Tese apresentada à Pró-Reitoria de Pós-Graduação e Pesquisa do Instituto Tecnológico de Aeronáutica, como parte dos requisitos para obtenção do título de Mestre em Ciências no Curso de Pós-Graduação em Engenharia Eletrônica e Computação, na Área de Informática.

Mauro Carlos Pichiliani

MAPEAMENTO DE SOFTWARE PARA PERMITIR A COLABORAÇÃO SÍNCRONA

Tese aprovada em sua versão final pelos abaixo assinados:

Prof. Dr. Celso Massaki Hirata
Orientador

Prof. Dr. Homero Santiago Maciel
Pró-Reitor de Pós-Graduação e Pesquisa

Campo Montenegro
São José dos Campos, SP - Brasil
2006

Mapeamento de Software para Permitir Colaboração Síncrona

Mauro Carlos Pichiliani

Composição da Banca Examinadora:

Prof. Dr. Clóvis Torres Fernandes	Presidente - ITA
Prof. Dr. Celso Massaki Hirata	Orientador - ITA
Prof. Dr. Edgar Toshiro Yano	Membro - ITA
Prof. Dr. Adilson Marques da Cunha	Membro - ITA
Prof. Dr. Hugo Huks - PUC/RJ	Membro Externo

ITA

Dedico este trabalho à memória de minhas avós Alice Ferreira do Santos Pichiliani e Maria Coelho Curvello, que em vida, sempre me agraciaram com seu afeto.

Agradecimentos

Ao Prof. Hirata, meu orientador, pela confiança, paciência e compreensão dedicada em todos os momentos. Sua orientação foi fundamental para tornar este trabalho uma realidade.

À toda minha família, em especial aos meus pais e irmãos, que sempre acreditaram no meu sucesso e criaram todas as condições para que eu pudesse alcançar meus objetivos.

Aos professores do ITA, em especial aqueles que ministraram disciplinas da pós-graduação durante a minha formação: Nei Yoshihiro Soma, Vakulathil Abdurahiman, Adilson Marques da Cunha, Felipe Afonso de Almeida e Edgar Toshiro Yano.

Aos amigos da pós-graduação que conheci no ITA neste período e que sem eles não seria possível a realização deste trabalho: Alexandre, Alysson, Celeny, Daniel, Davi, Diógenes, Elisa, Fabricio, Giuliano, Joubert, Luiz, Marck, Michele, Maria Luisa, Osvandre, Pascally, Regina, Tatiane e Vera.

Resumo

Atualmente, devido aos altos custos necessários para estabelecer reuniões presenciais e às novas formas de comunicação como correio eletrônico, conferência por telefone, vídeo conferências e listas de discussão, as empresas têm procurado reduzir a quantidade de reuniões presenciais. Para este objetivo, a colaboração remota auxiliada por aplicações de *groupware* apresenta-se como uma alternativa à realização de tarefas compartilhadas. Contudo, a disponibilidade atual de aplicações *groupware* é limitada a poucos domínios de problema, o que dificulta o trabalho colaborativo. Adicionalmente, existe um grande número de aplicações não colaborativas que poderiam ser transformadas em aplicações *groupware*, beneficiando-se das vantagens deste tipo de aplicação. Neste trabalho, apresenta-se um mapeamento dos principais componentes de aplicações não colaborativas baseadas no estilo arquitetural MVC (Model-View-Controller) para componentes de uma aplicação colaborativa com o objetivo de permitir a colaboração remota entre participantes. Usando o mapeamento, aplicações existentes podem ser estendidas para apoiar a colaboração síncrona durante a elaboração de tarefas compartilhadas. Para exemplificar a aplicação do mapeamento, a ferramenta CASE (Computer Aided Software Engineering) ArgoUML foi estendida de acordo com o mapeamento para apoiar a elaboração colaborativa de diagramas da UML (Unified Modeling Language) através da Internet. O protótipo construído, denominado CoArgoUML, não fornece apenas o compartilhamento do diagrama entre os usuários, mas também um meio de comunicação, dispositivos de percepção e um mecanismo para controlar o acesso concorrente aos elementos do diagrama. Uma experiência foi realizada em um ambiente controlado para avaliar a satisfação dos usuários com o protótipo construído. Os resultados desta experiência indicam que o protótipo atendeu aos requisitos especificados.

Abstract

Due to high costs demanded to accomplish physical meetings and the new ways of communication such as electronic mail, conference calls, video conferences, and discussion lists, companies are trying to reduce the occurrence of physical meetings. To this goal the remote collaboration provided by groupware applications are presented as an alternative to perform collaborative tasks. However, the availability of groupware applications is limited to few problem domains, which make the remote collaboration more difficult. Additionally there is a huge number of non collaborative applications that can be leveraged to groupware applications, achieving their benefits. This work presents a mapping from the main components of non collaborative applications based on the MVC architectural style to components of collaborative applications in order to provide a remote collaboration among participants. Using this mapping, existing applications can be extended to support synchronous collaboration during the elaboration of shared tasks. In order to exemplify the use of the mapping, a CASE (Computer Aided Software Engineering) tool called ArgoUML is extended according to the mapping to support the collaborative elaboration of UML (Unified Modeling Language) diagrams through the Internet. The prototype built, called CoArgoUML, not only allows the sharing of the diagrams among the users, but also provides a communication tool, awareness devices, and a mechanism to control the concurrent access to the diagram's elements. An experiment was conducted on a controlled environment to evaluate the user's satisfaction with the prototype. The results of the experiment indicate that the prototype comply with the specified requirements.

Sumário

Lista de ilustrações

Lista de Tabelas

Lista de abreviaturas e Siglas

ACT: Arquitetura de Colaboração Transparente.

API: Application Programming Interface.

BSCW: Basic Support for Cooperative Work.

CASE: Computer Aided Software Engineering.

CES: Collaborative Editing Systems.

CSCW: Computer-Supported Cooperative Work.

DVD: Digital Video Disc.

GEF: Graph Edition Framework.

I-CASE: Integrated CASE.

ICT: Intelligent Collaboration Transparency.

IDE: Integrated Development Environment.

ITA: Instituto de Tecnologia da Aeronáutica.

JRE: Java Runtime Environment.

MVC: Model-View-Controller.

MOF: Meta-Object Facility.

OCL: Object Constraint Language.

OMG: Object Management Group.

PAC: Presentation, Abstraction, Controller.

PGML: Portable Graphics Markup Language.

RUP: Rational Unified Process.

SATC: Sistemas de Apoio ao Trabalho Cooperativo.

SVG: Scalable Vectorial Graphics.

UML: Unified Modeling Language.

VOIP: Voz Sobre IP.

XML: eXtensible Markup Language.

XP: Extreme Programming.

XMI: XML Metadata Interchange.

WYGIWIG: What You Get Is What I Get.

WYSIAWIS: What You See Is Almost What I See.

WYSIWID: What You See Is What I Did.

WYSIWIS: What You See Is What I See.

WYSIWWS: What You See Is What We Share.

WWW: World Wide Web.

Capítulo I

Introdução

Este capítulo apresenta a motivação por trás da realização deste trabalho, evidenciando o problema de pesquisa em questão. Em seguida os objetivos são definidos, seguidos das contribuições pretendidas a partir dos resultados alcançados com este trabalho. Por fim, apresenta-se um breve resumo do conteúdo de cada um dos capítulos e apêndice contidos neste trabalho.

1.1 Motivação

A crescente busca por maior competitividade no mercado empresarial tem levado as empresas a procurar por alternativas para reduzir seus custos operacionais envolvendo as tarefas executadas no dia a dia. Cada vez mais as informações e responsabilidades tornam-se compartilhadas para obtenção dos seus objetivos de negócio, em especial nas empresas que possuem projetos de desenvolvimento de software. As tarefas estão se tornando responsabilidades de um grupo de participantes, em vez de centradas um único indivíduo, permitindo sua execução mais consistente e com menos riscos.

Durante a execução de uma tarefa compartilhada, os responsáveis são solicitados a interagir entre si para completá-la. Para promover interações em tarefas compartilhadas é comum se realizar reuniões presenciais onde os envolvidos se encontram fisicamente para realizar as operações referentes à tarefa. As operações podem incluir apresentação da tarefa, discussão sobre a forma de execução da tarefa, elaboração conjunta de documentos, verificação e validação dos documentos gerados e atribuição de responsabilidades. Além do aspecto operacional, as reuniões presenciais

também possuem um aspecto social, característico de encontros face a face facilitados pela comunicação pessoal.

Porém, nem sempre este tipo de reunião presencial pode acontecer, devido à quantidade de recursos necessários para elaborar uma reunião presencial. Disponibilidade dos participantes para viagens e disponibilidade de espaço físico adequado são apenas alguns exemplos de fatores que podem determinar o cancelamento de reuniões. O cancelamento de reuniões pode desperdiçar recursos já alocados e frustrar os demais participantes preparados para o encontro.

Deste modo, as empresas devem explorar alternativas à reunião presencial dos participantes envolvidos em uma tarefa compartilhada. A vídeo conferência é uma das alternativas adotadas pelas empresas para reduzir custos com reuniões presenciais. Entretanto, ela fornece apenas imagem e áudio dos participantes remotos, sem apresentar recursos para a colaboração na execução da tarefa, como o trabalho simultâneo em documentos, compartilhamento dos documentos, armazenamento das interações dos usuários e monitoramento das sessões colaborativas.

Outra possibilidade para se evitar reuniões presenciais é a colaboração e cooperação remota auxiliada por aplicações computacionais. O emprego destas aplicações visa diminuir a necessidade de reuniões presenciais com os envolvidos em alguma tarefa compartilhada e prover funcionalidades específicas à tarefa a ser executada colaborativamente. Um exemplo de tarefa compartilhada que pode ser beneficiada pelo de uso uma aplicação que suporte a colaboração e a coordenação é a modelagem de diagramas por meio de uma ferramenta CASE (*Computer Aided Software Engineering*) de modelagem, utilizada na geração de artefatos durante as fases do processo de desenvolvimento de software.

A área que estuda como as pessoas trabalham em grupo e como o computador pode ajudar no apoio a este grupo é conhecida como CSCW (*Computer-Supported Cooperative Work*). Esta área surgiu na década de oitenta e seu estudo é multifacetado, incluindo conceitos de outras áreas, como as área de psicologia, sociologia e educação, para apoiar o estudo de componentes organizacionais, individuais e de dinâmica de grupo.

A Internet, e em particular *WEB*, permite um compartilhamento de informações numa forma simples através do uso de uma estrutura descentralizada e

independente. No entanto, mesmo com o uso da Internet, é necessário vencer as barreiras impostas pela distância geográfica entre as equipes que desejam trabalhar de forma colaborativa.

Segundo Prikladnicki [63], vários ambientes e aplicações têm sido desenvolvidos, ao longo dos últimos anos, para ajudar na comunicação, coordenação e colaboração entre equipes distribuídas e conectadas pela Internet. No entanto, muitas destes ambientes e aplicações são restritos a apenas alguns domínios de problema.

Esta restrição se deve à grande demanda por recursos que viabilizem o desenvolvimento de novos ambientes e aplicações com suporte à comunicação, coordenação e colaboração. São exemplos destes recursos: ferramentas específicas, mão de obra especializada e a mudança no foco de desenvolvimento de aplicações com interação monousuária para aplicações com interação multiusuária. Em contrapartida, modificar aplicações com interação monousuário para que elas suportem a interação multiusuária demanda uma quantidade menor de recursos do que criar uma aplicação com a interação multiusuária a partir do zero [9].

De acordo com Xia et al. [73], estender aplicações de interação monousuária para que elas permitam colaboração entre múltiplos usuários possui o potencial de aumentar significativamente a disponibilidade de aplicações colaborativas e melhorar a usabilidade dessas aplicações. Durante a última década, alguns esforços na área de CSCW contribuíram para a redução da complexidade necessária para estender e adaptar aplicações de interação monousuárias, com o objetivo de apoiar a colaboração. As abordagens mais significativas envolvem a substituição e adaptações de componentes, indicadas para alterar aplicações existentes sem mudar seus componentes internos. Porém, estas abordagens não se concentram em modificações gerais, visando os estilos arquiteturais das aplicações.

Sendo assim, o problema de pesquisa deste trabalho é estender aplicações não colaborativas, que seguem o estilo arquitetural MVC, para permitir a colaboração síncrona, utilizando a Internet como rede de interconexão.

1.2 Objetivos

A solução proposta para o problema de pesquisa deste trabalho é um mapeamento que permite a extensão de aplicações não colaborativas para apoiar a colaboração síncrona, utilizando a Internet como rede de interconexão. Para aplicar este mapeamento, é necessário que a aplicação siga o estilo arquitetural MVC (*Model*, *View*, *Controller*). Esse estilo arquitetural separa os dados da aplicação (o modelo), do código de tratamento de dados (o controlador) e também da interface de visualização (a visão). O mapeamento é feito entre os componentes originais da aplicação e os componentes colaborativos, com base nos requisitos especificados.

