Tiago Caminha Gaspar

Linha de Produtos de Software para Comunicação Síncrona na Web

Dissertação apresentada ao Programa de Pós Graduação do Departamento de Computação - UFSCar, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Orientador:
Antonio Francisco do Prado

Co-orientador:
Cesar Augusto Camillo Teixeira

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

São Carlos, SP - Brasil

Junho de 2010

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

G249Lp

Gaspar, Tiago Caminha.
Linha de produtos de software para comunicação síncrona na web / Tiago Caminha Gaspar. -- São Carlos : UFSCar, 2012.
96 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2010.

1. Engenharia de software. 2. Linha de produtos de software. 3. Web 2.0. 4. Sistemas colaborativos. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

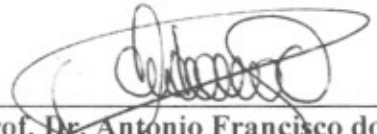
Programa de Pós-Graduação em Ciência da Computação

"Linha de Produtos de Software para Comunicação síncrona na Web"

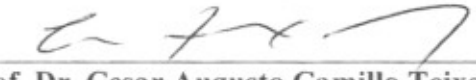
TIAGO CAMINHA GASPAR

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

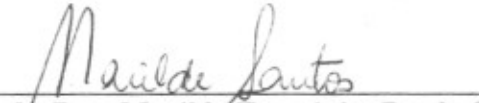
Membros da Banca:



Prof. Dr. Antonio Francisco do Prado
(Orientador - DC/UFSCar)



Prof. Dr. Cesar Augusto Camillo Teixeira
(Co-orientador - DC/UFSCar)



Profa. Dra. Marilde Terezinha Prado Santos
(DC/UFSCar)

Prof. Dr. Ivan Luiz Marques Ricarte
(UNICAMP)

São Carlos
julho/2010

Agradecimentos

Agradeço aos meus orientadores Antonio Francisco do Prado e Cesar Camillo Teixeira tanto pelos ensinamentos acadêmicos, que foram fundamentais na elaboração deste trabalho, quanto pelos ensinamentos de vida que levo comigo ao fim dessa jornada.

Agradeço aos membros da banca Ivan Ricarte e Marilde Santos pela gentileza de participar da minha defesa de mestrado. Agradeço em especial ao Ivan pelas relevantes contribuições oferecidas na qualificação de mestrado.

Agradeço à FAPESP pelo importante apoio durante este trabalho. Parte dos resultados obtidos foram decorrentes da minha participação no Projeto Tidia-Ae financiado pela FAPESP.

É com felicidade que posso dizer que não fiz uma jornada solitária. Muitas pessoas estiveram ao meu lado contribuindo com conhecimentos, ajudas práticas, conselhos espirituais, ou simplesmente trazendo a alegria do convívio. Os colegas de Lince, do Departamento de Computação da UFSCar e do Intermedia-ICMC-USP deram essas e muitas outras contribuições. Agradeço ao Daniel Cugler, Cristiane Yaguinuma, Henrique Mello, Arthur, Cláudia Roberta, Cássio Pereira e Flávia Linhalis pela participação no projeto Tidia-Ae. Agradeço ao Erick Melo e a Maria da Graça pelas boas idéias. Agradeço ao Victor Laguna e Ricardo de Almeida pela amizade e pela ajuda em outras atividades enquanto estava dedicado a escrita deste trabalho.

Agradeço ao Laboratório Lince pelas oportunidades e pela ótima estrutura, fundamental para o bom andamento das atividades realizadas nesta pesquisa.

Agradeço especialmente à Kamila Rios pela alegria, compreensão e apoio incondicional em todas as etapas deste trabalho. Agradeço também pelas importantes contribuições feitas na revisão deste trabalho.

Por fim, agradeço a minha família pelo apoio e carinho, em especial a minha mãe Anaíza Gaspar e ao meu pai Antônio Gaspar por sempre estarem ao meu lado. Agradeço também aos meus irmãos por sempre terem sido meus melhores amigos.

Resumo

O conjunto de conceitos e tecnologias que caracterizam a Web 2.0 revolucionou e estendeu as práticas de comunicação assistidas por sistemas computacionais. Aplicações com comunicação multimídia síncrona, interfaces ricas, tendo a Web como plataforma são exemplos dessa revolução. No entanto, esse tipo de aplicação ainda é pouco explorada apesar do seu potencial. A maioria das comunicações apoiadas por meios computacionais acontece de forma textual através de blogs, wikis e redes sociais. Entende-se que com a evolução dos meios de transmissão de dados, as comunicações apoiadas por computador podem ser melhoradas com a incorporação de áudio e vídeo de qualidade, tornando essas comunicações mais próximas daquelas que fazemos quotidianamente de forma presencial. Neste trabalho, apresenta-se instrumentos tanto teóricos quanto tecnológicos para facilitar desenvolvimento de aplicações de comunicação síncrona com interfaces ricas usando a Web como plataforma. A experiência no domínio permite identificar desafios inerentes ao desenvolvimento desse tipo de aplicações e características comuns a essas aplicações. Este trabalho descreve uma abordagem de reúso de software para construção de aplicações de comunicação síncrona baseada nas idéias de linha de produtos de software para promover a racionalização do esforço de desenvolvimento. Essa abordagem, chamada de Linha de Produtos de Software para Comunicação Síncrona na Web (LPSCSW), é realizada em duas etapas: Engenharia de Domínio (ED) e Engenharia de Aplicação (EA). Na ED são construídos os *assets* ou ativos da linha de produtos, e na EA são construídos os diferentes produtos com reúso desses *assets*. Os *assets* incluem componentes, arquiteturas, padrões de projeto, técnicas e ferramentas que auxiliem tanto a ED como a EA.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

Adobe AIR - *Adobe Integrated Runtime*

AJAX - *Asynchronous JavaScript and XML*

CASE - *Computer-Aided Software Engineering* / Engenharia de Software apoiada por Computador

CBO - *Coupling Between Object Classes* / Acoplamento entre Objetos

CI - Comunicador Instantâneo

C&K - Conjunto de métricas para sistemas orientados a objetos definido por Chindamber et. al.

CMS - *Content Menagement System* / Sistema de Gerenciamento de Conteúdo

DC - Departamento de Computação da UFSCar

Digae - *Distributed Gathering Environment)* / Ambiente Distribuído de Reuniões

DIT - *Depth of Inheritance Tree* / Profundidade da Árvore de Herança

EA - Engenharia de Aplicação

EaD - Educação a Distância

ED - Engenharia de Domínio

FMIS - *Flash Media Interactive Server*

FMJ - *Freedom for Media in JAVA*

GWT - *Google Web Toolkit*

IDE - *Integrated Development Environment* / Ambiente de Desenvolvimento Integrado

ICMC - Instituto de Ciências Matemáticas e de Computação

ISBN - *International Standard Book Number* / Padrão Internacional de Numeração de Livros

JMF - *JAVA Media Framework*

JMS - *JAVA Message Server* / Servidor de Mensagens JAVA

JSF - *JAVA Server Faces*

JVM - *JAVA Virtual Machine* / Máquina Virtual JAVA

LCOM - *Lack of Cohesion in Methods* / Falta de Coesão em Métodos

LMS - *Learning Management System* / Sistema de Gerência de Aprendizado

LOC - *Lines of Code* / Linhas de Código

LPS - Linha de Produtos de Software

LPSCSW - Linha de Produtos de Software para Comunicação Síncrona na Web

MSS - *Media Streaming Service)* / Serviço de Fluxo de Mídia Síncrona

MVC - *Model View Controller*

NOC - *Number of Children* / Número de Filhos

ORM - *Object Relational Mapping* / Mapeamento Objeto-Relacional

QTJava - *Quicktime for JAVA*

Reface - *Remote Face-to-Face Experience* / Experiência Remota Face a Face

REST - *Representational State Transfer* / Protocolo de Transferência de Estados

RFC - *Response For a Class* / Resposta Para Uma Classe

RFID - *Radio-Frequency Identification* / Identificação por Radio Frequência

RIA - *Rich Internet Applications* / Aplicações Web de Interface Rica

RTMP - *Real Time Messaging Protocol* / Protocolo de Transferência de Mensagens em Tempo Real

RWIC - *Rich Window Interface Components* / Componentes de Interface Rica

SADT - *Structured Analysis and Design Technique* / Técnica de Análise e Design Estruturada

SCS - *Synchronous Communication Service* / Serviço de Comunicação Síncrona

SOA - *Service Oriented Architecture* / Arquitetura Orientada a Serviços

Tidia-Ae - Tecnologia da Informação para Desenvolvimento da Internet Avançada - Aprendizado Eletrônico

UFAM - Universidade Federal do Amazonas

UFSCar - Universidade Federal de São Carlos

USP - Universidade de São Paulo

WMC - *Weighted Methods per Class* / Complexidade de Métodos por Classe

WYSIWYG - *What You See Is What You Get*

Sumário

1 Introdução

A Web 2.0 (OREILLY, 2005) é um termo que tem origem nos conceitos e tecnologias empregados por um conjunto de empresas durante "Bolha ponto-com"em 2000. Esse conjunto de empresas destacaram-se por meio de aplicações e serviços que tinham algumas características em comum. Essas características incluem interfaces ricas, uso da Web como plataforma, armazenamento de inteligência coletiva, software para multi-dispositivos, dentre outras. Geralmente uma aplicação Web 2.0 não possui todas essas características simultaneamente, mas um subconjunto delas.

Relações humanas e produção de conhecimento estão mudando rapidamente. Uma das razões dessas mudanças é a forma de produção e acesso a informação criados pela Web 2.0. A comunicação tem um importante papel nessas mudanças e é fundamental para a produção de conhecimento. Apesar das tecnologias de informação e comunicação propiciarem hoje maior facilidade para a comunicação entre pessoas, nota-se a preferência por comunicações assíncronas. Essa preferência talvez seja motivada pela comunicação assíncrona acontecer sem o engajamento simultâneo de pessoas na comunicação, ou talvez pela precariedade da qualidade das comunicações síncronas através de mídia contínua como áudio e vídeo, possibilitada pelas redes de dados de alta velocidade.

Nos dias atuais, a largura de banda para transmissão de dados justifica o acesso e produção de conteúdo de mídia rica. Sites como YouTube, Flicker e Justin.tv são alguns exemplos do crescente conteúdo de mídia rica disponível na Web. Dados levantados em 2008 apontam que 66% dos usuários brasileiros com acesso a Internet fizeram acessos a conteúdo multimídia (COMSCORE, 2009a). Apesar da largura de banda disponível hoje ser suficiente para transmissão de conteúdo mais rico, a maioria das comunicações síncronas, feitas principalmente em cliente de mensageiros instantâneos, ainda são textuais.

Comunicações síncronas podem evoluir fazendo uso mais frequente de mídias ricas. O cenário ideal é colaborar remotamente da mesma maneira como se faz face a face. Comunicações mediadas por computador oferecem possibilidades não encontradas em

comunicações face a face como armazenamento de informação, busca de conteúdo, suporte na produção de conteúdo e comunicação remota.

Aplicações de comunicações multimídia síncrona geralmente possuem características em comum como interações por áudio, vídeo e texto. Essas características podem ser vistas como unidades de interação que viabilizam uma forma de comunicação entre participantes. A representação dessas unidades de interação por serviços e componentes de interface gráfica possibilitam o reúso de software entre essas aplicações. Por exemplo, uma aplicação que necessita de comunicação síncrona por áudio e vídeo pode reutilizar os serviços e componentes gráficos que suportam essa funcionalidade.

O projeto Tidia-Ae (Tecnologia da Informação para Desenvolvimento da Internet Avançada - Aprendizado Eletrônico) (TIDIA-AE, 2009), no qual o autor deste trabalho atuou, é uma iniciativa que tem como objetivo a construção de um ambiente de aprendizado eletrônico na Web. O Ae é um portal de código aberto que conta com um conjunto de funcionalidades e ferramentas de apoio ao ensino. Parte desse projeto é dedicado à pesquisa e ao desenvolvimento de aplicações multimídia de comunicação síncrona com interfaces ricas. Várias dessas aplicações foram construídas ao longo do projeto. No entanto, o desenvolvimento de aplicações para esse domínio mostrou-se complexo.

Desenvolver aplicações Web de comunicação multimídia síncrona pode ser difícil e custoso. Comparadas com aplicações desktop, as RIAs (Aplicações Web de Interface Rica) devem satisfazer requisitos não funcionais relacionados à segurança, boa responsividade, interfaces gráficas elaboradas, integração com navegadores como suporte a histórico, e outros mais. Barreiras tecnológicas exigem esforços de equipes de desenvolvimento na solução de problemas relacionados a esses requisitos não funcionais. A heterogeneidade de navegadores, plataformas e linguagens de programação é outro problema relacionado à produtividade das equipes. Esse cenário complexo dificulta a formação de times, a previsão de custos de projetos e a manutenção de aplicações. Em vista desse cenário, o reúso de software pode ser uma boa maneira de racionalizar esforços no desenvolvimento de aplicações Web de comunicação multimídia síncrona.

Segundo Mili et al., reúso de software melhora a qualidade em geral de um sistema se componentes de qualidade são usados na construção desse sistema. Com um processo de reúso, a produtividade aumenta na mesma proporção em que sistemas de qualidade são sistematizados (MILI et al., 1995).

Conceitos de linha de produtos de software e desenvolvimento baseado em componentes podem ser uma maneira de viabilizar o reúso no domínio de aplicações Web de

comunicação multimídia síncrona. Para facilitar o desenvolvimento tanto de aplicações como de artefatos de reúso nesse domínio, descreve-se uma abordagem. Essa abordagem, chamada de Linha de Produtos de Software para Comunicação Síncrona na Web (LPSCSW), é realizada em duas grandes etapas: Engenharia de Domínio (ED) e Engenharia de Aplicação (EA). Na ED são construídos os *assets* ou artefatos da linha de produtos, e na EA são construídos produtos com reúso desses *assets*. Os *assets* desenvolvidos incluem componentes, arquiteturas, padrões de projeto, técnicas e ferramentas que auxiliem tanto a ED como a EA.

1.1 Objetivo

Este trabalho tem por objetivo incentivar o emprego da comunicação multimídia síncrona por meio de aplicações Web para facilitar o intercâmbio de informações entre participantes remotos. No entanto, notou-se que a construção de aplicações com essas características pode ser difícil devido às complexidades inerentes ao desenvolvimento para o ambiente oferecido pelos navegadores. Alguns exemplos dessas complexidades incluem a falta de compatibilidade na interpretação de código gerado e a dificuldade de construção de interfaces gráficas elaboradas com suporte à comunicação multimídia. A principal contribuição deste trabalho está em atenuar essa complexidade por meio do reúso de componentes e outros artefatos de software que possam diminuir o esforço de desenvolvimento para aplicações Web com comunicação multimídia síncrona.

O objetivo deste trabalho pode ser dividido nos seguintes itens:

1. Descrever as alternativas tecnológicas para desenvolvimento de aplicações Web de comunicação multimídia síncrona;
2. Disponibilizar uma abordagem de reúso baseada em linha de produtos de software para a construção de artefatos de reúso e para o desenvolvimento de aplicações com reúso desses artefatos;
3. Disponibilizar os componentes e artefatos construídos para reúso no desenvolvimento de aplicações Web de comunicação multimídia síncrona;
4. Disponibilizar aplicações Web de comunicação multimídia síncrona; e
5. Incentivar o uso da comunicação multimídia síncrona por meio das aplicações disponibilizadas;

1.2 Metodologia

A definição de um conjunto de tecnologias como linguagens de programação e plataformas de execução que suportem a comunicação multimídia síncrona em aplicações Web é uma decisão importante na construção dos artefatos de reúso e das aplicações. Essa decisão tem impacto no esforço de desenvolvimento, treinamento de equipes e custo de licenças. Várias alternativas tecnológicas em linguagens de programação e plataformas de execução suportam interfaces ricas e comunicação multimídia. A escolha de um conjunto de tecnologias a ser adotado deve atender a requisitos não funcionais como escalabilidade, usabilidade, compatibilidade com navegadores e facilidade de instalação.

Após a escolha do conjunto tecnológico, avalia-se as técnicas de reúso de software que podem auxiliar o desenvolvimento no domínio de aplicação de comunicação multimídia síncrona. O reúso de software tem sido realizado por técnicas de recortes de código, componentes, frameworks, linha de produtos de software e muitos outros. Um conjunto de técnicas apropriadas para o domínio deve ser identificado.

Com o apoio das técnicas de reúso selecionadas, uma abordagem de reúso é desenvolvida. Essa abordagem deve definir procedimentos para identificação e construção de artefatos de reúso. Esses artefatos são identificados por meio de comparações entre aplicações do domínio em busca de similaridades. A abordagem de reúso também deve definir como aplicações devem ser desenvolvidas fazendo reúso dos artefatos construídos.

A abordagem de reúso deve ser usada no desenvolvimento de aplicações. Essas aplicações devem ser disponibilizadas para facilitar a comunicação entre participantes remotos e para explorar atividades remotas que antes eram feitas de forma presencial.

As aplicações e artefatos resultantes da abordagem devem ser avaliados. Métricas de qualidade de software é uma forma de avaliação que permite a quantificação de atributos de qualidade. Esses atributos devem refletir o reúso feito pelos produtos, ganhos ou perdas de qualidade propiciados pelo reúso dos artefatos e a reusabilidade dos artefatos em si.

Na próxima seção descreve-se a organização do trabalho.

1.3 Organização da Dissertação

Este trabalho está organizado da seguinte maneira:

- No Capítulo 2 discute-se o conceito Web 2.0, define-se o que são aplicações Web

2.0, as principais tecnologias usadas no desenvolvimento de aplicações de interfaces ricas, as principais tecnologias usadas para multimídia na Web, particularmente relacionadas à captura de câmera e streaming de áudio e vídeo.

- No Capítulo 3 faz-se um resumo sobre o projeto Tidia-Ae que motivou este trabalho e descreve-se as principais aplicações desenvolvidas no âmbito do projeto.
- No Capítulo 4 abordam-se alguns conceitos sobre reúso de software, particularmente sobre linha de produtos de software.
- No Capítulo 5 comparam-se trabalhos relacionados a este.
- No Capítulo 6 delimita-se um domínio de atuação do trabalho e descreve-se a abordagem de reúso proposta.
- No Capítulo 7 quantificam-se atributos de qualidade de aplicações derivadas e dos *assets* construídos pela abordagem de reúso através de métricas de software.
- Por fim, no Capítulo 8 conclui-se a dissertação através de reflexões sobre o trabalho realizado e possíveis caminhos de evolução.

2 Web 2.0

A crise financeira experienciada pelas empresas de TI entre os anos de 1995 a 2000 ocasionou perdas financeiras severas e muitas falências. Entretanto, algumas empresas obtiveram perdas atenuadas no auge da crise e crescimento acentuado posteriormente. Notou-se que a crise ao invés de deteriorar o setor, influenciou novas iniciativas como *sites* e aplicações com mais inovação. O'Reilly 2005 notou que as empresas que sobreviveram à crise financeira compartilhavam algumas características e denominou o conjunto dessas características como Web 2.0. Geralmente as empresas não atendem a todas essas características simultaneamente, mas a um subconjunto delas. Conforme mostra-se na Figura 1, O'Reilly definiu 7 características que delimitam o termo Web 2.0:

Figura 1: Características da Web 2.0

- **A Web como plataforma**: O termo plataforma pode ter muitos significados no âmbito tecnológico: plataforma de execução, plataforma de desenvolvimento, plata-

forma de testes, dentre outros. Neste trabalho, entende-se que o termo plataforma está relacionado a um conjunto de ferramentas e serviços para execução de aplicações. O protocolo de comunicação é geralmente o HTTP, usa-se o navegador como ambiente de execução e na maioria das vezes as aplicações seguem a arquitetura cliente-servidor. Nota-se um aumento de aplicações tradicionalmente desktop que mudaram para a plataforma Web. Aplicações desktop como jogos, planilhas, editores de texto, agendas, clientes de email, até ERPs são exemplos dessa mudança;

- **Aglutinação de inteligência coletiva**: A característica central das empresas que sobreviveram à crise é armazenar e usar informações produzidas coletivamente. As aplicações Web 2.0 colecionam informações provenientes de seus usuários e melhoram a qualidade de seu serviço à medida que essas informações são processadas. Colecionar essas informações torna-se possível pelo uso da Web como plataforma.

- **Dado é o mais importante (*Data is the next 'Intel inside'*)**: A informação hoje é muito valorizada no âmbito das empresas de tecnologia. Adiciona-se ainda mais valor às informações se combinadas aos dados coletados dos usuários. O site *Amazon*, por exemplo, usa um catálogo de publicações ISBN (International Standard Book Number) fornecida por uma determinada empresa. No entanto, ao longo de anos de uso, os usuários do *Amazon* foram adicionando comentários, revisões e notas às publicações. Hoje o maior vendedor de catálogo de publicação não é a empresa que fornece catálogos ISBN, mas a própria *Amazon*;

- **Fim do ciclo tradicional de release de software**: Uma das principais características de aplicações Web 2.0 é que são oferecidas como serviços e não como produtos. Esses serviços estão sempre disponíveis e são modificados a partir das informações coletadas de seus usuários, fazendo da área de operações uma das mais importantes da empresa. Além disso, novas funcionalidades são testadas mensalmente, semanalmente ou até diariamente. Se os usuários não responderem às novas funcionalidades, as mesmas são retiradas e novas adicionadas. Portanto, aplicações podem receber o rótulo de *Beta* por muitos anos, aprimorando sua melhor configuração;

- **Modelo de programação leve (*lightweight*)**: O modelo de programação para aplicações Web 2.0 é diferente do tradicional. A programação para a Web 2.0 utiliza protocolos de comunicação mais simplificados como REST (Representational State Transfer) (FIELDING; TAYLOR, 2002), linguagens de programação menos complexas e dinâmicas como Ruby, metodologias de desenvolvimento ágeis e composição por serviços (*Mashups*). Esses serviços são desenvolvidos de forma fracamente acoplada,

possibilitando que um grande número de aplicações faça uso sem que o provedor do serviço precise ter qualquer conhecimento sobre quem os usa. O *Google Maps*, por exemplo, oferece serviços como um mapa com marcador que pode ser exibido em um email de um aniversário indicando o endereço da festa;

- **Software orientado a multi-dispositivos**: Outro aspecto da Web 2.0 é que não se limita a apenas um PC. Qualquer aplicação Web já satisfaz esse requisito, pois necessita de no mínimo um servidor e um cliente. No entanto, esse conceito vai um pouco além, fazendo com que aplicações não se limitem a arquitetura cliente-servidor como a arquitetura P2P, ou até mesmo rode em plataformas de hardware diferentes do PC como dispositivos móveis. *ITunes* é um bom exemplo desse conceito enviando conteúdo tanto para *IPods* quanto para PCs;

- **Interfaces ricas**: Aplicações desktop só puderam migrar para a plataforma Web pela evolução das tecnologias de apresentação nos navegadores. A evolução aconteceu gradualmente desde *applets* JAVA, javascript, flash, até culminar nas aplicações AJAX (*Asynchronous JavaScript and XML*) como as desenvolvidas pela Google como *Gmail* e *Google Maps*. As aplicações Web 2.0 contam com interfaces ricas que possibilitam ao usuário a experiência de trabalhar como se a aplicação tivesse rodando localmente, com interfaces atraentes e atualizações transparentes;

Uma das características de aplicações Web 2.0 é o emprego de interfaces ricas usando a Web como plataforma para disponibilização de serviços ao usuário. Na próxima seção descreve-se algumas das tecnologias envolvidas na construção desse tipo de aplicações.

2.1 Tecnologias de Interfaces Ricas para Web

As Aplicações Ricas para a Internet (do inglês *Rich Internet Applications - RIAs*) são aplicações Web que oferecem funcionalidade e recursos, similares àqueles presentes em aplicações *desktop* (DEITEL; DEITEL, 2008). O avanço das aplicações Web foi propiciado principalmente pela evolução dos meios de transmissão de dados para acesso à Internet e pela evolução de navegadores oferecendo suporte para renderização gráfica, captura de câmera, atualizações assíncronas de informações, suporte a *plugins*, entre outras funcionalidades.

Dentre as características de RIAs, podem-se destacar as seguintes:

- **Elementos de interface rica**: Recursos como captura e exibição de vídeos, mapas, campos dinâmicos em formulários, planilhas e editores de texto *online* são exemplos de elementos de interface rica que proporcionam maior interatividade, melhorando a experiência geral do usuário. Alguns *frameworks* disponibilizam tais elementos como componentes que podem ser reutilizados em diversas aplicações, facilitando, assim, o desenvolvimento de RIAs;

- **Processamento distribuído entre cliente e servidor**: Com navegadores mais robustos, suportando *plug-ins*, a carga de processamento pode ser compartilhada entre servidor e cliente. Desta forma, servidores podem distribuir sua capacidade de processamento para atender a mais clientes, pois estes são capazes de executar algumas tarefas localmente; e

- **Transmissão de dados assíncrona**: Nas aplicações Web tradicionais, o cliente realiza uma requisição para processamento no servidor, que retorna uma resposta para atualização completa da página Web (modelo de comunicação síncrona). Em RIAs, a comunicação entre cliente e servidor pode ocorrer de modo assíncrono - o cliente envia requisições assíncronas para efetuar atualizações parciais de página, sem ter que recarregar a página por completo (DEITEL; DEITEL, 2008). Essa técnica, chamada de AJAX, baseia-se em tecnologias existentes como XHTML, CSS, DOM, XML e JavaScript, e utiliza o objeto *XMLHttpRequest* para atualizar partes específicas de páginas Web, de modo que usuários sejam capazes de interagir com a página enquanto o servidor processa as requisições (DEITEL; DEITEL, 2008) (HOLDENER, 2008).

Existem diversos *frameworks* que facilitam o desenvolvimento de RIAs. Dentre os mais populares estão ZK (ZK, 2009), GWT (GWT, 2009), *RichFaces* (RICHFACES, 2009), *Flex* (FLEX, 2009), *JavaFX* (JAVAFX, 2009) e *Silverlight* (SILVERLIGHT, 2009). A seguir, são apresentadas mais informações sobre esses *frameworks*, considerando alguns exemplos de componentes de interface rica.

2.1.1 ZK

O *framework* ZK (ZK, 2009) oferece diversas facilidades para o desenvolvimento de aplicações RIA: conjunto de componentes de interface rica; ferramenta de desenvolvimento voltada para a o ambiente Eclipse com editor visual; integração com outras tecnologias como JSP (JSP, 2009), JSF (JSF, 2009), Spring (SPRING, 2009), Hibernate (HIBERNATE,

2009), entre outras. Além disso, ZK conta com uma comunidade de desenvolvedores, que disponibiliza seu código fonte por meio das licenças GPL e comercial.

