante pode ser definido um papel (aluno, professor) que o permita realizar certas operações tais como:
  - Existência de um mediador na sala – pode-se definir o papel de um mediador, que teria controle sobre os demais participantes, ou não ter essa figura presente;
  - 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.
- ***Definição de funcionalidades*** – a cada papel criado pode ser relacionada uma série de funcionalidades como, por exemplo:
  - Permissão para retirar um participante de uma sala;
  - Ativação de uma URL em todas as aplicações clientes;
  - 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*** – o texto de uma mensagem a ser enviada pode ter cor, tamanho da fonte e estilo definidos pelo usuário.
- ***Uso de som***.

A seguir será apresentado o modelo de classes do *Framework* proposto levando-se em consideração os aspectos já elicitados dos diversos sistemas de comunicação síncrona. O modelo será visto e comentado em partes de forma a proporcionar um melhor entendimento. Nesta descrição será destacado o *kernel* e os *hot-spots* do ambiente.

4.3 Descrição do *Framework* JLearningServices

Na **Figura 27** é mostrada a relação entre o servidor (classe *Server*) e as conexões (classe *Connection*). O servidor pode receber e controlar diversas conexões. Foi utilizado o *Design Pattern State* [22] (circundado por linhas curvas) para definir o estado de uma conexão - aberta ou fechada. Esse padrão comportamental permite que o comportamento de um objeto seja modificado de acordo com o estado em que ele se encontra. Isso pode permitir que se remova conexões "mortas", fazendo com que o servidor trabalhe de maneira mais eficiente. Este *Pattern* também foi utilizado para definir os possíveis estados do servidor: em funcionamento, esperando o recebimento de conexões e fechado. Assim tem-se um maior controle sobre o funcionamento do servidor.

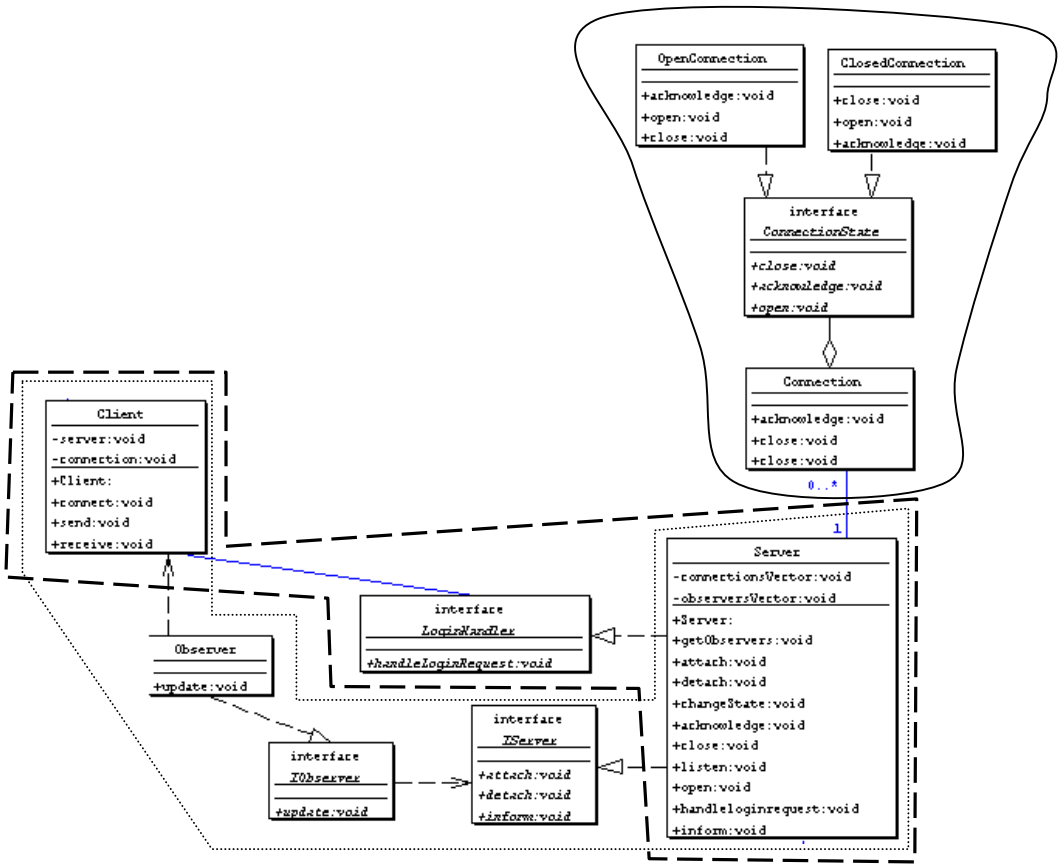


Figura 27 — Parte do modelo ilustrando o uso dos *Design Patterns State*, *Observer* e *Chain Of Responsibility*

O *Design Pattern Observer* [22] (tracejado pequeno) foi utilizado para notificar os clientes (classe *Client)* envolvidos em uma sessão de *chat*, por exemplo, a respeito de alterações que ocorram em um algum cliente específico. Esse padrão comportamental permite que, havendo a modificação de um objeto, os demais sejam notificados e atualizados em tempo de execução.

Um padrão também foi utilizado para controle das requisições de *login*, necessárias para que um sistema seja acessado. Esse controle é feito pelo *Design Pattern Chain of Responsibility* [22] (tracejado em negrito), um modelo comportamental que permite que várias requisições sejam feitas e acumuladas até que possam ser atendidas. Nesse contexto, existe ainda uma classe *UserInformation*, que não está na figura, mas 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*.

Na **Figura 28**, são apresentados os *hot-spots* que podem ser utilizados para todos os tipos de cliente. A classe *Resource* é especializada nas classes *Sound*, *Image*, *Icon*, *ActionButton*, *Text*, *Application* e *Graphic*, permitindo que sejam definidos recursos para uso de sons, imagens, ícones, botões de ação, texto, aplicações e elementos gráficos, respectivamente. 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* [22]. Assim, pode ser feito um desenho (classe *Picture)* composto por qualquer combinação de elementos que especializam a classe *Graphic*.

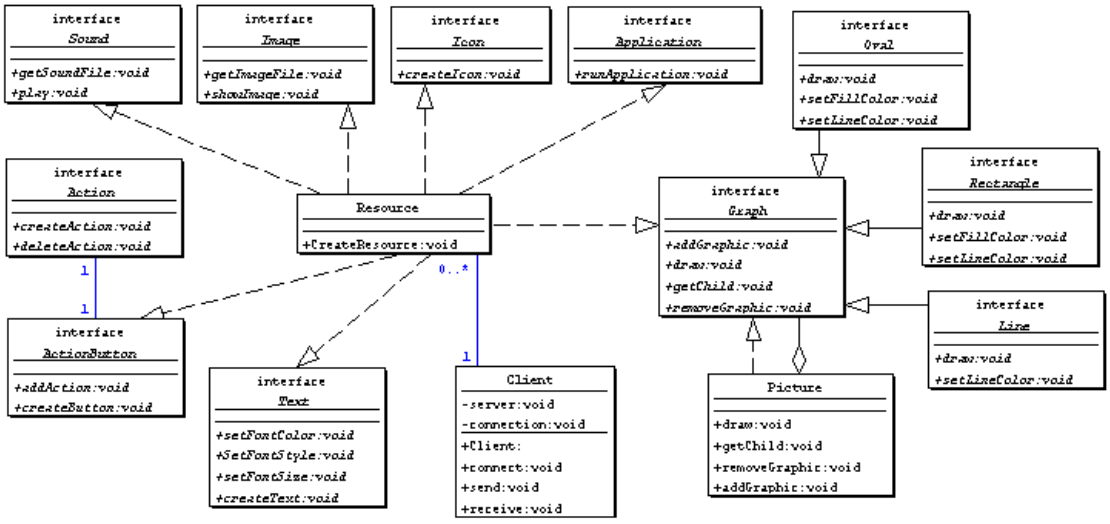


Figura 28 — *Hot-spots* do cliente (classe *Client*)

Os tipos de cliente que podem ser gerados com uso do *Framework* são mostrados na **Figura 29**. A classe *Client* possui especializações de acordo com o tipo de comunicação utilizado pela aplicação que será gerada: um-para-um, um-para-muitos ou muitos-para-muitos. Nesta etapa foi utilizado o *Desing Pattern Application Types* [16], que separa aplicações em categorias de acordo com seu comportamento e características.