Para aplicar o mapeamento de componentes é necessário estender e adaptar alguns componentes da aplicação existente. A modificação de componentes de uma aplicação é uma tarefa que requer o código fonte dos mesmos, pois o mapeamento indica quais componentes devem ser modificados para apoiar os novos aspectos funcionais da aplicação colaborativa.

As aplicações alvo do mapeamento, isto é, as aplicações computacionais cujos componentes podem ser mapeados para que suportem a colaboração, devem seguir o estilo arquitetural MVC e também possuir o código fonte disponível. Para se obter o conjunto de todas as aplicações alvo é preciso obter primeiro um subconjunto de todas as aplicações que seguem o estilo arquitetural MVC. Deste subconjunto de aplicações, obtém-se um novo subconjunto contendo apenas as aplicações que seguem o estilo arquitetural MVC e que também possuem o código livre, uma vez que o mapeamento implica na modificação dos componentes. Deste modo, a principal classe de aplicações beneficiada pelo mapeamento é a classe de aplicações de código livre, fortemente apoiada por uma grande comunidade de usuários e desenvolvedores.

Este trabalho inclui também o desenvolvimento de um protótipo por meio da aplicação do mapeamento proposto, a implementação de um mecanismo de controle de concorrência distribuído com níveis de controle de travas, a coleta de dados durante a utilização do protótipo e análises qualitativas e quantitativas dos dados coletados. Este protótipo permite a validação parcial do mapeamento proposto, pois ele é um estudo de caso da implementação do mapeamento.

1.3 Contribuições Pretendidas

Com a aplicação deste mapeamento, aplicações existentes podem ser estendidas para apoiar a colaboração síncrona, durante a elaboração de artefatos criados em um projeto de software, estendendo suas funcionalidades e, possivelmente, aumentando seu grau de usabilidade.

O mapeamento proposto fornece um guia com recomendações para a modificação dos componentes da aplicação, sem apresentar nenhum detalhe técnico de implementação. Utilizando este mapeamento, aplicações não colaborativas, construídas de acordo com o estilo arquitetural MVC, podem ser beneficiadas pelo mapeamento, tornando-se colaborativas. Uma vez que existe uma grande variedade de aplicações que se enquadram nesta categoria, o mapeamento pode ajudar os desenvolvedores a implementar funcionalidades colaborativas em suas aplicações, expandindo os benefícios do uso de aplicações colaborativas.

O protótipo apresenta recursos de certa forma limitados quando comparado com outros trabalhos, porém contêm uma combinação única de características que, se aperfeiçoadas de maneira adequada, oferece uma solução bastante poderosa para a edição colaborativa de diagramas UML (*Unified Modeling Language*). Entre as características que podem ser citadas neste sentido, têm-se as seguintes:

- A criação de dispositivos que facilitem a percepção e a comunicação do trabalho cooperativo: um *chat* para a troca de mensagens entre os participantes, *telepointers* para permitir a percepção remota e uma lista de elementos travados para cada participante;
- A implementação de uma estrutura de controle de concorrência distribuída de vários níveis, que garante a sincronia entre os diversos participantes;
- A capacidade de compartilhamento de diagramas UML por usuários dispersos geograficamente e conectados na Internet; e
- O controle da sessão colaborativa e o armazenamento remoto do diagrama compartilhado por meio de um servidor de colaboração.

A verificação da satisfação dos usuários com o protótipo, por meio da experiência controlada, fornece dados empíricos para a análise e avaliação dos resultados obtidos. As análises dos resultados desta experiência também apresentam

uma alternativa para a avaliação quantitativa de trabalhos efetuados colaborativamente. Esta análise da satisfação dos usuários também é utilizada para validar parcialmente a conformidade com os requisitos colaborativos especificados e que foram implementados no protótipo.

1.4 Resumo dos Próximos Capítulos

Uma breve descrição dos próximos capítulos é apresentada a seguir.

O Capítulo 2 reúne os conceitos necessários para se entender o contexto desta tese. Nele, os principais conceitos da área de trabalho cooperativo apoiado por computador e *groupware* são apresentados seguidos de uma pequena revisão sobre UML e ferramentas CASE. Este capítulo segue apresentando os conceitos que envolvem o Desenvolvimento Baseado em Componentes, os *frameworks*, os padrões de projetos e as arquiteturas de software. Por fim, as abordagens existentes permitindo que aplicações não colaborativas apóiem colaboração síncrona entre vários usuários são apresentadas.

No Capítulo 3, o mapeamento de componentes é apresentado com base nos componentes genéricos de uma aplicação que segue o estilo arquitetural MVC.

O Capítulo 4 apresenta a estrutura de funcionamento da ferramenta de código livre ArgoUML e descreve como o mapeamento proposto foi implementado nesta ferramenta para a criação do protótipo chamado CoArgoUML. Este capítulo apresenta ainda uma seção sobre o uso da abordagem proposta e uma comparação desta abordagem com os trabalhos correlatos.

No Capítulo 5, uma experiência controlada é apresentada. O ambiente, os usuários, as tarefas e algumas observações encontradas durante a experiência são descritos e detalhados.

No Capítulo 6, algumas análises dos dados coletados durante a experiência são apresentadas em conjunto com as variáveis observadas e principais resultados obtidos.

O Capítulo 7 apresenta as conclusões desta tese junto com as principais limitações encontradas, ressaltando as contribuições mais importantes e uma lista de trabalhos futuros.

O Apêndice A contêm os questionários de perfil, avaliação de esforço e avaliação final utilizados na coleta de dados antes, durante e depois da experiência.

O Apêndice B apresenta trechos dos registros de manipulação de elementos e comunicação armazenados pelo protótipo durante a elaboração dos diagramas na experiência.

O Apêndice C mostra alguns diagramas resultantes da elaboração colaborativa dos usuários na experiência comentados por especialistas em modelagem de diagramas UML.

Capítulo II

Contexto do Trabalho

Este capítulo apresenta os principais tópicos usados na elaboração deste trabalho. O capítulo começa apresentando os conceitos, classificações e características funcionais da área de trabalho cooperativo apoiado por computador e *groupware*. Em seguida, a UML é apresentada junto com seus principais diagramas. O capítulo segue discorrendo sobre ferramentas CASE e apresenta os conceitos que envolvem o Desenvolvimento Baseado em Componentes, os *frameworks* e os padrões de projeto. Uma breve revisão sobre estilos arquiteturais de software utilizados em aplicações é apresentada na seqüência. Por fim, o capítulo descreve as abordagens atuais para construir aplicações colaborativas e adaptar aplicações existentes para apoiar a colaboração.

2.1 Trabalho Cooperativo Apoiado por Computador

Nesta seção os conceitos de trabalho cooperativo apoiado por computador são apresentados juntamente com a definição de *groupware*. Em seguida, algumas classificações e as principais características funcionais de *groupware* são apresentadas.

2.1.1 *CSCW* e *Groupware*

À medida que os computadores pessoais se tornam mais eficazes e as redes de computadores mais abrangentes e mais rápidas, os computadores passam a ser usados não só em tarefas simples e individuais, mas também na comunicação e no auxílio ao trabalho em conjunto com outras pessoas. Reuniões, conversas telefônicas,

apresentações e encontros informais, entre outros, são atividades em grupo comuns necessárias na realização efetiva de trabalho. Essas atividades, em geral, ocupam grande parte do tempo das pessoas, tanto na preparação e execução como no deslocamento dos envolvidos.

A área de Trabalho Cooperativo Apoiado por Computador, abreviada pela sigla CSCW (*Computer-Supported Cooperative Work*) estuda como as pessoas trabalham em grupo e os meios de apoiá-las sob vários aspectos com a intenção de aumentar a produtividade do grupo. Alguns autores brasileiros preferem utilizar a sigla SATC (Sistemas de Apoio ao Trabalho Cooperativo), em vez de CSCW. Neste trabalho, dá-se preferência à sigla CSCW, devido ao seu extenso uso pelos pesquisadores da área.

Ellis et al. [10] definem CSCW como a área que observa como os grupos trabalham e procura descobrir como as tecnologias, especialmente as de computação, podem ajudá-los a trabalhar. Esta área surgiu na década de oitenta e seu estudo é multifacetado, incluindo conceitos de da Psicologia, Sociologia e Educação, para apoiar o estudo de componentes organizacionais, individuais e de dinâmica das interações do grupo. Na definição de Ellis, a área de CSCW envolve também aspectos sociais e cognitivos.

Enquanto a sigla CSCW tem sido usada para representar a pesquisa na área do trabalho cooperativo e a maneira pela qual o computador pode auxiliar os grupos na execução de suas atividades, o termo *groupware* tem sido usado para designar o hardware ou software gerados pela pesquisa em CSCW. Apesar disso, algumas vezes, tanto a sigla CSCW como o termo *groupware* são usados como sinônimos de trabalho cooperativo auxiliado por computador. Desta maneira, os produtos comerciais e outros softwares para auxiliar no trabalho cooperativo são normalmente referenciados como exemplos de *groupware*.

Segundo Ellis et al. [10], *groupwares* são sistemas baseados em computadores que apóiam grupos de pessoas engajadas em uma tarefa ou objetivo comum e que provêm uma interface para um ambiente compartilhado. O principal objetivo de um *groupware* é auxiliar os grupos no que diz respeito à realização de tarefas cooperativas através de aspectos da comunicação, colaboração e coordenação de suas atividades, quando são utilizadas tecnologias computacionais. Alguns exemplos de sistemas *groupware:* Sistemas de correio eletrônico, vídeo conferências, salas de reuniões eletrônicas e editores de texto colaborativos.

O conceito de *groupware* também está relacionado com o trabalho em grupo, mais especificamente em como desenvolver técnicas que forneçam apoio às formas nas quais as pessoas se comunicam e colaboram para atingir as metas do trabalho no contexto pessoal, gerencial ou organizacional. Em comparação com as aplicações não colaborativas, os sistemas *groupware* fornecem apoio às tarefas compartilhadas, por meio da colaboração entre os usuários.

As empresas, com suas necessidades por eficiência e resultados rápidos, têm demonstrando grande interesse em sistemas *groupware*, na última década, impulsionando o desenvolvimento desses sistemas e demandando soluções eficazes, de modo a propiciar maior competitividade [2].

Devido à extensa pesquisa na área, diversas características próprias podem ser encontradas nos sistemas *groupware*. Devido a essas características, os sistemas *groupware* podem ser classificados de diversas formas. A próxima subseção apresenta algumas dessas classificações.

2.1.2 Classificações em CSCW

A área de CSCW possui diversas formas para classificar os sistemas *groupware*. Os diferentes critérios utilizados em cada classificação permitem que a pesquisa na área seja dividida, facilitando a concentração de esforços de pesquisa em tipos específicos de aplicações.

Os critérios de classificação envolvem conceitos como tempo, espaço, tamanho do grupo, funcionalidades da aplicação e tipos de interface. A maioria dos critérios e classificações foi definida pelo trabalho seminal de Ellis et al. [10]. Posteriormente os critérios e classificações foram incrementados pontualmente por vários outros autores no decorrer da década de noventa.

A classificação de sistemas *groupware* mais aceita e conhecida na área foi proposta por Ellis et al. [10] e propõe que os critérios de tempo e espaço sejam analisados para identificar a qual grupo as aplicações devem pertencer. Os possíveis valores para estes critérios são dispostos em uma matriz de classificação de *groupwares*, apresentada na Figura 2.1.

	Mesmo Tempo	Tempo Diferente
Mesmo Local	Interação Síncrona (face a face)	Interação Assíncrona
Local Diferente	Interação Síncrona Distribuída	Interação Assíncrona Distribuída

Figura 2.1 - Matriz Espaço x Tempo proposta por Ellis et al. Traduzido de [10].

De acordo com a matriz apresentada na Figura 2.1, os sistemas *groupware* podem ser classificados em quatro categorias distintas, obtidas pela combinação dos valores das dimensões. Uma das dimensões considera o critério espaço e refere-se à localização física dos usuários, que pode assumir o valor “Mesmo Local” ou “Local Diferente”. A outra dimensão da matriz considera o critério de tempo, tratando do momento em que os participantes trabalham, que se divide em “Mesmo Tempo”, para representar interações síncronas, e “Tempo Diferente”, para representar interações assíncronas.

Um outro critério também freqüentemente utilizado na área de CSCW para classificar as interações dos usuários, no que diz respeito tanto à percepção de eventos da aplicação como à comunicação, divide as interações dos usuários com as aplicações tradicionalmente em dois modos: assíncrona e síncrona.

Na interação assíncrona, não existe o uso da simultaneidade no tempo dos eventos gerados por um usuário, de modo que a interação acontece por meio de recipientes que armazenam os eventos para notificar posteriormente a ocorrência destes eventos a outros usuários. Neste modo, os usuários trabalham sobre os mesmos objetos em tempos diferentes. Modificações nos objetos compartilhados são encaminhadas imediatamente a todos os membros e são observadas com algum atraso até o momento em que os membros se reconectem ao sistema.

Na interação síncrona, existe a noção de simultaneidade no tempo, pois eventos gerados por um usuário são imediatamente notificados para os outros usuários. Neste modo, os usuários trabalham ao mesmo tempo com os mesmos objetos. Modificações em um objeto compartilhado são encaminhadas automaticamente e observadas por outros usuários.