Com relação aos componentes de interface rica, ZK dispõe de uma série de recursos, tais como janelas customizáveis, editor de texto WYSIWYG (*What You See Is What You Get)*, barra de ferramentas com menus, geração de gráficos e relatórios, *download/upload* de arquivos, mapas (integração com *Google Maps*), planilhas, entre outros. Esses componentes são utilizados como *tags* em páginas Web, o que facilita o desenvolvimento e a integração com aplicações legadas. A Figura 2, extraída de (ZKDEMO, 2009), ilustra o uso de três componentes ZK em uma mesma aplicação - planilha, gráficos e integração com *Google Maps*.

Além dos componentes de interface rica, ZK oferece suporte a *server push* ou AJAX Reverso (MARTIN, 2008) (CRANE; MCCARTHY, 2008), que consiste em enviar conteúdo ao cliente sem uma requisição explícita, diante de algum evento ou condição estabelecida no servidor. AJAX Reverso não muda o paradigma do protocolo HTTP, essa técnica baseia-se em alguns artifícios para simular o envio de dados por parte do servidor, pois o cliente ainda continua a efetuar as requisições. ZK implementa duas abordagens para *server push*:

- **Polling**: cliente faz requisições ao servidor em intervalos de tempo pré-definidos para atualizar a interface. Essa técnica é de simples implementação, no entanto pode não ser indicada para aplicações com muitos usuários. O *overhead* de processamento de requisições pode ser grande, considerando-se que grande parte dessas requisições não trará informações relevantes.

- **Comet**: cliente realiza conexões HTTP persistentes (ou longa vida), permitindo várias atualizações em uma mesma requisição longa. Essa técnica é mais indicada para aplicações com grande número de usuários. As informações serão enviadas aos usuários somente quando o servidor tiver informações a serem transmitidas.

Algumas aplicações síncronas do projeto Tidia-Ae como Chat e Comunicador Instantâneo, descritas no Capítulo 3, beneficiam-se do uso de AJAX Reverso, pois demandam intensa atualização da interface, devido a troca de mensagens entre usuários. Além disso, através do emprego de AJAX, partes específicas da página Web são atualizadas, evitando-se o recarregamento da página por completo.

Além de ZK, existem diversos *frameworks* para desenvolvimento de RIAs - a próxima seção descreve sucintamente alguns deles, com enfoque em elementos de interface rica.

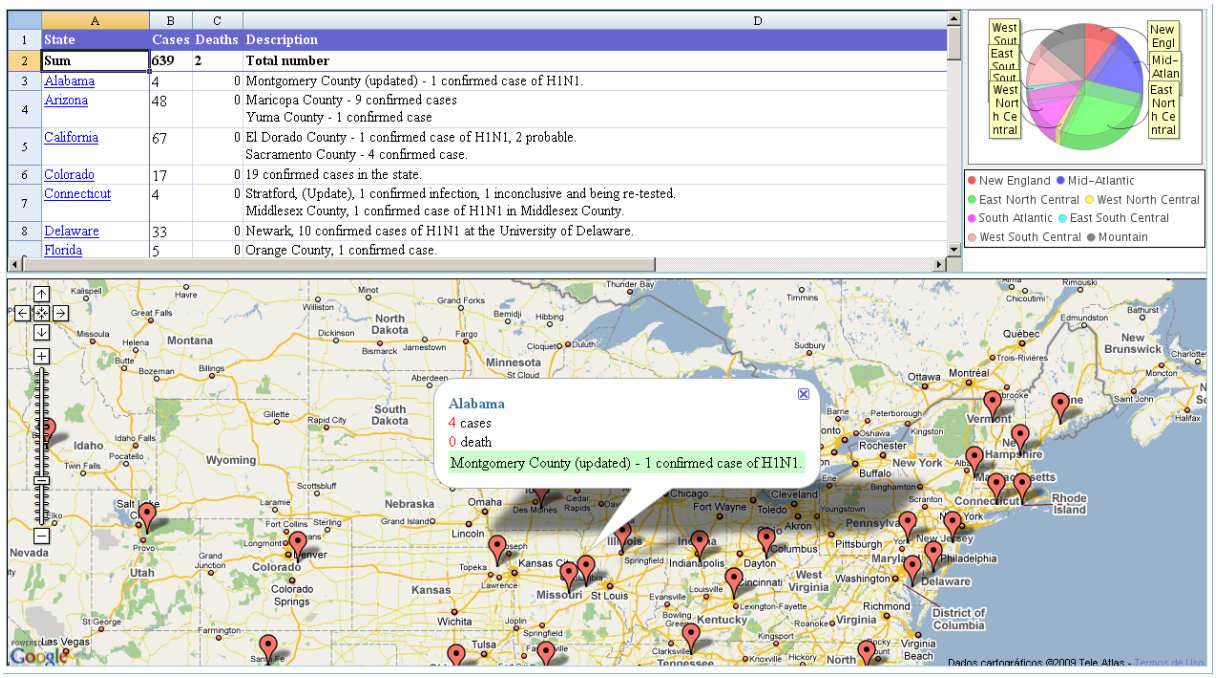


Figura 2: Componentes de interface rica do *framework* ZK (ZKDEMO, 2009).

2.1.2 Outras tecnologias - (GWT, RichFaces, Flex, JAVAFX, Silverlight)

Dentre as diversas tecnologias para desenvolvimento de interfaces ricas, é necessário avaliar qual delas é a mais adequada para os requisitos específicos da aplicação. Por exemplo, existem *frameworks* que não oferecem suporte a AJAX Reverso, portanto seriam menos indicados para o desenvolvimento de aplicações síncronas. Neste sentido, são apresentadas, a seguir, as principais características de alguns *frameworks* populares, como GWT, *RichFaces*, *Flex*,*JavaFX* e *Silverlight*.

O *Google Web Toolkit* (GWT) (GWT, 2009) permite que desenvolvedores programem o lado cliente na linguagem Java, pois esse *framework* realiza a compilação para código *javascript* otimizado, compatível com os principais navegadores. GWT é distribuído como software livre sob a licença Apache 2.0, contando com uma comunidade de contribuidores para desenvolvimento e suporte. Com relação a interface rica, GWT dispõe de componentes básicos como botões, listas, árvores, tabelas, painéis para organização dos elementos gráficos, edição de texto WYSIWYG, com destaque para a facilidade de integração com serviços Web do Google, por exemplo, *Google Maps*. Embora o conjunto de componentes não seja muito extenso, GWT permite que componentes mais complexos sejam definidos a partir dos existentes para atender os requisitos específicos das aplicações. No entanto, a versão atual (1.5) ainda não oferece suporte a AJAX Reverso, o que pode dificultar o

desenvolvimento de aplicações que demandam o recurso de *server push*.

RichFaces (RICHFACES, 2009), desenvolvido pela comunidade *JBoss* e distribuído sob a licença LGPL, disponibiliza uma biblioteca de componentes de interface rica com base na especificação JSF. Além dos componentes básicos (botões, links, menus, árvores, tabelas), *RichFaces* também possui componentes para calendário e integração com *Google Maps*. Vale ressaltar que *RichFaces* implementa AJAX Reverso com a abordagem *Polling*, facilitando a construção de aplicações síncronas que se baseiam na tecnologia JSF. Embora seja possível utilizar esses componentes em quaisquer ambientes de programação JAVA, a plataforma *JBoss Developer Studio*, que oferece recursos como editor visual para *RichFaces*, é proprietária - o que pode ser visto como uma desvantagem para o desenvolvimento de algumas aplicações.

Flex (FLEX, 2009) é uma linguagem de código aberto para a criação de RIAs, desenvolvida pela Adobe e disponível sob as licenças *Mozilla Public License* e comercial. *Flex* baseia-se em MXML, uma linguagem declarativa baseada em XML usada para descrever a interface de usuário, e *ActionScript*, usada para implementar a lógica de cliente. Sua biblioteca de componentes conta com mais de 100 componentes de interface rica, com destaque para integração com APIs de serviços Web e facilidades para criação de aplicações síncronas, considerando *Polling*, *Comet* e protocolo RTMP (*Real Time Messaging Protocol*) (ADOBE, 2009e) para implementar a técnica de *server push*. Além da diversidade de componentes, *Flex* possui integração com várias plataformas de desenvolvimento, como ColdFusion, PHP, .NET e JAVA. Outra característica interessante é a possibilidade de executar as aplicações tanto no navegador, usando o *plug-in Adobe Flash Player*, como no *desktop*, usando o *Adobe Integrated Runtime* (*Adobe AIR*). Esses ambientes de execução possuem licença proprietária, embora sejam gratuitos para uso. Assim como ocorre com *RichFaces*, o ambiente de programação *Adobe Flex Builder* também é proprietário.

A solução desenvolvida pela Sun Microsystems, *JavaFX* (JAVAFX, 2009), apóia-se na consolidada plataforma JAVA para o desenvolvimento de RIAs. *JavaFX* baseia-se na Máquina Virtual JAVA (*JAVA Virtual Machine* - JVM), explorando o conceito de aplicação multiplataforma, que pode ser executada em diversos dispositivos (navegadores, *desktop*, celulares, televisores), desde que tenham a JVM devidamente instalada. Com relação aos componentes de interface rica, *JavaFX* investe principalmente na integração com serviços Web e no uso de elementos multimídia como imagens, animações, gráficos e vídeos. Apenas alguns elementos de *JavaFX* possuem código livre, e em breve a Sun Microsystems pretende modificar sua licença para GPL 2.0. *JavaFX* ainda não oferece suporte para

AJAX Reverso - uma solução alternativa para o desenvolvimento de aplicações síncronas é o uso de *sockets* JAVA, embora haja problemas de escalabilidade no servidor em função das conexões persistentes.

A *Microsoft* também possui um *framework* para desenvolvimento de RIAs chamado *Silverlight* (SILVERLIGHT, 2009). Para a execução das aplicações, é necessário instalar o *plug-in Silverlight* nos navegadores, gratuito para uso no lado do cliente, assim como ocorre com *Adobe Flash Player*. *Silverlight* baseia-se nas tecnologias .NET e no ambiente *Visual Studio*, todos com licença proprietária, o que pode ser um obstáculo para a construção de algumas aplicações. Com relação a interface rica, os componentes contemplam tanto elementos básicos quanto recursos gráficos como animações, áudio e vídeo. Além disso, *Silverlight* implementa AJAX Reverso com base na abordagem *Polling*, facilitando o desenvolvimento de RIAs síncronas.

Um dos vários desafios na construção de aplicações multimídia síncronas usando interfaces ricas é a captura e transmissão de áudio e vídeo. Uma característica desejável é a integração das soluções de captura e transmissão ao navegador, evitando instalações complexas que dificultem a utilização das aplicações. A próxima seção aborda a captura e transmissão de áudio e vídeo na Web comparando as soluções disponíveis atualmente.

2.2 Captura e Transmissão de Áudio e Vídeo em Tempo Real na Web

O aumento da largura de banda na transmissão de dados para acesso a Internet tem intensificado a produção de conteúdo multimídia. Sites com o propósito de armazenar e disponibilizar esse tipo de conteúdo têm se tornado bastante populares. Apenas em Janeiro de 2009, 139 milhões ou 75% dos usuários de Internet norte-americanos tiveram acesso a conteúdo multimídia *online* (COMSCORE, 2009b). No Brasil nota-se também o interesse do público pelo acesso a Internet para a visualização desse tipo de conteúdo. Pesquisas realizadas em Julho de 2008 apontam que aproximadamente 66% do público brasileiro com acesso a Internet costuma acessar conteúdo multimídia *online* (COMSCORE, 2009a). A diferença entre o percentual de visualização de conteúdo multimídia entre o público norte-americano e o brasileiro talvez seja explicada pela baixa penetração do acesso a Internet através de banda larga de transmissão de dados no Brasil. No entanto, pesquisas brasileiras apontam o aumento dessas conexões de banda larga. O primeiro trimestre de 2009 apresenta um aumento de 5% no número de conexões em relação ao

último trimestre de 2008 (TELECO, 2009).

O interesse pelo conteúdo multimídia disponível hoje na Web e o aumento dos usuários de banda larga para transmissão de dados na Internet sugere um cenário favorável para colaborações síncronas multimídia. No entanto, a tecnologia para captura e transmissão de áudio e vídeo pela Web oferece reduzido número de alternativas. A limitação das alternativas deve-se principalmente à captura dos dispositivos de áudio e vídeo por aplicações Web. As soluções que se propõem a capturar os dispositivos de áudio e vídeo no cliente, com flexibilidade de sistema operacional e integradas a navegadores, são escassas.

Dentre essas soluções de captura de dispositivos de áudio e vídeo no cliente destacam-se: Flash da Adobe, JMF (JAVA Media Framework) da Sun, FMJ (Freedom for Media in JAVA) e QTJava (Quicktime for JAVA) da Apple. A Tabela 1 mostra um comparativo entre essas soluções.

Tabela 1: Soluções de Captura de Áudio e Vídeo

Soluções de Captura de Áudio e Vídeo						
Alternativa	**Linguagem**	**SW Cliente**	**Plataformas**	**Licença**	**Status**	**Custo**
JMF	JAVA	JVM + JMF	Windows, Linux e Mac OSX	SCSL	Deconti-nuado	0
FMJ	JAVA	JVM	Windows, Linux e Mac OSX	LGPL	Pouco Ativo	0
QTJava	JAVA	JVM + QT	Windows e Mac OSX	-	Pouco Ativo	0
Flash	Actionscript	Flash Player	Windows, Linux e Mac OSX	Proprie-tária	Ativo	0

JMF é uma API para manipulação de mídias em JAVA disponibilizada pela SUN. É necessária a instalação da JVM, JMF e, preferencialmente, de um pacote de desempenho (JMF Performance Pack) para usufruir-se dessa API. A instalação desse conjunto de softwares pode tornar complexa a configuração no cliente. Além disso, o JMF não é mais suportado pela Sun, a sua última atualização aconteceu em 2004.

Após a descontinuidade do JMF, o projeto de código aberto FMJ (Freedom for Media in JAVA) tenta preencher o espaço deixado. O projeto se propõe a ser compatível com softwares desenvolvidos para JMF disponibilizando interfaces idênticas. Apesar da instalação no cliente ser menos complexa quando comparada com o JMF, o projeto ainda parece imaturo e pouco ativo. No momento de escrita deste trabalho, Agosto de 2010, sua última versão era de Setembro de 2007.

QTJava é um projeto desenvolvido pela Apple para disponibilizar o acesso de códigos feitos em JAVA às bibliotecas do QuickTime (Player multimídia da Apple). A API QTJava torna possível a captura dos dispositivos de áudio e vídeo no cliente. Porém, o projeto talvez não seja prioridade para a Apple, pois o mesmo se encontra defasado. O suporte para Mac OSX é restrito à JVM 1.4.1. Outro fator limitante é que essa API funciona apenas no Windows e Mac OSX.

Outra alternativa consiste na utilização do plugin Flash para captura dos dispositivos de áudio e vídeo no cliente. A instalação do plugin Flash não representa um problema, pois está disponível em cerca de 99% dos desktops com acesso a Internet (ADOBE, 2009c). A plataforma Flash é hoje a mais utilizada para disponibilização de conteúdo em áudio e vídeo. Em janeiro de 2008 o *Youtube*, que usa a tecnologia Flash, foi responsável por 9.8 bilhões do total de 10.1 bilhões de vídeos assistidos nos Estados Unidos (COMSCORE, 2009b). Facilidade de instalação do Flash Player, presença desse software na maioria dos computadores com acesso a internet, integração com navegadores, suporte a vários sistemas operacionais e a popularidade do Flash para reprodução de áudio e vídeo fazem do Flash Player uma alternativa viável para captura de dispositivos de áudio e vídeo no cliente e para reprodução de conteúdo multimídia. Entretanto, para *streaming* multimídia *online* em Flash faz-se necessária a presença de um servidor de *streaming* de mídia.

Existem várias alternativas para servidores de *streaming* de mídia em Flash que vão de código aberto até proprietárias. Algumas dessas alternativas mais populares são Red5 (RED5, 2009) de código aberto, FMIS (*Flash Media Interactive Server*) (ADOBE, 2009b) proprietária e Wowza (SYSTEMS, 2009) proprietária. A mídia originada no cliente usa o formato FLV (ADOBE, 2009d) para codificação de áudio e vídeo e é transmitida ao servidor de mídia através do protocolo RTMP [1]. A Tabela 2 mostra um comparativo dos servidores para *streaming* de mídia *online*.

[1] O protocolo RTMP também pode ser usado com encapsulamento HTTP (RTMPT) evitando complicações relacionadas a *firewalls*.

Tabela 2: Algumas Alternativas para *Streaming* Multimídia

Alternativas para *Streaming* Multimídia						
Alternativa	**Linguagem**	**SW Cliente**	**Plataformas**	**Licença**	**Status**	**Custo**
FMIS	Actionscript	Flash Player	Windows e Linux	Proprietária	Ativo	$$
Wowza	JAVA	Flash Player	Windows, Linux e Mac OSX	Proprietária	Ativo	$
Red5	JAVA	Flash Player	Windows, Linux e Mac OSX	LGPL	Ativo	0

A Figura 3 ilustra a captura e transmissão multimídia usando o plugin Flash e servidores de mídia Flash. O cliente acessa uma página Web na qual reside uma aplicação Flash. Essa aplicação é responsável por capturar informações da câmera e microfone e da máquina cliente, e enviar o conteúdo multimídia através do protocolo RTMP para servidores de mídia Flash como FMIS ou Red5. De forma análoga, uma aplicação Flash recebe um fluxo multimídia ao vivo e o reproduz na máquina cliente.

Figura 3: Captura e transmissão multimídia em Flash.

No próximo capítulo, descreve-se o projeto Tidia-Ae e suas aplicações de comunicação multimídia síncrona, que foram desenvolvidas usando algumas das tecnologias descritas ao longo deste capítulo.

3 Projeto Tidia-Ae

Tidia-Ae (Tecnologia da Informação no Desenvolvimento da Internet Avançada - Aprendizado Eletrônico) foi um projeto que visou estimular a pesquisa na área de Tecnologia da Informação aplicada à Educação a Distância (EaD) tendo como premissa a disponibilidade de uma rede de alto desempenho (Internet Avançada). De forma mais específica, o projeto objetivou estimular a pesquisa para o desenvolvimento de um conjunto de ferramentas integradas, independentes de plataforma operacional e voltadas para EaD. O projeto contou com mais de 150 pesquisadores de 20 laboratórios das principais universidades do estado de São Paulo (FAPESP, 2008).

O Ae é um ambiente de apoio à colaboração, concebido pelo projeto Tidia-Ae, baseado no LMS (*Learning Management System*) Sakai (SAKAI, 2009). No âmbito do projeto, o autor deste trabalho esteve envolvido no desenvolvimento de aplicações Web de comunicação multimídia síncrona com interfaces ricas. Essas aplicações têm o objetivo de auxiliar a comunicação numa perspectiva educacional, porém não se limitam somente à educação. O caráter mais abrangente dessas aplicações permite que também sejam usadas para outros tipos de comunicação.

O núcleo São Carlos do projeto Tidia-Ae, composto principalmente pelos laboratórios Lince-UFSCAR e Intermedia-ICMC/USP, desenvolveu várias aplicações Web de comunicação síncrona multimídia com interfaces ricas. Essas aplicações foram desenvolvidas usando a abordagem de reúso descrita nesse trabalho, detalhada no Capítulo 6. Dentre essas aplicações, destacam-se:

- Comunicador Instantâneo: aplicação síncrona para comunicações por texto, áudio e vídeo;
- Chat: aplicação síncrona com moderação de mensagens para comunicação por texto em uma sala de conversas;
- Whiteboard: lousa eletrônica para edição comunicação síncrona por tinta eletrônica;

- Reface (Remote Face-to-Face Experience): aplicação síncrona para aula remota com apoio a monitores;
- Digae (Distributed Gathering Environment): aplicação síncrona para reuniões com suporte a configuração automática de uma sala baseado na identificação através de RFID (*Radio-Frequency IDentification*) dos participantes;

As próximas sessões descrevem essas aplicações em detalhes.

3.1 Comunicador Instantâneo

Aplicação no estilo *Instant Messenger* que possibilita a comunicação por texto, áudio e vídeo entre participantes. O Comunicador Instantâneo (CI) é integrado ao ambiente Ae propiciando um meio ágil para troca de informações entre os usuários. As mensagens textuais podem ser armazenadas e indexadas para buscas futuras. É uma aplicação Web que utiliza as tecnologias AJAX e Flash (para áudio e vídeo) acessando as informações do Ae para montar seu contexto.

Um usuário, a partir de uma lista de contatos *online* e *offline*, pode escolher outro usuário para iniciar uma conversa. Essa conversa pode ser por texto, com apoio de *emoticons*, por áudio e vídeo. Caso o usuário encontre-se *offline*, as mensagens enviadas lhe serão entregues quando o mesmo encontrar-se novamente *online* no CI. A aplicação também conta com interface gráfica de janelas flutuantes que possibilita o livre posicionamento e redimensionamento de janelas de conversa dentro da área da página.

A Figura 4 mostra um *screenshot* de uma conferência com 4 usuários no CI . A área 1 representa a lista de contatos (*online* ou *offline*), a área 2 indica o espaço para visualização de mensagens enviadas, na área 3 indica-se o vídeo dos demais usuários que participam da conversa, a área 4 da figura representa um campo para inserção de mensagens de texto e na área 5 mostra-se o vídeo do usuário que está logado na aplicação.

3.2 Chat

A aplicação Chat (MORAES et al., 2008) possibilita a troca de texto, *emoticons* e arquivos a um grupo de participantes ligados a um mesmo tópico ou sala. O Chat é integrado ao Ae e proporciona um meio para debates síncronos entre um grupo de participantes. Pode-se moderar mensagens enviadas assegurando que as mesmas são apropriadas para a

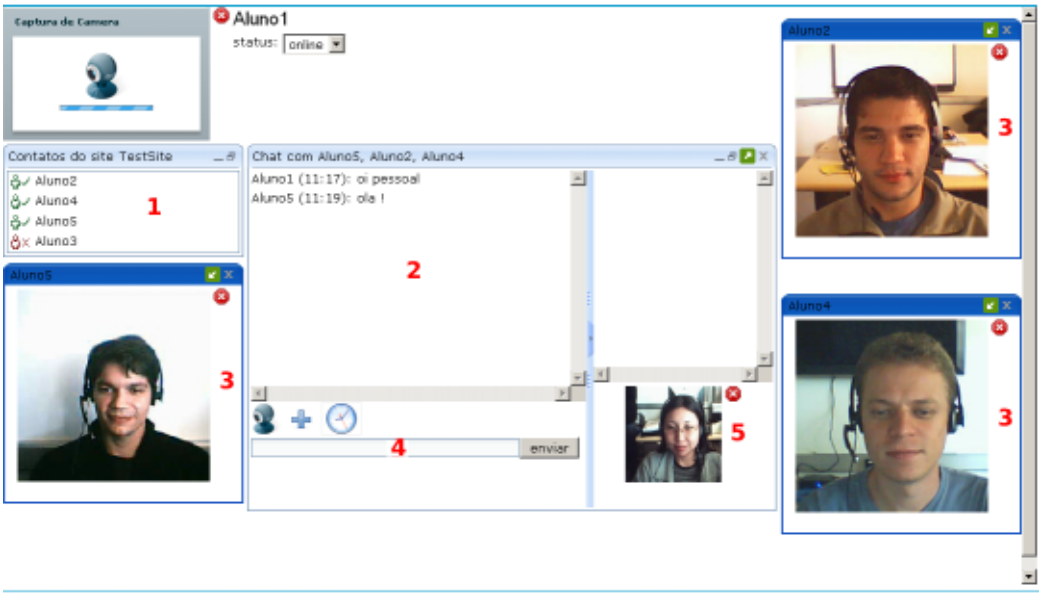


Figura 4: *Screenshot* da aplicação Comunicador Instantâneo.

sala. O Chat ainda conta com funcionalidades como ajuste de estilo da fonte e troca de mensagens reservadas.

A Figura 5 mostra um *screenshot* da aplicação. A área 1 mostra o espaço para vizualização de mensagens enviadas, a área 2 representa a lista de contatos (*online* ou *offline*), a área 3 da figura indica o campo para digitação de mensagens e a área 4 indica a barra de ferramentas.

3.3 Whiteboard

A Whiteboard (CATTELAN et al., 2003) é uma aplicação que possibilita a interação síncrona por meio de tinta eletrônica na qual participantes podem fazer anotações colaborativamente em um canvas. A aplicação fornece um conjunto de operações cujas funções permitem: uso de diferentes espessuras de caneta e cores, uso de figuras geométricas pré-definidas, uso de recursos para copiar/recortar/colar/mover objetos, desfazer e refazer ações, apagar objetos, duplicar ou criar novos slides, linhas de grade (útil para gráficos). Para acesso aos slides, pode-se navegar para o próximo, anterior ou por intermédio de miniaturas. A ferramenta permite a identificação dos participantes conectados ou não na sessão, e registra o autor de cada ação durante a captura. Após a finalização de uma sessão (de ensino-aprendizagem), pode-se disponibilizar as anotações na Web, na forma de imagens com o conteúdo capturado.

A Figura 6 mostra um *screenshot* da aplicação Whiteboard. A área 1 da Figura indica

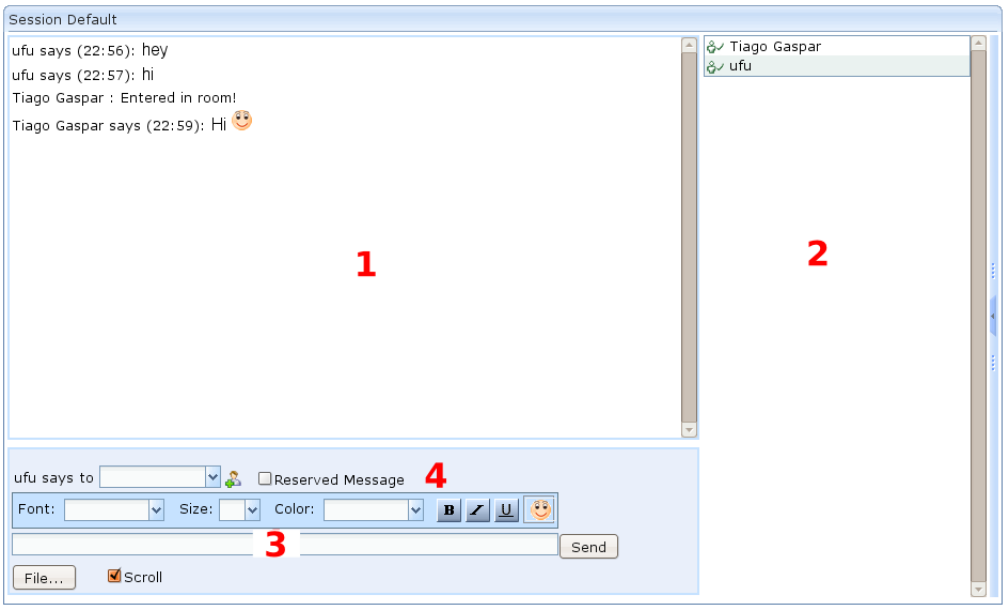


Figura 5: *Screenshot* da aplicação Chat.

o canvas onde as anotações são realizadas, esse canvas pode ter um fundo de slides, como ilustrado na Figura, ou um fundo branco. A área 2 indica a lista de contatos de usuários que encontram-se *online* ou *offline*. A partir dessa lista, o professor ou outro usuário com papel de administrador pode dar permissão de edição no canvas a qualquer dos usuários. A área 3 mostra os slides em miniatura permitindo a fácil navegação entre eles. A área 4, barra de ferramentas, concentra a maior parte das funcionalidades da aplicação como cor do traço, figuras geométricas, navegação de slides, dentre outros.