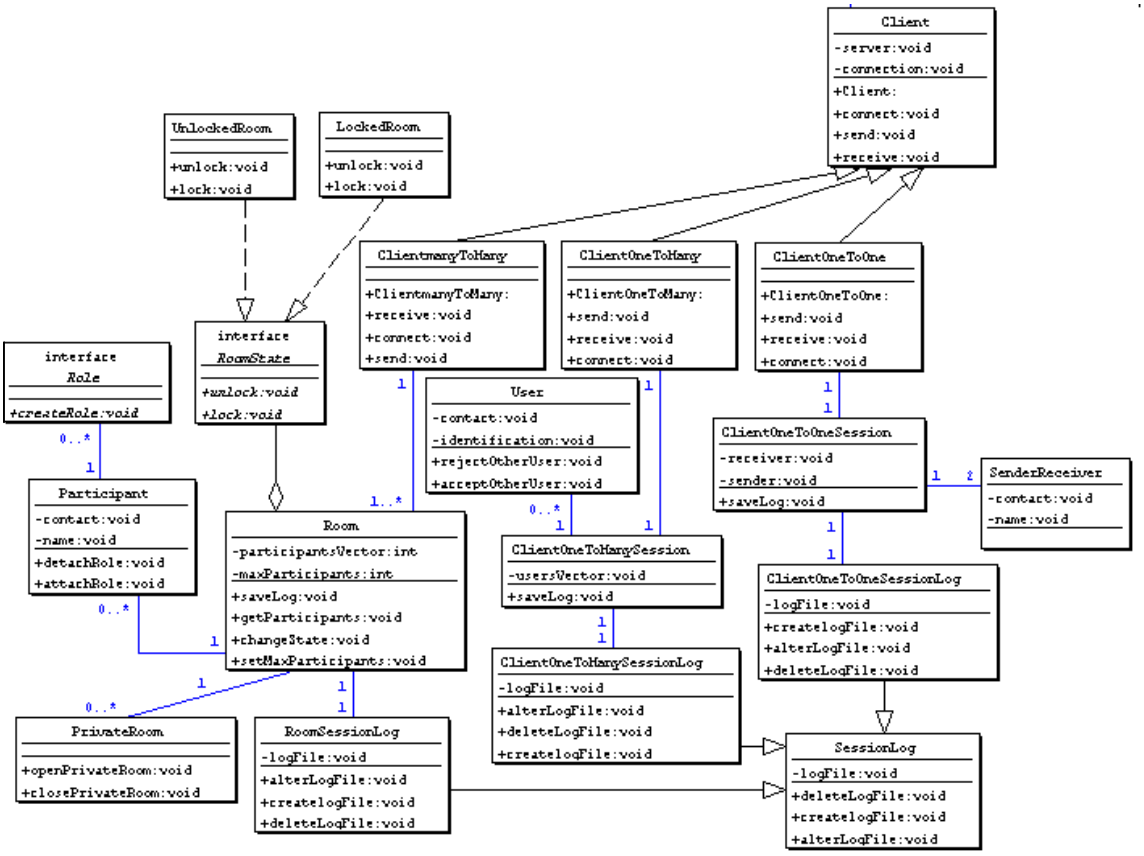


Figura 29 — Tipos de cliente e suas classes

Aplicações muitos-para-muitos (classe *ClientManyToMany*), semelhantes a *chats*, podem ter uma ou mais salas (classe *Room*) 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*.

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* foi especializada para contemplar a gravação desses arquivos por qualquer um dos tipos de aplicação (cada tipo de aplicação implementa esse recurso de acordo com suas particularidades).Aplicações um-para-muitos (classe *ClientOneToMany*) consistem de uma sessão de troca/recebimento de mensagens (classe *ClientOneToManySession*) que é válida enquanto o usuário estiver conectado. Um usuário (classe *User*) recebe de outros solicitação para manter contato, as quais podem ser aceitas ou rejeitadas. Da mesma forma, pode solicitá-la a outros e ser aceito ou rejeitado. Aplicações um-para-um (classe *ClientOneToOne*) também possuem uma sessão de envio/recebimento de mensagens (classe *ClientOneToOneSession*), porém essa interação só pode ocorrer entre duas pessoas (classe *SenderReceiver)*.

Na **Figura 30**, estão definidos os *hot-spots* referentes a aplicações muitos-para-muitos (classe *ClientManyToMany*). Trata-se dos papéis (classe *Role*) que podem ser assumidos por um participante (classe *Participant*).

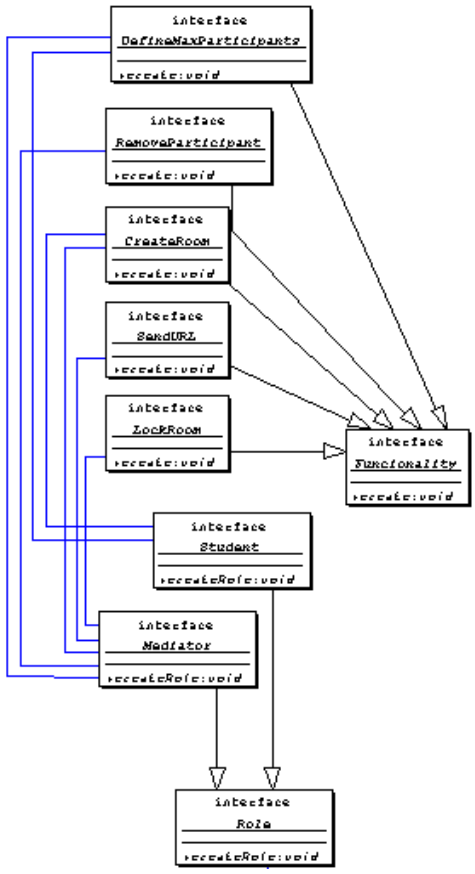


Figura 30 — *Hot-spots* associados ao participante (classe *Participant*)

Dois papéis existentes são *Mediator* e *Student*, mas outros podem ser definidos através da especialização da classe *Role* por meio de herança. A cada papel podem estar associadas diversas funcionalidades (classe *Funcionality*) como, por exemplo, o de retirar um participante da sala. Existem funcionalidades definidas – *RemoveParticipant*, *SendURL*, *CreateRoom*, *LockRoom* e *DefineMaxParticipants* – mas, assim como a classe *Role*, a classe *Funcionality* pode ser especializada para a adição de novas funcionalidades, as quais podem ser associadas a um ou mais papéis.