Os sistemas *groupware* são agrupados em cada uma das quatro categorias definidas na Figura 2.1. Como exemplo de classificação de aplicações de acordo com esta matriz, a categoria onde os valores dos critérios espaço e tempo assumem os valores “Local Diferente” x “Mesmo Tempo”, respectivamente, incluem as aplicações de vídeo conferência e ensino à distância.

Além dos dois modos de classificação das interações dos usuários tradicionais, Molli et al. [59] sugerem um terceiro modo chamado de modo multisíncrono. Neste modo, cada usuário possui uma cópia dos objetos compartilhados que podem ser modificados em paralelo. Este modo alterna a interação dos usuários entre fases de divergência e convergência. Durante as fases de divergência, cada usuário trabalha isoladamente e, durante as fases de convergência, os usuários sincronizam suas cópias distintas para restabelecer uma visão comum dos objetos. Atividades individuais posteriores causam novamente a divergência, necessitando novamente de sincronização e assim por diante.

Por ser uma definição posterior às classificações tradicionais, o modo de interação multisíncrona não é considerado nas principais classificações de sistemas *groupware*.

Uma modificação na matriz de espaço versus tempo, proposta por Grudin et al. [40], inclui novos valores para os critérios de tempo e espaço. Estes valores representam a característica de previsibilidade de tempo e espaço, dividindo os valores das dimensões e inserindo uma nova linha e uma nova coluna na matriz original.

Na matriz proposta por Grudin o valor da coluna “Tempo Diferente” dividiu-se entre nos valores “Tempo Diferente, mas Previsível”, indicando a possibilidade de prever o momento em que os participantes estarão ativos, e no valor “Tempo Diferente e Imprevisível”, indicando a impossibilidade de previsão temporal. A coluna “Local Diferente” também é dividida no valor “Local Diferente, mas Previsível”, para quando é possível prever o local onde estará cada participante, e no valor “Local Diferente e Imprevisível”, para indicar quando não é possível afirmar qual será a localização física dos participantes. A Figura 2.2, traduzida de [40], apresenta a matriz tempo versus espaço modificada para acomodar os novos valores dos critérios tempo e espaço.

	Mesmo Tempo	Tempo Diferente, Mas Previsível	Tempo Diferente e Imprevisível
Mesmo Local	Sala de encontro	Deslocamentos de trabalho	Salas de equipes
Local diferente, Mas Previsível	Conferência eletrônica de vídeo e destkop	Correio eletrônico	Escrita Colaborativa
Local diferente e Imprevisível	Seminários interativos	Quadro de anúncios	Workflow

Figura 2.2 - Classificação de sistemas *groupware* de acordo com o Tempo, o Espaço e a Previsibilidade. Traduzida de [40].

Baseado na matriz original de espaço versus tempo, Nunamaker [35] sugere a inserção de uma nova dimensão como critério de classificação com o objetivo de considerar a quantidade de indivíduos envolvidos na utilização do sistema. Argumentando que o tamanho do grupo vai determinar uma série de cuidados para que o apoio à comunicação e ao compartilhamento de informações seja adequado, mesmo para grandes grupos dispersos, a nova dimensão, chamada de Tamanho do Grupo, indica a classificação levando em consideração a quantidade de pessoas que utilizam o sistema.

A primeira dimensão da nova matriz, como na classificação original, atribui os valores ”Mesmo Tempo” e “Tempo Diferente” para a dimensão Tempo. A segunda dimensão, denominada Espaço ou Proximidade do grupo, é dividida entre os valores “Indivíduo distribuído”, “Grupo distribuído” e “Indivíduo distribuído em Grupo”. A última dimensão, chamada Tamanho do Grupo, é dividida entre os valores “de 3 a 7 pessoas” e “de 7 a n pessoas”. Dias [52] (apud Nunamaker [35]) apresenta a matiz tridimensional utilizada na classificação proposta por Nunamaker, apresentada na Figura 2.3.

Os critérios de tempo, espaço, e tamanho do grupo utilizados para classificar os sistemas *groupware* fornecem informações gerais sobre as propriedades das aplicações, mas não classificam as funcionalidades e propósitos específicos das mesmas, como em um catálogo que agrupa produtos de acordo com suas funcionalidades.

Ellis et al. [10] também propõem uma classificação de acordo com as funcionalidades de um sistema *groupware*, onde um mesmo sistema pode ser

classificado em mais de uma categoria devido às suas diversas funcionalidades. Esta classificação tem o objetivo apresentar uma idéia geral dos diferentes tipos de funcionalidades abordados pelos sistemas *groupware*. Outras classificações abrangem critérios de funcionalidade para propósitos mais específicos, como a classificação de Grudin [39], que aborda com maior ênfase os sistemas de *workflow*.

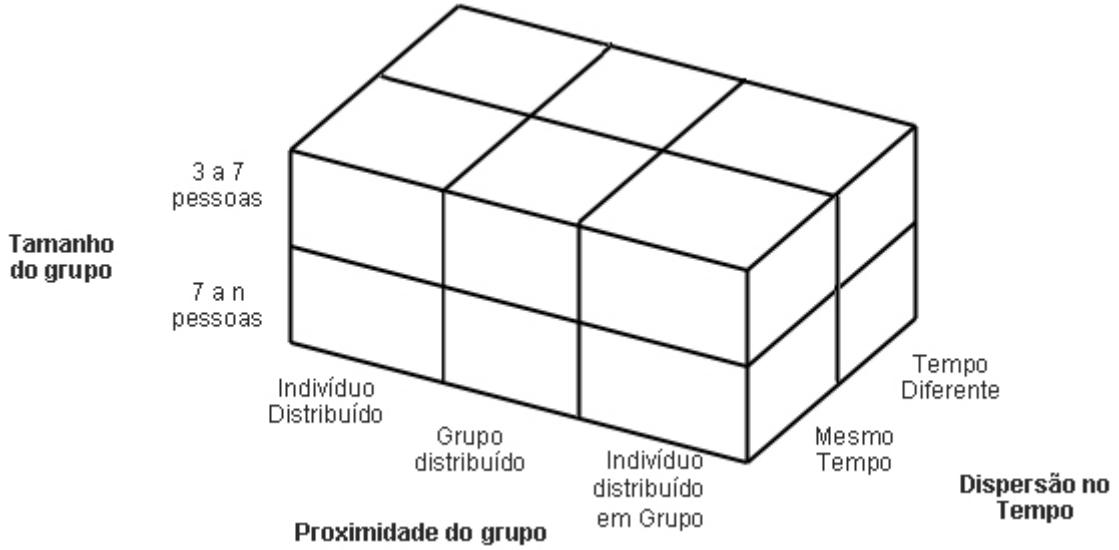

Figura 2.3 - Classificação de sistemas *groupware* de acordo com o Tempo, o Espaço e o Tamanho do Grupo [52].

De acordo com as funcionalidades levantadas por Ellis, as seguintes classes de aplicações foram definidas:

Sistemas de Mensagens - Este tipo de aplicação permite a troca assíncrona de mensagens textuais entre grupos de usuários. Os sistemas de correio eletrônico junto com os murais e fórum eletrônicos são os principais representantes desta categoria.

Editores MultiUsuários - Os editores de interação multiusuária permitem a composição e edição colaborativa de um documento compartilhado, dividindo o documento em partes para facilitar a edição concorrente de vários usuários. Estes editores também fazem, de forma transparente aos usuários, o controle de acesso e a sincronização entre os segmentos. Somente alguns exemplos deste tipo de aplicações adotam um modelo de notificação, como o *Basic Support for Cooperative Work* (BSCW) [68].

Sistemas de Apoio à Decisão do Grupo (*Group Decision Support Systems*) e Salas de Reunião Eletrônicas - Estes sistemas exploram problemas não estruturados em

um grupo, por meio de computadores, e junto com as salas de reunião eletrônicas objetivam aumentar a produtividade de reuniões para a tomada de decisão, através da aceleração deste processo ou do aumento da qualidade da decisão tomada [13].

Conferências - Estes sistemas são subdivididos em: conferências em tempo real, que permitem a interação síncrona durante uma reunião virtual de usuários dispersos geograficamente; teleconferências, que apóiam as interações do grupo por meio de telecomunicações; e *desktop conferencing*, que fazem uso de estações de trabalho como interface da conferência, mas também rodam outras aplicações compartilhadas entre os participantes.

Agentes inteligentes - Nem sempre os participantes de uma reunião eletrônica são pessoas. Alguns participantes podem ser agentes inteligentes responsáveis por um conjunto específico de tarefas empregadas para simular ações de participantes reais. Um exemplo desses sistemas pode ser encontrado em Schmidt et al. [3], com participantes representados no sistema por um conjunto de agentes, executando ou antecipando para o usuário várias tarefas a serem elaboradas.

Sistemas de coordenação - Estes sistemas são responsáveis pela integração e o ajuste harmonioso dos esforços individuais dos participantes em rumo à obtenção de um objetivo geral. Três categorias podem ser utilizadas para distinguir as funcionalidades mais específicas deste tipo de sistemas: sistemas orientados a procedimentos, que vêem os procedimentos organizacionais como processos programáveis; sistemas orientados à conversação, baseados na observação de que as pessoas coordenam suas atividades através de comunicação; e modelos orientados a estruturas, os quais descrevem as atividades organizacionais em termos de relacionamentos de papéis.

A interface de uma aplicação, seja ela gráfica ou não, fornece ao usuário acesso às funcionalidades da aplicação e é responsável tanto pela captação das interações do usuário com a aplicação como pela apresentação das informações para o usuário. Interfaces de *groupware* possuem características específicas para apoiar o comportamento complexo das interações dos usuários com a aplicação e envolvem outras características funcionais além daquelas normalmente consideradas no projeto de uma interface para um único usuário.

Willians et al. [42] apontam as características funcionais das interfaces de *groupware* como a maior diferença entre um *groupware* e um software qualquer. De

acordo com Willians, ao contrário dos outros softwares, que procuram isolar um usuário dos demais, um *groupware* deve dar ênfase ao ambiente de interação multiusuária, coordenando e orquestrando as atividades, de modo que os usuários possam interagir uns com os outros para completar tarefas compartilhadas.

Devido aos problemas de projeto apresentados pelas interfaces de *groupware*, algumas técnicas e conceitos foram sugeridos com o objetivo de resolver as questões de interface das aplicações que interagem com vários usuários. As principais abordagens sugeridas foram utilizadas para classificar as interfaces, e as aplicações que as utilizam, de acordo com os modelos adotados por cada abordagem. Dias [52] compilou várias abordagens encontradas na literatura e apresenta quatro modelos de interfaces possíveis, descritos abaixo:

WYGIWIG (*What You Get Is What I Get*) - A principal característica deste modelo de interface é o compartilhamento de janelas entre os usuários. Pequenas inconsistências nas versões dos documentos dos usuários chegam a ser toleradas pelo sistema, desde que não comprometam as informações que podem ser recuperadas do espaço compartilhado. Há também a presença de restrições de coordenação e atrasos na propagação das atualizações nos documentos compartilhados;

WYSIWID (*What You See Is What I Did*) - Neste modelo, as interações entre os usuários dão-se de forma assíncrona, pois a propagação das alterações para os demais usuários pode ser adiada até que determinadas condições sejam estabelecidas, tornando defasada a visão dos usuários em relação à real situação atual;

WYSIAWIS (*What You See Is Almost What I See*) - Neste modelo de interfaces a programação das ações dos usuários no ambiente compartilhado é incompleta, ou seja, as ações são encaminhadas com pequenas variações que não chegam a prejudicar a interpretação dos resultados no ambiente compartilhado; e

WYSIWIS (*What You See Is What I See*) - Este modelo de interface garante que o contexto compartilhado pelos participantes seja idêntico para todos, ou seja, os participantes possuem a mesma visão do trabalho compartilhado em todos os momentos.

O modelo WYSIWIS é a interface mais utilizada em *groupware*, graças ao seu forte senso de contexto de compartilhamento e à facilidade de implementação. Entretanto, alguns autores consideram este tipo de interface como inflexível devido à

falta de controle sobre certos aspectos das interfaces individuais dos usuários. Esta falta de controle evita que os usuários configurem aspectos de suas interfaces da maneira que mais lhes agrada, tornando o uso de um *groupware* semelhante ao uso de uma aplicação não colaborativa.

Um relaxamento do tipo de interface WYSIWIS, proposto por Stefik et al. [54], propicia aos usuários o controle sobre aspectos de suas interface individuais permitindo que eles trabalhem de forma mais natural. WYSIWIS relaxado torna o *groupware* mais flexível, aproximando os usuários das maneiras com as quais eles utilizam as aplicações não colaborativas.

O modelo de interfaces WYSIWWS (*What You See Is What We Share*), proposto por Boulila et al. [41], sugere uma interface que compartilhe o modelo entre os usuários. Neste modelo de interface, os usuários não compartilham janelas ou aplicações, mas sim o modelo, o que permite uma visão individual do mesmo, de acordo com as configurações de visualização da interface escolhidas pelo usuário.