3.4 Reface

A aplicação Reface (Remote Face-to-Face Experience) proporciona recursos para que uma aula remota seja ministrada com a participação de professores, monitores e alunos. A aplicação oferece funcionalidades de forma a tentar aproximar-se a uma aula convencional traçando paralelos entre ferramentas de aula convencionais e ferramentas computacionais como quadro negro e Whiteboard; conversa com monitor e Comunicador Instantâneo com monitor; ouvir/ver o professor e áudio/vídeo do professor; e conversa com colegas e Comunicador Instantâneo com colegas. Reface incorpora algumas aplicações já desenvolvidas para prover suas funcionalidades como o Comunicador Instantâneo para conversas privadas, Chat para discussões de sala de aula e Whiteboard como lousa.

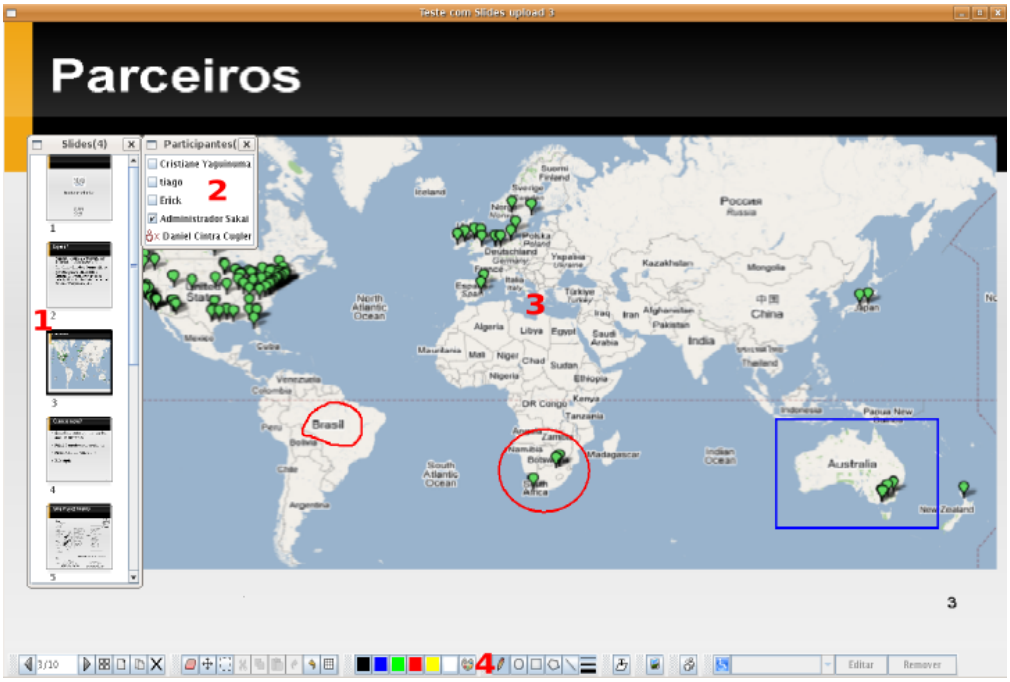


Figura 6: *Screenshot* da aplicação Whiteboard.

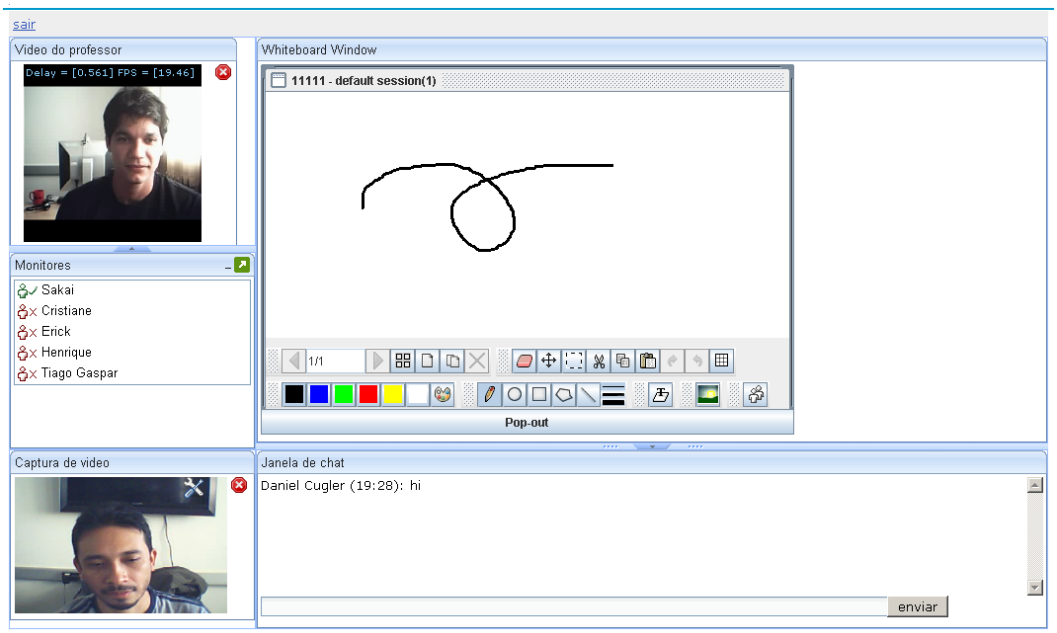


Figura 7: *Screenshot* da aplicação Reface.

3.5 Digae

O Digae é uma aplicação que viabiliza reuniões com áudio, vídeo, texto e Whiteboard para participantes distribuídos geograficamente. Na configuração de sessão da aplicação é possível escolher os participantes, assunto, duração e tipos de interação síncrona como texto, áudio/vídeo ou Whiteboard. A aplicação foi projetada para integração com um

ambiente físico de forma que o ambiente se configure automaticamente após identificação do usuário, abrindo sessões de Whiteboard em uma tela, exibindo vídeos dos outros participantes em outra tela e possibilitando interações textuais. O ambiente identifica através de RFID o participante, estabelece uma conexão com os demais participantes da reunião agendada e instancia a melhor configuração para o perfil do participante e da reunião.

3.6 Outras Aplicações

A aplicação Téte-a-Téte é uma aplicação para reuniões que procura manter o contato visual, estilo "olho-no-olho", entre os participantes de uma conversa. Teleprompters refletem a imagem da tela em um espelho, atrás do qual está uma câmera, permitindo que participantes se olhem diretamente. A aplicação prevê a comunicação entre vários participantes, fazendo-se necessário descrever a posição de cada um deles. Com base nessa descrição, é possível montar uma sala em que um monitor ocupa a posição de cada participante, exceto aquele que está presente. A reunião na sala acontece de maneira similar a uma reunião normal. Quando um participante olha, por exemplo, para sua esquerda, os outros participantes remotos notarão que ele está conversando com o participante que está relativamente ao seu lado esquerdo.

A aplicação Grava-Vídeo possibilita a gravação de notificações de áudio e/ou vídeo no Ae. Permite, por exemplo que um professor, ou um aluno, grave mensagens e as coloquem em wikis, emails, notificações e etc.

3.7 Participação do Autor

O autor deste trabalho participou durante dois anos e meio (2007 a 2009) da gerência de desenvolvimento do laboratório Lince-DC-UFSCar no Projeto Tidia-Ae sob a coordenação do Professor Cesar Teixeira. O time de desenvolvedores do projeto no Lince contou com a participação de 10 desenvolvedores que se alternaram durante o andamento do projeto. O autor também coordenou a colaboração dos desenvolvedores realizada com o laboratório Intermídia-ICMC-USP. Os componentes e serviços propostos ao longo deste trabalho foram em grande parte descobertos pelas necessidades apresentas tanto pelas aplicações desenvolvidas pelo Lince, quanto pelas aplicações desenvolvidas pelo Intermídia.

O autor participou como desenvolvedor nas aplicações Comunicador Instantâneo, Reface e Grava-Vídeo, além de atuar no desenvolvimento dos componentes e serviços propostos ao longo desse trabalho.

4 Reúso de Software

Reutilização de Software é uma prática antiga, compreendendo desde o recorte de código e bibliotecas de serviços, até os componentes de software, serviços Web, padrões e *frameworks*. Em níveis de abstrações mais altos pode-se ter o reúso da análise e projeto de software, como ocorre com o uso de padrões e modelos de *frameworks*. O reúso da modelagem, somada com a reutilização no nível de implementação resulta num grande ganho de qualidade e produtividade, além de diminuir os custos e tempo do desenvolvimento de software.

Diferentes métodos, técnicas e ferramentas têm sido pesquisadas visando tanto o "desenvolvimento para reúso", como o "desenvolvimento com reúso". No desenvolvimento para reúso o objetivo é prover recursos para a reutilização. Por exemplo, componentes de software dos diferentes domínios, como GUI (*Graphical User Interface*) e Banco de Dados, são construídos e disponibilizados para reúso por diferentes aplicações. No desenvolvimento com reúso, pratica-se a reutilização dos recursos disponíveis, como por exemplo, quando se reutiliza componentes para construção de uma interface gráfica ou para acesso a banco de dados.

Uma das formas mais praticadas de reúso acontece por meio de componentes. Clemens (1998) define um componente de software como uma unidade de composição com contratos de interface bem definidos e contextos de dependências explícitos. Um componente de software pode ser executado independentemente ou passível à composição. O autor se refere a componentes como unidades bem definidas de funcionalidade, com interfaces claras e que podem ser combinadas com componentes diferentes. Componentes de GUI são bons exemplos de componentes que podem ser combinados para compor uma interface gráfica de usuário através de botões, imagens e caixas de texto, por exemplo.

Outra forma de reúso bastante difundida é através de *frameworks*. Framework é uma aplicação reusável semi-completa que pode ser especializada para produzir aplicações customizadas (FAYAD; SCHMIDT, 1997). O framework ou arcabouço provê uma estrutura

ou esqueleto para que seja preenchido com as especificidades da aplicação. Hibernate é um exemplo de framework bastante usado que provê funcionalidades de mapeamento objeto relacional com base em POJOs (Plain Old JAVA Objects). Através das relações de herança ou associação de um conjunto de POJOs, Hibernate gera a estrutura de tabelas de banco de dados que reflete as relações desse conjunto.

Linha de Produtos de Software (LPS) é uma forma de reúso de software que vem crescendo em importância. Essa forma de reúso é voltada para um processo sistemático em que produtos são derivados usando-se artefatos de reúso combinados com partes específicas de cada aplicação. LPS difere de componentes e *frameworks* de três formas:

- Contém uma arquitetura que é compartilhada por todos os produtos da linha. *Frameworks* também provêem uma arquitetura, porém essa arquitetura não é necessariamente central, pode ser responsável por um aspecto da aplicação como persistência;
- Define um processo para reúso. Esse processo é sistematizado e abrange tanto a construção de artefatos de software para reutilização como um processo de construção de produtos com reúso desses artefatos; e
- Admite artefatos outros além de código fonte como artefatos de análise ou projeto.

LPS não é oposta a componentes ou *frameworks*, pelo contrário, incentiva essas formas de reúso e as complementa com processos sistematizados. No entanto recomenda-se usar a LPS para domínios mais restritos, em que se tem similaridades bem definidas. Caso contrário, corre-se o risco de ter uma LPS muito abrangente, e como consequência, muito custosa. No entanto, o domínio não pode ser muito restrito, pois a LPS teria perspectiva de poucos produtos e também não seria economicamente viável.

Na próxima seção, discute-se alguns conceitos de LPS.

4.1 Linha de Produtos de Software

Nos dias atuais, um número crescente de empresas de desenvolvimento de software percebem a importância do reúso sistemático através de linhas de produto de software (LPS) ou família de produtos de software. De forma análoga à produção industrial, os produtos de software derivados de uma linha de produtos compartilham uma infra-estrutura comum, um processo de produção e peças, ou componentes, para seu desenvolvimento.

Empresas como Hewlett-Packard, Motorola e Nokia tem usado com sucesso linhas de produto no desenvolvimento de software (NORTHROP, 2002).

Pelo levantamento realizado por Cohen (2002), uma definição de linha de produtos de software bem aceita é dada por Clements e Northrop (2001):

> "Uma linha de produtos de software é um conjunto de sistemas de software compartilhando um conjunto gerenciável de características que satisfazem necessidades de uma missão ou segmento de mercado específico e que são desenvolvidos a partir de um núcleo comum de artefatos (*core assets*) de uma maneira sistemática."

Essa definição introduz conceitos chaves para o entendimento de linhas de produto:

- *core assets*[1]: é um conjunto de *assets* ou artefatos prontos para serem reusados no desenvolvimento de novos produtos. Os *core assets* podem ser componentes de software, padrões de projeto, documentos utilizados no desenvolvimento, arquitetura, cronogramas e outros artefatos. Os *core assets* encontram-se presentes em todos os produtos da linha. A arquitetura é um desses *core assets* e carrega consigo as possibilidades de variabilidade da linha. O *core asset* arquitetura, em particular, é visto como ponto chave de uma linha de produtos, caso seja mal projetado pode inviabilizar todo o projeto;
- Maneira Sistemática: cada *asset* tem um processo associado que define como o mesmo poderá ser usado no desenvolvimento de novos produtos. O conjunto desses processos combinados com processos que descrevem o acoplamento dos *assets* definem um plano de produção para cada produto. O plano de produção define um procedimento de derivação para um novo produto (CLEMENTS et al., 2007); e
- Missão ou segmento particular: o segmento de atuação de uma linha de produtos deve ser delimitado e bem compreendido. As estratégias de negócio são traçadas dentro desse segmento de mercado. Se o escopo do segmento for muito grande também será o seu custo. É complexo gerenciar uma linha de produtos muito genérica. Além da base de *assets* ser grande, a complexidade de cada *asset* também será, pois deverão suportar muitas variabilidades;

A Figura 8 ilustra o conceito de *assets* e *core assets*. Os *assets* não são artefatos obrigatórios em todos os produtos. Por exemplo, o produto P3 tem artefatos em comum

[1]Neste trabalho serão usados os termo em inglês *Core Asset* e *Asset* para evitar ambiguidades.

com o produto P1, no entanto não são os mesmos artefatos que P3 tem em comum com P2. *Core assets* são artefatos que estão presentes em todos os produtos da linha, como o losango central mostrado na figura.

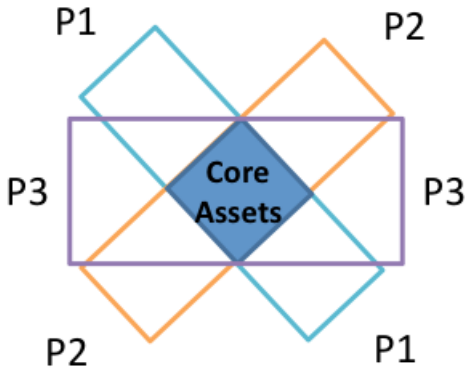


Figura 8: *Core assets* de uma Linha de Produtos de Software

A Principal diferença entre o desenvolvimento de sistemas convencionais individuais e a abordagem de linhas de produto é o enfoque. Nos sistemas convencionais enfoca-se um único sistema por vez, ao passo que o desenvolvimento de linhas de produto visa um conjunto de sistemas. A mudança de enfoque é fundamentalmente de estratégia de negócios. Enquanto o desenvolvimento de sistemas convencionais funciona de maneira *ad-hoc*, ou seja, orientada a contratos, o desenvolvimento de linhas de produto tem uma visão estratégica do nicho de mercado (LINDEN et al., 2007). Tem uma postura pró-ativa de desenvolvimento, buscando com isso diminuir o tempo de entrega de produtos para o mercado (*time-to-market*).

Ao contrário da abordagem tradicional, que em termos de reúso se preocupa principalmente com o código, o reúso da linha de produtos envolve todos os *assets* relevantes do ciclo de vida de desenvolvimento. *Assets* que variam de requisitos, arquitetura, até testes. Esses *core assets* podem ser reusados através da linha de produtos pois refletem em si requisitos de variabilidades.

Discute-se a seguir alguns pontos relevantes de LPS no âmbito da Engenharia de Software.

4.1.1 Engenharia de Requisitos

A Engenharia de Requisitos define uma maneira sistemática e repetível de técnicas para obtenção de completude, consistência e relevância dos requisitos de sistema (SUMMERVILLE, 2006). Segundo Jay e Mayer (JAY; MAYER, 1990), requisitos descrevem o que

o sistema deve fazer, como deve se comportar, propriedades que deve conter, qualidades que deve possuir, e as restrições que o sistema e o seu desenvolvimento devem satisfazer. Requisitos são definidos como:

1. Uma condição ou propriedade necessária ao usuário para resolver um problema ou atingir um objetivo;

2. Uma condição ou propriedade que o sistema ou um componente do sistema devem possuir para satisfazer um contrato, um padrão, especificação ou outro documento formal; e

3. Uma representação documentada de uma condição ou característica como dos itens 1 ou 2.

O entendimento dos requisitos é fundamental para construção de softwares. Porém, uma definição técnica clara e precisa do que o software deve atender é uma tarefa complicada. O uso de linguagens naturais, como português, para especificação traz ambiguidades à definição dos requisitos. Outro fator complicador é a extração dos requisitos a partir dos especialistas de domínio, a comunicação envolvida é complicada e a visão dos especialistas pode se modificar durante o ciclo de desenvolvimento do software.

No âmbito de LPS, requisitos são *assets* tangíveis, ou seja, são artefatos reusáveis que definem atributos e restrições de produtos. Os requisitos estão alinhados com a definição de escopo da linha de produtos e evoluem de forma conjunta (CLEMENTS; NORTHROP, 2001). Requisitos comuns entre os produtos da linha podem ser definidos com a inclusão de pontos de variação. Esses pontos de variação ao serem preenchidos formam a especificação de um produto da linha. Os pontos de variação podem ser simples, como por exemplo, trocando-se um valor numérico: "numero_max_de_usuários = 50", ou mais complexos, deixando-se sessões inteiras para preenchimento de acordo com as especificidades de cada produto.

4.1.2 Definição de Arquitetura

Bass et al. (BASS et al., 2003) define uma arquitetura de software como uma estrutura ou conjunto de estruturas de um programa ou sistema computacional que compreende elementos de software, as propriedades externamente visíveis desses elementos e a relação entre os elementos.

As propriedades externamente visíveis e a relação entre os elementos são decisões de mais alto nível e afetam o sistema como um todo. Decisões que não tenham ramificações em todo sistema não são arquiteturais.

A arquitetura é um dos *core assets* mais importantes de uma LPS, talvez o mais importante. Geralmente começa-se a modelar os requisitos a partir do artefato de arquitetura. Atributos de qualidade de um sistema como performance, variabilidade e disponibilidade são influenciados em grande parte pelas decisões arquiteturais. Requisitos não funcionais como os citados devem ter reflexo na definição de uma arquitetura. Sem esse apoio arquitetural, a tarefa de realização desses requisitos não funcionais torna-se mais complicada (CLEMENTS et al., 2007).

Toda arquitetura é por definição uma visão de nível mais alto do sistema, portanto admite um certo grau de generalidade. No entanto, para LPSs a arquitetura deve ser bem ponderada para atender a um conjunto de possíveis produtos que compartilhem a mesma arquitetura. As possibilidades de variação de produtos de uma LPS devem estar contidas nas decisões de arquitetura.

4.1.3 Desenvolvimento de Componentes

Segundo Clements et al. (2007), numa linha de produtos de software, uma das tarefas de um engenheiro de software é produzir um conjunto de componentes que comporão a arquitetura. Os componentes se encaixam na arquitetura e provêem funcionalidades aos produtos. Os componentes desenvolvidos podem tanto ser incorporados ao repositório de *assets* da linha possibilitando a construção de produtos ou os componentes podem ser empregados por um produto específico. Os componentes pertencentes ao repositório de *assets* devem ser flexíveis o suficiente para satisfazer os pontos de variação especificados na arquitetura ou nos requisitos da linha de produtos. A funcionalidade a ser atendida é definida no contexto da arquitetura da linha.

Os componentes de uma linha de produtos devem ter mecanismos de variação determinados pela abordagem de variabilidade da linha. Essas variações devem ser escolhidas adequadamente para suportar os requisitos de estratégia e produção. Os componentes que integram o repositório de *assets* devem ter processos de uso bem definidos. Esses processos guiam o uso dos componentes na composição de produtos da linha. O desenvolvimento desses componentes e seus artefatos (especificação de interfaces, descrição de mecanismos de variabilidade, testes, dentre outros) constituem grande parte do repositório de *assets* de uma linha de produtos. Consequentemente, a atividade de desenvolvimento

desses componentes é grande parte do esforço operacional de construção de uma linha de produtos de software.

4.1.4 Variabilidade

De acordo com Linden et al. (2007), a engenharia de linhas de produtos de software tem como objetivo suportar uma gama de produtos. Esses produtos podem ser direcionados a clientes específicos ou a segmentos de mercados diferentes. A variabilidade é uma abordagem fundamental para atingir tais objetivos.

Assets podem ser usados por vários produtos sem a necessidade de adaptação. Porém, em muitos casos, adaptações são necessárias para que os *assets* atendam a um domínio mais amplo. Mecanismos de variabilidade ajudam a controlar as adaptações necessárias e a auxiliar os desenvolvedores de produtos na tarefa.

Um aspecto importante no esforço de uma linha de produtos diz respeito ao momento de identificação dos pontos de variação. Os pontos de variação podem ser identificados durante a definição da arquitetura da linha ou podem ser descobertos durante a construção de produtos. Herança é um mecanismo que permite a criação de pontos de variação sem que se tenha conhecimento de quem o usará. Esse mecanismo de variabilidade é geralmente definido na fase de concepção da linha. Instanciação por *templates* pode permitir a descoberta de novos parâmetros de variação depois de sua criação. Neste caso, requisitos de variabilidade são descobertos durante a construção de produtos.

Anastasopoulos e Gacek (2001) discutem implementações de mecanismos de variabilidade. A Tabela 3 lista alguns mecanismos de variabilidade definidos por esse autor:

Tabela 3: Mecanismos de Variabilidade (ANASTASOPOULOS; GACEK, 2001)

Mecanismos de Variabilidade	
Mecanismo	**Descrição**
Agregação	Técnica orientada a objetos na qual a funcionalidade de um objeto é estendida ao delegar o trabalho a outro.
Herança	Um objeto herda um conjunto de atributos e funcionalidades de outro objeto, podendo também especializar as funcionalidades e atributos herdados.

Continuação na próxima página

Tabela 3 – Continuação da página anterior

Tipos de Mecanismos de Variabilidade	
Parametrização	Um conjunto de parâmetros define o comportamento do subsistema.
Sobrecarga	Reusa o nome de uma funcionalidade para suportar diferentes tipos de dados. A sobrecarga possibilita o reúso de código ao custo de legibilidade e complexidade.
Compilação Condicional	Múltiplas implementações de um modulo coexistem em um mesmo arquivo. Uma é escolhida em tempo de compilação de acordo com o parâmetros passados ao pré-processador.
Padrões de Projeto	Soluções conhecidas e catalogadas para problemas recorrentes. Alguns padrões como *Adapter* (GAMMA et al., 1995) são boas abordagens para flexibilização de código.

4.1.5 Considerações

Linha de Produtos de Software complementa as abordagens de reúso de componentes e *frameworks* oferecendo um método de derivação de produtos e uma arquitetura única projetada para o reúso. Também proporciona catalogação e gerenciamento dos *assets*, otimizando a capacidade de reúso.

Porém, o custo de uma LPS é geralmente elevado. Desenvolver uma arquitetura e demais *core assets* pode requerer grandes investimentos. Essa abordagem não é recomendada para qualquer tipo de software. No entanto, o ganho é proporcional ao número de produtos gerados pela linha. Se no escopo definido existe perspectiva para vários produtos derivados da linha, a sua adoção pode ter um grande custo-benefício.

No próximo capítulo discute-se alguns trabalhos relacionados a reúso de software para o domínio de aplicações Web com interfaces ricas.

5 Trabalhos Correlatos

Poucos trabalhos foram encontrados sobre reúso em aplicações Web de interface rica com comunicação multimídia. A maioria dos trabalhos na área de comunicação concentram-se em discussões e modelos como em (SHOTSBERGER, 2000), (SINGH, 1999) e (BORGHOFF; SCHLICHTER, 2000); aplicações como em (MURPHY; LAFERRIERE, 2007) e (OTTER; EMMITT, 2007); e e-learning como em (HRASTINSKI, 2008) e (CHEN et al., 2004). Na área de linha de produtos de software, grande parte dos esforços tem sido voltado para práticas como em (CLEMENTS et al., 2007); metodologias como em (KANG et al., 2002), (WEISS; LAI, 1999) e (AMERICA et al., 2000); e aplicações tecnológicas como em (LEE et al., 2006) e (VOELTER; GROHER, 2007).

Wang (2008) propõe um ambiente para colaboração na Web (*PowerMeeting*) baseado no framework AJAX GWT (Google Web Tool Kit). *PowerMeeting* emprega uma arquitetura transacional e provê serviços como gerenciamentos de usuários, sessão, grupos, transações e persistência. O ambiente conta com aplicações de agenda, apresentação de slides, *brainstorming*, planejamento de tarefas e chat. *Powermeeting* possui serviços similares aos propostos neste trabalho, porém o ambiente proposto por Wang não tem preocupações de reúso. Outra diferença é o conjunto de aplicações adotado, *Powermeeting* é voltado para colaboração enquanto este trabalho propõe uma abordagem de comunicação multimídia.

Ng (2007) realiza uma comparação entre o aprendizado realizado sincronamente a distância em contraste com o aprendizado face à face em salas de aulas convencionais. O trabalho usa um sistema para aprendizado síncrono remoto chamado *Interwise* que possibilita a transmissão de conteúdo através de áudio, vídeo, texto e whiteboard. Permite que os participantes (alunos, professores e tutores) conversem entre si com suporte multimídia. Porém, este trabalho é essencialmente na área de e-learning e não aborda aspectos de reúso.

Balzerani et al. (2005) propõem uma arquitetura (*Koriandol*) para aplicações na Web.

Koriandol define uma aplicação Web como um conjunto de páginas, arranjadas hierarquicamente, e componentes. A arquitetura é composta por: um módulo de execução (renderiza páginas dinâmicas), *engine* para tradução de modelos XSLT (XSLT para HTML); componentes; e controladores de acesso, de configuração e de componentes. A arquitetura, especificamente o conceito de páginas, não se adapta perfeitamente a interfaces ricas.