Todos os *hot-spots* do modelo possuem métodos para sua criação e manipulação, e por serem classes abstratas devem ser implementados na ocasião da instanciação do *Framework*.

A **Figura 31**, na página a seguir, mostra uma visão geral do modelo do JLearningServices.

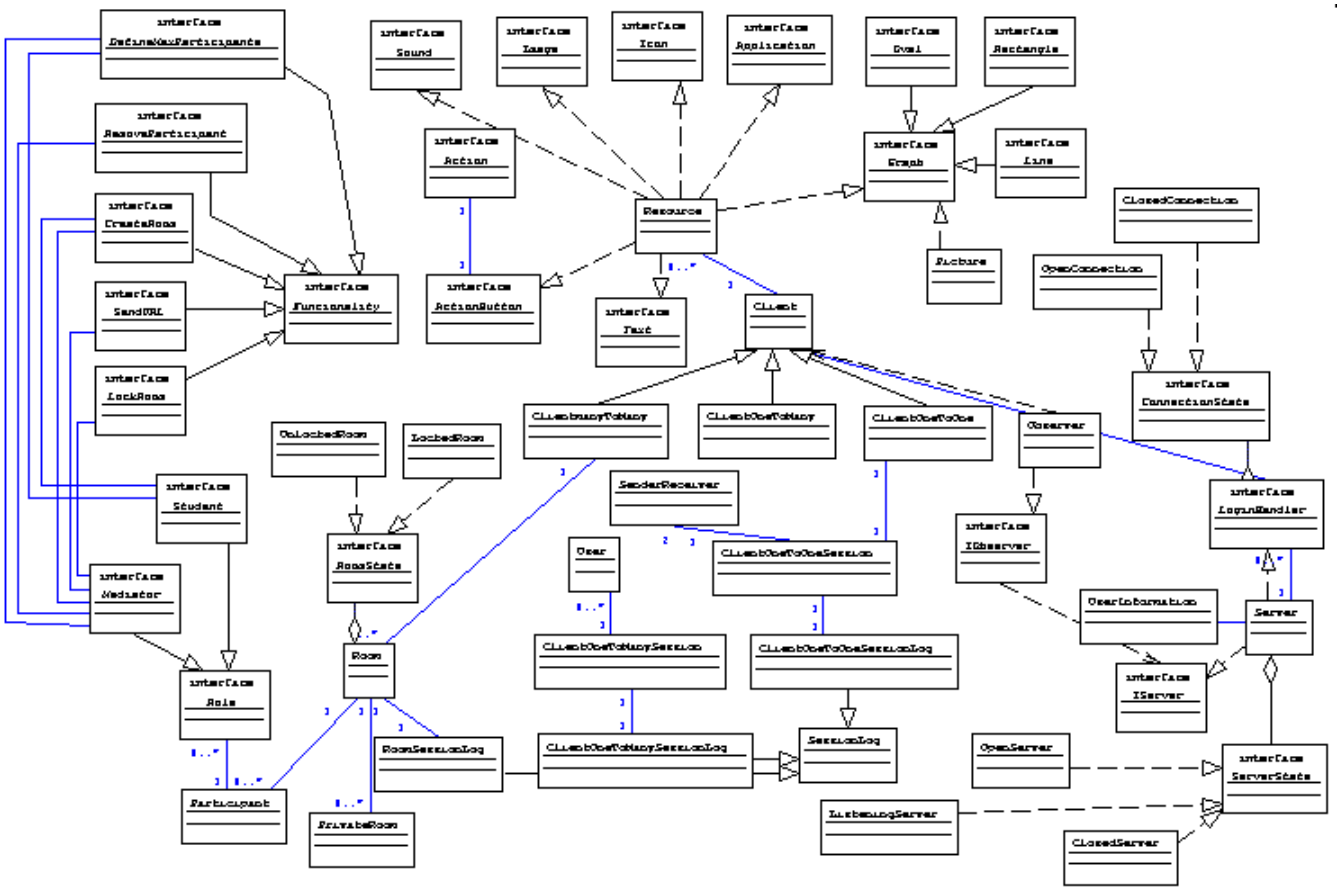


Figura 31 — Visão geral do modelo do JLearningServices

A seção seguinte justifica a utilidade do *Framework* JLearningServices para a área de EAD.

4.4 JLearningServices e EAD

A contribuição do *Framework* JLearningServices é a possibilidade de gerar aplicações síncronas próprias para EAD, com características/funcionalidades específicas desta área.

Podem ser criadas, por exemplo, diferentes aplicações de *chat*, cada uma com algumas características/funcionalidades em comum e outras distintas. Cada uma destas aplicações atenderá a um certo público, buscará satisfazer necessidades pertinentes ao ambiente de EAD em que for incorporada. Isto não se consegue com a simples adaptação de ferramentas já existentes.

Assim, pode-se obter resultados mais proveitosos com o uso das aplicações geradas por este *Framework* (que podem ser, de certa forma, personalizadas para o tipo de ambiente a que se destinam).

Este capítulo apresentou o desenvolvimento do modelo do Framework JLearningServices. No próximo capítulo, teremos uma breve explicação de como instanciá-lo para gerar aplicações síncronas.

5 Instanciação do *Framework*

Este capítulo dará uma breve explicação sobre como deve-se instanciar o Framework para que se obtenha uma aplicação de chat.

Após a modelagem do *Framework*, foi realizada sua implementação. Para isto, foi utilizada a linguagem Java, mais especificamente o JDK (*Java Development Kit*) na versão 1.3, encontrado para *download* gratuito no endereço *www.java.sun.com*. Esta etapa tornou necessário o estudo de recursos da linguagem Java [12][14][29], especialmente recursos para programação distribuída [15][28][38] e implementação de *Design Patterns* [24][25][38].

O JLearningServices é um *Framework* totalmente *white-box*, ou seja, exige que se conheça o código-fonte para realizar sua instanciação. A instanciação pode ser feita estendendo as classes do *Framework* ou complementando o código de classes já existentes.

Como visto no capítulo anterior, o *Framework* possui três tipos de aplicação cliente definidos em seu modelo. Cada uma destas aplicações é uma especialização da classe *Client*, onde são implementados os métodos básicos para receber e enviar mensagens no cliente. Estas classes especializadas herdam os métodos da classe *Client* e implementam o comportamento típico de cada classe de serviço síncrono.

Para, por exemplo, se obter um *chat*, deve-se escolher e utilizar a classe que implementa serviços muitos-para-muitos e as classes a ela relacionadas. A classe que implementa esses serviços é a classe *ClientManyToMany*. Ela está associada à classe *Room*, onde é implementada a estrutura da sala de *chat*. Pode-se estender ou utilizar diretamente a classe *Room* para adicionar elementos de interface gráfica e inserir os *hot-spots* relacionados ao cliente e outros elementos relacionados à classe *Room* da maneira desejada. O mesmo pode ser feito com as demais classes associadas, por exemplo: modificar ou complementar a programação de alguns métodos nas classes *Server*, *ServerState*, *OpenServer*, *ListeningServer* e *ClosedServer* para definir como será tratada a troca de estados do servidor, o que será executado de acordo com o estado que ele assumir. Ou então, complementar a classe *ClientManyToMany* para definir uma interface adicional (uma tela inicial que permita acesso às salas de *chat*, por exemplo) e/ou outros controles ainda não definidos. Pode-se também criar novas classes e associá-las às classes existentes de modo a criar novos recursos para a aplicação.