O protótipo construído nesta tese, que será apresentado no Capítulo 4, segue os princípios do modelo de interface WYSIWIS relaxado e também do modelo WYSIWWS para o compartilhamento de um modelo representado por um diagrama da UML.

2.1.3 Características Funcionais

Os sistemas g*roupware* são compostos por várias características funcionais que nem sempre estão presentes em todos os sistemas colaborativos, porém qualquer sistema considerado *groupware* deve possuir ao menos algumas dessas características.

Um *groupware* empregado para auxiliar a elaboração de alguma tarefa compartilhada deve levar em consideração algumas características funcionais básicas para poder apoiar a colaboração. Santoro et al. [23] citam como principais características funcionais de um *groupware* a comunicação, a coordenação, a percepção (*awareness*), o compartilhamento de informações e a designação de papéis.

A comunicação é de extrema importância em situações de trabalho em grupo, seja ela utilizada para trocar idéias, discutir, aprender, negociar ou para tomar decisões.

Da mesma maneira que as interações dos usuários, a comunicação pode ser classificada em síncrona, assíncrona ou multisíncrona, dependendo do momento que os membros receberam as mensagens enviadas pelo canal de comunicação. Diferentes canais de comunicação podem ser utilizados para tornar a comunicação mais eficiente e produtiva como, por exemplo, *chat* ou vídeo conferência.

Outro modo de classificar a comunicação se refere à existência de ligações entre os participantes, através tanto de canais de comunicação direta, como troca de mensagens e reuniões, quanto por meio de um canal indireto através da memória de grupo, onde a construção e o compartilhamento do conhecimento comum podem ser considerados interfaces de comunicação.

A coordenação do grupo que trabalha com um *groupware* síncrono pode exigir um sofisticado mecanismo de controle das ações para garantir o sucesso da colaboração. Mecanismos de controle de concorrência, como travas ou transformação, são utilizados para garantir a consistência dos elementos manipulados pelos participantes da colaboração no documento compartilhado.

Em situações onde a probabilidade de conflitos entre as ações dos participantes for baixa ou existir um moderador para coordenar as ações, o controle de concorrência pode ser dispensado. Nessas situações, a coordenação das ações pode ficar a cargo do chamado protocolo social, caracterizado pela ausência de mecanismos de controle explícitos entre as atividades e pela confiança nas habilidades dos participantes de mediar por si só as interações por meio de um canal de comunicação.

Durante o trabalho em grupo, a percepção é de grande importância e se relaciona com a coordenação, a comunicação e a cooperação durante a execução da tarefa compartilhada. A percepção em si é relativa ao ser humano, enquanto os elementos de percepção estão relacionados à interface do ambiente [29]. Através da percepção, um indivíduo organiza e interpreta as suas impressões sensoriais para atribuir significado ao seu meio. Do ponto de vista psicológico, a percepção envolve também os processos mentais, a memória e outros aspectos que podem influenciar na interpretação dos dados percebidos.

Já a cognição, derivada da palavra latina *cognitione*, significa a aquisição de um conhecimento através da percepção, de acordo com Ferreira [8]. É o conjunto dos

processos mentais usados no pensamento, na classificação, reconhecimento e compreensão que auxiliam o julgamento através do raciocínio.

Nas ferramentas colaborativas síncronas, a percepção permite que um participante, fisicamente separado, esteja ciente da presença e das ações dos demais participantes da colaboração facilitando o trabalho em conjunto. A percepção também permite socializar virtualmente o ambiente e pode servir para indicar o esforço dos participantes em interagir com a ferramenta e com os demais participantes.

A percepção, em geral, é implementada nas ferramentas colaborativas síncronas por meio de elementos de percepção que fornecem informações imediatas a respeito do estado dos participantes na colaboração e de suas interações com a ferramenta. Contudo, apesar das vantagens dos elementos de percepção de um *groupware*, Santoro et al. [23] notam que tais elementos ainda não conseguem se aproximar da riqueza de informações proporcionada pela interação face a face.

A colaboração necessita que os participantes compartilhem informações. Este aspecto é essencial para os grupos, devido à necessidade de evitar esforços repetitivos e assegurar que todos os participantes do grupo estejam utilizando a mesma informação, de forma que não haja inconsistências. Os sistemas *groupware* geralmente fazem uso de documentos compartilhados como mecanismo de compartilhamento de informações e de armazenamento da memória do grupo, que deve registrar todo o processo de interação, como a própria comunicação realizada e passos desencadeados, bem como todos os produtos gerados durante a colaboração.

A utilização de papéis predefinidos em um sistema *groupware* tem como objetivo principal estruturar as interações entre os participantes do grupo, definir tarefas e gerenciar o acesso aos elementos do documento compartilhado. Um papel descreve como um conjunto de indivíduos se relaciona com algum elemento compartilhado e com os outros participantes restantes do grupo por meio da especificação dos direitos e responsabilidades sobre diferentes atividades a serem realizadas pelo grupo.

A definição de papéis também é utilizada como um mecanismo de coordenação das atividades dos participantes e também como um mecanismo de controle de acesso a elementos do documento compartilhado. No entanto, sua atribuição precipitada pode restringir o potencial criativo dos participantes, ou seja, cada indivíduo

somente efetuaria as operações específicas de seu papel sem que todo o seu potencial tenha sido alcançado.

Outra característica funcional presente em *groupware* é a sessão colaborativa, que pode ser definida como uma atividade compartilhada em andamento. Ela é composta por um grupo de participantes que compartilham documentos e mensagens de forma coordenada durante um período de tempo com o objetivo de realizar uma tarefa compartilhada. O acesso à sessão colaborativa, os privilégios de cada participante e as propriedades da sessão podem ser controlados automaticamente pelo sistema ou serem gerenciados explicitamente por um dos membros envolvidos na sessão colaborativa.

2.2 UML

A UML incorpora a noção do desenvolvimento de software totalmente visual, baseando-se na modelagem de diagramas classificados em visões de abstração. Essas visões de abstração resultam em uma linguagem de modelagem bem definida, expressiva, poderosa e que poderia ser aplicada a uma grande variedade de tipos de problemas [28].

A definição da UML contém uma notação e um metamodelo. A notação compreende todos os elementos de representação gráfica vistos no modelo (retângulo, setas, o texto etc), compondo sintaxe do modelo de linguagem. Por exemplo, a notação do diagrama de classe define a representação de itens e conceitos tais como: classe, associação e multiplicidade. Um metamodelo é definido como a descrição abstrata dos modelos que definem quais componentes da linguagem de modelagem são necessários para descrever os modelos de um dado domínio de aplicação.

A UML disponibiliza uma forma padrão de modelagem de projetos de sistemas, incluindo seus aspectos conceituais, tais como processos de negócios e funções do sistema, além de itens concretos como as classes escritas em determinada linguagem de programação, processos de banco de dados e componentes de software reutilizáveis.

A UML aborda o caráter estático e dinâmico do sistema a ser analisado levando em consideração, já no período de modelagem, todas as futuras características do sistema. Na prática, a UML é aplicada por meio de um conjunto de diagramas

utilizados para modelar um sistema a partir de diferentes perspectivas. Os diagramas UML podem ser agrupados em duas categorias: diagramas estruturais e diagramas comportamentais. São quatro os principais diagramas estruturais utilizados para visualizar, especificar, construir e documentar os aspectos estáticos de um sistema [28]:

Diagrama de classes - Descrevem os tipos de objetos (classes, tipos de dados e interfaces) e os diferentes tipos de relacionamentos estáticos que os conectam;

Diagramas de objetos - Representam uma instância em particular de um diagrama de classes;

Diagrama de componentes - Descrevem os componentes de software do sistema, tais como código fonte, código binário e executáveis, bem como suas dependências; e

Diagrama de instalação (desenvolvimento) - Descrevem a configuração, em tempo de execução, dos nós de processamento e seus respectivos componentes.

Os diagramas comportamentais são utilizados para visualizar, especificar, construir e documentar os aspectos dinâmicos de um sistema. São cinco os principais diagramas comportamentais definidos pela UML.

Diagrama de casos de uso - Definem os atores, os casos de uso do sistema e seus relacionamentos. Um caso de uso é uma unidade de funcionalidade que representa uma seqüência de ações que o sistema executa e que produz um resultado observável para os atores deste sistema;

Diagrama de estados - Descrevem o comportamento de um sistema através da representação de um conjunto de estados conectados por transições;

Diagrama de atividades - Estes diagramas são considerados uma variação do diagrama de estados e correspondem as atividades que representam a execução de operações;

Diagrama de seqüência - Descrevem uma interação através das mensagens trocadas pelos objetos envolvidos nesta interação, organizadas em uma seqüência temporal; e

Diagrama de colaboração - Descrevem uma interação através de objetos, suas conexões e fluxos de mensagens associadas às conexões.

A primeira parte da versão atual da UML 2.0, que descreve os novos diagramas e os elementos de modelagem disponíveis, foi adotada pelo OMG (*Object Management Group*) em outubro de 2004. As outras partes da especificação da UML, a infra-estrutura, a linguagem de restrição de objetos (*Object Constraint Language*) e a interoperabilidade entre diagramas não foi completada deste novembro de 2005, data da última reunião do comitê de padronização.

Devido ao alto nível de comunicação e a necessidade de interação entre os desenvolvedores durante a elaboração de diagramas UML, a introdução de uma ferramenta colaborativa para auxiliar a modelagem de diagramas entre desenvolvedores dispersos geograficamente pode trazer benefícios na elaboração de diagramas UML. Com base neste cenário, o Capítulo 4 apresenta um protótipo que auxilia a modelagem de diagramas UML por meio da edição colaborativa síncrona de diagramas.

2.3 Ferramentas CASE

Com a utilização de ferramentas CASE (*Computer Aided Software Engeneering)* as informações envolvidas no processo de desenvolvimento de software podem ser criadas, modificadas e reutilizadas mais facilmente do que em um processo manual. Documentos de requisitos, modelos gráficos e textuais e outras informações são manipulados por essas ferramentas proporcionando uma maior flexibilidade e rapidez no processo de desenvolvimento de software. As demais características dessas ferramentas, como a capacidade de geração de código e a consistência entre modelos, fornecem recursos para aumentar a eficiência e qualidade do processo de desenvolvimento de software.

Ferramentas CASE de modelagem são um tipo particular de ferramenta CASE amplamente utilizadas na indústria, em especial as ferramentas CASE que utilizam a UML. Essas ferramentas auxiliam na criação de modelos de diversos aspectos de um sistema de software, tais como o seu comportamento, sua arquitetura ou a sua estrutura. Além de serem editores gráficos, que facilitam a criação de modelos e diagramas por meio de uma notação, essas ferramentas normalmente oferecem mecanismos para auxiliar a criação de outros artefatos durante o projeto de software.

Uma ferramenta CASE de modelagem auxilia diversas fases do ciclo de desenvolvimento de um sistema, desde a fase de análise até a fase de testes, apresentando utilidade também durante a fase de manutenção do sistema já implantado. Essas ferramentas armazenam informações em sua base de dados, tanto em forma de textos como em forma gráfica, possibilitando a comunicação entre os envolvidos no projeto e a aprovação das definições de processos.

As principais vantagens decorrentes do uso de ferramentas CASE incluem o aumento da qualidade e produtividade dos produtos finais, a redução de trabalho manual e o aumento da eficiência na elaboração de modelos. Como conseqüência destas vantagens, os envolvidos no projeto podem dedicar mais tempo para a tomada de decisões e para a gerência das mudanças, além de outras tarefas do projeto como a documentação e o controle de defeitos, erros e falhas.

A sigla I-CASE (*Integrated CASE*) é empregada para definir ferramentas CASE que se relacionam entre si, abrangendo desde as fases de análise/projeto até a geração do código em um mesmo ambiente. Neste ambiente integrado, a troca de informações entre diferentes de artefatos é promovida pela integração entre as ferramentas, o que facilita a utilização de processos unificados durante o desenvolvimento de um software.

Um aspecto importante que deve ser considerado é a interoperabilidade entre diferentes ferramentas CASE. Segundo Pressman [70], o grande potencial da Engenharia de Software Assistida por Computador só é alcançado integrando-se diversas ferramentas individuais, pois uma ferramenta complementa a funcionalidade da outra, contribuindo com todo o processo. Essa abordagem oferece várias vantagens se comparada ao uso de ferramentas individuais, incluindo a transferência de informações entre as ferramentas, a redução do esforço necessário para realizar atividades que se expandem durante o projeto e o aumento do controle de projeto, entre outras.

A integração, contudo, é difícil de ser obtida, pois envolve diversos fatores como o uso de repositórios compartilhados, padronização de entradas e saídas, integração por eventos, entre outros. Desta maneira, é necessária uma arquitetura completa de integração, para que todas as ferramentas possam oferecer o máximo de seu potencial e contribuir para o melhoramento do processo de desenvolvimento como um todo.