Altintas et al. (2005) propõem uma LPS multi-camada para desenvolvimento de interfaces ricas na Web voltada para a área financeira (*Aurora*). *Aurora* define um *core asset* composto de: um CMS (Content Menagement System) responsável por comunicação, autenticação e sessão; uma linguagem (EBML) para especificação de interfaces ricas; suporte a Web services; dentre outros. O trabalho proposto por Altintas et al. apresenta estudos de caso em que a linha propiciou significativa melhora nos produtos desenvolvidos. Apesar de abordar interfaces ricas, *Aurora* não tem no seu escopo comunicação ou multimídia.

Observa-se na literatura trabalhos teóricos sobre metodologias de reúso, porém foram poucos os registros sobre o uso dessas metodologias em trabalhos de carácter prático, com desenvolvimento efetivo de software. Os poucos registros encontrados não oferecem análise suficientemente detalhada dos resultados atingidos em termos de reúso.

No próximo capítulo, apresenta-se uma abordagem de reúso que utiliza conceitos de LPS de forma mais pragmática, baseada em dois anos e meio de desenvolvimento de software no projeto Tidia-Ae. No Capítulo 7, os resultados obtidos por este trabalho são quantificados por meio de métricas de software, permitindo uma avaliação segura dos benefícios propiciados por esta abordagem de reúso.

6 *Linha de Produtos de Software para Comunicação Síncrona na Web (LPSCSW)*

O domínio de aplicações Web de comunicação multimídia síncrona pode ser bastante abrangente para ser abordado por este trabalho. Comunicação síncrona pode acontecer de diversas formas, por variados meios, inclusive presencialmente. Multimídia também é um termo que pode ser amplo, significando diferentes meios para transmissão de conteúdo. Este trabalho limita-se ao domínio das aplicações Web de comunicação síncronas através de texto, áudio, vídeo e tinta eletrônica. Pretende-se apresentar uma maneira de sistematizar o reúso com um conjunto de artefatos que podem ser ampliados de forma a possibilitar um maior alcance da abordagem de reúso.

Uma das tarefas mais difíceis em qualquer abordagem de reúso é a descoberta do que deve ser reusado ou dos requisitos de domínio. Nesta proposta, os requisitos de domínio foram descobertos de maneira incremental, desenvolvendo um conjunto de aplicações para o domínio Web de comunicação síncrona e identificando similaridades entre as aplicações desenvolvidas, conforme descrito na Seção 6.2.1.

Realizando esse procedimento para as aplicações construídas no projeto Tidia-Ae, foi possível perceber que todas as aplicações desenvolvidas apresentam dois requisitos funcionais em comum:

1. Meios para mandar e receber mensagens síncronas, ou serviço de comunicação; e
2. Meios para obter informações sobre os participantes da comunicação, ou serviço de sessão

Como trata-se de aplicações Web, esses requisitos funcionais demandam requisitos não funcionais como:

1. Interfaces ricas para viabilizar o recebimento de mensagens sem a necessidade de recarregar a página[1]; e

2. Uma arquitetura para facilitar a conexão com os serviços citados e demais artefatos. Toda aplicação necessitará de acesso aos serviços de comunicação e sessão, portanto faz sentido, em termos de reúso, ter uma arquitetura bem definida para facilitar esse acesso.

Dentre as formas de reúso pesquisadas no Capítulo 4, a LPS pode ser uma boa alternativa para viabilizar o reúso dentro do domínio deste trabalho. Essa forma de reúso permite ter um conjunto de *core assets*, que está presente em todos os produtos da linha, e permite um reúso mais abrangente de artefatos de Análise ou Projeto. As aplicações pertencentes ao domínio deste trabalho também podem possuir um conjunto de características em comum, como os requisitos funcionais e não funcionais citados. Essas características podem ser transformadas em *core assets* para viabilizar o reúso por todas as aplicações do domínio.

6.1 LPSCSW

Motivados pelas dificuldades encontradas no desenvolvimento para o domínio de aplicações Web de comunicação multimídia síncrona com interfaces ricas e pelas características em comum que essas aplicações apresentam, uma abordagem de reúso baseada em linha de produtos de software é descrita com intuito de diminuir esforços e aumentar a qualidade dessas aplicações.

Linha de Produtos de Software para Comunicação Síncrona na Web (LPSCSW) (GASPAR et al., 2009) (GASPAR et al., 2009) é uma abordagem de reúso baseada em LPS construída para o domínio descrito neste trabalho. LPSCSW possui um conjunto de *core assets*, incluindo uma arquitetura única, e *assets* para possibilitar o reúso de software através da construção de produtos. O desenvolvimento da LPSCSW suporta, além dos *core assets* e *assets* mencionados, um conjunto de técnicas, ferramentas e um processo de construção tanto de *assets* como de produtos.

Conforme mostra a Figura 9, a abordagem de reúso LPSCSW é dividida em duas etapas: Engenharia de Domínio (ED), onde se constrói os *assets*, e Engenharia de Aplicação (EA), onde se constrói produtos ou aplicações fazendo reúso dos *assets*.

[1]Recarregar a página periodicamente em busca de novas mensagens pode ser oneroso para o servidor, além de prejudicar a usabilidade da aplicação.

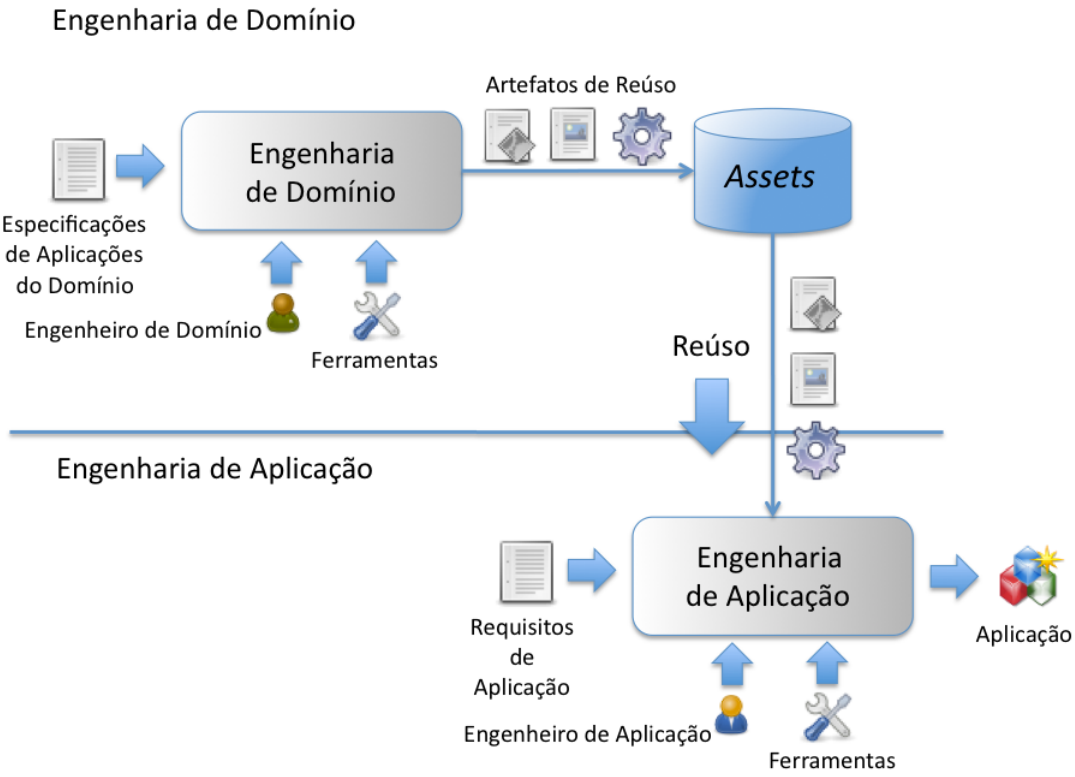


Figura 9: Abordagem LPSCSW.

A Engenharia de Domínio identifica aspectos comuns de um domínio para que sejam reutilizadas por aplicações desse domínio. O enfoque é centrado nas similaridades apresentadas por um conjunto de aplicações. A ED tem como objetivo transformar essas similaridades em *assets* tangíveis como componentes, *frameworks*, padrões de projeto, especificações de análise, dentre outros. O desenvolvimento desses *assets* acontece através de prototipação evolutiva com iterações de curta duração apoiadas pelo modelo em espiral proposto por Boehm (1986). Os *assets* provenientes desta etapa são armazenados no repositório de *assets* para que se sejam reusados posteriormente.

Diferentemente da Engenharia de Domínio, que visa similaridades em um conjunto de aplicações, a Engenharia de Aplicação tem como objetivo desenvolver aplicações reusando os *assets* provenientes da Engenharia de Domínio. Os produtos ou aplicações instanciadas fazem uso dos *assets* para compor parte de sua funcionalidade. Quanto maior o reúso de *assets* alcançados pelos produtos, mais eficiente é a LPS. A EA, assim como a ED, usa o modelo em espiral com iterações de curta duração.

Por mais completo que seja o repositório de *assets*, toda aplicação terá sua especificidade e necessitará de código customizado. É importante ter em mente, que embora seja mais custoso, deve se considerar, no desenvolvimento de novas funcionalidades ou na

modificações de existentes, a possibilidade de incorporá-las ao repositório de *assets* para que sejam reusadas posteriormente. Esse tipo de atitude aumenta o grau de reúso da LPS a medida que esta vai sendo utilizada, propiciando retorno a médio e longo prazo.

6.2 Engenharia de Domínio

A idéia de duas atividades separadas para construção de artefatos de software para reúso e desenvolvimento de aplicações não é nova. O termo Engenharia de Domínio apresenta sinônimos na literatura como Fábrica de Experiências ou Capital de Reúso (MILI et al., 1995). No entanto, o conceito é o mesmo, centrado no desenvolvimento de artefatos de software, ou *assets*, para reúso posterior.

A definição de um processo para a construção de *assets*, como em toda atividade de desenvolvimento de software, possibilita a sistematização, documentação e previsibilidade de resultados. O processo de Engenharia de Domínio proposto neste trabalho, é dividido nas seguintes disciplinas: Requisitos de Asset, Análise de Asset, Projeto de Asset, Implementação de Asset e Testes de Asset. Figura 10 mostra o diagrama SADT (*Structured Analysis and Design Technique*) (MARCA; MCGOWAN, 1987) desse processo. As setas que entram no lado esquerdo de cada disciplina representam a entrada de dados, as setas ao lado direito representam a saída de dados. As setas no topo representam os controles que influenciam internamente cada disciplina. No fundo de cada disciplina estão os participantes e ferramentas usadas.

O tambor com rótulo Assets na Figura 10 representa um repositório de artefatos para reúso ou um repositório de *assets*. Os *assets* contidos no repositório podem ser *assets* de análise (modelos de casos de uso), *assets* de projeto (modelos UML, padrões de projeto, ou arquiteturas) e *assets* de implementação (componentes ou *frameworks*). A navegação das disciplinas para o repositório de *assets* significa a produção de novos *assets* e seus armazenamentos no repositório. A navegação do repositório de *assets* para as disciplinas também acontece e representa o reúso interno feito na Engenharia de Domínio. É provável que um novo componente reuse *assets* existentes no repositório, como outro componente, por exemplo.

O processo de desenvolvimento realizado na Engenharia de Domínio segue a prototipação evolutiva, através do modelo em espiral, conforme pode ser visto na Figura 11. A cada iteração, ou volta na espiral, acontecem as disciplinas de Requisitos de Asset, Análise de Asset, Projeto de Asset, Implementação de Asset e Testes de Asset. Dependendo

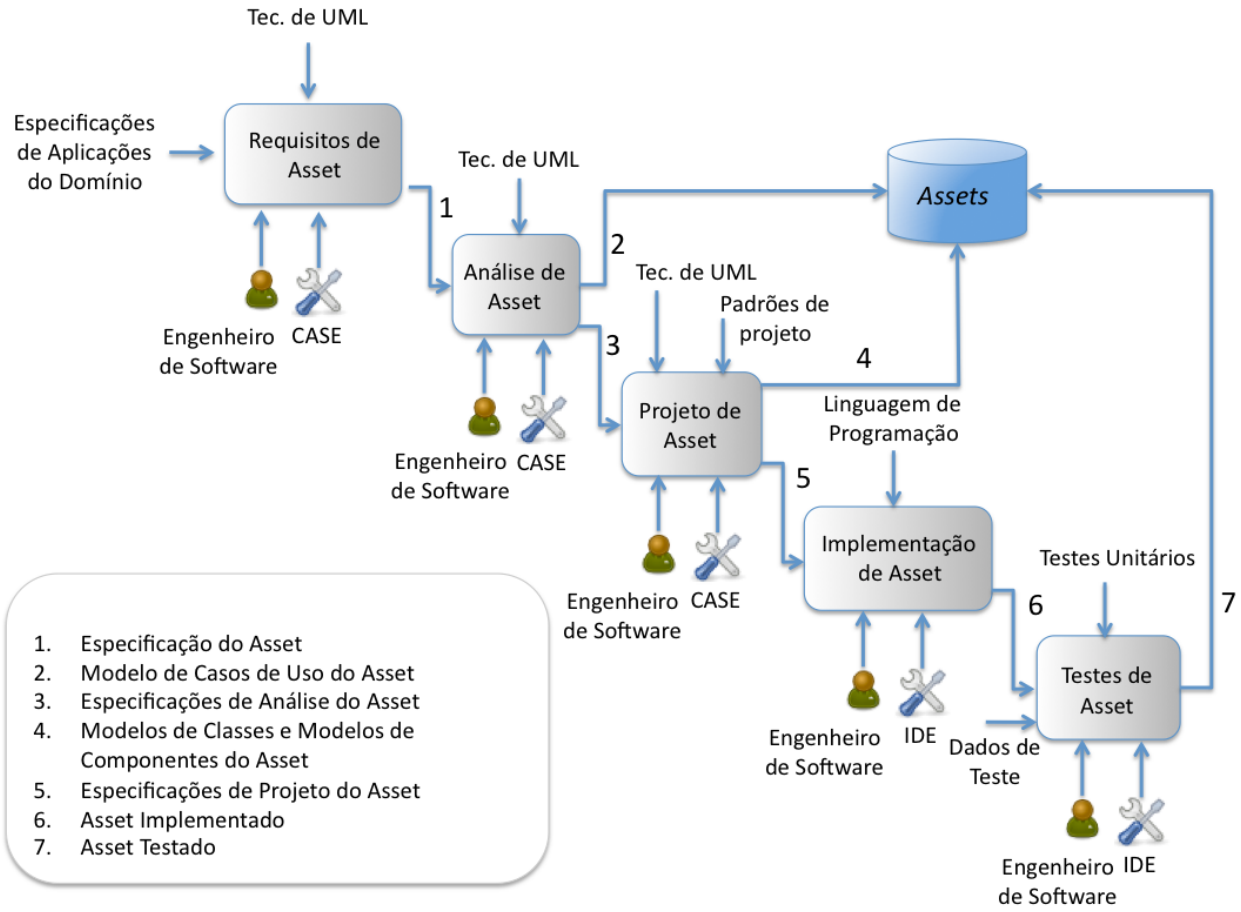


Figura 10: Disciplinas realizadas na Engenharia de Domínio.

da iteração, as disciplinas têm menor ou maior intensidade. No começo do processo, as disciplinas de Requisitos de Asset e Análise de Asset são bastante enfatizadas, gradualmente, à medida que o número de iterações avança, o enfoque passa para Implementação de Asset e posteriormente para Testes de Asset. As iterações são de curta duração e ao fim de cada uma tem-se pequenos entregáveis.

A seguir, detalha-se as atividades realizadas em cada disciplina da Engenharia de Domíno:

6.2.1 Requisitos de Asset

A disciplina de Requisitos de Asset tem por objetivo identificar as similaridades que as aplicações do domínio apresentam. Uma maneira de identificar essas similaridades é através da comparação de aplicações do domínio. As aplicações não precisam estar construídas para a identificação das similaridades, mas precisam estar especificadas na forma de documentos de requisitos e modelos de casos de uso. Algumas similaridades

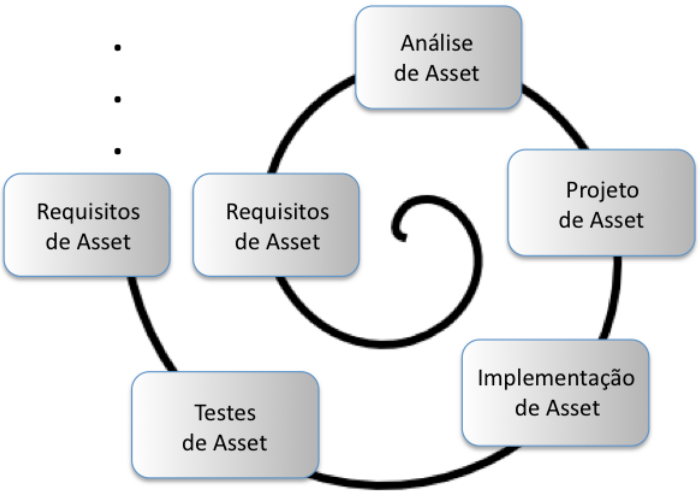


Figura 11: Modelo em Espiral praticado na LPSCSW.

podem não ser de interesse da Engenharia de Domínio por não representar perspectiva de reúso para um número significativo de aplicações do domínio. A decisão de construir ou não a similaridade, ou o *asset*, não é uma tarefa simples e requer uma boa visão do domínio. A Figura 12 mostra um fluxograma que ilustra o procedimento realizado para descoberta dos requisitos de domínio neste trabalho. Explica-se, a seguir, os processos internos do fluxograma:

1. **Comparar Aplicações**: Toma-se uma aplicação a ser desenvolvida como referência e compara-se os documentos de requisitos e modelos de casos de uso com aplicações existentes em busca de semelhanças[2]. Os documentos de requisitos fornecem informações sobre os requisitos não funcionais e uma descrição dos requisitos funcionais de cada aplicação. Os modelos de casos de uso fornecem detalhes sobre os requisitos funcionais de cada aplicação. Por exemplo, a primeira aplicação que o Laboratório Lince desenvolveu no projeto Tidia-Ae foi o Comunicador Instantâneo que possui funcionalidades de texto e áudio/vídeo síncronos. Como o laboratório Intemídia-ICMC-USP, parceiro do Lince no projeto, necessitava desenvolver uma aplicação de chat (Chat), as duas aplicações foram comparadas em busca de semelhanças. Os documentos de requisitos e modelos de casos de uso das duas aplicações foram comparados e identificou-se que as duas têm em comum funcionalidades de comunicação síncrona por texto;

2. **Avaliar Viabilidade**: depois das similaridades identificadas, é necessário fazer um esboço dos artefatos a serem desenvolvidos para avaliação da viabilidade. Esse

[2]A comparação também ser realizada em um conjunto de aplicações a serem desenvolvidas.

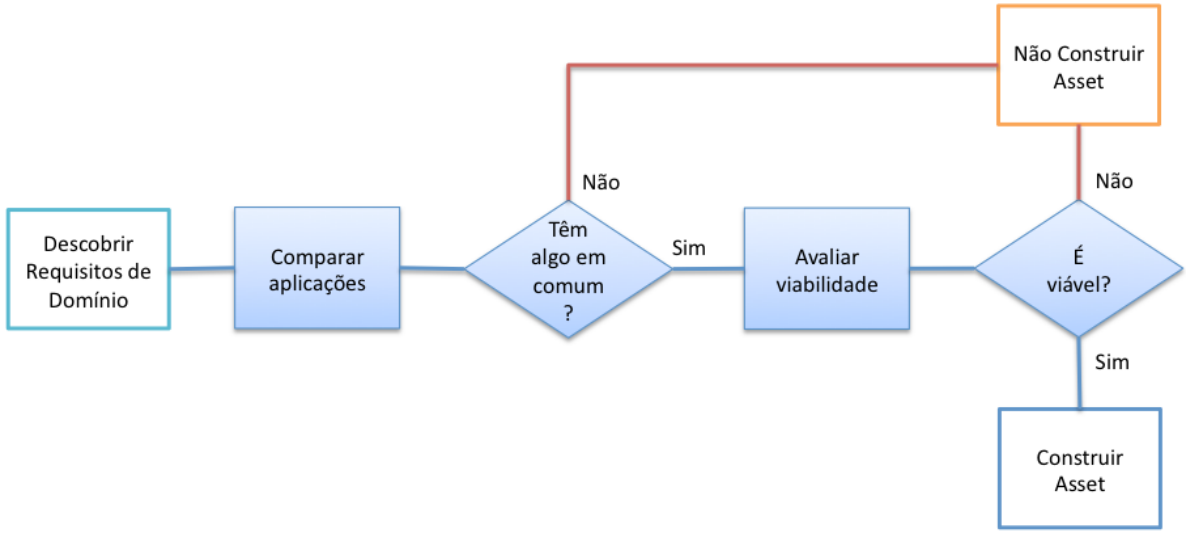


Figura 12: Descoberta de similaridades e decisão de construção de *assets*.

esboço é informal e pode ser uma descrição textual ou um diagrama de classes simplificado. O artefato pode ser um componente de GUI, um padrão de projeto, um serviço, dentre outros. Com base no esboço e na experiência de desenvolvimento de aplicações no domínio, é possível fazer uma estimativa preliminar de esforço. Levando-se o esforço estimado em consideração, deve-se analisar quantas aplicações se beneficiarão do artefato e decidir se desenvolver o artefato é viável. Usando novamente o projeto Tidia-Ae como exemplo, depois da identificação de similaridades entre as aplicações Comunicador Instantâneo e Chat, dois artefatos foram esboçados: um serviço de comunicação para troca de mensagens síncronas e um componente GUI para comunicação textual. O desenvolvimento desses artefatos demandaria um esforço considerável. No entanto, mais duas aplicações que necessitariam das funcionalidade de comunicação textual estavam previstas para desenvolvimento (Reface e Digae). Portanto, apesar do desenvolvimento dos artefatos não ser trivial, ele seria reusado por várias aplicações. Nesse caso, decidiu-se pela construção.

A Figura 13 mostra o diagrama simplificado de casos de uso da aplicação Comunicador Instantâneo e a Figura 14 mostra o diagrama de casos de uso simplificado da aplicação Chat. Por meio da comparação dos documentos de requisitos e dos modelos de casos de uso de ambas as aplicações, notou-se que CI-1 e CH-1, CI-2 e CH-2, CI-3 e CH-3, e CI-4 e CH4 apresentam similaridades. CI-5 e CI-6 são outras similaridades do domínio que foram descobertas posteriormente, comparando o Comunicador Instantâneo com outras aplicações que também demandariam comunicação por áudio e vídeo como o Reface e Digae.

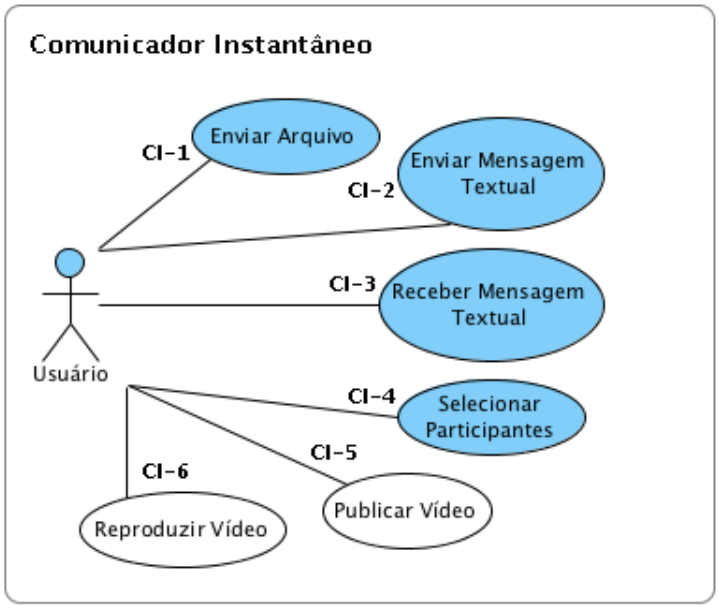


Figura 13: Parte do diagrama de casos de uso da aplicação Comunicador Instantâneo.

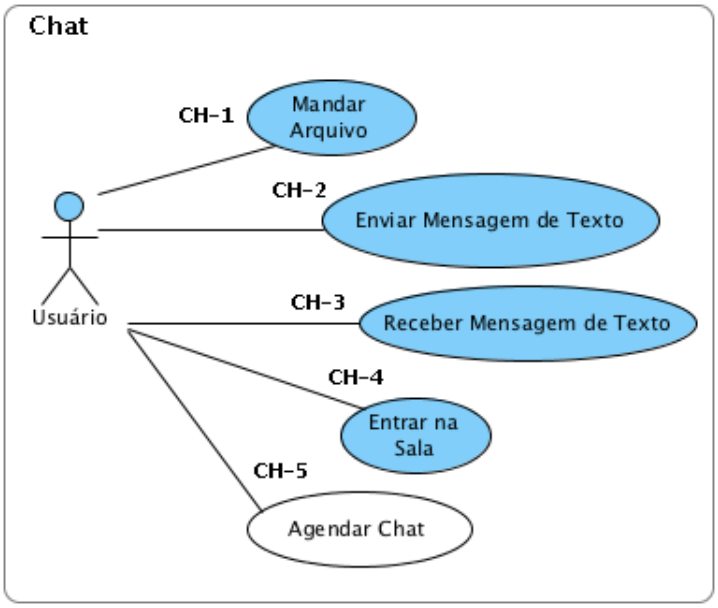


Figura 14: Parte do diagrama de casos de uso da aplicação Chat.

Com base nas similaridades descobertas, dois *asset* foram desenvolvidos: um serviço de comunicação síncrona (SCS), descrito na Seção 6.2.3.2, e um componente de GUI para comunicação textual (*Text*), descrito na Seção 6.2.3.1. A Listagem 6.2.1 mostra um trecho da descrição do componente de GUI para comunicação por texto. Essa descrição é uma Especificação de Asset, um exemplo da saída produzida por esta disciplina conforme Figura 10. No restante deste capítulo, adota-se o *asset Text* como exemplo. Descreve-se as etapas de construção do *Text* em cada uma das disciplinas da ED. *Text* também será usado nas disciplinas de EA para exemplificar o reúso de *assets*.

As similaridades entre as aplicações Comunicador Instantâneo e Chat foi de simples identificação. No entanto, essa similaridades podem não ser tão aparentes, por isso é importante a experiência e conhecimento do domínio.

Nome: *Text*

Descrição: *Asset de interface gráfica para envio e visualização de mensagens síncronas textuais*

Funcionalidades:

1. O *asset* deve permitir o envio e visualização de mensagens textuais de forma síncrona por meio de uma interface gráfica Web.
2. O *asset* deve ter duas regiões: uma para exibir mensagens enviadas, e outra para enviar mensagens.
3. Cada mensagem exibida deverá ser no formato:"nome_do_usuario (hora:minuto): texto_da_mensagem"
4. O *asset* deve atualizar a barra de rolagem para exibir a última mensagem.
5. O *asset* deve ter uma caixa de texto para envio da mensagem.
6. O *asset* deve ter um botão para envio da mensagem.
7. O *asset* deve usar o serviço de comunicação síncrona SCS para envio e recebimento de mensagens.
8. O *asset* deve receber como parâmetro o tópico de conversa, o id do participante e o nome do usuário.