No caso dos *hot-spots*, deve-se estender a classe do *hot-spot* escolhido e realizar a implementação dos métodos existentes em sua classe, bem como adicionar outros comportamentos desejados.

Por exemplo, se é desejado criar um *chat* com uso de ícones, deve-se estender a classe *Icon* e realizar a implementação do método *ShowIcon()*, bem como outros

métodos que se queira implementar. A classe *Icon* está associada à classe *Resource*, ligada à classe *Client*, que é estendida pela classe *ClientManyToMany*; através desta última pode ser acessada pela classe *Room* para definir onde colocar os ícones e como utilizá-los.

A seguir serão listados os principais métodos das classes mais importantes – *Server* e *Client* – assim como métodos das principais classes associadas a elas, de forma a fornecer mais detalhes sobre como proceder na instanciação do JLearningServices.

5.1 Servidor

As principais classes que formam a estrutura de servidor e que serão descritas a seguir são: *Server* mais *ServerState* e suas subclasses (servidor e seus estados), bem como *AppConnection* mais *ConnectionState* e suas subclasses (conexão e seus estados).

5.1.1 Classe Server

O construtor da classe *Server* apenas inicializa o estado da aplicação servidora (**Figura 32**). A classe *ServerState* define esse valor inicial como *open* (aberto, ou seja, em funcionamento).

```
public Server() {
  state = ServerState.start();
}
```

Figura 32 — Construtor da classe *Server*

O método *connect()*, mostrado na **Figura 33**, coloca a aplicação servidora disponível em uma determinada porta, onde ficará aguardando pedidos de conexão de aplicações cliente. A aplicação servidora entra em estado *listening* (escutando a porta), cria um *socket* servidor que ficará responsável por aceitar, ou não, as conexões solicitadas e cria *streams* de comunicação (um para recepção e outro para envio de informações).

```
protected void connect() throws IOException {
  state = state.processEvent(LISTEN_EVENT);
  conn = new AppConnection();
  conn.createConnection(new ServerSocket(getPort()));
  input = new DataInputStream(new BufferedInputStream(conn.getConnection().getInputStream()));
  output = new DataOutputStream(new BufferedOutputStream(conn.getConnection().getOutputStream()));
}
```

Figura 33 — Método *connect()* da classe *Server*

O método *connect()* utiliza o método *getPort()* para obter o valor da porta a ser utilizada pela aplicação servidora. O código do método *getPort()* é mostrado na **Figura 34**.

```
private int getPort() {
  ResourceBundle resources = null;
  try {
    resources = ResourceBundle.getBundle("PROPERTIES" + System.getProperty(file.separator") + "SERVER", Locale.getDefault());
  }
  catch(MissingResourceException mre) {
    state = state.processEvent(CLOSE_EVENT);
    System.err.println("Server: file Server.properties not found!");
    System.exit(0);
  }
  int port = new Integer(resources.getString("ServerPort")).intValue();
  return(port);
}
```

Figura 34 — Método *getPort()* da classe *Server*

Foi definido um arquivo de configuração (*Server.properties*) para tornar facilmente acessíveis os valores do *host* e da porta que a aplicação servidora utiliza. Este arquivo deve permanecer no diretório *Properties*, que deve ser criado dentro do diretório onde ficam gravadas as classes da aplicação cliente ou da aplicação servidora. Caso este arquivo não seja encontrado, a aplicação servidora assume o estado *closed* (fechada, ou seja, não mais em funcionamento), emite uma mensagem de erro e é fechada.

Para fechar uma conexão estabelecida com o servidor, deve ser utilizado o método *disconnect()* (**Figura 35**), que fecha os *streams* de comunicação e a conexão que está sendo utilizada.

```
protected void disconnect() throws  IOException {
  input.close();
  output.close();
  conn.closeConnection();
}
```

Figura 35 — Método *disconnect()* da classe *Server*

Há ainda os métodos *send(String msg)* e *receive()*, mostrados na **Figura 36**, responsáveis pelo envio e recebimento das informações através dos *streams* de comunicação.

```
protected String receive() throws IOException {
  return(input.readUTF());
}

protected void send(String msg) throws IOException {
  output.writeUTF(msg);
}
```

Figura 36 — Métodos *receive()* e *send(String msg)* da classe *Server*

A seguir, será mostrada a implementação das classes que representam os estados que uma aplicação servidora pode assumir.

5.1.2 Classe ServerState

A implementação do *Design Pattern State* que representa os estados do servidor ficou definida da seguinte forma: a classe *ServerState* possui dois métodos abstratos – *processEvent(int event)* e *enter()* – que devem ser sobrescritos por suas subclasses e um método estático chamado *start()*, que inicializa o estado do servidor como *open* (**Figura 37**).

```
public abstract ServerState processEvent(int event);


public abstract void enter();


public static ServerState start() {
  ClosedServer cServer = new ClosedServer();
  return(cServer.processEvent(OPEN_EVENT));
}
```

Figura 37 — Métodos da classe *ServerState*

As subclasses de *ServerState* (*OpenServer*, *ListeningServer* e *ClosedServer*) implementam os métodos abstratos de forma semelhante. Em ListeningServer, por exemplo, a implementação foi feita da seguinte forma (**Figura 38**): o método *processEvent(int event)* recebe o valor correspondente a um evento que simboliza um estado no qual o servidor deve entrar e realiza esta mudança; o método *enter()* executa uma ação a ser realizada quando o servidor entrar no estado *listening*.

```
public ServerState processEvent(int event) {
  switch (event) {
    case OPEN_EVENT:
      OpenServer oServer = new OpenServer();
      oServer.enter();
      return(oServer);
    case LISTEN_EVENT:
      return this;
    case CLOSE_EVENT:
      ClosedServer cServer = new ClosedServer();
      cServer.enter();
      return(cServer);
    default:
      throw new IllegalArgumentException("Unexpected event!");
  }
}


public void enter() {
  System.out.println("Server is now listening...");
}
```

Figura 38 — Métodos da classe *ListeningServer*

Quando em estado *open*, uma aplicação servidora gerencia solicitações para conexão. A forma como foi implementada a classe que representa as conexões pode ser vista a seguir.

5.1.3 Classe *AppConnection*

A classe *Connection*, que no modelo representa as conexões, no momento da implementação ganhou o nome de *AppConnection* devido a um conflito de nome com a interface *java.sql.Connection* do *Java Development Kit* (JDK). Desta forma, problemas como erros de compilação podem ser evitados.

A classe *AppConnection* guarda o *socket* de uma conexão e possui dois métodos para a criação de conexões: um utiliza o *socket* da aplicação servidora (**Figura 39**) e o outro utiliza o *host* e a porta da aplicação servidora (**Figura 40**).

```
public void createConnection(ServerSocket server) throws IOException {
  socket = server.accept();
  state = ConnectionState.start();
}
```

Figura 39 — Método *createConnection(ServerSocket server)* da classe *AppConnection*

```
public void createConnection(String host, int port) throws IOException {
  try {
    socket = new Socket(host, port);
    state = ConnectionState.start();
  }
  catch(UnknownHostException uhe) {
    System.err.println("Error creating connection: host unknown");
  }
}
```

Figura 40 — Método *createConnection(String host, int port)* da classe *AppConnection*

A classe *AppConnection* ainda possui um método para obter o *socket* da conexão – *getConnection()* – e um método para fechar a conexão – *closeConnection()* (**Figura 41)**.

```
public Socket getConnection() {
  return(socket);
}

public void closeConnection() throws IOException {
  state = state.processEvent(CLOSE_EVENT);
  socket.close();
}
```

Figura 41 — Métodos da classe *ListeningServer*

Conexões também podem assumir alguns estados. A forma como isto foi implementado é explicada a seguir.