2.3.1 Exemplos de Ferramentas CASE para Modelagem

A maioria das ferramentas CASE faz uso de um conjunto básico de diagramas UML. Algumas ferramentas CASE possuem recursos adicionais, propiciando: a geração de código fonte em várias linguagens a partir de diagramas; a integração entre elementos de diferentes diagramas; o controle de versões; e o trabalho cooperativo, entre outros. Porém, muitas ferramentas são consideradas simples editores gráficos que manipulam os elementos de modelagem de alguns dos principais diagramas UML. As ferramentas comerciais disponíveis que permitem a edição de diagramas UML incluem: a IBM Rational Rose; a Borland Together Designer; e a GentleWare Poseidon UML. As seguintes ferramentas gratuitas se destacam:

Violet - Totalmente escrita em Java, permite a edição de alguns diagramas UML, como diagramas de classes e de seqüência. Ela permite a geração de código a partir do modelo e outros recursos adicionais;

EclipseUML - Essa ferramenta é um *plug-in* para a plataforma *Eclipse* que permite a edição de diversos diagramas UML e a geração de código. Os recursos para trabalho em equipe e o acesso a repositórios é oferecido em uma versão comercial desta ferramenta;

JUDE - O JUDE (*Java and UML Developers Environment*) é uma ferramenta de modelagem de diagramas UML, desenvolvida em Java, e que combina recursos de mapas mentais. Ela permite a edição de 14 diagramas UML, além de recursos para a geração de código fonte a partir de diagramas;

Umbrello UML Modeler - Esta ferramenta, projetada para ser executada somente no ambiente gráfico KDE (*K Desktop Environment*), foi construída em C/C++ e atende ao padrão XMI (*XML Metadata Interchange*), além de permitir a exportação de modelos no formato SVG (*Scalable Vectorial Graphics*);

UMLSculptor - Uma ferramenta simples para a edição de diagramas de classes da UML. A sua principal característica é a facilidade com que os diagramas podem ser criados;

DIA - O DIA (*Diagram Editor*) é um editor de gráficos criado a partir da biblioteca GTK+ (GIMP toolkit) e que faz parte do projeto GNOME. Inspirado no software *Microsoft Visio*, além da edição de diagramas UML, o DIA permite que os

diagramas sejam exportados em vários formatos, incluindo EPS, SVG, XFIG, WMF e PNG; e

ArgoUML - A ferramenta ArgoUML é totalmente escrita em Java, possui mecanismos para a geração de código, engenharia reversa e um sistema de críticas aos modelos gerados. Esta ferramenta, atualmente, é considerada a ferramenta gratuita mais utilizada pelos desenvolvedores, permitindo o desenvolvimento da ferramenta CASE comercial GentleWare Poseidon UML.

Nenhuma das ferramentas CASE de modelagem citadas contêm recursos para a edição colaborativa de diagramas UML, evidenciando a falta opções colaborativas para os usuários, no que diz respeito à modelagem de diagramas UML.

Em geral, as ferramentas CASE que suportam a modelagem de diagramas UML apresentam poucos recursos para o compartilhamento de seus diagramas, fazendo com que os desenvolvedores busquem alternativas para compartilhar os dados gerados pela ferramenta. Na maioria das vezes, a alternativa escolhida pelos desenvolvedores envolve o uso de uma ferramenta auxiliar que faz o controle de versões de arquivos, onde os desenvolvedores trabalham individualmente nos seus diagramas para que futuramente sejam sincronizados em uma única versão ao final do diagrama. Esta abordagem não fornece um espaço de trabalho compartilhado tampouco mecanismos de percepção de trabalho compartilhado e canais de comunicação entre os desenvolvedores durante a tarefa.

Transformar uma ferramenta CASE de modelagem síncrona em um editor colaborativo de diagramas traz para a área de engenharia de software algumas das vantagens e desvantagens que um *groupware* pode oferecer. Por meio da colaboração os usuários podem concentrar os esforços dos desenvolvedores distantes entre si em um ambiente de trabalho compartilhado.

As ferramentas CASE que apresentam recursos para a modelagem colaborativa de software são incluídas na categoria de CES (*Collaborative Editing Systems*). onde os participantes do grupo colaborativo trabalham editando um modelo compartilhado. Em sistemas síncronos os usuários podem visualizar imediatamente as mudanças introduzidas por outros usuários no modelo compartilhado.

Os editores colaborativos CUTE *(Collaborative UML Technique Editor)* [51] e CO2DE *(Collaborate to Design)* [48] são exemplos de iniciativas que permitem a

edição colaborativa de diagramas UML, assim como as ferramentas CASE comerciais GentleWare Poseidon UML e Magic Draw.

Apesar de apresentarem soluções prontas para o desenvolvimento colaborativo de diagramas UML, os trabalhos publicados não apresentam técnicas ou métodos para a modificação de uma aplicação já existente com o objetivo de promover a colaboração. Os trabalhos publicados relatando o desenvolvimento de *groupware* para a modelagem colaborativa mostram que os resultados finais envolvem mais a criatividade dos desenvolvedores do que a aplicação de métodos ou mesmo trabalhos anteriores. Nota-se que o desenvolvimento de *groupware* ainda é utilizado de maneira *ad hoc*, com pouca ou nenhuma técnica sistemática de desenvolvimento que possa ser seguida como um guia para promover não apenas o compartilhamento de modelos em aplicações, mas também os principais conceitos de *groupware*.

2.4 Desenvolvimento Baseado em Componentes

Antes de definir o que é o Desenvolvimento Baseado em Componentes, faz-se necessário apresentar uma definição adequada para componentes de software.

Componentes de software, doravante denominados componentes, são blocos de construção prontos para serem instalados e executados. Nesta definição, apresentada por Szyperski [12], os componentes são independentes e reusáveis, provendo serviços quando utilizados individualmente em composição com outros componentes que oferecem outros serviços.

Um componente provê acesso ao seu serviço através de uma ou mais interfaces, definidas como um conjunto de operações que especificam os serviços oferecidos pelos componentes. Estes serviços podem, normalmente, ser personalizados ajustando-se os valores das propriedades do componente.

O Desenvolvimento Baseado em Componentes (DBC) faz uso de componentes substituíveis, reusáveis e interoperáveis para compor a aplicação final, de acordo com Gimenes e Huzita [32]. O Desenvolvimento Baseado em Componentes têm como objetivo prover o reuso dos componentes no nível de implementação e aumentar a interoperabilidade entre as partes do software. Deste modo, os componentes representam parte da funcionalidade que está pronta para ser instalada e executada em

vários ambientes. Em uma abordagem voltada para o Desenvolvimento Baseado em Componentes, os seguintes princípios são desejáveis:

- Cada componente deve prover uma funcionalidade específica (serviço único);
- As dependências entre componentes devem ser minimizadas;
- Os componentes devem ser genéricos, a fim de serem reutilizados em um maior número possível de aplicações;
- Os componentes devem ser facilmente combináveis a fim de facilitar a montagem da aplicação;
- Os componentes devem ser projetados de tal forma que possam ser futuramente estendidos, dado que não é possível prever todas as necessidades de um serviço oferecido por um componente em um primeiro momento;
- Os componentes devem ser utilizados em uma grande variedade de plataformas operacionais (interoperabilidade); e
- Os componentes devem possuir propriedades as quais podem ser modificadas para adaptar o serviço provido pelo componente a fim de atender uma necessidade ou situação específica.

Em resumo, o Desenvolvimento Baseado em Componentes pode facilitar o processo de desenvolvimento de sistemas, para que estes sejam mais flexíveis. Deste modo, o Desenvolvimento Baseado em Componentes propicia a criação de sistemas modulares compostos de componentes, que podem ser adaptados e combinados na medida da necessidade, tendo em mente futuras manutenções.

2.5 *Framework*

O conceito de *framework* foi proposto com o objetivo de maximizar a reusabilidade de vários artefatos e conceitos durante o processo de desenvolvimento de software. Na área de Engenharia de Software, existem várias definições para o termo *framework,* cada uma focando um aspecto em particular. Nesta tese optou-se pela

definição que aborda principalmente a reusabilidade de projeto. Orfaly [21] apresenta a seguinte definição de *framework*:

"Um *framework* é o projeto de um conjunto de objetos que colaboram para por em prática um conjunto de responsabilidades. *Frameworks* são um modo de reutilizar projetos de alto nível."

Os *frameworks* são genéricos e devem ser estendidos para permitir a criação de aplicações específicas ou outros subsistemas. Devido a esta generalidade, as aplicações construídas a partir de um mesmo *framework* apresentam aspectos semelhantes, diferenciando-se apenas em seu comportamento implementado na aplicação. Isto torna as aplicações desenvolvidas a partir dos *frameworks* mais fáceis de manter e mais consistentes para os usuários, que não precisam aprender diferentes aplicações.

Um *framework* pode fornecer um grau de reutilização de até 80% [19], pois além de permitir a reutilização de código, ele também oferece reutilização de projeto, liberando o desenvolvedor dos aspectos comuns do domínio de aplicação. Reutilizar um projeto eleva a um novo patamar o conceito de reutilização em software, permitindo que projetos complexos sejam beneficiados por soluções já implementadas, testadas e documentadas. Além disso, a reutilização de projeto é mais importante que a reutilização de código, pois reutilizar o projeto apresenta mais dificuldade do que reutilizar o código [65].

A reutilização em projetos de software vem sendo praticada há algum tempo pelo uso de bibliotecas de procedimentos e componentes de software contendo bibliotecas de classes orientadas a objetos. A principal diferença entre *framework* e estas soluções amplamente utilizadas é que um *framework* corresponde a um projeto de alto nível, consistindo de classes abstratas e concretas que especificam uma arquitetura para aplicações.

Componentes contendo classes, onde nem todas as classes relacionam-se, fornecem um conjunto de serviços disponibilizado através da interface pública de suas classes e não contêm conhecimentos do domínio. De maneira similar, uma biblioteca de procedimentos fornece uma série de serviços através de chamadas às suas rotinas. Nenhuma das duas abordagens permite a reutilização de projeto.

O modo como os *frameworks* operam também é diferente de outras abordagens. Em um *framework* bem projetado, o desenvolvedor especifica somente o

código de métodos de classes específicas que corresponderão a sua aplicação. O próprio *framework* se encarrega de chamar o código da aplicação quando necessário, pois é ele quem trata as particularidades da aplicação. Em componentes ou bibliotecas procedurais, além do código da aplicação propriamente dita, o desenvolvedor deve ser responsável pelo comportamento funcional da aplicação, como a especificação do fluxo de execução e a estrutura do programa.

Reutilizar um projeto por meio do uso de um *framework* requer a execução de várias tarefas. Inicialmente deve-se escolher um *framework* que forneça o molde de projeto desejado no qual a aplicação final poderá se encaixar. Em seguida, é necessário verificar como o *framework* deve ser estendido para que a aplicação faça uso de seu projeto. O uso de um *framework* geralmente requer a adição de classes concretas, geradas a partir da especialização das classes abstratas do *framework*.

Por definição, *framewoks* são incompletos, reusáveis e não devem ser considerados aplicações por si só. Eles contêm funcionalidades abstratas, com pouca ou nenhuma implementação, que devem ser completadas para que o desenvolvedor possa aproveitar suas vantagens. Boa documentação, facilidade de uso, extensão e eficiência são apenas algumas das características que um bom *framework* deve possuir.

Vários autores propõem *frameworks* para o desenvolvimento de *groupware*. Prakash [7] propõe um *framework* para gerência de operações em uma ferramenta colaborativa, possibilitando registrar e desfazer operações. Ortega [14] descreve um modelo e um *framework* que permitem a elaboração de *groupware* com interfaces personalizáveis. Pinheiro [44] propõe um *framework* para gerenciar as informações de percepção de um *groupware*.

Com foco na área de modelagem de artefatos em um projeto de software, Boulila et al. [41] propõem o *framework* D-Meeting (*Distributed Meeting*) para apoiar a modelagem distribuída síncrona de software. O D-UML (*Distributed UML*) é uma implementação em Java do *framework* D-Meeting que suporta a modelagem de diagramas UML durante reuniões síncronas por meio de manipulações e compartilhamento de informações.

Lozano [49] apresenta um *framework* para consistência de artefatos em ferramentas de modelagem de software colaborativas. Este *framework* se propõe a resolver problemas de sistemas distribuídos, em especial os problemas relacionados

com a divergência, violação da ordem casual e violação das intenções dos usuários. Com o objetivo de implementar os algoritmos e esquemas apresentados neste *framework*, Lozano apresenta o protótipo DArgoUML (*Distributed ArgoUML)*, que é uma versão distribuída da ferramenta CASE ArgoUML. Apesar de implementar elementos de controle de concorrência e tornar distribuída uma aplicação já existente, não há indícios de como implementar, de forma sistemática, requisitos importantes em *groupware* como, por exemplo, a comunicação, o gerenciamento de sessão e dispositivos de presença.

Os *frameworks* para a construção de *groupware* auxiliam a criação de novas aplicações, contudo eles fornecem poucos benefícios para o processo de adaptação de aplicações existentes para que elas apóiem a colaboração. Devido à forma de uso dos *frameworks*, os desenvolvedores que desejarem implementar funcionalidades colaborativas podem reaproveitar apenas partes do *framework*. Este trabalho segue esta idéia, uma vez que o protótipo descrito no Capítulo 4 utiliza o *framework* GEF (*Graph Edition Framework*).