Listagem 6.2.1: Trecho do documento de Especificação do *asset* de GUI para comunicação textual síncrona.

6.2.2 Análise de Asset

A disciplina Análise de Asset recebe como entrada um documento de especificação de asset, produzido na disciplina Requisitos de Asset. Embora a Figura 10 não mostre, esta disciplina também se apóia nos documentos de requisitos e modelos de casos de uso

das aplicações que motivaram a construção do *asset*. Esses documentos e modelos são importantes para que o *asset* a ser desenvolvido seja compatível com essas aplicações.

O objetivo da Análise de Asset é obter os requisitos do *asset* a ser construído. Diferentemente da Análise realizada convencionalmente no desenvolvimento de aplicações, onde se tem um cliente que conhece os requisitos, a Análise de Asset descreve os requisitos com base nas similaridades do domínio. Nesta disciplina, é fundamental contar com um especialista de domínio para definir requisitos que atendam aplicações ainda não especificadas.

A Análise de Asset descreve os requisitos do *asset* por meio de um documento de requisitos e modelos de casos de uso. Usa-se ferramentas CASE (*Computer-Aided Software Engineering*) como ferramenta para produção dos documentos e modelos necessários. Durante o desenvolvimento dos *assets* da LPSCSW usou-se a ferramenta CASE Jude (JUDE, 2010).

Na Seção anterior, o *asset Text*, *asset* de GUI para comunicação síncrona textual, foi identificado. O diagrama simplificado de caso de uso desse *asset* pode ser visto na Figura 15. A descrição do caso de uso Enviar Mensagem Textual pode ser viso na Figura 16. Nota-se que a primeira ação realizada na descrição desse caso de uso é feita pelo Sistema. As aplicações Comunicador Instantâneo e Chat têm ações iniciais diferentes. Os usuários do CI devem selecionar um grupo de contatos para inicio da comunicação textual, enquanto os usuários do Chat entram na sala de chat. Para fins de compatibilidade, apenas as funcionalidades em comum devem ser extraídas.

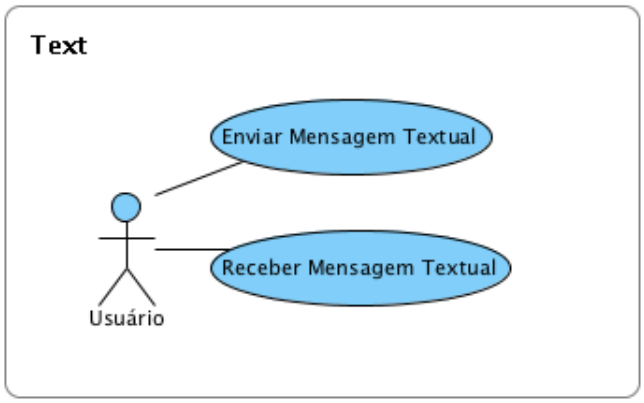


Figura 15: Diagrama de casos de uso do *asset Text*

Casos de uso genéricos podem ser transformados em *assets* para reúso por produtos da LPSCSW. O diagrama de casos de uso mostrado na Figura 15 e a descrição de casos de uso na Figura 16 são um exemplos de *assets* da Análise de Asset que podem compor a

ID do Caso de Uso:	4.1
Nome do Caso de Uso:	Enviar Mensagem Textual
Criado por:	Tiago Gaspar
Data de Criação:	29/08/08
Última Atualização Realizada por:	-
Data da Última Atualização:	-
Atores:	Usuário
Descrição:	Esse caso de uso corresponde à funcionalidade de enviar uma mensagem textual.
Prioridade:	1 (alta)
Pré-condições:	1. Usuário deve estar logado
	2.
Pós-condições:	1. Mensagem textual enviada
	2.

Fluxo Básico de Eventos	
Ações do Ator:	**Ações do Sistema:**
	1. Sistema provê uma tela onde o usuário pode escrever o texto que deseja. Essa tela é dividida em duas partes, uma inferior onde o usuário escreve o texto, e uma superior onde são exibidas as mensagens já enviadas e recebidas. A tela tem um botão para envio de mensagem.
2. Usuário escreve o texto e por meio da interface do sistema e clica no botão para envio de mensagem.	3. Sistema envia a mensagem para todos os participantes ligados a conversa. O sistema exibe na tela superior da tela a mensagem enviada pelo usuário.

Figura 16: Descrição do fluxo principal do caso de uso Enviar Mensagem Textual.

especificação de uma aplicação. A reutilização desses *assets* na Análise de Aplicação pode ser útil para diminuir esforços de confecção de documentos e modelos, e principalmente, para reutilizar o conhecimento de domínio adquirido na construção de diversas aplicações.

6.2.3 Projeto de Asset

A disciplina Projeto de Asset recebe como entrada os documentos produzidos na disciplina de Análise de Asset e tem o objetivo de realizar um plano detalhado para implementação de um *asset*. Normalmente, ferramentas CASE com suporte a UML são usadas como ferramentas para especificação do plano de implementação. As atividades realizadas nesta disciplina incluem diagramas de classe, diagramas de sequência, diagramas de componentes, definições tecnológicas (plataformas e linguagens de programação),

escolha de padrões de projeto, dentre outras. As definições tecnológicas adotadas pela LPSCSW são ZK, JSF (JAVA Server Faces), HTML, Javascript e Flash para apresentação; Red5, JAVA e Spring para lógica; e Hibernate e MySQL para persistência. *Assets* de projeto como padrões de projeto e arquiteturas também são criados nesta disciplina.

O desenvolvimento de algumas aplicações no domínio como Comunicador Instantâneo, Grava-Vídeo, Reface e acompanhamento das aplicações desenvolvidas pelo Laboratório Intermedia-ICMC-USP, como Chat e Digae, permitiu a especificação de uma arquitetura que facilita o reúso dos *assets* e a organização interna dos produtos da LPS descrita. Essa arquitetura divide aplicações e *assets* em camadas e pode ser vista na Figura 17.

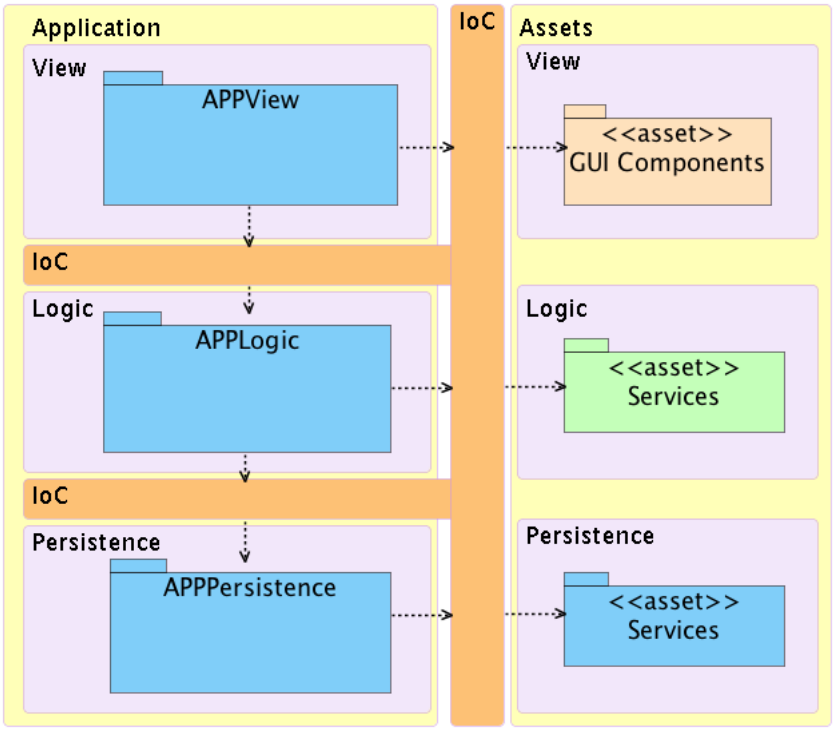


Figura 17: Arquitetura compartilhada entre os produtos da LPSCSW.

As atividades de ED e EA devem ser diferenciadas. Apesar da arquitetura mostrada na Figura 17 referir-se a uma aplicação, a arquitetura é tratada aqui como um *asset* comum a todos os produtos da linha, ou *core asset*.

O *core asset* arquitetura é dividido em três camadas, assim como os *assets* do repositório. A camada APPView é responsável pelas funcionalidades gráficas de apresentação, a APPLogic concentra a lógica de negócios e a APPPersitence é responsável pela persistência. As chamadas de métodos que acontecem entre camadas devem ser de cima para baixo, ou seja da camada mais superior para a mais inferior, evitando assim o salto entre camadas não vizinhas.

De maneira análoga, os *assets* também são divididos nas três camadas. A LPSCSW tem *assets* que devem ser usados na camada APPView da aplicação, outros na camada APPLogic e por fim, *assets* da APPPersistence. A dependência dos *assets* com a aplicação e entre as camadas da aplicação são gerenciadas pelo padrão de projeto Inversão de Controle (IoC) (FOWLER, 2009).

A Inversão de Controle diminui o acoplamento através da gerência de dependências. Na prática, um arquivo de configuração *XML* guarda todas as dependências de um produto. Nesse arquivo, são descritas as implementações concretas dos *assets*. Caso haja necessidade da troca de algum desses *assets*, a alteração fica restrita apenas ao arquivo. IoC diminui acoplamento dos produtos com implementações específicas e evita a recompilação do código a cada mudança. A Listagem 6.1 mostra a injeção da camada de persistência (*DAO*) e do serviço de comunicação SCS no componente Comunicador Instantâneo.

```
<!-- Bean de logica da aplicacao Comunicador Instantaneo -->
<bean id="br.fapesp.tidia.ae.im.logic.IMLogic" class="br.fapesp.tidia.ae.im.logic.impl.IMLogicImpl"
    init-method="init" singleton="false">
    <property name="dao" ref="br.fapesp.tidia.ae.im.dao.IMDao"/>
    <property name="sCSLogic" ref="br.fapesp.tidia.ae.scs.logic.LogicImpl"/>
<bean>
```

Listagem 6.1: Exemplo de IoC usado na LPSCSW.

Para exemplificar como IoC funciona, suponha, na Listagem 6.1, que a camada de persistência (representada para propriedade *dao*) é dependente do SGBD MySql. Se esse SGBD necessita ser alterado para PostgreSQL, por exemplo, então, considerando que a classe DAO do PostgreSQL implementa uma interface em comum com a classe DAO MySql, a única modificação necessária é a atualização do valor da propriedade *dao*. Um *container* de IoC, como Spring, lê esse arquivo de configuração e injeta a implementação do DAO na aplicação.

Descreve-se a seguir, o projeto realizado para os *assets* desenvolvidos na LPSCSW agrupados por camada, conforme a Figura 17.

6.2.3.1 *Assets* de Interface Gráfica do Usuário (GUI)

Os componentes de interface gráfica ou componentes de GUI encapsulam complexidades de interfaces ricas baseadas em AJAX e Flash para troca de texto, áudio, vídeo e tinta eletrônica. Usa-se o *framework* ZK e *tags* JSF no padrão MVC (*Model View Controller*)

para construir as interfaces e facilitar o reúso dos componentes existentes. As principais formas de reúso de componentes gráficos da LPSCSW acontecem por uso da biblioteca de *tags* (*taglib*) JSF RWIC (*Rich Window Interface Components*). O uso dessas *tags* permite que os componentes sejam instanciados através de linguagem declarativa, simplificando o reúso na construção de interfaces.

RWIC é mostrado na Figura 18 e abstrai complexidades inerentes a uma janela de interface rica na Web proporcionando suporte para atualizações via AJAX[3], controle de janela (minimizar, maximizar, redimensionar, *popup*, arrastar, dentre outros), gerenciamento de layout e facilidades de integração com os serviços SCS e MSS, ambos descritos na seção 6.2.3.2.

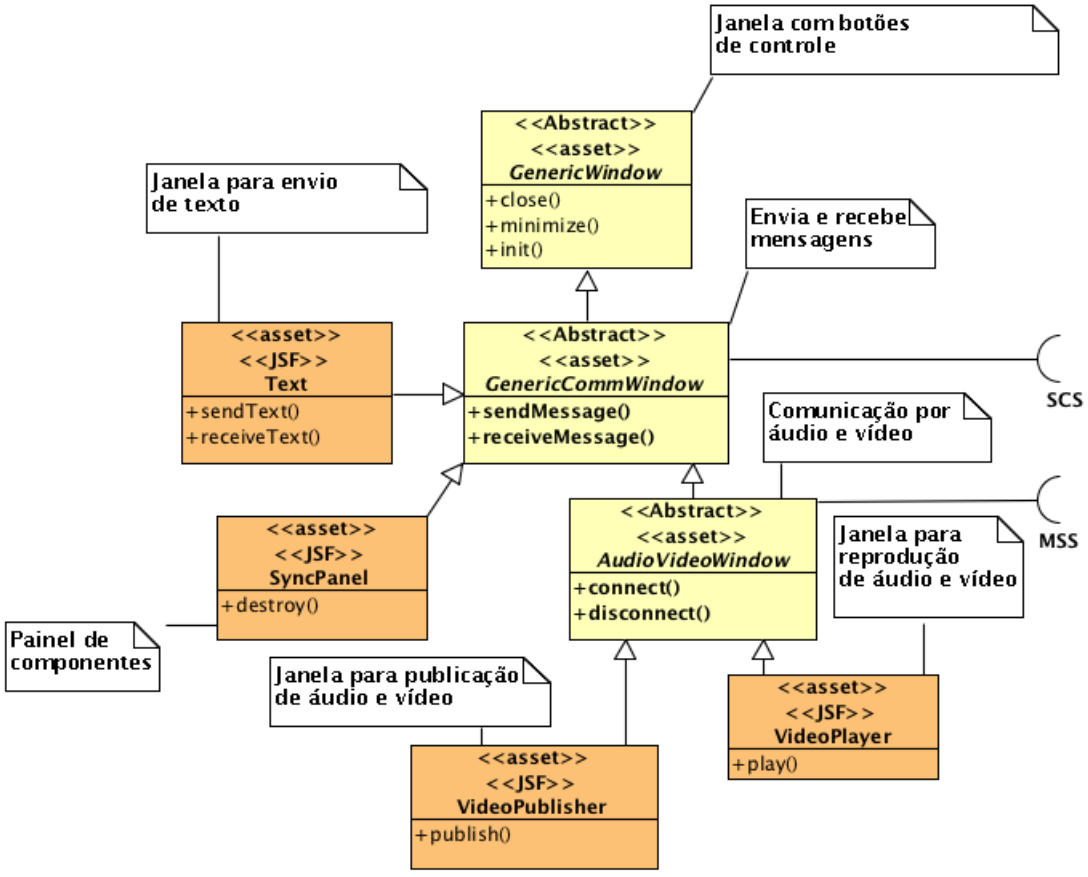


Figura 18: Arquitetura da biblioteca RWIC.

Como mostra a Figura 18, classes abstratas como *GenericCommWindow* e *AudioVideoWindow* abstraem complexidades de interfaces ricas e de comunicação. *Text*, que foi usada como exemplo nas disciplinas de Especificação de Asset e Análise de Asset, é uma classe concreta que representa uma janela para troca de mensagens de texto, essa classe

[3]Com o uso do AJAX, embora na prática as atualizações aconteçam de forma assíncrona, como o consta no próprio acrônimo, o usuário tem a impressão de atualizações síncronas devido ao curto período de resposta (Reverse AJAX (BOZDAG et al., 2007)).

também oferece parâmetros para ativação de uma barra de tarefas para envio de arquivos e exibição de histórico de mensagens textuais. *VideoPublisher* é uma classe concreta responsável pela captura e publicação de áudio e/ou vídeo. *VideoPlayer* também é uma classe concreta que reproduz um fluxo multimídia. Todas as classes concretas mostradas na figura são mapeadas para *tags* JSF e consideradas componentes de interface gráfica.

A Figura 18 mostra apenas alguns componentes da *taglib* RWIC. A seguir, descreve-se brevemente o restante dos componentes:

- *ContacList*: é um componente de interface gráfica que exibe os participantes presentes em uma colaboração. ContactList possibilita que ações de seleção em membros da lista de contatos sejam customizadas para chamar outros componentes como *Text* ou *VideoPlayer*. Ícones de participantes *online*, *offline*, ausentes ou de papéis específicos podem ser customizados;

- *Mosaic*: é um componente de interface gráfica que exibe um painel de vídeos de participantes e os arranja da melhor maneira levando em consideração sua dimensão e a dimensão dos objetos que encapsula. Esse componente é um gerenciador de *layout* que contém várias instâncias de *VideoPlayer*;

- *PlayerPublisher*: é um componente de interface gráfica que possibilita a reprodução e publicação de áudio/vídeo no mesmo componente. O vídeo em reprodução ocupa a tela, enquanto um pequeno quadrado no canto inferior direito mostra o vídeo em publicação. Esse componente é útil quando não se tem muito espaço na tela do usuário, pois combina os componentes *VideoPlayer* e *VideoPublisher*;

- *Whiteboard*: é um componente de interface gráfica de lousa eletrônica que possibilita a vários participantes compartilharem de forma síncrona a mesma visão de um *canvas*. Nesse *canvas* é possível realizar anotações com tinta eletrônica, figuras geométricas, upload de um conjunto de slides, navegação sincronizada de slides, dentre outros. O componente permite também a persistência do que foi anotado na lousa para recuperação posterior; e

- *SyncPanel*: é um componente de interface gráfica responsável por notificar outros componentes RWIC para que realizem operações de limpeza quando um navegador é fechado, ou quando se navega para outra página.

6.2.3.2 *Assets* de Lógica

Os serviços ou componentes de lógica suportam funcionalidades de mais baixo nível comparados com os componentes de GUI. Esses serviços são responsáveis pela comunicação síncrona e pelo gerenciamento de sessão. Descreve-se a seguir o *core asset* SCS, o *asset* MSS e o *core asset* Serviço de Sessão:

O *core asset* Serviço de Comunicação Síncrona (SCS - *Synchronous Communication Service*) suporta a troca de mensagens síncronas de n-para-n participantes, seguindo o padrão de projeto Publish-Subscribe (GAMMA et al., 1995), possibilita a troca de mensagens de controle, de texto, de envio de arquivos e mensagens customizadas. O serviço também suporta o envio de mensagens para participantes *offline*. Nesse caso, as mensagens *offline* são entregues quando os participantes encontrarem-se *online*. SCS proporciona a semântica de domínio de comunicação síncrona para a troca de mensagens e depende de um serviço externo genérico como JMS (*JAVA Message Server*)(MICROSYSTEMS, 2009) ou Microcomm[4].

Conforme Figura 19, a classe SCSClientReceiver atua como um mediador para um serviço de mensagens externo, recebendo mensagens e as repassando. Clientes devem especializar essa classe para receber mensagens. SCSLogic oferece, principalmente, métodos para estabelecimento de conexão entre clientes e envio de mensagens. A classe SCSProviderAdapter segue o padrão de projeto *Adapter* (GAMMA et al., 1995) para diminuir o acoplamento com o serviço de mensagens externo[5]. Tanto SCSClientReceiver quanto SCSLogic são injetadas pelo *container* de IoC nas aplicações.

O *asset* Serviço de Fluxo de Mídia Síncrona (MSS - *Media Streaming Service*) permite streaming e persistência de áudio e vídeo em tempo real usando o protocolo RTMP[6]. Uma ponte de comunicação foi construída com SCS para notificação de eventos como início e término de um fluxo de vídeo, ou início e término de gravação de mídia. Uma aplicação pode ter necessidade de ser notificada do início de publicação de áudio ou vídeo de outra instância. Dessa forma, poderá, por exemplo, abrir uma janela *VideoPlayer* para exibir o vídeo do participante.

O *core asset* Serviço de Sessão oferece facilidades para gerenciamento de sessão como

[4]Microcomm é o serviço de mensagens adotado por este trabalho e foi desenvolvido durante o projeto Tidia-Ae como trabalho de conclusão de curso do bolsista Cássio Pereira.

[5]A mudança de um serviço de mensagens específico durante projeto Tidia-Ae motivou a adoção do padrão de projeto.

[6]Atualmente, é adotado um servidor de fluxo de mídia *Flash* código aberto chamado *Red5* (RED5, 2009). O Serviço de Fluxo de Mídia Síncrona também é compatível com o *Flash Media Server* (ADOBE, 2009a).

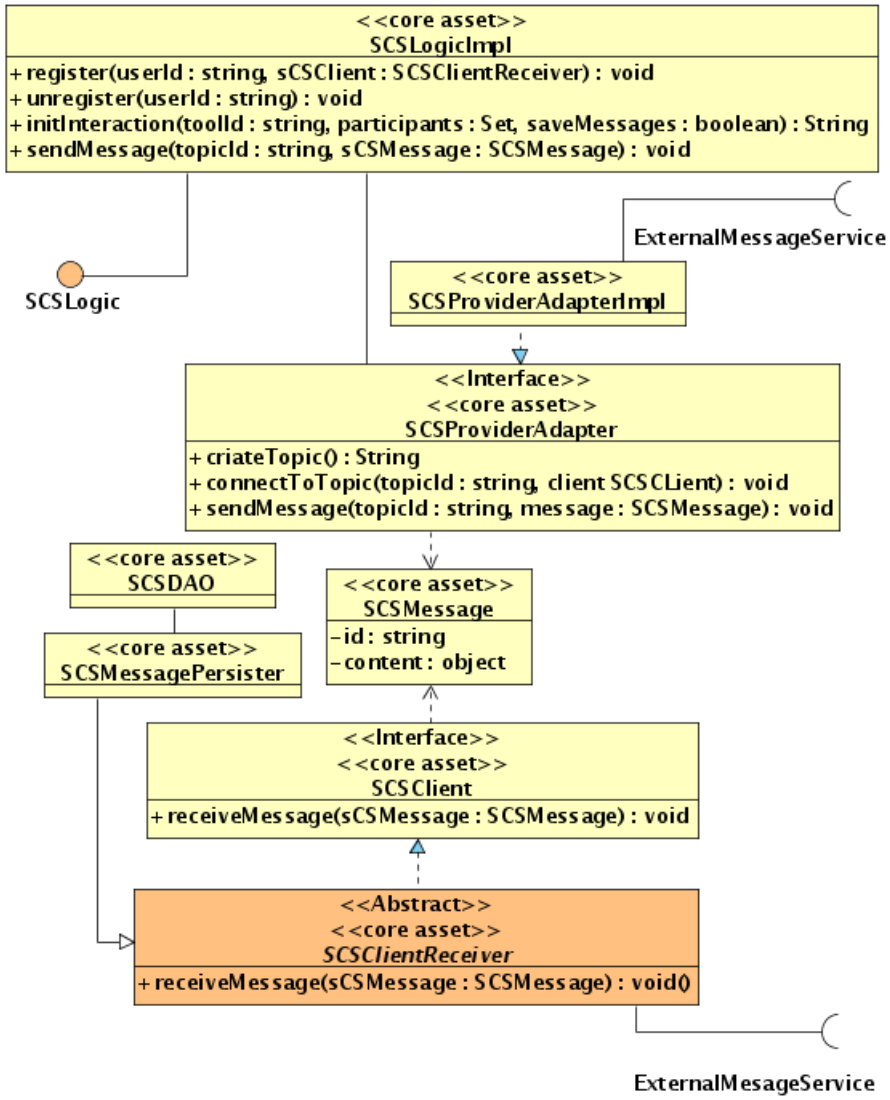


Figura 19: Modelo de classes do Serviço de Comunicação Síncrona (*SCS*).

inclusão/remoção/edição de participantes, notificações por e-mail, notificação de tempo esgotado, configuração de parâmetros, conflito de horários, entre outros. A sessão possui um período de tempo em que participantes atuarão na comunicação. O agendamento dessa sessão significa a escolha de um período, um grupo de participantes e um conjunto de parâmetros de configurações iniciais.

6.2.3.3 *Assets* de Persistência

O *asset* Serviço de Persistência oferece suporte para persistência de dados. Provê funcionalidades genéricas para consulta, criação, edição e remoção. A Figura 20 mostra o modelo de classes do Serviço de Persistência. A classe *LPSCSWGenericDAO* implementa as interfaces *Modifier*, que especifica métodos para alteração de dados, e *Finder*, interface

responsável pela busca de dados. Os objetos e classes que aparecem nos métodos de *LPSCSWGenericDAO* fazem parte do mapeamento ORM (*Object Relational Mapping*) do produto.

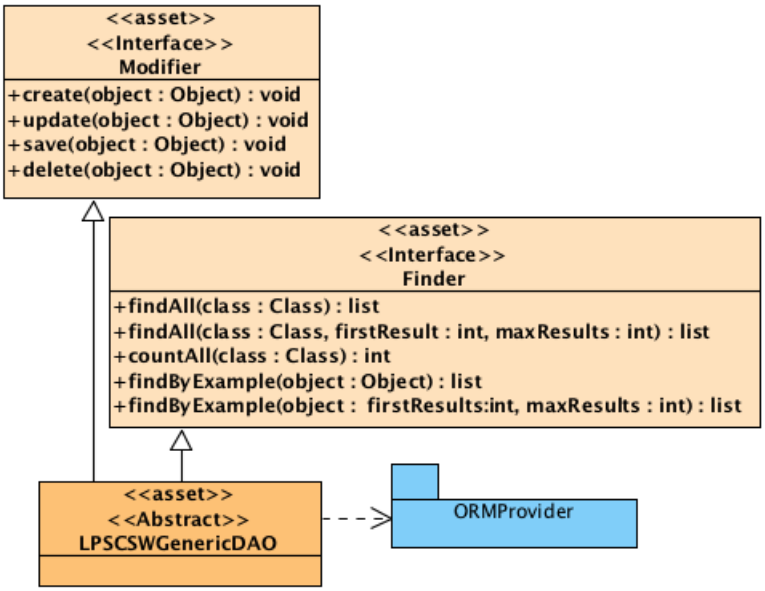


Figura 20: Modelo de classes do Serviço de Persistência.

De forma análoga ao SCS, o serviço é independente de implementações específicas de *frameworks* ORM, como *TopLink* (ORACLE, 2009) ou *Hibernate* (HIBERNATE, 2009) permitindo a mudança do serviço sem que o produto precise ser modificado. *LPSCSW-GenericDAO* é injetado no produto através de IoC.

6.2.4 Implementação de Asset

Esta disciplina é bastante parecida com a disciplina de implementação feita normalmente. A Implementação de Asset recebe como entrada um projeto detalhado para implementação de um *asset*. Esse projeto conta com diagramas de classes, componentes, sequências, dentre outros. Com base nos diagramas e especificações, realiza-se a implementação das funcionalidades. Essa disciplina tem a IDE (*Integrated Development Environment*) como principal ferramenta para confecção de código fonte. Em grande parte da codificação dos *assets* da LPSCSW usou-se J2EE com apoio da IDE Eclipse para desenvolvimento. Entretanto, também foram feitas codificações em *Flex* (linguagem para Flash) e Javascript.