5.1.4 Classe *ConnectionState*

A classe *ConnectionState* e suas subclasses *OpenConnection* e *ClosedConnection*, que compõem o *Design Pattern State* para os estados de uma conexão, foram implementadas de forma semelhante ao padrão que representa os estados da aplicação servidora. Apenas os comportamentos que uma conexão passa a ter ao assumir cada estado, ou as ações que ela executa é que são diferentes.

A próxima seção trata da implementação das classes referentes à estrutura de aplicações clientes.

5.2 Cliente

As principais classes da estrutura cliente do *Framework* são as subclasses de *Client*. Das diversas outras classes associadas a estas subclasses, foi priorizado mostrar a implementação da classe *Room*, que representa uma sala de *chat* e é utilizada na aplicação exemplo mostrada neste artigo, bem como a implementação da classe *SessionLog* e suas subclasses, que constituem um recurso interessante para as aplicações.

5.2.1 Subclasses de *Client*

As subclasses da classe *Client* (*ClientOneToOne*, *ClientOneToMany* e *ClientManyToMany*) possuem os métodos para conexão com a aplicação servidora e envio/recebimento de informações. Na classe *ClientManyToMany*, por exemplo, encontram-se os seguintes métodos (**Figura 42** e **Figura 43**): *connect()*, que cria uma conexão e *streams* para comunicação; *disconnect()*, que fecha os *streams* de comunicação e a conexão que estava sendo utilizada; *getHost()* e *getPort()*, que obtém os valores do *host* e da porta utilizados pela aplicação servidora no arquivo de configuração *Server.properties*; *send()* e *receive()*, que envia e recebe, respectivamente, as informações através dos *streams* de comunicação; *getRooms()*, que atua de forma semelhante a *getHost()* e *getPort()*, buscando os nomes das salas existentes em um arquivo de configuração chamado *Rooms.properties*. .

```
public void connect() throws IOException {
  conn = new AppConnection();
  conn.createConnection(getHost(), getPort());
  input = new DataInputStream(new BufferedInputStream(conn.getConnection().getInputStream()));
  output = new DataOutputStream(new BufferedOutputStream(conn.getConnection().getOutputStream()));
}

public void disconnect() throws IOException {
  input.close();
  output.close();
  conn.closeConnection();
}
```

Figura 42 — Métodos *connect()* e *disconnect()* da classe *ClientManyToMany*

```
public String getHost() {
  ResourceBundle resources = null;
  try {
   resources = ResourceBundle.getBundle("PROPERTIES" + System.getProperty("file.separator") + "SERVER", Locale.getDefault());
  }
  catch(MissingResourceException mre) {
    System.err.println("File Properties/Server.properties not found!");
    dispose();
  }
  String host = resources.getString("Host");
  return(host);
}

public int getPort() {
  ResourceBundle resources = null;
  try {
    resources = ResourceBundle.getBundle("PROPERTIES" + System.getProperty("file.separator") + "SERVER", Locale.getDefault());
  }
  catch(MissingResourceException mre) {
    System.err.println("File Properties/Server.properties not found!");
  }
  int port = new Integer(resources.getString("Port")).intValue();
  return(port);
}

public String receive() throws IOException {
  return(input.readUTF());
}

public void send(String msg) throws IOException {
  output.writeUTF(msg);
  output.flush();
}

public Enumeration getRooms() {
  ResourceBundle resources = null;
  try {
   resources = ResourceBundle.getBundle("PROPERTIES" + System.getProperty("file.separator") + "ROOMS", Locale.getDefault());
  }
  catch(MissingResourceException mre) {
    System.err.println("File Properties/Rooms.properties not found!");
  }
  return(resources.getKeys());
}
```

Figura 43 — Demais métodos da classe *ClientManyToMany*

A seguir, é mostrada a implementação da classe *Room*.

5.2.2 Classe *Room*

A classe *Room* possui métodos para obter/definir o número de participantes de uma sala, obter o nome da sala, etc, e utiliza métodos da classe *ClientManyToMany* para conexão e envio/recebimento de mensagens. O mais importante é que ela implementa a interface *Runnable*, que implica na criação do método *run()* (**Figura 44**), responsável pela execução das principais funções da sala de *chat*. Isto permite a criação de *Threads* para as salas.

```
public void run() {
  try {
    (…)
    while (true) {
      String line = client.receive();
      (…)
      outputArea.setEditable(true);
      outputArea.append(line + "\n");
      outputArea.setEditable(false);
    }
  }
  catch(Exception e) { System.err.println("Exception: " + e.getMessage()); }
  finally {
    try {
      client.disconnect();
    }
    catch(IOException ioe) { System.err.println("IOException: " + ioe.getMessage()); }
  }
  SwingUtilities.invokeLater(new Runnable() {
    public void run() { repaint(); }
  });
}
```

Figura 44 — Método *run()*

Salas de *chat* podem ter sessões gravadas em arquivos de *log*. A implementação da classe *SessionLog* é descrita a seguir, mostrando a ferramenta desenvolvida para lidar especificamente com *log* de salas de chat.

5.2.3 Classe *SessionLog*

Um recurso implementado foi a possibilidade de gravação de *logs* das sessões. A classe *SessionLog* possui métodos para a criação, deleção, leitura e gravação dos arquivos de *log – createLog(String fileName)*, *deleteLog(File file)*, *readLog(File file)* e *writeLog(File file, String textLine)*. Estes métodos podem ser vistos na **Figura 45** e na **Figura 46**.

```
public File createLog(String fileName) throws IOException {
  return(new File(fileName));
}

public void deleteLog(File file) throws IOException {
  if (file.exists()) {
    file.delete();
  }
}
```

Figura 45 — Métodos *createLog(String fileName)* e *deleteLog(File file)* da classe *SessionLog*

```
public String readLog(File file) throws IOException {
  BufferedReader reader = new BufferedReader(new FileReader(file.getPath()));
  String text = "";
  while (true) {
    String line = reader.readLine();
    if (line != null) {
      text += line + '\n';
      reader.skip(1);
    }
    else {
      break;
    }
  }
  reader.close();
  return(text);
}

public void writeLog(File file, String textLine) throws IOException {
  BufferedWriter writer = new BufferedWriter(new FileWriter(file.getPath(), true));
  writer.write(textLine);
  writer.newLine();
  writer.close();
}
```

Figura 46 — Demais métodos da classe *SessionLog*

As subclasses de *SessionLog* implementam outros recursos para o trabalho com os arquivos de *log*. Na classe *RoomSessionLog*, por exemplo, é definido um editor para leitura, edição e gravação deste tipo de arquivo. Uma visão da interface deste editor é mostrada a seguir (**Figura 47**).