2.6 Padrões de Projeto

Padrões de projeto são um meio para representar, registrar e reutilizar micro-arquiteturas de projeto repetitivas, bem como a experiência acumulada por projetistas durante o desenvolvimento de software [22]. A micro-arquitetura mencionada neste contexto tem o objetivo de servir como um molde, permitindo vários tipos de implementações em projetos distintos. Outras definições para padrões de projeto podem ser encontradas na literatura, tendo como ponto comum entre elas a idéia básica de reutilizar experiências passadas.

Os padrões de projeto, em geral, não contêm idéias novas e nem soluções originais. Sua origem consiste na observação de soluções freqüentemente utilizadas para resolverem problemas comuns em um determinado contexto. A adoção das idéias de problema, solução e contexto aplicadas aos padrões de projeto em software fazem com que um padrão apresente quatro propriedades essenciais para sua posterior utilização [22]: o nome do padrão, o problema, a solução e as conseqüências.

O nome do padrão é usado para sintetizar um problema, sua solução e suas conseqüências. Deste modo, o vocabulário dos projetistas é ampliado, facilitando o desenvolvimento de software, uma vez que existirá um vocabulário comum que permitirá a troca de experiência entre projetistas através da comunicação das decisões de projeto. Apenas dizer que um sistema implementa o padrão X, por exemplo, pode comunicar uma decisão de projeto de forma sintética e mais precisa do que uma explicação complexa, assumindo que ambos os lados conheçam o padrão X.

O problema descreve quando aplicar o padrão. Explica o problema e o contexto em que ele está inserido. Pode descrever problemas específicos ou genéricos. É comum encontrar pelo menos dois exemplos de problemas para cada padrão documentado.

A solução descreve os elementos que formam o padrão, seus relacionamentos, responsabilidades e documentação. A solução é apresentada para um problema genérico e que contém uma estrutura de fácil implementação.

As conseqüências são os resultados e os impactos gerados no sistema que utilizou um determinado padrão. A descrição destas conseqüências é vital para uma avaliação dos custos e benefícios de um padrão e, consequentemente, para auxiliar sobre a decisão a respeito do seu uso. As conseqüências de um padrão incluem, por exemplo, seu impacto sobre aspectos operacionais do sistema, como a portabilidade e expansibilidade do sistema.

A utilização de padrões de projetos apresenta as seguintes vantagens: facilidade de comunicação entre os projetistas, redução da complexidade do sistema, larga reutilização de arquiteturas de software e redução do tempo de aprendizado. Padrões de projetos também podem ser vistos como blocos de construções, fornecendo idéias básicas para a criação de aplicações mais complexas [22].

Buschmann [24] sugere um agrupamento dos padrões de projeto existentes e futuros em três categorias: Padrões de Arquitetura, Padrões de Projeto e Padrões de Idioma.

- Padrões de Arquitetura - Expressam um esquema da organização global da estrutura do sistema. Eles fornecem um conjunto de subsistemas predefinidos, especificando os relacionamentos entre eles e estabelecendo regras para esses relacionamentos;

- Padrões de Projeto propriamente dito - Fornecem um esquema para refinar os subsistemas ou componentes do sistema de software. Esses padrões possuem um grau de granularidade considerado médio e são independentes de linguagem de programação. Padrões de projeto correspondem a uma abstração de duas, três ou um pequeno número de classes, utilizadas em exaustão no desenvolvimento de software orientado a objetos; e
- Padrões de Idioma - São os padrões de nível mais baixo, específicos para uma determinada linguagem de programação. Eles descrevem como implementar aspectos particulares de componentes e dos relacionamentos entre eles usando características específicas da linguagem alvo.

Além desta classificação, os padrões podem ser categorizados segundo o domínio no qual se aplicam. Nesta classificação existem padrões genéricos, que podem ser aplicados em diferentes domínios; mas também podem existir padrões específicos para determinados domínios como, por exemplo, tolerância à falhas, criptografia, telecomunicações etc.

Com o objetivo de tornar fácil a utilização de padrões de projetos, Gamma et al. [22] propuseram a criação de um catálogo de padrões de projeto genéricos. Baseado na experiência prática dos autores na resolução de problemas comuns, este catálogo conta com vinte e três padrões bem documentados e com ao menos um exemplo de aplicação prática de cada padrão.

Muitos desenvolvedores confundem o conceito de padrões de projeto, freqüentemente relacionando o conceito de padrões de projetos com aspectos mais técnicos de implementação como listas encadeadas ou algoritmos específicos. Gamma et al. [22] procuram esclarecer esta falha de entendimento apresentando os padrões de projeto como descrições de objetos e classes que se comunicam e devem ser personalizados para resolver problemas gerais de projeto em um contexto particular.

A construção de sistemas *groupware* também é beneficiada pelos padrões de projeto. Lukosch e Schümmer [72] propõem uma linguagem para a especificação de padrões para *groupware*, visando instrumentar o desenvolvimento, de modo a favorecer o reuso de soluções e capacitar membros inexperientes no desenvolvimento de

groupware. Além da linguagem de especificação de padrões, Lukosch e Schümmer apresentam um catálogo de padrões de projetos específicos para sistemas *groupware*.

Apesar de auxiliarem no desenvolvimento de sistemas *groupware*, os padrões de projetos não fornecem soluções para as questões ligadas à adaptação e modificação de uma aplicação para que ela propicie colaboração síncrona. Contudo, as aplicações existentes geralmente fazem uso de diversos padrões de projetos genéricos, de modo que o bom conhecimento destes padrões pode auxiliar no processo de adaptação e modificação dos componentes utilizados pela aplicação.

2.7 Arquiteturas de Software

À medida que o tamanho dos sistemas cresce, também cresce a sua complexidade e torna-se mais difícil satisfazer um número cada vez maior de requisitos assim como atender às restrições de orçamento e cronograma.

Numa visão que reúne estratégia de reuso de software, tem-se enfatizado a importância de reuso centrado na arquitetura para o desenvolvimento de software, durante todo seu ciclo de vida.

A efetividade do uso de arquitetura como estrutura para reuso pode ser vista através de analogias em domínios de conhecimento bem estabelecidos, como o de Engenharia Civil e Química. A arquitetura de software serve como uma estrutura que permite o entendimento dos componentes de um sistema e seus inter-relacionamentos, especialmente daqueles atributos consistentes ao longo do tempo [5]. Elas influenciam vários aspectos da aplicação, como a tolerância à falhas, a facilidade de modificação, o desempenho da aplicação e a quantidade de esforço necessário na implementação da aplicação e reuso de código.

Um estilo arquitetural descreve uma família de arquiteturas de software que compartilha propriedades, como por exemplo, os componentes permitidos, as restrições e interações entre os componentes, as invariantes, o modelo computacional etc. [37].

Com o objetivo de facilitar a organização dos componentes de uma aplicação, diversos tipos de arquiteturas de software foram desenvolvidos. Esta seção apresenta algumas das principais arquiteturas de software e estilos arquiteturais utilizados no

desenvolvimento de software não colaborativo e colaborativo relacionados com o contexto deste trabalho. É importante notar que as principais arquiteturas descritas nesta seção focam a organização dos componentes de uma aplicação em relação à interface do usuário. O estilo arquitetural MVC, em particular, surgiu da necessidade de separar os componentes da interface do usuário dos demais componentes da aplicação.

2.7.1 Arquiteturas para o Desenvolvimento de Software não Colaborativo

Aplicações que fornecem um alto nível de interação com o usuário são geralmente desenvolvidas utilizando arquiteturas de aplicações não colaborativas. As principais arquiteturas utilizadas na construção de software não colaborativo relacionados com esta tese incluem o modelo Arch, o estilo arquitetural PAC (*Presentation, Abstraction, Controller*) e a arquitetura MVC (*Model, View, Controller*).

O modelo Arch [1] pode ser descrito como um modelo de referência. Ele define uma decomposição funcional geral de um sistema interativo e propõe um alto nível de dependência para os componentes de domínio e de interface de usuário. O modelo Arch divide um sistema interativo em cinco camadas ou componentes: 1) Componentes do Núcleo Funcional (*Functional Core*); 2) Componentes de Adaptação do Núcleo Funcional (*Functional Core Adapter*); 3) Componentes de Interação Física (*Physical Interaction*); 4) Componentes de Interação Lógica (*Logical Interaction*); e 5) Componentes de Controle de Diálogo (*Dialog Controller*). A Figura 2.4 mostra como esses componentes se relacionam no modelo Arch.

O estilo arquitetural PAC (*Presentation, Abstraction, Controller*) decompõe um sistema em uma hierarquia de agentes PAC. Cada agente inclui três facetas: 1) Apresentação, que representa a interface de usuário; 2) Abstração, que mantém os dados; e 3) Controle, que coordena toda a comunicação entre as facetas de apresentação e abstração. A faceta de Controle também coordena todo o tipo de comunicação entre os agentes da hierarquia, como mostrado na Figura 2.5. Neste estilo arquitetural, cada agente é visto como autônomo e pode executar em um processo ou *thread* independente [38].

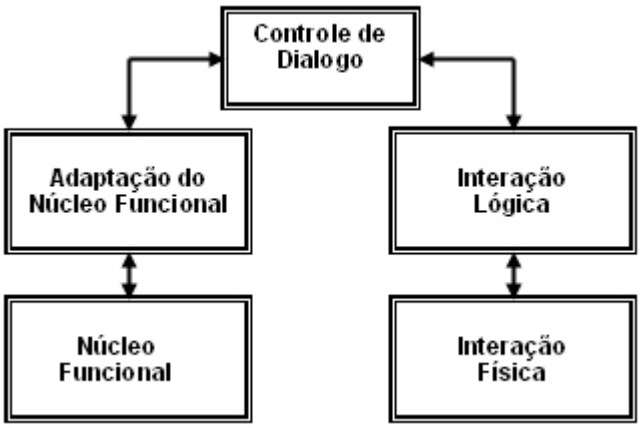

Figura 2.4 - Modelo referencial Arch. Traduzido de [57].

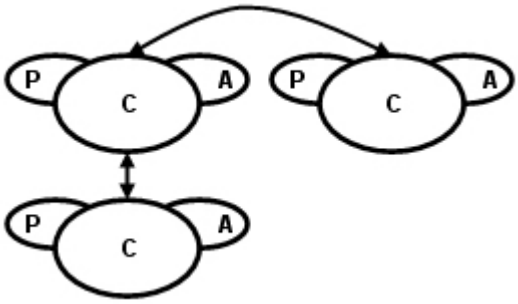

Figura 2.5 - Estilo Arquitetural PAC [38].

Suthers [17] define o estilo arquitetural MVC por meio da separação das partes da aplicação: Modelo, Visão e Controle. Na definição de Suthers, Modelo é a representação interna de modelo semântico de um problema do domínio e implementa os dados da aplicação, Visão é responsável pela visualização dos dados do Modelo por meio de alguma representação visual, e Controle habilita o usuário ou o ambiente a modificar o estado do Modelo.

O Controle é responsável por traduzir as interações do usuário com o sistema em atualizações no estado do Modelo e também por notificar os componentes da Visão que o estado do modelo foi modificado, possivelmente desencadeando alguma modificação na interface do usuário.

Os componentes Controle e Visão devem se registrar como observadores do Modelo, de modo que as modificações no estado do modelo automaticamente resultem em atualizações no Controle e também da Visão. A Figura 2.6 mostra alguns dos fluxos de mensagens da arquitetura MVC da maneira que ela é implementada na maioria dos softwares básicos.

Uma característica inerente ao estilo arquitetural MVC é o baixo acoplamento entre Visão, Controle e Modelo. Devido a este baixo acoplamento, em teoria, uma aplicação que segue o estilo arquitetural MVC pode utilizar várias implementações de Visão e Controle para um mesmo Modelo, pois as maneiras nas quais estas partes são conectadas na aplicação geralmente é flexível, a ponto de permitir que outras implementações da Visão ou Controle sejam registradas para se comunicarem com o modelo. Gamma et al. [18] sugerem o uso de vários padrões de projetos para auxiliar a implementação de uma aplicação MVC como, por exemplo, o uso do padrão *Observer* como mecanismo para notificar Visão e Controle das mudanças no estado do Modelo.