A saída desta disciplina consiste num conjunto de arquivos nos quais estão contidos os códigos-fonte. A Listagem 6.2 mostra um exemplo de código fonte. O método da classe

Text mostrado na listagem é responsável pelo envio de mensagens. A linha 10 mostra a mensagem textual sendo obtida do campo de entrada de dados, a linha 13 constrói uma mensagem SCS, e a linha 16 envia a mensagem pelo SCS.

```
1  public class Text extends GenericCommWindow{
2      ...
3
4      /**
5       * Send text message to participants
6       */
7      public void sendText() {
8
9          // getting message content
10         String messageContent = inputText.getValue();
11
12         //Constructs a message given its content
13         CommMediaMessage message = new CommMediaMessage(CommMessageType.TEXT,
               messageContent, this.getUser().getId(), topicId);
14
15         // send message using SCS
16         this.sendMessage(message);
17
18         // clear Text inputArea content
19         inputText.setRawValue("");
20     }
21
22     ...
23 }
```

Listagem 6.2: Trecho de código responsável pelo envio de mensagens pertencente ao core asset SCS.

6.2.5 Testes de Asset

Essa disciplina recebe como entrada um conjunto de códigos-fonte para teste. Normalmente, realizam-se testes unitários para verificar a conformidade do código. Esses testes, depois de desenvolvidos, podem ser automatizados e aplicados a cada manutenção do código. A automatização de unitários testes é útil para manutenções, pois pode-se usar novamente os testes unitários desenvolvidos para avaliar a conformidade das mudanças. Após a execução dos testes unitários, são feitos testes de integração, caso o *asset* em teste tenha dependência com outro componente, por exemplo.

Nem todas as funcionalidades podem ser testadas por testes unitários. Conectividade e qualidade de vídeo ou áudio, ou elementos de interface gráfica são alguns exemplos. Nesse caso o *asset* deve ser testado manualmente.

Se o *asset* apresentar erros, esses erros são reportados e o conjunto de códigos-fonte volta para a Implementação de Asset para correção. Se os testes forem bem sucedidos, o *asset* é inserido no repositório de *assets* e está pronto para o reúso.

Listagem 6.3 mostra o caso de teste unitário para o método de envio de mensagens textuais da classe *Text*. Esse teste simula o envio de uma mensagem e verifica se a mesma foi recebida. Na linha 8 mostra-se a criação da classe *Text*, na linha 9 mostra-se a configuração do serviço SCS, na linha 13 a mensagem é enviada, na linha 18 verifica-se se a mensagem enviada foi recebida.

```
1  /**
2   * Unit Test for testSendText
3   * @author Tiago Gaspar
4   */
5
6  public void testSendText() {
7
8      Text testText = new Text();
9      testText.setSCS(new MockSCS());
10
11     String sentMsg = "Geronimooo!";
12     testText.getInputText().setValue(sentMsg);
13     testText.sendText();
14     this.waitForAnswer(1000);
15
16     String receivedMsg = getLastMsgString(testText.getOutputText().getValue());
17
18     Assert.assertEquals(sentMsg, receivedMsg);
19 }
```

Listagem 6.3: Método para envio de mensagens da classe Text.

Além de testes unitário, alguns *assets* podem necessitar de testes manuais. Principalmente os *assets* de interface gráfica. Figura 21 mostra o teste manual realizado no *asset Text*.

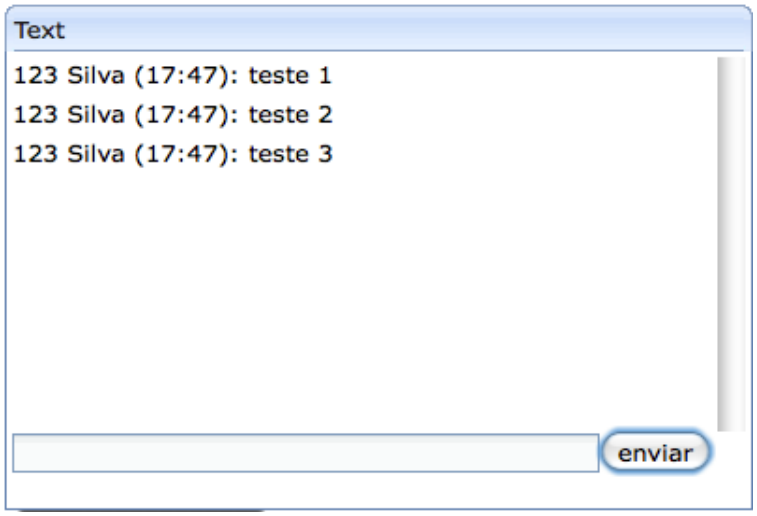


Figura 21: Teste manual do *asset* de GUI *Text*.

6.3 Engenharia de Aplicação

Produtos ou aplicações resultantes da LPSCSW são construídos na Engenharia de Aplicação (EA). Nessa etapa, o objetivo principal é construir produtos fazendo reúso de software. Cada produto resultante da linha tem em comum o reúso do conjunto de *core assets*.

A EA estabelece uma metodologia para construção dos produtos através de iterações de curta duração. Em cada iteração, acontecem as disciplinas de Análise de Aplicação, Projeto de Aplicação, Implementação de Aplicação e Testes de Aplicação. A Figura 22 mostra o diagrama SADT da EA. O reúso dos *assets* pode acontecer em algumas disciplinas, cada uma delas conta com *assets* específicos. Por exemplo, os *assets* produzidos durante a disciplina de Análise de Asset podem ser usados na disciplina Análise de Aplicação. No entanto, esses *assets* não poderão ser usados no Projeto de Aplicação, que tem como objetivo especificar o projeto da aplicação e não a análise.

6.3.1 Análise de Aplicação

A Análise de Aplicação recebe os Requisitos de Aplicação como entrada. Esses requisitos são definidos pelo cliente, principal interessado na construção da aplicação. Com base nas informações providas pelo cliente, são feitos os documentos de análise e casos de uso da aplicação. Os *assets* disponíveis devem influenciar a produção desses documentos, inclusive complementado-os com os *assets* de análise disponíveis.

As ferramentas utilizadas nessa disciplina são editores de texto para os documentos

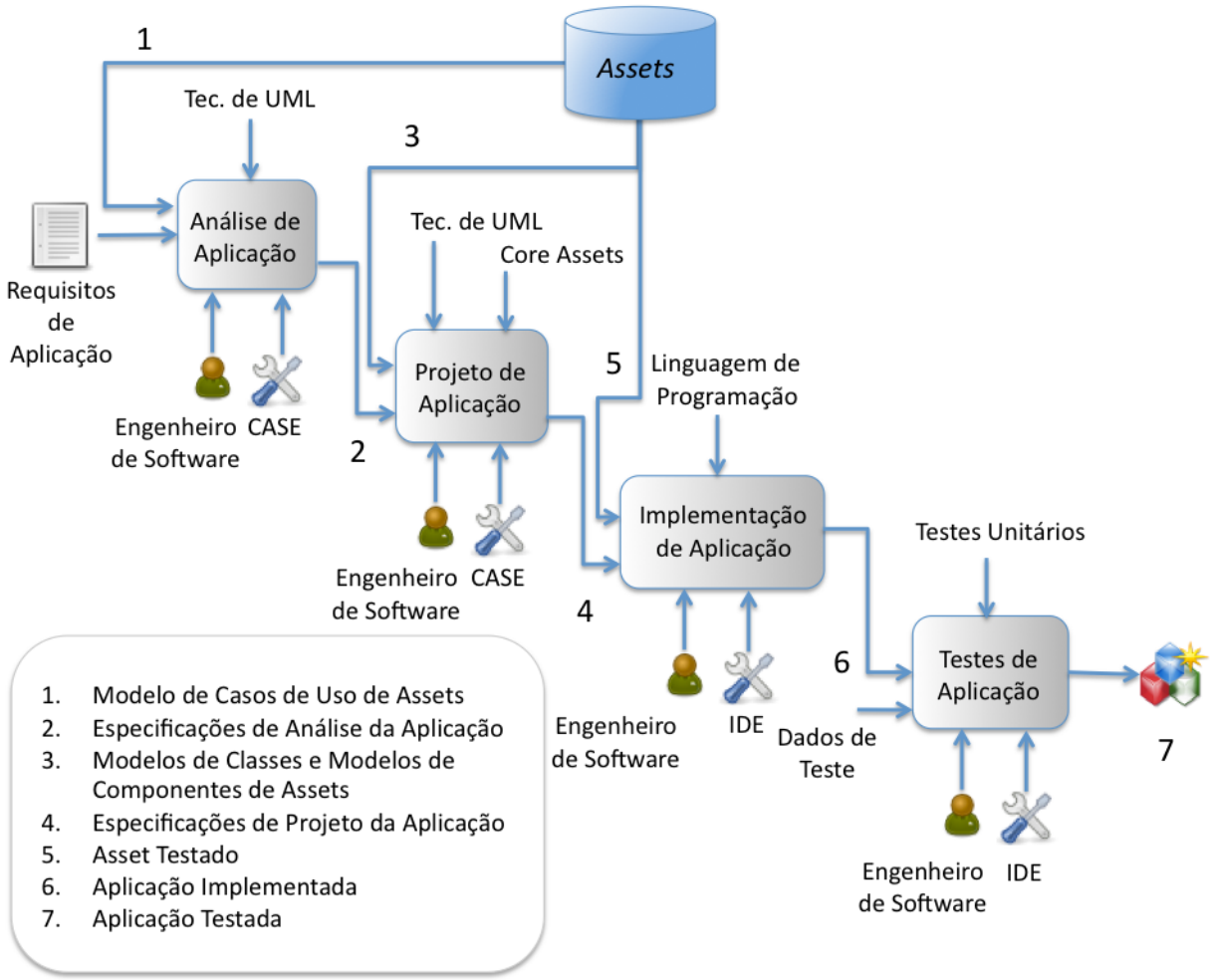


Figura 22: Disciplinas realizadas na Engenharia de Aplicação.

de requisitos e casos de uso, e uma IDE com suporte a UML para os diagramas de casos de uso.

Para ilustrar esta e as demais disciplinas de EA, constrói-se uma aplicação exemplo. Essa aplicação exemplo chamada de Meeting é uma aplicação de comunicação síncrona para reuniões que permite aos participantes conversar, ver uns aos outros, rabiscar na lousa eletrônica, trocar mensagens textuais e, ao fim da reunião, compor uma ata com o assunto discutido. Esse produto exemplifica o processo de construção usando a LPS descrita neste trabalho.

Pela descrição da aplicação, identifica-se o reúso dos *assets*: *Mosaic* para exibição de áudio e vídeo dos participantes; *VideoPublisher* para captura e publicação de câmera e microfone; *Whiteboard* para a lousa eletrônica; e *Text*, asset exemplo construído na ED, para comunicação por mensagens textuais. No entanto, o requisito ata de reunião não está no repositório de *assets*, pois é uma especificidade da aplicação Meeting.

Figura 23 mostra o diagrama de casos de uso simplificado da aplicação Meeting. Os casos de uso sombreados, são aqueles identificados como *assets* e pertencentes ao repositório. Dois casos de uso, Acessar Ata e Compor Ata, não são cobertos pelos *assets* existentes e, portanto, devem ser descritos.

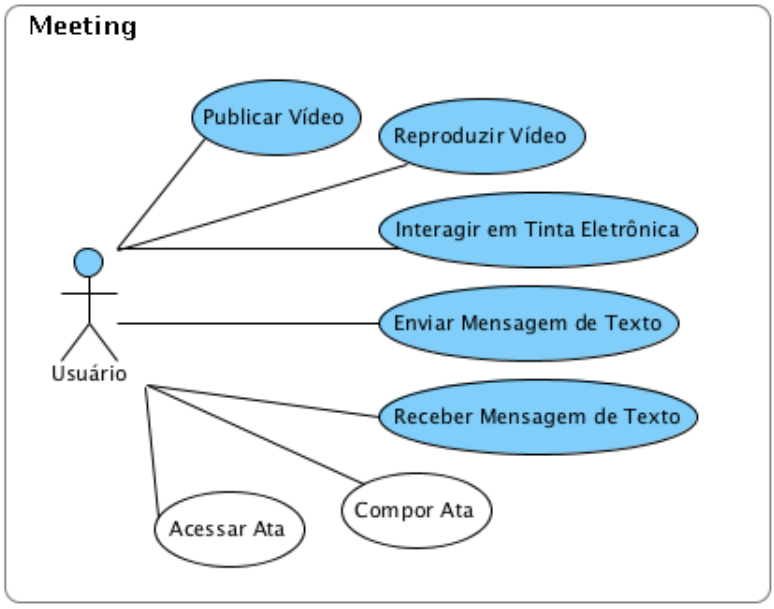


Figura 23: Diagrama de caso de uso simplificado da aplicação Meeting.

6.3.2 Projeto de Aplicação

Projeto de Aplicação recebe os documentos de requisitos e casos de uso produzidos na disciplina de Análise de Aplicação. Possivelmente alguns *assets* já foram identificados como candidatos a reúso e já estão inclusos nesses documentos. Outros *assets* de projeto como padrões de projeto e a arquitetura serão reusados nesta disciplina a partir do repositório de *assets*.

Nesta disciplina, assim como na disciplina Projeto de Asset, define-se o conjunto de tecnologias que será adotado pelas aplicações. As tecnologias usadas na LPCSCSW por esta disciplina são: JSF para apresentação, JAVA e Spring para a lógica, e Hibernate e MySQL para persistência. Há uma diminuição do conjunto de tecnologias envolvidas em relação à disciplina Projeto de Asset. A diminuição desse conjunto deve-se, principalmente, pelas abstrações feitas no processo de construção dos *assets*.

A arquitetura da LPSCSW, definida na Figura 17, é compartilhada por todos os produtos, portanto, o primeiro passo realizado no Projeto de Aplicação é criar um diagrama de classes com base na arquitetura proposta. Em seguida, os *assets* identificados são

conectados ao diagrama de classes da arquitetura. As partes específicas da aplicação são modeladas com base na arquitetura, seguindo as separações de camadas em AppView, AppLogic e AppPersistence.

Geralmente, as ferramentas utilizadas nesta disciplina são IDEs com suporte à UML. As saídas são compostas por modelos detalhados para implementação com diagramas de classes, componentes, sequências, dentre outros.

A Figura 24 mostra o diagrama de classes da aplicação exemplo Meeting. Esse diagrama mostra o reúso dos *assets* RWIC na camada APPView, Serviço de Sessão na camada APPLogic e Serviço de Persistência na camada APPPersistence e o padrão de projeto Inversão de Controle para gerenciamento de dependências. Embora o *core asset* SCS não seja usado diretamente pela Meeting, ele é usado pelos componentes RWIC. Portanto, como todo *core asset* da LPSCSW, ele está presente na Meeting.

Na camada *MeetingView* da aplicação, conforme Figura 24, a página JSF *Main* agrega os *assets* de interface gráfica e a página JSF *Minute* foi desenvolvida para permitir a criação da ata de reunião. Complexidades relativas a interfaces ricas, serviços de comunicação, áudio e vídeo, whiteboard, e etc, estão encapsuladas nos *assets* de GUI. Na camada de lógica, a classe *MeetingLogic* trata basicamente da lógica de negócio relativa ao requisito ata de reunião e acessa o Serviço de Sessão para obter informações sobre os participantes da reunião. Por fim, na camada de persistência, *MeetingDAO* persiste a ata da reunião.

6.3.3 Implementação de Aplicação

A Implementação de Aplicação recebe como entrada os artefatos produzidos no Projeto de Aplicação. Esses artefatos já refletem o reúso planejado, pois os *assets* selecionados fazem parte dos modelos. O esforço de implementação é direcionado às funcionalidades específicas da aplicação e a instanciação dos *assets* selecionados. IDEs são usadas para confecção dos códigos fontes específicos e para conexão aos *assets* selecionados. Ao fim desta disciplina, tem-se um conjunto de arquivos de código fonte prontos para testes.

A Listagem 6.4 mostra os *assets* que foram usados na página JSF *Main*[7]da aplicação Meeting. Como esses *assets* estão disponíveis através de *taglibs*, é possível usá-los declarativamente numa página JSF. Linguagens declarativas geralmente são mais produtivas na construção de interfaces gráficas, pois tendem a ser menos complexas quando comparadas com linguagens imperativas.

[7]O código listado foi simplificado para ressaltar o uso da *tags* RWIC.

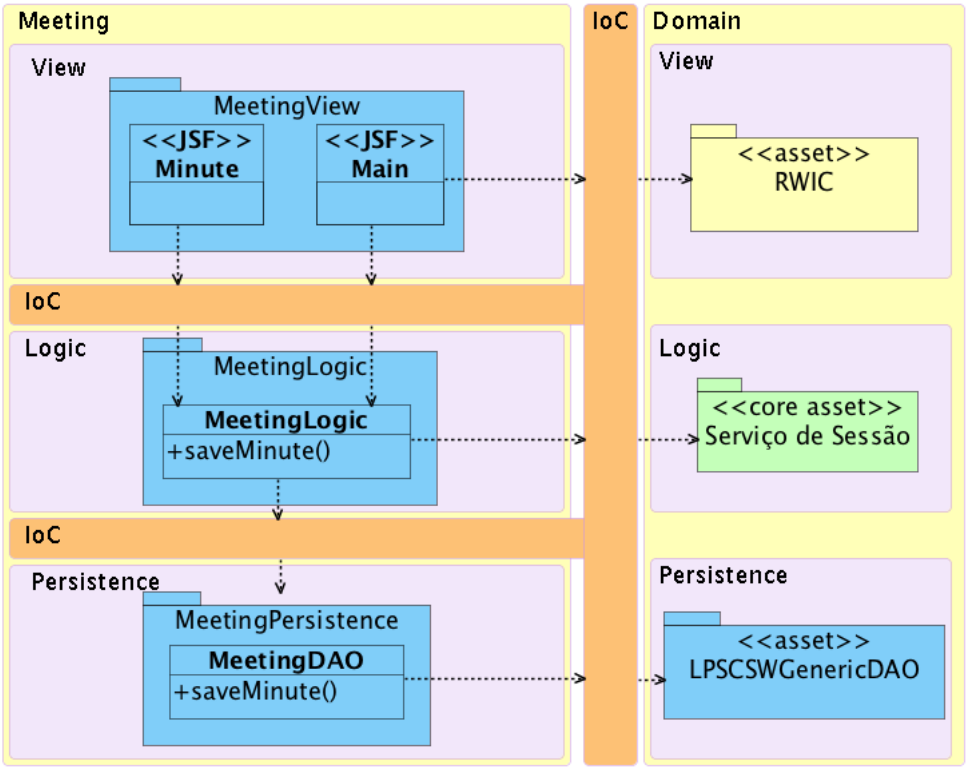


Figura 24: Arquitetura da aplicação Meeting.

```
1  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2  <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3  <%@ taglib uri="http://br.fapesp.tidia.ae.lpscsw/jsf/rwic" prefix="rwic" %>
4
5  <f:view>
6  <h:form>
7  <rwic:syncpanel>
8      <h:commandLink value="sair" action="finish" style="...">
9      <rwic:whiteboard participants=#{MeetingLogic.participants} editAll="true" style="...">
10     <rwic:text participants=#{MeetingLogic.participants} showtoolbar="true" style="...">
11     <rwic:videopublisher serverUrl=#{MeetingLogic.MSSServerUrl} stream=#{MeetingLogic.userId}
           debug="true" style="...">
12     <rwic:mosaic serverUrl=#{MeetingLogic.MSSServerUrl} streams=#{MeeintLogic.participantsId}
           debug="true" style="...">
13 </rwic:syncpanel>
14 </h:form>
15 </f:view>
```

Listagem 6.4: Trecho de código da página JSF *Main*.

6.3.4 Testes de Aplicação

Os testes realizados nesta disciplina tem o objetivo de identificar erros no conjunto de códigos-fonte produzidos na disciplina de Projeto de Aplicação. Em primeiro lugar, testa-se o código específico da aplicação através de testes unitários. Em seguida, realizam-se testes de integração com os *assets* selecionados. E por fim, testa-se as aplicações através do uso, simulando ações de usuário. Caso algum erro seja identificado, o mesmo é reportado para a equipe de programadores e retorna para a disciplina Projeto de Aplicação. Caso o código não apresente erro, finaliza-se a iteração.

Não é necessário fazer testes direcionados aos *assets*, pois já foram validados na ED e, possivelmente, estão em produção em outras aplicações. No entanto, caso seja identificado algum erro nos *assets*, esses erros são reportados para a ED.

As ferramentas usadas nessa disciplina são IDEs de teste unitário como *JUnit* e navegadores para simulação de uso. Quando trata-se de aplicações Web, é necessário realizar os testes de uso para cada navegador, ou no mínimo para os quatro mais comuns do mercado.

A Figura 25 mostra o teste de uso realizado no navegador Firefox para a página *Main* da aplicação Meeting. No canto esquerdo superior tem-se o componente *Whiteboard*, no canto direito superior mostra-se o componente *Text*, no canto inferior esquerdo o vídeo do participante é publicado através do componente *VideoPublisher* e no canto inferior direito pode-se ver os participantes através do componente *Mosaic*.

No próximo capítulo, quantifica-se alguns *assets* da linha e alguns produtos desenvolvidos através de métricas de qualidade. Descreve-se, também, os cenários de uso dos produtos decorrentes da LPSCSW.

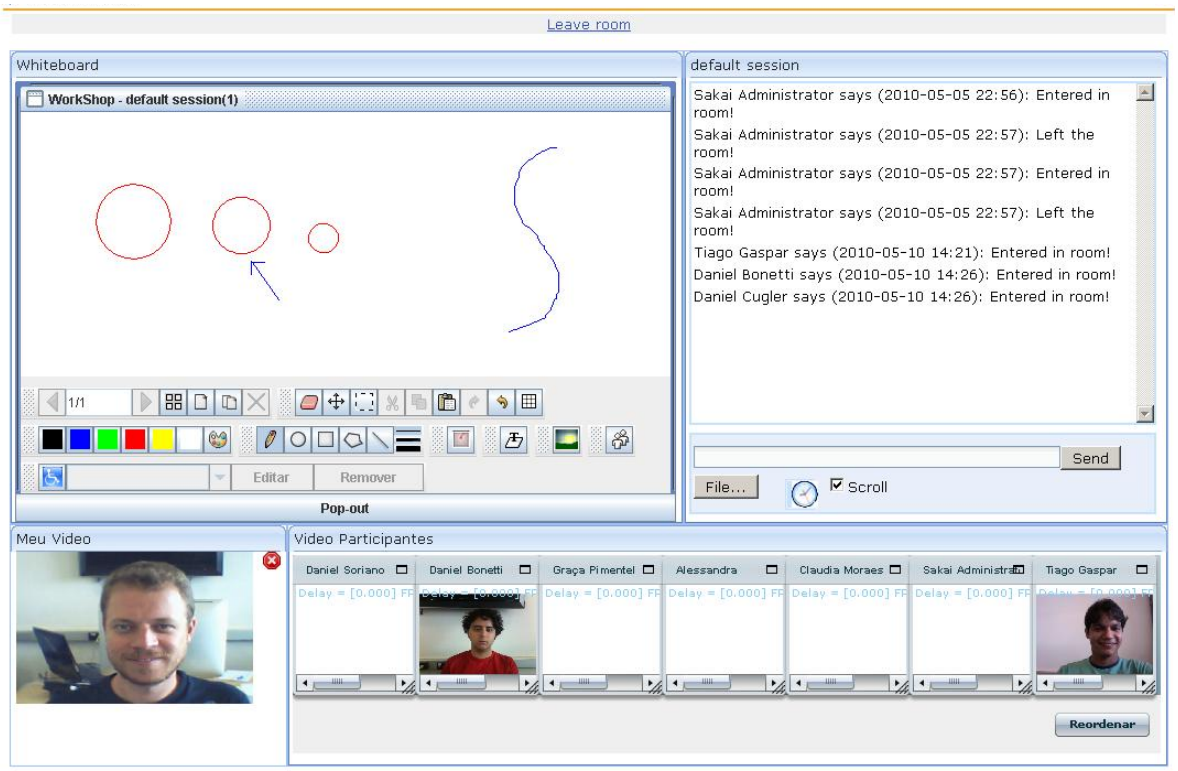


Figura 25: Aplicação Meeting derivada da LPSCSW.

7 Validação do Trabalho

Na medida em que empresas e instituições procuram implementar programas de reúso de software como forma de obter melhorias na produtividade e qualidade, é preciso medir o progresso desses programas e identificar as estratégias de reúso mais eficientes. Modelos e métricas de reúso servem a esse propósito (FRAKES; TERRY, 1996).

Uma boa maneira de validar este trabalho é através do emprego de métricas de software para avaliar a qualidade e o reúso realizado na construção de produtos da LPS descrita. Comparar a evolução das métricas entre duas versões de uma mesma aplicação, uma construída com a abordagem de reúso e outra construída sem a abordagem, pode ser significativo no que tange a evolução de qualidade e reúso propiciado por este trabalho. A aplicação Comunicador Instantâneo (CI), por ser a primeira desenvolvida, possui uma versão não componentizada. Dessa forma, pode-se comparar essa versão com a última versão do CI reconstruída a partir da abordagem de reúso descrita.

Quanto mais aplicações fazem reúso de software, melhor se valida a porção de software reutilizado. Apesar da abordagem de reúso descrita neste trabalho ter sido usada na construção das aplicações do projeto Tidia-Ae, a quantidade dessas aplicações ainda é pequena quando comparada com aplicações decorrentes do reúso praticado pela indústria de software. Portanto, outra forma de avaliar a LPSCSW é medir a reusabilidade de seus *assets*, ou seja, medir as características que potencializam o reúso futuro.

Outra forma de validação, e talvez a mais importante para avaliar a LPSCSW, é o uso efetivo das aplicações resultantes do reúso. Uma das vantagens de se aplicar pesquisa em projetos de desenvolvimento de software com usuários reais, como o Tidia-Ae, é a oportunidade de testar e evoluir o software construído com retornos balizados pelo uso real.

Na próxima seção, faz-se uma introdução sobre qualidade e métricas de software. Em seguida, compara-se as melhorias das métricas de qualidade com relação a tendência de erros (*error-prone*) e manutenibilidade para duas versões de uma mesma aplicação.

7.1 Qualidade e Métricas

Qualidade de software pode ser algo bastante subjetivo. Um software com boa qualidade para um usuário de Unix como o editor de texto Emacs, pode não ser visto com a mesma satisfação por um usuário Windows, sem a mesma desenvoltura técnica. Apesar da subjetividade, a ISO estabeleceu o padrão ISO/IEC 9126 (ISO/IEC, 2001), embora com controvérsias no meio acadêmico, pois nem sempre os atributos definidos pelo padrão significam qualidade para o usuário final (JUNG et al., 2004).