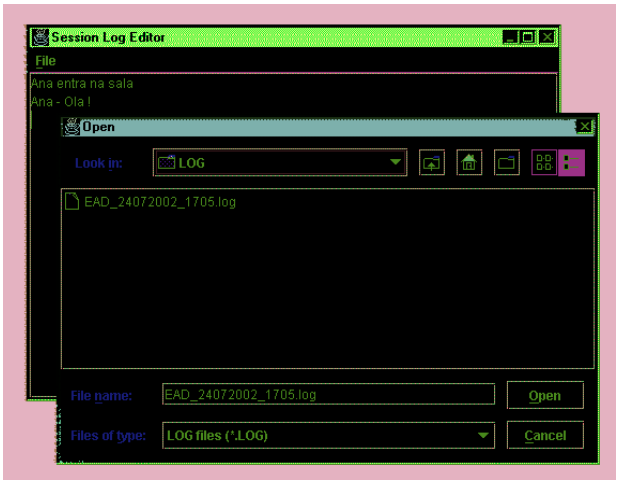


Figura 47 — Editor de arquivos de *log*

A próxima sessão traz uma orientação de como proceder para implementar os *hot-spots* do *Framework*.

5.3 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.

Um *hot-spot* existente no modelo do JLearningServices é *Funcionality*, que representa funcionalidades que podem ser associadas a determinados papéis (classe *Roles*) assumidos por participantes (classe *Participant*) de uma sala de *chat*. Para criar uma nova funcionalidade, é preciso implementar essa classe de interface, codificando na nova classe os métodos especificados em *Funcionality* e criando outros métodos conforme desejado.

A seguir é mostrado o código da interface *Funcionality* e a estrutura de uma funcionalidade criada a partir de sua implementação, chamada *SendURL* e que já faz parte do *Framework* (**Figura 48**).

```
public interface Funcionality {
  public void defineFuncionality();
}



public class SendURL implements Funcionality {

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

  (…)  // podem ser criados outros métodos aqui
}
```

Figura 48 — Classes *Funcionality* e *SendURL*

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

5.4 Exemplo de Instanciação

Para exemplificar o uso do JLearning Services, 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*, mostrada na **Figura 49**.

```
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 49 — Estrutura de código principal da classe *ChatServer*

Na **Figura 50**, pode ser vista uma tela com a execução da aplicação servidora.

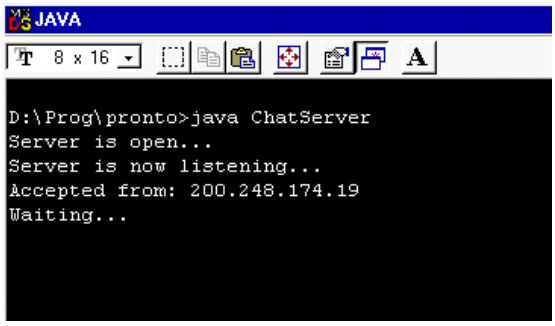


Figura 50 — Exemplo de execução de aplicação servidora

Para a aplicação cliente foi complementada a programação das classes *ClientManyToMany* e *Room* a fim de definir a interface gráfica do *chat* e seu comportamento.

Para executar a aplicação desejada, deve-se definir classes que controlam como será feita essa execução. Por exemplo, para executar a aplicação de *chat*, pode-se definir a classe da **Figura 51** para chamá-la.

```
import java.lang.*;

public class ClientApplication {

  public ClientApplication(String client) {
    Client obj = null;
    Class c  = null;
    try {
      c = Class.forName(client);  // cria uma classe da aplicação ClientManyToMany, definida por parâmetro
     obj = (Client)c.newInstance();  // sobrescreve os métodos abstratos de Client com os de ClientManyToMany
    }
    catch(ClassNotFoundException cnfe) {
      System.err.println("ClassNotFoundException: " + cnfe.getMessage());
    }
    catch(IllegalAccessException iae) {
      System.err.println("IllegalAccessException: " + iae.getMessage());
    }
    catch(InstantiationException ie) {
      System.err.println("InstantiationException: " + ie.getMessage());
    }
  }

  public static void main(String[] args) {
    ClientApplication("ClientManyToMany");
  }