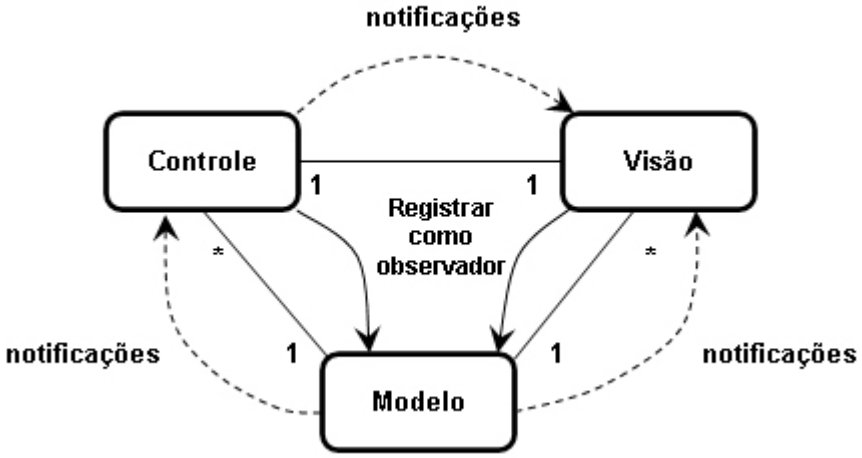

Figura 2.6 - Estilo arquitetural MVC.

A arquitetura MVC é empregada de diversas formas nas aplicações colaborativas. Suthers [17] apresenta algumas maneiras pelas quais as aplicações colaborativas de ensino organizaram seus componentes na arquitetura MVC, de acordo com o nível de acoplamento dos componentes. Apesar de focar apenas aplicações colaborativas na área de ensino, as arquiteturas apresentadas por Suthers são genéricas a ponto de poderem ser consideradas para aplicações colaborativas de propósito gerais.

A Figura 2.7 apresenta os quatro tipos de organização do modelo MVC, de acordo com o acoplamento dos componentes sugerido por Suthers: 1) Arquitetura Centralizada; 2) Arquitetura Replicada; 3) Arquitetura Distribuída; e 4) Arquitetura Híbrida.

A Arquitetura Centralizada sugere que somente uma aplicação contenha todos os componentes do MVC. Os clientes contêm apenas cópias distribuídas da Visão e do

Controle, alimentadas pelo envio dos eventos gerados na GUI para todos os clientes participantes. Neste caso, os componentes Visão, Controle e Modelo permanecem apenas em uma máquina, como a Figura 2.7.a apresenta. Um exemplo de aplicação que utiliza a Arquitetura Centralizada é o NetMeeting [43], onde o software básico é responsável pela captura e reprodução, para todos os clientes, dos eventos gerados pelo usuário. O software compartilhado pelo NetMeeting é executado em um único processo e em uma única máquina, portanto classificado como centralizado devido ao fato que todos os componentes Modelo, Visão e Controle residem em apenas um local.

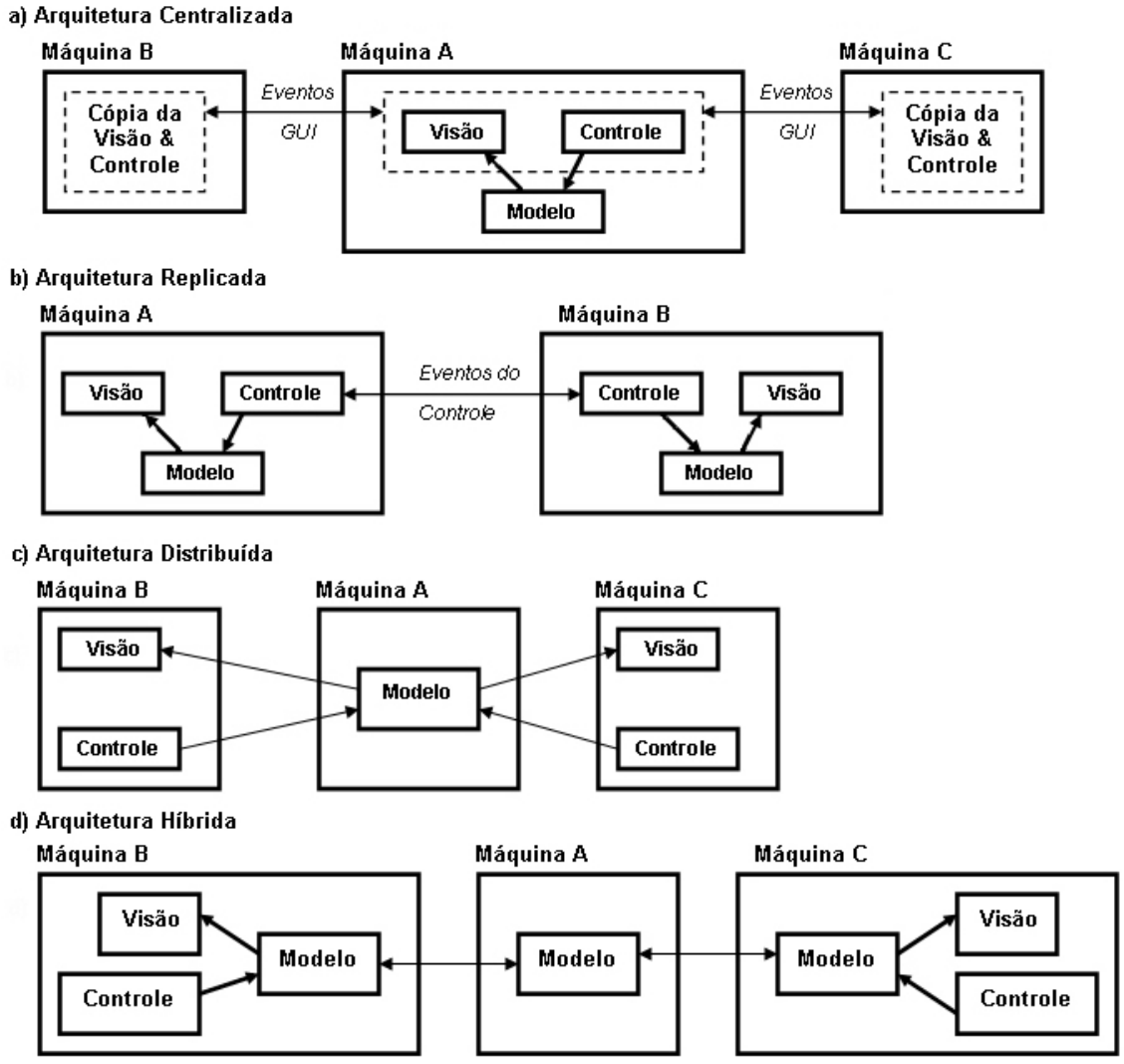

Figura 2.7 - Organizações dos componentes do MVC em: a) Arquitetura Centralizada, b) Arquitetura Replicada, c) Arquitetura Distribuída e d) Arquitetura Híbrida. Traduzida de [17].

Na Arquitetura Replicada, todos os componentes do software básico são instalados e configurados em cada uma das máquinas clientes. Esta arquitetura é caracterizada pela replicação de todos os componentes da arquitetura MVC em todos os clientes. A Figura 2.7.b apresenta a Arquitetura Replicada, onde dois clientes utilizam uma aplicação colaborativa.

A arquitetura Replicada faz um uso eficiente dos recursos de rede, pois apenas os dados relacionados com os eventos do Controle são transmitidos, diferentemente da Arquitetura Centralizada onde os dados relacionados com os eventos do Controle e também da Visão são transmitidos pela rede. Apesar de ser possível utilizar as aplicações que seguem a Arquitetura Replicada, sem a presença de uma conexão constante com a rede, estas arquiteturas, em geral, são utilizadas apenas quando há uma conexão constante entre todos os componentes. O Habanero [4] e o MatchMaker [25] são exemplos de sistemas *groupware* que utilizam a arquitetura Replicada.

A Arquitetura Distribuída é caracterizada pela distribuição dos componentes do MVC em várias estações. Tipicamente, o Modelo é mantido em um servidor compartilhado, e cada cliente possui os componentes da Visão e do Controle, como mostrado na Figura 2.7.c.

O exemplo mais familiar do uso da Arquitetura Distribuída pode ser encontrado em aplicações de comércio eletrônico, onde o usuário utiliza um navegador que fornece os componentes Visão e Controle para que ele interaja com os dados do Modelo armazenados em um servidor. Este tipo de Arquitetura Distribuída compartilha algumas características da Arquitetura Centralizada no sentido que as especificações da Visão e do Controle são construídas, implantadas e armazenadas no servidor, para depois serem enviadas aos clientes.

Se a Arquitetura Distribuída for corretamente implementada, somente as atualizações dos eventos no Modelo precisam ser transmitidas, fazendo com que a Arquitetura utilize os recursos de rede de forma eficiente. Nesta arquitetura, os usuários podem colaborar no mesmo modelo de dados enquanto utilizam diferentes representações visuais. Contudo, a falta de um Modelo local nas aplicações faz com que elas dependam de uma conexão permanente com a rede para serem executadas.

Na Arquitetura Híbrida, apresentada na Figura 2.7.d, cada cliente mantém uma cópia local dos componentes da Visão, do Controle e do Modelo. Nela, os softwares

básicos podem ser executados sem depender dos demais componentes, armazenando o estado dos dados do Modelo local em um arquivo e possuindo diferentes representações visuais de cada modelo. O ambiente Belvedere [18] é um exemplo de uso da arquitetura Híbrida.

Para compartilhar os dados entre os usuários, um servidor remoto é responsável para armazenar uma cópia compartilhada do Modelo que deverá sincronizar as modificações locais dos modelos em um único repositório central de dados. Assim, somente o envio dos eventos do Controle pela rede são necessários para que o modelo do software armazenado no servidor remoto possa ser atualizado. Nesta arquitetura os softwares básicos devem possuir algum mecanismo não só para enviar e receber as modificações do modelo armazenado no servidor compartilhado, mas para também atualizar os componentes de Visão.

A Arquitetura Híbrida é o único tipo de arquitetura que pode ser utilizada na implementação do protótipo descrito no Capítulo 4, uma vez que o mapeamento proposto não sugere nenhuma modificação estrutural no software básico original. Esta restrição traz para a abordagem proposta todas as vantagens e desvantagem que a arquitetura Híbrida proporciona.

Atualmente, o estilo arquitetural MVC é mais comum em softwares não colaborativos, em particular nos softwares desenvolvidas para Internet, pois fornece aos desenvolvedores uma organização de componentes desejável para este contexto. Com modificações específicas nos componentes organizados de acordo com o estilo arquitetural MVC, um software pode ser desenvolvido para gerar aplicações colaborativas derivadas de aplicações não colaborativas. Por exemplo, o sistema Groove [11] mostrou que o estilo arquitetural MVC pode ser modificado para incluir um processador de comando que intercepta mudanças no modelo local e manda estas mudanças para os outros agentes do MVC no momento que as modificações acontecem ou quando os outros agentes se conectam à sessão colaborativa.

2.7.2 Arquiteturas para o Desenvolvimento de Software Colaborativo

Estilos arquiteturais específicos para o desenvolvimento de software surgiram ao mesmo tempo em que os sistemas *groupware* tornaram-se populares. As arquiteturas colaborativas foram modeladas, por meio da expansão e modificação dos componentes e de sua organização.

Dewan [62] propõe uma Arquitetura Colaborativa Genérica que estrutura um *groupware* em um número variável de camadas que compreendem desde o nível de domínio do problema até o nível de hardware. Nesta arquitetura, apresentada na Figura 2.8, uma camada é considerada um componente de software que corresponde a um nível específico de abstração.

Uma camada de nível inferior, isto é, uma camada que está mais perto do usuário, é responsável pelo gerenciamento de objetos que interagem com os objetos na camada imediatamente superior. A camada inferior desta arquitetura representa as estações de trabalho (sistema operacional e o hardware), responsáveis pelo gerenciamento dos dispositivos de entrada e saída da aplicação.

A comunicação entre as camadas é feita por meio de eventos. Neste contexto, os eventos são considerados mecanismos de alto nível que não devem ser classificados com base na simultaneidade de comunicação entre as camadas.

A camada superior da arquitetura é chamada de camada semântica, pois contém objetos que representam, internamente, os objetos abstratos que o usuário do software colaborativo utiliza no contexto da sua tarefa.

Apesar de ser considerado uma Arquitetura Colaborativa Genérica, nem todos os módulos de um software colaborativo são organizados nas camadas abstratas apresentadas na arquitetura da Figura 2.8. As camadas e os módulos de uma arquitetura colaborativa incluem componentes da aplicação, implementados para prover as funcionalidades colaborativas do software, e componentes do sistema, que suportam a infra-estrutura necessária na colaboração. Indicar a arquitetura de uma aplicação colaborativa na verdade é caracterizar a interação e os aspectos dos componentes definidos pela aplicação.