Apesar das controvérsias, a norma ISO/IEC 9126 é amplamente usada como parâmetro de qualidade em softwares e prevê os seguintes atributos e subatributos de qualidade:

- Funcionalidade: Adequação, Exatidão, Interoperabilidade, Segurança e Conformidade;
- Confiabilidade: Maturidade, Tolerância a Falhas, Facilidade de Recuperação e Conformidade;
- Usabilidade: Facilidade de Compreensão, Facilidade de Aprendizagem, Operacionalidade, Atratividade, Conformidade;
- Eficiência: Comportamento em Relação ao Tempo, Utilização de Recursos, Conformidade;
- Manutenibilidade: Facilidade de Análise, Facilidade de Modificação, Estabilidade, Facilidade de testes e Conformidade; e
- Portabilidade: Adaptabilidade, Facilidade de Instalação, Coexistência, Facilidade de Substituição e Conformidade.

Algumas métricas foram propostas na literatura para medir atributos de qualidade em softwares (FENTON; NEIL, 1999), particularmente para sistemas orientados a objetos (BANSIYA; DAVIS, 2002), (ABREU; MELO, 1996), (LORENZ; KIDD, 1994). Uma das métricas mais citadas da literatura e com trabalhos de validações empíricas substanciais são as propostas por Chindamber et al. (C&K) (CHIDAMBER et al., 1994).

É importante usar métricas validadas pela comunidade científica, mas a praticidade dessas métricas também é fundamental. Aplicar métricas sem um conjunto de ferramentas adequado tende a ser improdutivo. Analisar milhares de linhas de código manualmente e realizar cálculos complexos com esses resultados é impraticável, além de ser propenso

a erros. Portanto, a existência de ferramentas para cálculo das métricas foi outro fator determinante na escolha das métricas C&K.

Algumas ferramentas para cálculo de métricas suportam grande parte das métricas C&K. No entanto, essas ferramentas não o fazem com precisão. Lincke et al. (2008) realizaram experimentos para avaliação de um mesmo conjunto de classes, com o mesmo conjunto de métricas, porém com ferramentas de cálculos diferentes. Esses autores constatam diferenças significativas entre os resultados obtidos.

A falta de precisão das ferramentas não representou um problema para a avaliação deste trabalho. O mesmo conjunto de ferramentas foi usado para cálculo das métricas de duas versões da mesma aplicação. A falta de precisão foi a mesma para as duas medidas. Portanto, é seguro criticar os resultados de forma relativa, mas não absoluta.

A seguir, define-se o conjunto de métricas C&K, juntamente com a interpretação dos autores:

1. Complexidade de Métodos por Classe (*Weighted Methods per Class - WMC*). WMC consiste na soma da complexidade de todos os métodos de uma classe.
   Seja $c_0, c_1, ..., c_n$ a complexidade de cada método de uma classe. WMC é a seguinte soma:

   $$\text{WMC} = \sum_{i=0}^{n} c_i.$$

   Se a complexidade de cada método for igual a 1, então WMC é a contagem dos métodos de uma classe.

   Interpretação:

   - O número de métodos e a complexidade dos métodos envolvidos é um indício do tempo e esforço requerido para desenvolver e manter a classe;
   - Quanto maior o número de métodos na classe, maior o potencial de impacto nos filhos, desde que os filhos herdem todos os métodos definidos na classe; e
   - Classes com grande número de métodos têm maior probabilidade de realizarem funções específicas na aplicação, limitando a reusabilidade.

2. Profundidade da Árvore de Herança (*Depth of Inheritance Tree - DIT*). DIT é a soma dos nós do caminho de maior profundidade entre a raiz e as folhas de uma árvore de herança.

   Interpretação:

- Quanto mais longe da raiz uma classe está na hierarquia, maior o número de métodos que a classe poderá herdar, tornando mais difícil prever seu comportamento;
- Árvores profundas são mais complexas pelo fato de existirem mais métodos e classes envolvidas; e
- Quanto mais longe da raiz na hierarquia uma classe se encontra, maior o potencial de reúso dos métodos herdados.

3. Número de Filhos (*Number of Children - NOC*). Número de subclasses diretas de uma classe (primeiro nível abaixo de uma classe na árvore de herança).

   Interpretação:

   - Como herança é uma forma de reúso, quanto maior o número de filhos, maior o reúso;
   - Uma classe que tenha um grande número de filhos talvez seja usada de forma imprópria; e
   - O número de filhos dá uma idéia do potencial de influência que a classe tem no design. Se uma classe tem um grande número de filhos, ela pode demandar testes mais rigorosos.

4. Acoplamento entre Objetos (*Coupling Between Object Classes - CBO*). CBO representa o número de outros objetos que estão acoplados a um determinado objeto. Acoplamento entre objetos significa que um objeto age em outro. Por exemplo, dois objetos estão acoplados se métodos de um objeto mandam mensagens ou acessam variáveis de outro objeto.

   Interpretação:

   - Acoplamento excessivo entre objetos aponta pobre design modular e previne o reúso. Quanto mais independente uma classe é, mais fácil o reúso da classe por outra aplicação;
   - Para melhorar a modularidade e promover o encapsulamento, deve-se minimizar o acoplamento intra-objetos. Quanto maior o acoplamento, maior a propagação de mudanças em um sistema, portanto a manutenção torna-se mais difícil; e
   - A medição do acoplamento é útil para prever a complexidade de testes de um sistema. Quanto maior o acoplamento, mais rigorosos os testes devem ser.

5. Resposta Para uma Classe (*Response For a Class - RFC*). RFC reflete o efeito de mensagens colaterais a partir de chamadas de métodos de uma classe. Mais precisamente, RFC pode ser definido como:
   RFC = |RS|, onde RS é o conjunto de respostas para uma classe.
   RS pode ser definido como:
   RS=$\{M\} \bigcup_{all\ i} \{R_i\}$, onde $\{R_i\}$ é o conjunto de métodos chamados por um método i e {M} é o conjunto de todos os métodos da classe.

   Interpretação:

   - Se muitos métodos podem ser chamados em resposta a uma mensagem, testes e depurações da classe tornam-se mais complicados, pois requerem um nível maior de entendimento do responsável pelos testes;
   - Quanto maior o número de métodos que podem ser chamados a partir de uma classe, maior a complexidade dessa classe; e
   - O pior valor de respostas possível pode ajudar na alocação apropriada de tempo para testes.

6. Falta de Coesão em Métodos (*Lack of Cohesion in Methods - LCOM*). LCOM representada o grau de independência de métodos de uma classe com relação ao compartilhamento de atributos da classe. Uma definição mais precisa é apresentada a seguir: Considere uma classe C_1 com n métodos $M_1, M_2, ..., M_n$. Seja $\{I_i\}$ o conjunto de variáveis de instancias usadas pelo método M_i. Existem n conjuntos $\{I_1\}, ..., \{I_n\}$.

   Seja P = $\{(I_i, I_j) | I_i \bigcap I_j = \emptyset\}$ e Q = $\{(I_i, I_j) | I_i \bigcap I_j \neq \emptyset\}$. Se todos n conjuntos $\{I_1\}, ..., \{I_n\}$ são $\emptyset$ então P = $\emptyset$.

   $$\text{LCOM} = |\text{P}| - |\text{Q}|, \text{ se } |\text{P}| > |\text{Q}|. \text{ LCOM} = 0, \text{ se } |\text{P}| < |\text{Q}|.$$

   Interpretação:

   - Classes com métodos coesos são desejáveis, pois promovem o encapsulamento;
   - A falta de coesão implica que a classe provavelmente necessite ser divida em duas ou mais subclasses; e
   - Baixa coesão aumenta a complexidade e tem como consequência a maior probabilidade de erros durante o processo de desenvolvimento.

As ferramentas Metrics 1.3.6 (METRICS-1.3.6, 2010) e Analyst4J (SWAT, 2010) foram usadas para análise do código e cálculo das métricas. Foi necessário combinar os resultados das duas ferramentas, pois as mesmas não possuíam suporte a todo o conjunto de métricas. A Tabela 4 mostra o resultado das métricas aplicadas ao CI V1.0, versão não componentizada, e CI V2.0, que faz uso dos *assets* descritos neste trabalho. A métrica WMC foi aplicada em conjunto com a métrica de complexidade ciclomática proposta por McCabe (MCCABE, 1976).

Tabela 4: Métricas C&K aplicadas à aplicação CI V1.0 e CI V2.0.

Métricas C&K (Média Aritimética)							
Aplicações	**WMC**	**DIT**	**RFC**	**NOC**	**LCOM**	**CBO**	**LOC**
CI V1.0	22,78	0,65	40,47	0	0,71	10,35	3315
CI V2.0	12,25	1,62	21	0	0,40	5,63	547

Nas próximas sessões analisam-se os dados da Tabela 4 com relação à tendência a erros e manutenibilidade.

7.1.1 Análise de Tendência a Erros

Basili et al. (1996) realizaram um estudo quantitativo da correlação entre a ocorrência de erros em programas orientados a objetos e as métricas C&K. O estudo avaliou um conjunto de 180 classes comparando os resultados dos erros detectados na fase de testes e valores das métricas C&K para cada classe. Ficou constatada a correlação entre as classes defeituosas e as métricas, em particular para WMC, DIT, RFC, NOC e CBO. A seguir, analisa-se a Tabela 4 de acordo com a correlação entre as métricas C&K e a possibilidade de ocorrência de erros nas aplicações CI V1.0 e CI V2.0.

- A métrica WMC é uma medida de complexidade. Quanto mais complexidade uma classe ou método tiverem, maior a possibilidade de erros. O CI V2.0 apresentou uma diminuição de WMC da ordem de 46%;
- A métrica DIT mede a profundidade da hierarquia de classes. Quanto mais profunda uma classe se encontra na hierarquia, mais métodos serão herdados e para uma hierarquia muito profunda, maior a possibilidade de detecção de erros. O CI V2.0 apresentou uma DIT aumentado em 148%. O aumento da métrica é consequência do reúso realizado através de hierarquia de classes (discutido na Seção 7.3). Nesse caso o CI V2.0 não apresentou melhoras.

- A métrica RFC mede o efeito de mensagens colaterais de uma classe iniciadas por uma mensagem local. Classes com maior RFC tendem a ser mais complexas e, de forma análoga a WMC, tendem a ser mais suscetíveis a falhas. O CI V2.0 apresentou uma diminuição de 48% em relação ao CI V1.0;
- A métrica NOC mostrou-se inalterada para as duas versões, permanecendo em zero;
- De acordo com Basili et al., a métrica LCOM não apresenta correlação significativa entre a possibilidade de detecção de erros; e
- A métrica CBO mede o acoplamento entre classes. Classes mais acopladas têm maior possibilidade de falhas, pois dependem fortemente de outras. Novamente nota-se que o CI V2.0 apresenta melhores resultados, uma diminuição de aproximadamente 46%.

7.1.2 Análise de Manutenibilidade

Li e Henrry realizaram um estudo correlacionando as métricas C&K e a manutenibilidade em sistemas implementados em Classic-Ada (LI; HENRY, 1993). Segundo esses autores, o esforço de manutenibilidade foi previsto de forma bastante satisfatória em cerca de 90%. Analisa-se a seguir, os valores das métricas C&K para o CI V1.0 e V2.0, mostrados na Tabela 4, com base na interpretação de Li e Henrry:

- Quanto maior o WMC, maior a complexidade da classe. Quanto maior o número de loops aninhados maior é a dificuldade de se entender a funcionalidade da classe, e portanto, maior será o esforço de modificações. WMC aponta que o CI V2.0 teve uma redução de complexidade de 46%;
- A métrica DIT mede a profundidade da hierarquia de classes. Quanto mais profunda uma classe se encontra na hierarquia, maior a possibilidade dessa classe fazer acessos a atributos da superclasse. O encapsulamento é violado se classes acessam os atributos da superclasse sem fazer uso dos métodos definidos para isso. Portanto, uma classe com DIT grande pode ser difícil de se manter. O CI V2.0 apresentou uma DIT aumentado em 148%. Não houve melhoras de manutenibilidade em relação ao CI V1.0 para essa métrica;
- A métrica RFC mede o efeito de mensagens colaterais de uma classe iniciadas por uma mensagem local. Quanto maior o número de RFC maior será a propagação de mudanças realizadas na classe. O CI V2.0 apresentou uma diminuição de 48% em relação ao CI V1.0;

- A métrica NOC mede o número de filhos que uma classe possui. Essa métrica mede outro ponto de vista em relação à DIT. Quanto mais filhos uma classe possui, maior será a propagação de mudanças para subclasses. Com relação à NOC, não houve mudanças significativas; e
- A métrica LCOM mede a falta de coesão dentro de uma classe. A coesão é caracterizada pela acesso a propriedades locais dentro dos métodos de uma classe. Se uma classe não é coesa, ou seja seus métodos não acessam grande parte dos atributos locais de uma classe, é sinal de modularização ruim. A modularização ruim afeta o esforço de manutenção, pois possivelmente uma classe com baixa coesão deverá ser divida em duas ou mais. O CI V2.0 apresentou uma diminuição de LCOM em 44%.
- A métrica CBO não foi usada nos estudos de Li e Henry.

Outro resultado positivo do CI V2.0 foi a diminuição substancial de linhas de código (LOC), em torno de 84%. Um código mais conciso e com menor complexidade tem o custo de manutenção diminuído.

Na próxima seção, quantifica-se o reúso real de software realizado no CI V2.0.

7.2 Quantificação de Reúso

Reúso de software pode ser medido em diferentes níveis. O reúso pode acontecer, por exemplo, na fase de análise ou projeto com o aproveitamento de documentos e modelos anteriores ou com a adoção de padrões de análise ou padrões de projeto. Esse tipo de reúso faz parte da abordagem de reutilização de software descrita neste trabalho, como o *core asset* arquitetura discutido na Seção 6.2. No entanto, quantificar precisamente o reúso através da adoção de um padrão de projeto pode não ser simples.

Um nível de reúso que pode ser medido com relativa precisão é feito através de análise do código fonte. A análise do reúso através do código fonte é um tema bastante discutido na literatura (DEVANBU et al., 1996), (PRIETO-DIAZ, 1991) e (BIEMAN, 1991). Algumas das métricas mais citadas na literatura são definidas por Frakes e Terry (FRAKES; TERRY, 1996).

Esses autores propuseram uma métrica simples para quantificação de reúso chamada Quantidade de Reúso. Quantidade de Reúso pode ser definida como:

$$QR = \frac{Linhas\ de\ codigo\ reusado\ em\ sistemas\ ou\ modulos}{Total\ de\ linhas\ de\ codigo\ no\ sistema\ ou\ modulo}$$

A fórmula acima é usada para calcular a Quantidade de Reúso feita pelo CI V2.0. Mas antes, os componentes usados pela aplicação e seus respectivos tamanhos (LOC - *Lines of Code*) precisam ser identificados. A Tabela 5 mostra o número de linhas de código dos componente usados no CI V2.0 e da própria aplicação[1].

Tabela 5: Número de linhas de código dos componente usados no CI V2.0 e da própria aplicação.

LOC das aplicações e seus componentes		
ID	**Nome**	**LOC**
A	SessionService	659
B	RWIC	1426
APP	CI V2.0	547

Usando a fórmula Quantidade de Reúso para o CI V2.0:

$$QR_{APP} = \frac{LOC(A) + LOC(B)}{LOC(A) + LOC(B) + LOC(APP)} = 0,79$$

Esse métrica significa a porcentagem de código reusado em uma aplicação. Portanto, a aplicação CI V2.0 é composta de cerca de 79% de código reusado.

Na próxima seção, avalia-se alguns *assets* da LPSCSW com relação ao atributo reusabilidade.

7.3 Análise de Reusabilidade

A reusabilidade é um conjunto de características que um artefato de software deve possuir para ser reusado a posteriori (FRAKES; TERRY, 1996). Uma das formas de avaliar os assets na LPSCSW é medir sua reusabilidade. No entanto, alguns assets da linha são de difícil avaliação. Não foram encontradas na literura métricas para avaliação de assets de análise ou de projeto. Entretanto, alguns componentes da linha como os GUI e de lógica podem ser ser avaliados por métricas para OO (CHIDAMBER et al., 1994) (LORENZ; KIDD, 1994) (ABREU; MELO, 1996) e por métricas para componentes (NARASIMHAN; HENDRADJAYA, 2005) (WASHIZAKI et al., 2003).

Existem algumas dificuldades para se avaliar reusabilidade nos assets de GUI e lógica da LPSCSW. Os autores das métricas que podem ser usadas para avaliar reusabilidade não definem intervalos de confiança, ou seja, valores que possam definir boa reusabilidade

[1] O LOC mostrado na tabela foi calculado com base nas linhas de código dos componentes utilizados pela aplicação.

para componente ou classes. A falta de ferramentas para cálculo das métricas é outro fator que complica a avaliação de reusabilidade.

As métricas C&K contam com suporte de ferramentas para cálculo e têm sido usadas para avaliação de reusabilidade. As métricas WMC, DIT, CBO e LCOM, em particular, podem ser usadas como parâmetro de reusabilidade (WASHIZAKI et al., 2003), (ROSENBERG; HYATT, 1997) e (CHIDAMBER et al., 1994).

Uma maneira de atenuar as dificuldades para se precisar a boa reusabilidade dos assets pode ser através de parâmetros de referência. Pode-se comparar os assets a componentes extensivamente reusados. Se esses componentes são amplamente reusados, é provável que tenham boa reusabilidade. No entanto, comparar componentes com funcionalidades distintas pode não ser significativo. Portanto, compara-se os *assets* de GUI da LPSCSW a outros componentes de GUI amplamente usados: o Ajax4JSF da JBoss. Ajax4JSF é usado por grande parte dos desenvolvedores que desejam adicionar suporte a AJAX para páginas JSF. O pacote java.lang do JDK 1.5 foi escolhido como referência para os *assets* que disponibilizam serviços de mais baixo nível como SCS e Session Service. java.lang é o pacote mais usado na linguagem JAVA. Nele, encontram-se tipos primitivos como Object, String, Integer, e a classe utilitária Math.

Uma aplicação que não foi projetada pra reúso é comparada aos *assets* e aos componentes de referência para se ter uma idéia dos atributos de reusabilidade que uma aplicação comum oferece. Espera-se que os *assets* assemelhem-se mais aos componentes de referência do que a uma aplicação que não serve ao reúso.

A Tabela 6 mostra os resultados das métricas de reusabilidade para o componente de referência Ajax4JSF, para o *asset* RWIC e para a aplicação CI V1.0. Os valores exibidos na tabela foram medidos pelas ferramentas Metrics 1.3.6 e Analyst4J.

Tabela 6: Métricas de reusabilidade aplicadas ao Ajax4JSF, RWIC e CI V1.0.

Reusabilidade - RWIC				
Componentes / Aplicações	**WMC**	**DIT**	**CBO**	**LCOM**
Ajax4JSF	18,33	1,11	6,67	0,33
RWIC	17,90	2,05	6,32	0,41
CI V1.0	22,78	0,65	10,35	0,71

Seguem algumas interpretações sobre reusabilidade com base nos valores das métricas medidos:

- A métrica WMC medida para o *asset* RWIC é similar, cerca de 2,5% menor, ao

WMC medido para os componentes Ajax4JSF. O CI V1.0 apresenta WMC elevada, cerca de 24% maior que Ajax4JSF. O alto WMC pode significar que algumas classes têm grande número de métodos específicos, dificultando o reúso;

- A métrica DIT medida para o *asset* RWIC é cerca de 84% maior que a DIT medida para os componentes Ajax4JSF. O CI V1.0 apresenta DIT cerca de 41% menor que Ajax4JSF. O alto DIT apresentado para RWIC justifica-se pela estrutura de herança dos componentes, conforme Figura 18, Sessão 6.2.3.1. RWIC define uma hierarquia de classes abstratas de baixa complexidade para propiciar o reúso. O baixo DIT apresentado pelo CI V1.0 pode indicar que a aplicação não apresenta muitas abstrações internas, o que pode ser um indício de baixo grau de reusabilidade;

- A métrica CBO medida para o *asset* RWIC é novamente semelhante, cerca de 5,5% menor, à CBO medida para os componentes Ajax4JSF. O CI V1.0 apresenta CBO elevada, cerca de 55% maior. O alto valor de CBO dificulta o reúso, pois significa pobre design modular e acoplamento excessivo entre objetos; e

- A métrica LCOM medida para o *asset* RWIC é cerca de 24,5% maior que a LCOM medida para os componentes Ajax4JSF. O CI V1.0 apresenta LCOM elevada, cerca de 115% maior. O alto valor de LCOM pode dificultar o reúso, pois aponta métodos com funções distintas em uma classe. Uma classe que não possui função bem definida dificilmente estará apta ao reúso.

A Figura 26 mostra graficamente os resultados obtidos com as métricas de reusabilidade. Nota-se que o *asset* RWIC e Ajax4JSF têm resultados semelhantes para a maioria das métricas de reusabilidade. No entanto, a aplicação CI V1.0 destoa visivelmente dos outros valores obtidos.

A Tabela 7 mostra os resultados das métricas de reusabilidade para os componentes java.lang, SCS, Session Service e para a aplicação CI V1.0. Novamente usa-se uma aplicação que não foi projetada para reúso para se ter uma idéia dos valores obtidos.

- A métrica WMC medida para os componentes java.lang é mais elevada do que para os componentes SCS e Session Service. O WMC mais alto de java.lang pode ser explicado pela alta complexidade das classes voltadas para cálculos matemáticos como Math;

- Os *assets* SCS e Session Service apresentam valores similares ao componente java.lang para a métrica DIT. A aplicação CI V1.0 apresenta DIT 56% menor que java.lang;

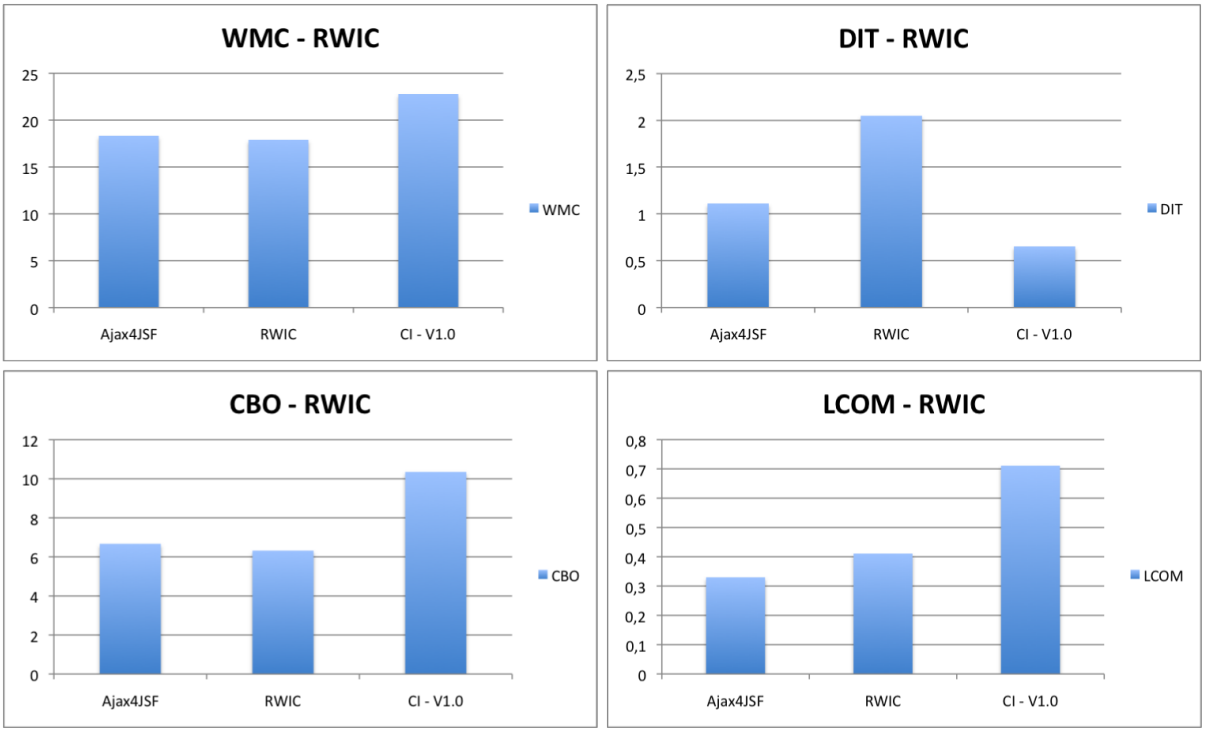


Figura 26: Métricas de reusabilidade para Ajax4JSF, RWIC e CI V1.0.

Tabela 7: Métricas de reusabilidade aplicadas ao java.lang, SCS, Session Service e CI V1.0.

Reusabilidade – Outros componentes				
Componentes / Aplicações	**WMC**	**DIT**	**CBO**	**LCOM**
java.lang	18,43	1,46	3,78	0,30
SCS	14,66	1,79	1,33	0,43
Session Service	14,33	1,56	1,43	0,41
CI V1.0	22,78	0,65	10,35	0,71

- Os *assets* SCS e Session Service apresentam CBO visivelmente inferior a java.lang, o que é uma característica desejável de reusabilidade; e
- A métrica LCOM medida para os *assets* SCS e Session Service apresentam valores similares a LCOM medida para java.lang. O CI V1.0 apresenta discrepância de 137% em relação a java.lang.

A Figura 27 mostra graficamente os resultados obtidos com as métricas de reusabilidade. A aplicação CI V1.0 novamente destoa visivelmente dos valores obtidos pelos componentes.