}
```

Figura 51 — Exemplo de classe que executa uma aplicação cliente

Este tipo de implementação pode ser usado sempre que precisarmos substituir métodos de classes abstratas pelos métodos das classes concretas que as estendem, a fim de executar os métodos concretos.

Na **Figura 52**, pode ser vista a aplicação de *chat* sendo executada.

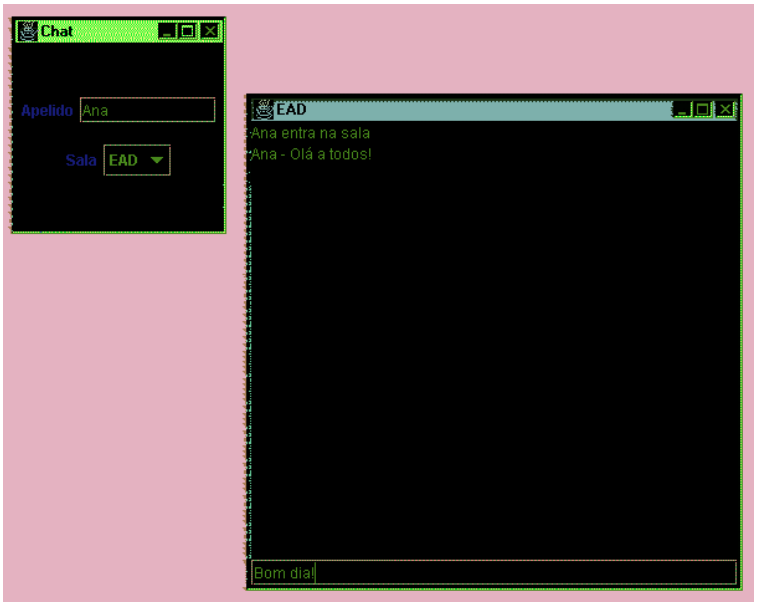


Figura 52 — Aplicação exemplo (*chat)*

Como trabalho futuro seria necessária a revisão e complementação tanto do modelo como da implementação dos métodos básicos do *Framework*, para torná-lo o mais completo e eficiente possível. O *Framework* pode tornar-se mais e mais *blackbox*, permitindo a geração automática de alguns elementos. Novas funcionalidades podem também ser integradas, melhorando o funcionamento do *Framework* e trazendo mais opções de *hot-spots* para a geração de seus tipos de aplicação. Este é um trabalho complexo e que necessita de bastante tempo para ser realizado.

Este último capítulo apresentou uma breve descrição de como criar uma aplicação de chat utilizando o Framework JLearningServices e colocou a necessidade de futura revisão e complementação para torná-lo uma ferramenta mais completa e eficiente.

6 Conclusão

A tecnologia de *Frameworks* aliada ao uso *Design Patterns* para desenvolvimento de sistemas proporciona redução de tempo e custo, com aumento da qualidade do *software* produzido. Pode-se obter diferentes aplicações (pertencentes a um mesmo domínio) sem a necessidade de novo trabalho de análise, projeto e codificação de todo o ambiente. Apenas se define o comportamento dos *hot-spots* desejados para aquela aplicação. O uso de *Design Patterns* facilita a modelagem de *Frameworks*.

Essa união de tecnologias se mostra adequada para áreas onde os requisitos mudam rapidamente, como o EAD. Com seu uso, novas funcionalidades podem ser mais facilmente integradas a uma ferramenta do que através do método tradicional de desenvolvimento orientado a objetos.

A metodologia utilizada (processo baseado na análise de domínio) mostrou-se consistente para o desenvolvimento do JLearningServices, uma vez que havia experiência adquirida no domínio de problema ao qual o *Framework* está relacionado.

Em se tratando de ferramentas síncronas, o processo de análise de requisitos não apresentou problemas no que diz respeito à estrutura das aplicações, por se tratarem de tipos de sistemas conhecidos. Porém, praticamente não existem aplicações desse tipo especificamente destinadas ao ensino. Assim, a busca de informações junto a pessoas ligadas à área do ensino, bem como a busca de aplicações que tivessem o maior número de características aproximadas a esse domínio, foi essencial para complementar a visão dos requisitos necessários ao *Framework*.

A principal contribuição do trabalho é o desenvolvimento de um *Framework* para a área de EAD, carente de recursos, e que permitirá 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 essas ferramentas, que existem já há muito tempo, de recursos/características para EAD, de forma que possam ser utilizadas em ambientes de ensino de maneira mais proveitosa.

O ambiente foi desenvolvido segundo uma metodologia de desenvolvimento em espiral. Nas fases de implementação e instanciação do protótipo utilizou-se a linguagem Java. Recursos de EAD foram sendo incorporados ao *Framework* durante seu desenvolvimento.

Este trabalho está vinculado ao projeto *WebComposeJ*: Uma Linguagem para Composição de Serviços *Web* em Espaços Compartilhados, que tem enfoque em *Web-based education*.

O desenvolvimento deste *Framework* foi de um grande aprendizado e propiciou o contato com áreas que despertaram um grande interesse. O conhecimento adquirido com o estudo de *Frameworks* e *Design Patterns* foi importante. Foi possível perceber na prática que, apesar da complexidade que há na criação deste tipo de ferramenta, os benefícios obtidos são muito grandes e vale a pena investir em sua criação.

Por ser aplicado à área de EAD, o trabalho tornou-se ainda mais proveitoso. O aprendizado foi muito grande, tendo em vista o pouco contato anterior com assuntos relacionado à educação, principalmente com o EAD. Ficou clara a importância dessa modalidade de ensino através dos tempos e principalmente hoje, quando se tem na *Web* uma grande fonte de recursos e oportunidades. Sem dúvida, é um tema de muita relevância e que merece toda a atenção de educadores, pesquisadores e instituições. Mas isso não quer dizer que se deva dar menor atenção ao ensino presencial, até mesmo porque muitas práticas de EAD são baseadas em práticas utilizadas no ensino presencial. Deve-se buscar a evolução dessas duas modalidades, sanando problemas pertinentes a cada uma delas, fortalecendo aquilo que têm de melhor e propiciando a colaboração entre elas, pois cada uma tem seu papel e sua importância. A implementação de um exemplo de aplicação do *Framework* foi trabalhosa, mas propiciou um melhor entendimento do desenvolvimento de *Frameworks* e uso de *Design Patterns*, e um maior conhecimento da linguagem Java, que se mostra adequada para o desenvolvimento de diversos tipos de aplicações.

Um aspecto importante a ressaltar é que se busca não somente que ferramentas síncronas sejam desenvolvidas a partir do uso de *Frameworks*, mas também ferramentas assíncronas. Existem diversas partes de sistemas para EAD que podem ser geradas desta forma, trazendo todos os benefícios já citados neste trabalho.

Referências

[1] ALEXANDER, Christopher; ISHIKAWA, Sara. SILVERSTEIN, Murray; JACOBSON, Max; FIKSDAHL-KING, Ingrid; ANGEL, Shlomo. **A Pattern Language**. Oxford University Press, New York, 1977. ***In [22]***.

[2] BASSET, Paul. **Framing Software Reuse: Lessons From the Real World**. Yourdon Press Computing Series, 1997.

[3] BELLONI, Maria Luiza. **Educação a Distância**. 2 ed. Campinas: Autores Associados, 2001.

[4] BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **The Unified Modeling Language User Guide**. Reading: Addison-Wesley, 1999.

[5] BRITAIN, Sandy; LIBER, Oleg. **A Framework for Pedagogical Evaluation of Virtual Learning Environments**. University of Wales, Bangor.

[6] COAD, Peter; MAYFIELD, Mark. **Projeto de Sistemas em Java: Construindo Aplicativos e Melhores Applets**. São Paulo: MAKRON Books, 1998.

[7] CRAIG, Larman. **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design**. Upper Saddle River: Prentice Hall, 1998.

[8] CRESPO, Sérgio; FONTOURA, Marcus; LUCENA, Carlos José. **Using Viewpoints, Frameworks, and Domain-Specific Languages to Enhance Software Reuse**. European Reuse Workshop - ERW'98. Madrid, Spain, 1998.

[9] CRESPO, Sérgio; FONTOURA, Marcus; LUCENA, Carlos José. **A Web-Based Educational Environments Comparison Using a Conceptual Model Compatible with the EDUCOM/IMS Platform**. Simpósio Brasileiro de Informática na Educação (SBIE'98). Fortaleza, Brasil, 1998.

[10] CRESPO, Sérgio; LUCENA, Carlos José; STAA, Arndt V. **FrameConstructor: Uma Ferramenta para Suporte a Construção Sistemática de Frameworks**. In: Monografias em Ciências da Computação. MCC42/98. PUC-RIO, Departamento de Informática, 1998.

[11] CRESPO, Sérgio. **Composição em WebFrameworks**. Tese de Doutorado. PUC-Rio, Departamento de Informática, 2000.

[12] DACONTA, Michael; MONK, Eric; KELLER, J Paul; BOHNENBERGER, Keith. **Java Pitfalls: Time-Saving Solutions and Workarounds to Improve Programs**. New York: John Wiley & Sons, 2000.

[13] DELLING, Rudolf Manfred. **Towards a Theory of Distance Education**. ICDE Bulletin, 13, 21-25. 1987. ***In [54]***.

[14] ECKSTEIN, Robert. **Java Swing**. Sebastopol: O'Reilly & Associates, 1998.

[15] FARLEY, Jim. **Java Distributed Computing**. Sebastopol: O'Reilly & Associates, 1998.

[16] FAYAD, Mohamed; SCHMIDT, Douglas; JOHSON, Ralph. **Building Application Frameworks: Object-Oriented Foundations of Framework Design**. New York: John Wiley & Sons, 1999.

[17] FAYAD, Mohamed; SCHMIDT, Douglas; JOHSON, Ralph. **Implementing Application Frameworks: Object-Oriented Frameworks at Work**. New York: John Wiley & Sons, 1999.

[18] FONTOURA, Marcus; PREE, Wolfgang; RUMPE, Bernhard. **UML-F: A Modeling Language for Object-Oriented Frameworks**. Princeton University, Department of Computer Science.

[19] FONTOURA, Marcus; HAEULSER, E. H.; LUCENA, Carlos José. **A Framework Design and Instantiation Method Based on Viewpoints**. In: Monografias em Ciência da Computação. MCC/98. PUC-RIO, Departamento de Informática, 1998.

[20] FONTOURA, Marcus; CRESPO, Sérgio; LUCENA, Carlos José; ALENCAR, Paulo; COWAN, Donald. **Using Viewpoints to Derive Object-Oriented Frameworks: A Case Study in the Web-Based Education Domain**. The Journal of Systems and Software, no. 54, Amsterdam, 2000.

[21] FROEHLICH, Garry; HOOVER, James; LIU, Ling; SORENSON, Paul. **Designing Object-Oriented Frameworks**. University of Alberta, Canada.

[22] GAMMA, Erich; HELM, Richard; JOHSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software**. Reading: Addison-Wesley, 1995.

[23] GARRISON, D. R.; SHALE, D. **Mapping the Boundaries of Distance Education: Problems in Defining the Field**. The American Journal of Distance Education, 1(1), 7-13. 1987. ***In [54]***.

[24] GRAND, Mark. **Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML**. 1 v. New York: John Wiley & Sons, 1998.

[25] GRAND, Mark. **Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML**. 2 v. New York: John Wiley & Sons, 1998.

[26] GRINGS, Eliane; MALLMANN, Marli; DAUDT, Sônia. **Ambiente Virtual de Aprendizagem: Uma Experiência Interdisciplinar no Ensino Superior**. XI Simpósio Brasileiro de Informática na Educação (SBIE'2000). Maceió, Brasil, 2000.

[27] GUAREZZI, Rita de Cássia; CAMILOTTI, Luciane. **Novas Tecnologias e Qualidade na Educação**. XI Simpósio Brasileiro de Informática na Educação (SBIE'2000). Maceió, Brasil, 2000.

[28] HAROLD, Elliote Rusty. **Java Network Programming**. 2 ed. Sebastopol: O'Reilly & Associates, 2000.

[29] HORSTMANN, Cay. **Core Java 1.2**. Palo Alto : Sun Microsystems, 1999.

[30] JOHNSON, R.E. **Reusing Object-Oriented Design**. University of Illinois, Technical Report UIUCDCS 91-1696, 1991. ***In [11]***.

[31] JOHNSON, R.E.; FOOTE, B. **Designing Reusable Classes**. Journal of Object-Oriented Programming, June 1988. ***In [11]***.

[32] KAYE, Anthony. **Origins and Structures**. In A. Kaye & G. Rumble (Eds.), Distance Teaching for Higher and Adult Education (pp. 15-31), London: Croom Helm, 1981. ***In [54]***.

[33] KEEGAN, Desmond. **The Foundations of Distance Education**. London: Croom Helm, 1986. ***In [54]***.

[34] KOSKIMIES, Kai; MÖSSENBÖCK, Hanspeter. **Designing a Framework by Stepwise Generalization**. Proceedings of the 5th European Software Engineering Conference. Lecture Notes in Computer Science 989. Springer-Verlag, 1995.

[35] KRAMER, Erika; MIRAGEM, Erminda; MOREIRA, Maria; SILVA, Mariza; OLIVEIRA, Maria; SOUZA, Olga; DANIEL, Péricles. **Educação a Distância: Da Teoria à Prática**. Porto Alegre: Alternativa, 1999.

[36] KREUTZ, Reinhard; KIESOW, Sven; SPITZER, Klaus. **NetChat: Communication and Collaboration via WWW**. Technical University Aachen, Institute for Medical Computer Science, Germany.

[37] KRISTENSEN, Bent Bruun. **Roles: Conceptual Abstraction Theory and Practical Language Issues**. Special Issue of Theory and Practice of Object Systems (TAPOS) on Subjectivity on Object-Oriented Systems, 1996.

[38] LEA, Doug. **Concurrent Programming in Java: Design Principles and Patterns**. 2 ed. Boston: Addison-Wesley, 2000.

[39] LUCENA, Carlos; FUKS, Hugo. **Professores e Aprendizes na Web: A Educação na Era da Internet**. Rio de Janeiro: Clube do Futuro, 2000.

[40] MAIA, Carmem. **EAD-BR: Educação a Distância no Brasil na Era da Internet**. São Paulo: Universidade Anhembi Morumbi, 2000.

[41] MAIA, Carmem. **Guia Brasileiro de Educação a Distância**. São Paulo: Editora Esfera, 2001.

[42] MARKIEWICZ, Marcus; LUCENA, Carlos José. **Understanding Object-Oriented Framework Engineering**. Inf. MCC38/00. PUC-RIO, Outubro 2000.

[43] MARKIEWICZ, Marcus; LUCENA, Carlos José. **Issues on Object-Oriented Framework Development**. ACM Crossroads, 7.4, Summer 2001.

[44] MATTSON, Michael. **Object-Oriented Frameworks: A Survey of Methodological Issues**. Licentiate Thesis, Department of Computer Science, Lund University, CODEN: LUTEDX/(TECS-3066)/1-130/(1996), also as Technical Report, LU-CS-TR: 96-167, Department of Computer Science, Lund University, 1996. ***In [11]***.

[45] MATTSON, Michael; BOSCH, Jan; FAYAD, Mohamed. **Framework Integration: Problems, Causes and Solutions.** Communications of ACM, Vol. 42, No 10 (Oct. 1999), pp 80-87, 1999. ***In [11].***

[46] MATTSON, Michael. **Evolution and Composition Object-Oriented Frameworks**. PhD Thesis. University of Karlskrona/Ronneby, Department of Software Engineering and Computer Science, 2000. ***In [11]***.

[47] PETERS, Otto. **Didática do Ensino a Distância: Experiências e Estágio da Discussão numa Visão Internacional**. São Leopoldo: UNISINOS, 2001.

[48] PREE, Wolfgang. **Design Patterns for Object-Oriented Software Development**. Reading: Addison-Wesley, 1995.

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

[50] PRESSMAN, Roger. **Software Engineering: A Practitioner's Approach**. 5 ed. Boston: McGraw-Hill, 2001.

[51] RHEINHEIMER, Letícia Rafaela; CRESPO, Sérgio. **JLearning Services: Um Framework para Serviços Síncronos em Ambientes para EAD**. XII Simpósio Brasileiro de Informática na Educação (SBIE'2001). Espírito Santo, UFES, Brasil, 2001.

[52] SANCHES, Iolanda; PRADO, Antonio. **Framework para Ensino a Distância via Web: Ferramentas para Internet/Intranet/WWW**. XI Simpósio Brasileiro de Informática na Educação (SBIE'2000). Maceió, Brasil, 2000.

[53] SCHMIDT, Douglas; FAYAD, Mohamed; JOHNSON, Ralf. **Software Patterns**. Communications of the ACM, 39(10), October 1997.

[54] SIMONSON, Michael; SMALDINO, Sharon; ALBRIGHT, Michael; ZVACEK, Susan. **Teaching and Learning at Distance: Foundations of Distance Education**. New Jersey: Prentice Hall, 2000.

[55] SOMMERVILLE, Ian. **Software Engineering**. 6 ed. Harlow: Addison-Wesley, 2001.

[56] TRAVIS, Greg. **Building a Java Chat Server**. Pode ser encontrado em http://www.ibm.com/developerWorks/.

[57] UMAR, Amjad. **Application (RE) Engineering Building Web-Based Applications and Dealing with Legacies**. New Jersey: Prentice Hall, 1997.

[58] WIRFS-BROCK, Rebecca; WILKERSON, Brian; WIENER, Lauren. **Designing Object-Oriented Software**. Englewood Cliffs: Prentice-Hall, 1990.

[59] _____. **Leveraging Object-Oriented Frameworks**. 1993. Pode ser encontrado em http://www.ibm.com/developerWorks/.

[60] _____. **Building Object-Oriented Frameworks**. Pode ser encontrado em http://www.ibm.com/developerWorks/.