Tanto os componentes de GUI (Ajax4JSF e RWIC) quanto os componentes de lógica (java.lang, SCS e Session Service) mostraram métricas de reusabilidade similares. Como esperado, uma aplicação que não foi projetada para reúso (CI1.0) obteve métricas de

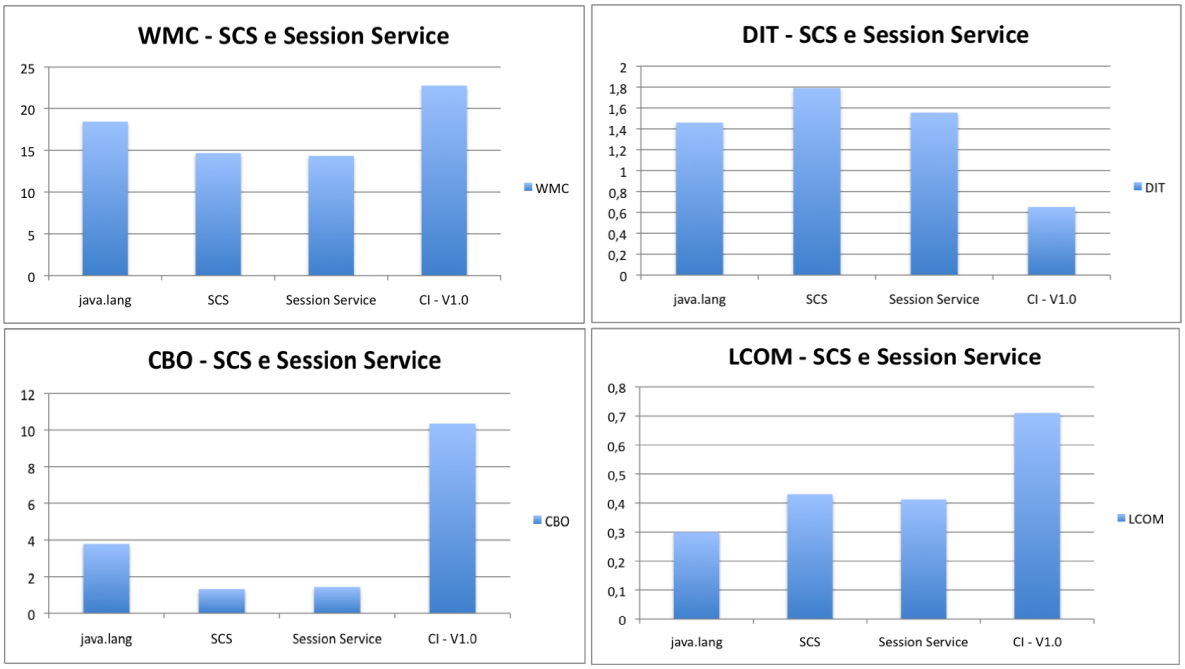


Figura 27: Métricas de reusabilidade para java.lang, SCS, Session Service e Comunicador Instantâneo V. 1.0.

reusabilidade piores do que os componentes avaliados.

Embora seja possível quantificar o grau de reusabilidade de um componente, as métricas de reusabilidade não levam alguns atributos relevantes de reúso em consideração, especialmente para os componentes de GUI. O uso de *tags* JSF é a principal forma de reúso desses componentes e as métricas não são capazes de fazer esse tipo de análise. Métricas específicas para reusabilidade em linguagem declarativa não foram encontradas na literatura.

Na próxima seção descrevem-se os cenários de uso das aplicações decorrentes da LPSCSW como forma de avaliar a linha de produtos desenvolvida.

7.4 Cenários de Uso das Aplicações

As aplicações Tidia-Ae de comunicação síncrona desenvolvidas pelo Lince, descritas no Capítulo 3, fizeram uso da abordagem de reúso descrita neste trabalho. Essas aplicações fazem parte da versão oficial do LMS (Learning Management System) Ae e estão disponíveis para uso gratuito[2].

As aplicações resultantes da LPSCSW também têm sido usadas como ferramentas

[2]Disponível em http://agora.tidia-ae.usp.br/portal

de comunicação no Departamento de Computação da UFSCar (DC) principalmente para apresentações de qualificação de mestrado e aulas remotas. Ao todo foram realizadas quatro qualificações de mestrado desde 2009. A Figura 28 mostra a qualificação de mestrado realizada em Junho de 2009, na qual o autor deste trabalho apresentou seu trabalho de qualificação fazendo uso da aplicação Whiteboard e de uma versão preliminar da Reface.

Figura 28: Qualificação de mestrado remota.

A Reface também foi usada pelo DC para aulas remotas nos cursos de graduação e pós-graduação. Essa aplicação foi útil para professores que precisaram se ausentar durante o período letivo e com o uso da Reface foram capazes de prosseguir com suas atividades didáticas de forma remota. A Figura 29 mostra o ambiente montado para uma ação de aprendizagem, na qual se explorou uma aula completamente remota, em que professores e alunos estavam em locais diferentes, porém interagindo de forma natural. O Prof. Dr. Marco Cristo da Universidade Federal do Amazonas (UFAM) gentilmente participou durante três dias do experimento, ministrando a aula Recomendação Web para Tv Digital para alunos de pós-graduação do DC e do ICMC-USP. A sala do professor, mostrada na figura, tem duas TVs, uma para cada turma e um computador sensível a toque para interações de tinta eletrônica. Uma parte da turma estava localizada no DC, em sala diferente do professor, e outra no ICMC-USP. Usou-se a rede de alta velocidade Kyatera (KYATERA, 2009) para transmissão de áudio e vídeo em alta qualidade. A Figura

30 mostra a visão que a turma teve do professor. A Figura 31 mostra a turma localizada no DC.

Figura 29: Sala do professor montada para ação de aprendizagem da Aplicação Reface.

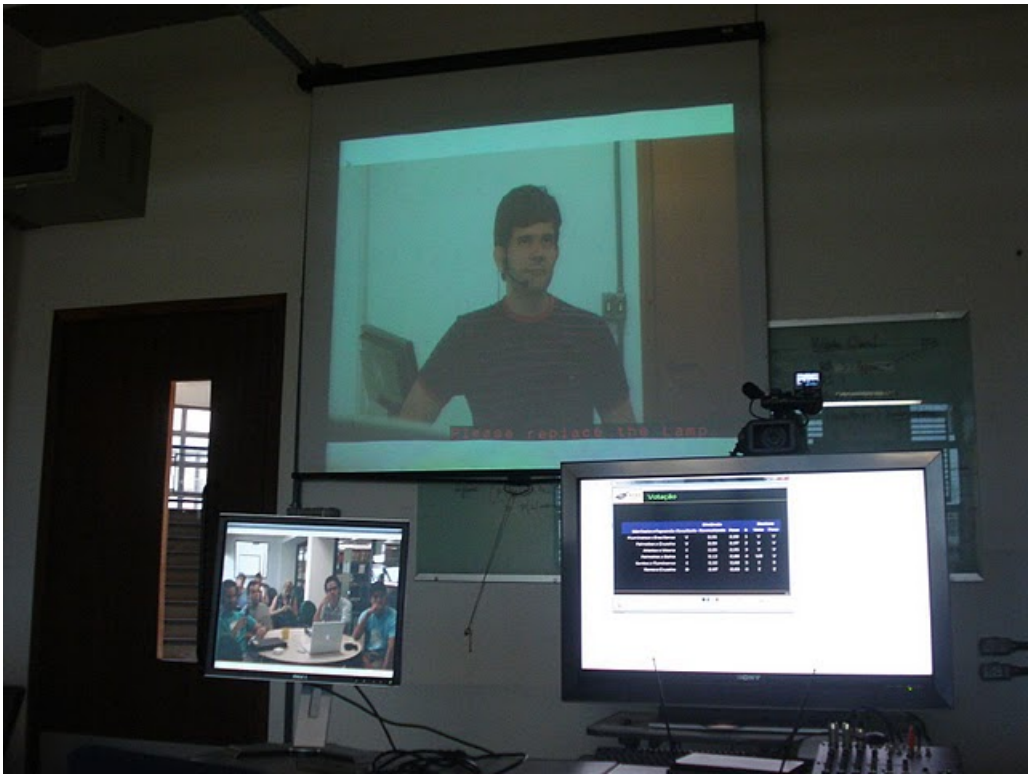


Figura 30: Visão da turma na ação de aprendizagem da Aplicação Reface.

Figura 31: Turma que participou da ação de aprendizado no DC.

8 Conclusão

Realizou-se um estudo das alternativas tecnológicas disponíveis para comunicação multimídia síncrona, conforme o item 1 da Seção1.1(Objetivo). As tecnologias ZK para interfaces ricas, Flash e Red5 para comunicação multimídia foram escolhidas para adoção da LPSCSW.

Disponibilizou-se a LPSCSW como abordagem de reúso para o domínio de aplicações Web de comunicação multimídia síncrona, conforme o item 2 da Seção 1.1. A construção da LPSCSW foi dividida em duas atividades: Engenharia de Domínio, onde desenvolve-se os *assets*, e Engenharia de Aplicação, onde aplicações, ou produtos da linha, são desenvolvidos com reúso desses *assets*. A LPS desenvolvida simplifica e possibilita a sistematização do desenvolvimento de *assets* para reúso e de produtos com reúso dos *assets*. O emprego desses *assets* torna transparente para o desenvolvedor algumas das dificuldades encontradas no domínio. O *asset* RWIC, por exemplo, requer do desenvolvedor apenas conhecimentos de JSF. Esse *asset* tem suporte a vários navegadores, portanto, o desenvolvedor não precisa se preocupar com códigos específicos para cada navegador.

Os *assets* e aplicações construídos na LPSCSW estão disponíveis através da licença LGPL, conforme os itens 3 e 4 da Seção 1.1. A licença LGPL permite a livre distribuição, alteração e comercialização dos *assets* e aplicações por qualquer interessado.

O uso das aplicações construídas foi incentivado através da inclusão das mesmas no ambiente Ae. A disponibilização dessas aplicações no instalador Ae permite a instalação do ambiente e das aplicações da LPSCSW de forma facilitada a qualquer interessado. As aplicações foram usadas pelo Departamento de Computação da UFSCar para viabilizar aulas e qualificações de mestrado remotas. Com isso atende-se ao item 5, último item, da Seção 1.1.

O emprego da LPSCSW resultou em melhorias significativas de qualidade de software. A comparação de duas aplicações com funcionalidades idênticas, uma versão feita sem reúso e outra construída com a LPSCSW foi importante para mensurar os ganhos obtidos

com uma estratégia de reúso. A comparação das aplicações CI V1.0 e CI V2.0 serviram a esse propósito. Mostrou-se melhoras de qualidade no quesito tendência de erros e manutenibilidade, além de uma redução de linhas de código na ordem de 84%, significando uma considerável redução no esforço de desenvolvimento.

A experiência no projeto Tidia-Ae, com apoio da FAPESP (FAPESP, 2010), foi fundamental para a concepção da proposta, possibilitando uma visão do domínio e dos riscos relacionados ao desenvolvimento desse tipo de aplicação. A participação no projeto direcionou esta pesquisa para resolução de problemas concretos e possibilitou apresentar um resultado mais maduro, fruto do desenvolvimento de várias aplicações e da avaliação do uso dessas aplicações por usuários reais.

8.1 Trabalhos Futuros

A LPSCSW é uma linha de produtos restrita à plataforma JAVA. Para aumentar a adoção dos produtos e reúso dos *assets* descritos neste trabalho é fundamental que os mesmos sejam compatíveis com várias plataformas. O Moodle (MOODLE, 2010), por exemplo, é um LMS colaborativo, assim como o Ae, muito popular entre universidades e empresas que oferece facilidades para a comunicação remota. No entanto, esse LMS não possui um conjunto de aplicações de comunicação síncrona. A compatibilidade da LPSCSW com esse LMS, ou qualquer outro ambiente, pode ser atingida por uma arquitetura interoperável de baixo acoplamento. SOA (Service Oriented Architecture) pode servir a esse propósito. Essa arquitetura é um paradigma para execução e manutenção de processos de negócio direcionado a sistemas distribuídos. SOA é baseado em três conceitos tecnológicos: serviços, interoperabilidade e baixo acoplamento (JOSUTTIS, 2007).

A adoção de SOA pode aumentar o potencial de reúso da LPSCSW, pois possibilita que produtos e *assets* não se restrinjam a limites de plataforma, podendo ser interoperáveis com JAVA, PHP, .NET, e etc. Para isso, é necessário disponibilizar os *assets* da linha através de *webservices* e possibilitar a descoberta dos mesmos através de um repositório de serviços. Alguns trabalhos têm sido realizados nesse sentido (GASPAR et al., 2010).

Outra forma de continuação deste trabalho é a generalização dos *assets* e produtos da LPSCSW para dispositivos móveis. A comunicação síncrona multimídia por meio de smartphones, por exemplo, possibilitará a participação em reuniões, aulas, ou comunicação instantânea a qualquer instante e em qualquer lugar. O desenvolvimento de software para dispositivos móveis é caracterizado pela agilidade, visto que o tempo médio de re-

torno de investimentos para a industria de celulares é de 4 meses. Um processo de reúso baseado em metodologias ágeis pode ser de grande valia para diminuição de custos de desenvolvimento e tempo de mercado (*time-to-market*).

Referências

ABREU, F. e; MELO, W. Evaluating the impact of object-oriented design on software quality. In: IEEE. *Proceedings of the 3rd International Software Metrics Symposium: March 25-26, 1996, Berlin, Germany*. [S.l.], 1996. v. 96, p. 90.

ADOBE. *Adobe Flash Media Server products*. May 2009. Http://www.adobe.com/products/flashmediaserver.

ADOBE. *Flash Media Interactive Server*. Jul 2009. Http://www.adobe.com/products/flashmediainteractive/.

ADOBE. *Flash Player Statistics*. Jul 2009. Http://www.adobe.com/products/player_census/flashplayer/.

ADOBE. *FLV/F4V Technology Center*. Jul 2009. Http://www.adobe.com/devnet/flv/.

ADOBE. *Real-Time Messaging Protocol*. May 2009. Http://www.adobe.com/devnet/rtmp/.

ALTINTAS, N. et al. Aurora software product line. In: *Turkish Software Architecture Workshop, Ankara*. [S.l.: s.n.], 2005.

AMERICA, P. et al. Copa; a component-oriented platform architecting method for families of software-intensive electronic products. In: *Tutorial for the First Software Product Line Conference, Denver, Colorado*. [S.l.: s.n.], 2000.

ANASTASOPOULOS, M.; GACEK, C. Implementing product line variabilities. *SOFTWARE ENGINEERING NOTES*, ACM Press; 1999, v. 26, n. 3, p. 109–117, 2001.

BALZERANI, L. et al. A product line architecture for web applications. In: ACM NEW YORK, NY, USA. *Proceedings of the 2005 ACM symposium on Applied computing*. [S.l.], 2005. p. 1689–1693.

BANSIYA, J.; DAVIS, C. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, Published by the IEEE Computer Society, p. 4–17, 2002.

BASILI, V.; BRIAND, L.; MELO, W. A validation of object-oriented design metrics as qualityindicators. *IEEE Transactions on Software Engineering*, v. 22, n. 10, p. 751–761, 1996.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software architecture in practice*. [S.l.]: Addison-Wesley Professional, 2003.

BIEMAN, J. Deriving measures of software reuse in object oriented systems. *Formal Aspects of Measurement*, Citeseer, p. 63–83, 1991.

BOEHM, B. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, ACM New York, NY, USA, v. 11, n. 4, p. 14–24, 1986.

BORGHOFF, U.; SCHLICHTER, J. *Computer-supported cooperative work: Introduction to distributed applications*. [S.l.]: Springer, 2000.

BOZDAG, E.; MESBAH, A.; DEURSEN, A. van. A comparison of push and pull techniques for ajax. *ArXiv e-prints*, jun. 2007.

CATTELAN, R.; BALDOCHI, L.; PIMENTEL, M. Processing and storage middleware support for capture and access applications. In: *Companion Proc. ACM/IFIP/USENIX International Middleware Conference*. [S.l.: s.n.], 2003. p. 315.

CHEN, N.; KINSHUK, K.; LIN, T. Synchronous learning model over the internet. Citeseer, 2004.

CHIDAMBER, S.; KEMERER, C.; MIT, C. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, v. 20, n. 6, p. 476–493, 1994.

CLEMENS, S. *Component Software. Beyond Object-Oriented Programming*. [S.l.]: Addison-Wesley Reading, MA, 1998.

CLEMENTS, P.; NORTHROP, L. et al. *A framework for software product line practice, Version 5.0*. July 2007. Http://www.sei.cmu.edu/productlines/framework.html.

CLEMENTS, P.; NORTHROP, L. M. *Software product lines: practices and patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN 0-201-70332-7.

COHEN, S. *Product Line State of the Practice Report*. [S.l.], 2002.

COMSCORE. *Brazilians' Engagement with Online Multimedia Content Impeded By Lack of Home Broadband Penetration*. Jul 2009. Http://www.comscore.com/Press_Events/Press_Releases/2008/09/Brazil_Broadband_Penetration/(language)/eng-US.

COMSCORE. *YouTube.com Accounted for 1 Out of Every 3 U.S. Online Videos Viewed in January*. Jul 2009. Http://www.comscore.com/Press_Events/-Press_Releases/2008/03/YouTube_Usage.

CRANE, D.; MCCARTHY, P. *Comet and Reverse Ajax: The Next-Generation Ajax 2.0*. [S.l.]: Apress, 2008.

DEITEL, P. J.; DEITEL, H. M. *Ajax, Rich Internet Applications e desenvolvimento Web para programadores*. [S.l.]: Pearson Prentice Hall, 2008.

DEVANBU, P. et al. Analytical and empirical evaluation of software reuse metrics. In: IEEE COMPUTER SOCIETY. *Proceedings of the 18th international conference on Software engineering*. [S.l.], 1996. p. 199.

FAPESP. *Tidia-Ae lança nova plataforma de e-learning*. August 2008. Http://incubadora.fapesp.br/forum/forum.php?forum_id=2136.

FAPESP. *Fundação de Amparo à Pesquisa do Estado de São Paulo*. January 2010. Http://www.fapesp.br/.

FAYAD, M.; SCHMIDT, D. Object-oriented application frameworks. *Communications of the ACM*, ACM, v. 40, n. 10, p. 38, 1997.

FENTON, N.; NEIL, M. Software metrics: successes, failures and new directions. *Journal of Systems and Software*, Elsevier, v. 47, n. 2-3, p. 149–157, 1999.

FIELDING, R.; TAYLOR, R. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, ACM New York, NY, USA, v. 2, n. 2, p. 115–150, 2002.

FLEX. *VisÃ£o geral do Flex*. Jul 2009. Http://www.adobe.com/br/products/flex/overview/.

FOWLER, M. *Inversion of Control Containers and the Dependency Injection Pattern*. July 2009. Http://martinfowler.com/articles/injection.html.

FRAKES, W.; TERRY, C. Software reuse: metrics and models. *ACM Computing Surveys (CSUR)*, ACM, v. 28, n. 2, p. 415–435, 1996.

GAMMA, E. et al. *Design patterns: elements of reusable object-oriented software*. [S.l.]: Addision-Wesley, 1995.

GASPAR, T. C.; PRADO, A. F.; TEIXEIRA, C. A. Linha de produtos de software para colaboração síncrona na web 2.0. In: *Anais do Simpósio Brasileiro de Sistemas Multimídia e Web (Webmedia)*. [S.l.: s.n.], 2009.

GASPAR, T. C.; PRADO, A. F.; TEIXEIRA, C. A. A service oriented approach for synchronous collaborative web 2.0 rias. In: *Anais do Simpósio Brasileiro de Sistemas Multimídia e Web (Webmedia)*. [S.l.: s.n.], 2010.

GASPAR, T. C.; YAGUINUMA, C. A.; PRADO, A. F. Desenvolvimento de aplicações colaborativas síncronas na web 2.0. *Anais do XV Simpósio Brasileiro de Sistemas Multimídia e Web - Minicursos*, v. 3, p. 168–207, 2009.

GWT. *Google Web Toolkit*. Jul 2009. Http://code.google.com/intl/pt-BR/webtoolkit/.

HIBERNATE. *Relational Persistence for Java and .NET*. May 2009. Https://www.hibernate.org/.

HOLDENER, A. T. *Ajax: The Definitive Guide*. [S.l.]: O'Reilly, 2008.

HRASTINSKI, S. Asynchronous & synchronous e-learning. *Educause Quarterly*, EDUCAUSE. 4772 Walnut Street Suite 206, Boulder, CO 80301-2538. Tel: 303-449-4430; Fax: 303-440-0461; e-mail: info@ educause. edu; Web site: http://www. educause. edu, v. 31, n. 4, p. 5, 2008.

ISO/IEC. *9126-1:2001 - Software engineering - Product quality*. 2001. Http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749.

JAVAFX. *JavaFX Overview*. Jul 2009. Http://javafx.com/about/overview/.

JAY, F.; MAYER, R. Ieee standard glossary of software engineering terminology. *IEEE Std*, p. 610–12, 1990.

JOSUTTIS, N. *Soa in practice*. [S.l.]: O'Reilly, 2007. ISBN 9780596529550.

JSF. *JavaServer Faces Technology*. Jul 2009. Http://java.sun.com/javaee/javaserverfaces/.

JSP. *JavaServer Pages Technology*. Jul 2009. Http://java.sun.com/products/jsp/.

JUDE. *JUDE : UML, ER, CRUD, DFD, Flowchart and Mind Map: Design & Modeling Tool*. 03 2010. Http://jude.change-vision.com/jude-web/index.html.

JUNG, H.; KIM, S.; CHUNG, C. Measuring software product quality: A survey of ISO/IEC 9126. *IEEE software*, IEEE Computer Society, p. 88–92, 2004.

KANG, K.; LEE, J.; DONOHOE, P. Feature-oriented product line engineering. *IEEE software*, v. 19, n. 4, p. 58–65, 2002.

KYATERA. *Rede de alta velocidade Kyatera*. May 2009. Http://kyatera.incubadora.fapesp.br/portal.

LEE, K. et al. Combining feature-oriented analysis and aspect-oriented programming for product line asset development. In: *Software Product Line Conference, 2006 10th International*. [S.l.: s.n.], 2006. p. 10.

LI, W.; HENRY, S. Object-oriented metrics that predict maintainability. *Journal of systems and software*, Elsevier, v. 23, n. 2, p. 111–122, 1993.

LINCKE, R.; LUNDBERG, J.; LÖWE, W. Comparing software metrics tools. In: ACM. *Proceedings of the 2008 international symposium on Software testing and analysis*. [S.l.], 2008. p. 131–142.

LINDEN, F. Van der; SCHMID, K.; ROMMES, E. *Software Product Lines in Action the Best Industrial Practice in Product Line Engineering: The Best Industrial Practice in Product Line Engineering*. [S.l.]: Springer, 2007.

LORENZ, M.; KIDD, J. Object oriented metrics. *Editorial Prentice-Hall*, 1994.

MARCA, D.; MCGOWAN, C. *SADT: structured analysis and design technique*. [S.l.]: McGraw-Hill, Inc. New York, NY, USA, 1987.

MARTIN, K. *Developing Applications Using Reverse Ajax*. Jul 2008. Http://today.java.net/pub/a/today/2007/03/22/developing-applications-using-reverse-ajax.html.

MCCABE, T. J. A complexity measure. *IEEE Transactions on Software Engineering*, v. 2, n. 2, p. 308–320, December 1976.

METRICS-1.3.6. *Eclipse Metrics Plugin*. June 2010. Http://metrics.sourceforge.net/.

MICROSYSTEMS, S. *Java Message Service*. May 2009. Http://java.sun.com/products/jms/.

MILI, H.; MILI, F.; MILI, A. Reusing software: Issues and research directions. *IEEE Transactions on Software Engineering*, Published by the IEEE Computer Society, p. 528–562, 1995.

MOODLE. *Moodle.org: open-source community-based tools for learning*. 2010. Http://moodle.org/.

MORAES, C. et al. *Uma Ferramenta de Chat com Ajax e Spring para um Ambiente de Aprendizado Eletrônico*. [S.l.]: WebMedia, 2008.

MURPHY, E.; LAFERRIERE, T. Adopting tools for online synchronous communication: Issues and strategies. *Making the Transition to E-learning: Strategies and Issues*, Information Science Publishing, p. 318, 2007.

NARASIMHAN, V.; HENDRADJAYA, B. Theoretical considerations for software component metrics. In: CITESEER. *Proceedings of World Academy of Science, Engineering and Technology*. [S.l.], 2005. v. 10, p. 165–170.

NG, K. Replacing face-to-face tutorials by synchronous online technologies: Challenges and pedagogical implications. *International Review of Research in Open and Distance Learning*, v. 8, n. 1, 2007.

NORTHROP, L. M. Sei's software product line tenets. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 19, n. 4, p. 32–40, 2002. ISSN 0740-7459.

ORACLE. *Oracle Top Link*. June 2009. Http://www.oracle.com/technology/products/ias/toplink/index.html.

OREILLY, T. *What is Web 2.0: Design patterns and business models for the next generation of software*. September 2005. Http://oreilly.com/web2/archive/what-is-web-20.html.

OTTER, A. den; EMMITT, S. Exploring effectiveness of team communication: Balancing synchronous and asynchronous communication in design teams. *Engineering, Construction and Architectural Management*, Emerald Group Publishing Limited, v. 14, 2007.

PRIETO-DIAZ, R. Implementing faceted classification for software reuse. *Communications of the ACM*, ACM, v. 34, n. 5, p. 88–97, 1991.

RED5. *Red5: Open Source Flash Server Open Source Flash*. May 2009. Http://osflash.org/red5.

RICHFACES. *RichFaces provides out-of-the-box "richness"for JSF Web applications*. Jul 2009. Http://jboss.org/jbossrichfaces/.

ROSENBERG, L.; HYATT, L. Software Quality Metrics for Object-Oriented Environments. *Crosstalk Journal, April*, Citeseer, 1997.

SAKAI. *Sakai Collaboration and Learning Environment (CLE)*. June 2009. Http://sakaiproject.org/portal.

SHOTSBERGER, P. The human touch: Synchronous communication in web-based learning. *Educational Technology*, v. 40, n. 1, p. 53–56, 2000.

SILVERLIGHT. *Silverlight powers rich application experiences wherever the Web works*. Jul 2009. Http://www.microsoft.com/silverlight.

SINGH, M. Write asynchronous, run synchronous. *IEEE Internet computing*, IEEE Computer Society, v. 3, n. 2, p. 4–5, 1999.

SPRING. *Spring Source Community*. Jul 2009. Http://www.springsource.org/.

SUMMERVILLE, I. *Software Engineering, 8-th Edition*. [S.l.]: Addison-Wesley, 2006.

SWAT, C. *Analyst4J - Product Overview*. June 2010. Http://www.codeswat.com/cswat/index.php?option=com_content&task=view&id=43&Itemid=63.

SYSTEMS, W. M. *Flash Media Server*. Jul 2009. Http://www.wowzamedia.com/.

TELECO. *Market Share de Banda Larga no Brasil*. Jul 2009. Http://www.teleco.com.br/blarga.asp.

TIDIA-AE. *Tecnologia da Informação para o Desenvolvimento da Internet Avançada - Aprendizado Eletrônico*. May 2009. Http://tidia-ae.incubadora.fapesp.br/portal.

VOELTER, M.; GROHER, I. Product line implementation using aspect-oriented and model-driven software development. In: *Software Product Line Conference, 2007. SPLC 2007. 11th International*. [S.l.: s.n.], 2007. p. 233–242.

WANG, W. Powermeeting: gwt-based synchronous groupware. In: ACM NEW YORK, NY, USA. *Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*. [S.l.], 2008. p. 251–252.

WASHIZAKI, H.; YAMAMOTO, H.; FUKAZAWA, Y. A metrics suite for measuring reusability of software components. In: CITESEER. *Proceedings of the 9th International Symposium on Software Metrics*. [S.l.], 2003. p. 211.

WEISS, D.; LAI, C. *Software product-line engineering: a family-based software development process*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

ZK. *ZK - Direct RIA*. Jul 2009. Http://www.zkoss.org/.

ZKDEMO. *Spreadsheet 1.0.0 RC2 Live Demo*. Jul 2009. Http://zssdemo.zkoss.org/.