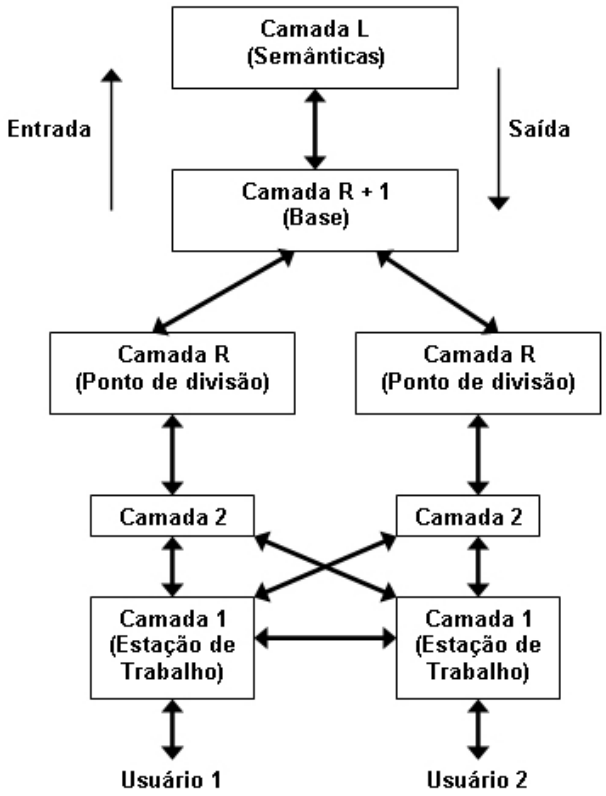

Figura 2.8 - Arquitetura Colaborativa Genérica. Traduzida de [62].

PAC* [26] é uma extensão colaborativa do modelo arquitetural PAC estruturado de acordo com o estilo sugerido pela Arquitetura Colaborativa Genérica de Dewan, mas com a diferença de permitir diferentes divisões e junções. O modelo PAC foi estendido para permitir que os controladores não apenas se comuniquem com os controladores de outros agentes do software colaborativo, mas que também interajam com outros softwares que utilizam o PAC*, sejam eles colaborativos ou não. A Figura 2.9 apresenta o Modelo arquitetural PAC*.

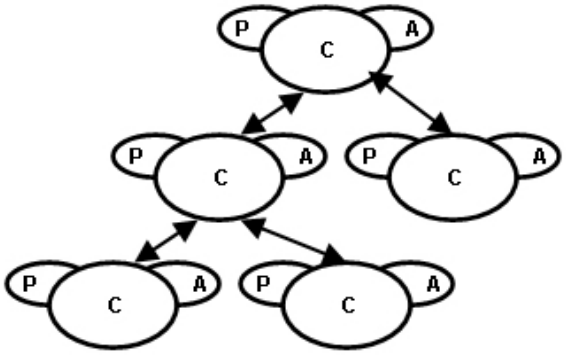

Figura 2.9 - Modelo arquitetural PAC* [26].

O modelo arquitetural Clover [74] é derivado da arquitetura de Dewan [62] e utiliza as camadas do modelo Arch [1]. Contudo, no lugar das cinco camadas do modelo

Arch, o modelo Clover contêm seis camadas. Os componentes do Núcleo Funcional (FC) foram separados em dois componentes: os componentes de Núcleo Funcional compartilhados e os componentes de Núcleo Funcional replicados. O metamodelo Clover, uma generalização do estilo arquitetural Clover, pode ser construído sobre outras arquiteturas e estilos arquiteturais como o Arch, o PAC* ou o MVC. A Figura 2.10 apresenta o metamodelo Clover.

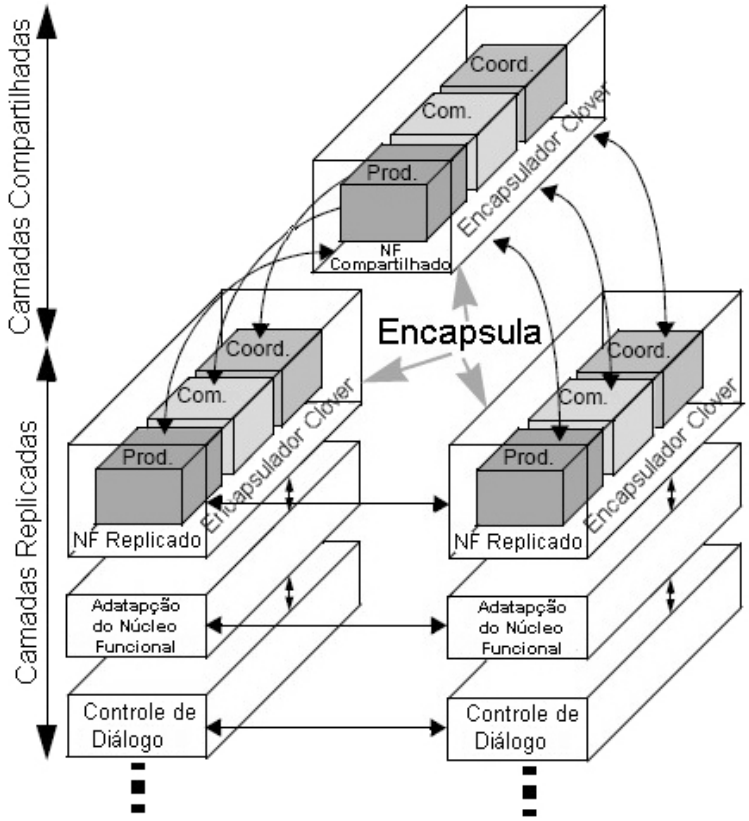

Figura 2.10 - Modelo arquitetural Clover. Traduzido de [74].

Outras abordagens para arquiteturas colaborativas incluem o estilo arquitetural Clock [58], o sistema Suite [61] e os estilos arquiteturais ALV (*Abstraction, Link, View*) [64] e Chiron-2 [69].

2.8 Trabalhos Correlatos

Esta seção descreve as abordagens existentes para construir aplicações colaborativas e para transformar aplicações não colaborativas existentes em aplicações

colaborativas. As principais abordagens incluem o uso de *toolkits*, sistemas de colaboração transparente, substituição dos componentes da aplicação por componentes colaborativos e a estratégia de adaptação transparente.

2.8.1 *Toolkits*

Numa tentativa de reduzir a complexidade de implementação de sistemas *groupware*, vários pesquisadores têm explorado o uso de *toolkits*. Um *toolkit* é definido como um conjunto de componentes predefinidos e reutilizáveis, com a finalidade de oferecer ferramentas e infra-estrutura suficientes para permitir que o programador desenvolva sistemas *groupware* de alta qualidade com razoável esforço [71].

Os *toolkits* para a construção de sistemas *groupware* facilitam a criação de novas aplicações colaborativas por meio de componentes de grupo e controles de percepção construídos para serem reutilizáveis, permitindo a criação de poderosas aplicações colaborativas. Da mesma forma que casas pré-fabricadas são construídas a partir da montagem de partes individuais, o uso de *toolkits* na construção de software colaborativo tornam a tarefa do desenvolvedor mais simples (e com sorte mais rápida), uma vez que o trabalho principal, desta maneira, consistirá apenas na montagem e configuração dos componentes em vez da criação deles a partir do zero. Pode-se dizer que o uso de *toolkits* para a construção de sistemas *groupware* segue os princípios envolvidos no Desenvolvimento Baseado em Componentes.

Muitos *toolkits* amplamente utilizados como o GroupKit [9], o COAST (*COoperative Application Systems Toolkit*) [13] e o MAUI (*Multi-User Awareness UI toolkit*) [34], fornecem *frameworks*, ambientes, bibliotecas de funções, componentes e controles para tornar mais rápida e fácil a construção de novas aplicações colaborativas.

Além de reduzir o esforço necessário e acelerar o desenvolvimento de aplicações colaborativas, o reuso de componentes de um *toolkit* oferece um conceito comum para o desenvolvimento de aplicações, o que pode auxiliar os desenvolvedores de software colaborativo. Com o objetivo de fornecer um conjunto valioso e amplamente aplicável de componentes de *software*, o desenvolvedor do *toolkit* deve antecipar as necessidades dos usuários prevendo as funcionalidades dos softwares colaborativos que serão criadas a partir dos componentes do *toolkit*.

O reuso dos componentes de *toolkits* pode ajudar o desenvolvimento de novos softwares colaborativos, entretanto, os componentes fornecidos pelo *toolkit* não podem auxiliar a adaptação de um software não colaborativo existente para que ele apresente recursos para a colaboração síncrona. Em geral isso acontece devido à necessidade de implementação de detalhes específicos do *toolkit*, que nem sempre são compatíveis com a estrutura da maioria das aplicações existentes.

Um exemplo desta incompatibilidade é a necessidade de sobrescrever somente certos métodos de alguns componentes fornecidos pelo *toolkit*. Além disso, detalhes técnicos como a linguagem de programação e a arquitetura utilizada na construção dos componentes do *toolkits* podem tornar o reuso dos componentes inviável em certas situações. Mesmo nos casos onde os detalhes técnicos não inviabilizam a utilização do *toolkit*, a quantidade de esforço necessário para a reutilização dos componentes pode não compensar.

Apesar dos *toolkits* não serem úteis para ajudar a extensão de um software existente, eles podem fornecer inspiração e idéias necessárias para implementar aspectos essenciais de comunicação, colaboração e cooperação em softwares existentes.

2.8.2 Sistemas de Colaboração Transparente

Os sistemas de colaboração transparente fornecem um meio de compartilhar o uso de aplicações existentes de forma síncrona. O compartilhamento fornecido por esta abordagem é transparente ou desconhecido para a aplicação e seus desenvolvedores [33]. A emulação das entradas e saídas de dados dos usuários da aplicação fornece aos usuários a impressão de que todos utilizam a mesma interface da aplicação, ao mesmo tempo e no mesmo local.

Estes sistemas utilizam o conceito de caixa preta de colaboração transparente, onde alguma aplicação externa é responsável pelas comunicações e notificações necessárias para apoiar aspectos básicos da colaboração entre os usuários remotos. Os mecanismos por trás desses sistemas incluem uma arquitetura centralizada, a captação e entrega dos dados de entrada dos usuários para múltiplos locais e a transmissão da interface gráfica da aplicação para todos os participantes.

Exemplos de sistemas de colaboração transparente comerciais incluem o Microsoft Netmeeting [43], o XTV [31] e o SharedX [15].

Colaboração Transparente Inteligente, ou ICT (*Intelligent Collaboration Transparency*) [20], apresenta uma evolução dos sistemas de colaboração transparente, pois eles utilizam uma infra-estrutura compartilhada entreposta entre aplicações heterogêneas e o sistema operacional para compartilhar os ambientes gráficos de cada local.

Devido à heterogeneidade das aplicações envolvidas, a simples reprodução dos eventos de uma aplicação em outras aplicações diferentes não faz sentido. Uma operação semântica de uma aplicação geralmente é implementada de forma diferente em outra aplicação, requerendo um mecanismo que, além de compreender as operações dos usuários, deve traduzir essas operações em seqüência de operações equivalentes a outras aplicações, antes que elas sejam reproduzidas. Deste modo, o mecanismo deve ser inteligente o suficiente para traduzir as operações de uma aplicação em uma seqüência de operações compatíveis em outras aplicações, por meio do conhecimento semântico das ações realizadas pelos usuários.

Com o uso deste tipo de sistemas, os usuários podem colaborar em uma tarefa comum, através de suas aplicações favoritas. Contudo, um módulo de captura/reprodução de eventos específico para cada plataforma deve ser utilizado para tratar da comunicação e do controle de concorrência, propiciando a interoperabilidade entre as aplicações.

Uma segunda geração de aplicações baseadas na abordagem do ICT foi elaborada por Lu et al. [36], denominada ICT2. A principal diferença entre a abordagem ICT clássica e a ICT2 é que a última não intercepta e compreende os eventos da aplicação. Em vez disso, a ICT2 utiliza uma versão adaptada de um algoritmo que gera as seqüências de edição entre os diferentes estados dos documentos locais dos usuários. Para juntar as ações concorrentes dos usuários, uma versão modificada do algoritmo de transformação operacional é utilizada.

Mangan [45] propõe uma variação da técnica de colaboração transparente. Esta proposta chama-se Arquitetura de Colaboração Transparente (ACT) e amplia as arquiteturas transparentes com o acréscimo do conceito de eventos semânticos. Neste contexto, os eventos descrevem o significado das operações realizadas pelo usuário da

aplicação e são gerados a partir de interpretações dos eventos de entrada e dos eventos de aplicação das arquiteturas transparentes. Com estes eventos, é possível alimentar modelos de percepção que oferecem informações úteis para mecanismos: de apoio à colaboração assíncrona; de descoberta de perfil de usuário; e de histórico das experiências dos usuários [45].

Apesar de apresentar uma alternativa rápida para compartilhar aplicações sem modificações no código fonte, os sistemas de colaboração transparente, a ACT e as abordagens ICT e ITC2 receberam muitas críticas afirmando que estes sistemas não suportam a colaboração adequadamente. O motivo principal dessas críticas envolve o uso ineficiente dos recursos de rede e a imposição de um estilo de colaboração inflexível e altamente acoplado que não suporta algumas funcionalidades comuns aos sistemas *groupware* como, por exemplo, controle de concorrência, interface WYSIWIS relaxada, percepção em grupo e tarefas colaborativas delegadas [33].

2.8.3 Substituição de Componentes

A abordagem de substituição de alguns componentes de uma aplicação por componentes colaborativos foi introduzida pelo Flexible JAMM (*Java Applets Made Multi-user*) [33]. Esta abordagem permite múltiplos estilos de colaboração, através da substituição dinâmica de certos objetos da interface do usuário por componentes de interfaces colaborativas. Para utilizar esses componentes colaborativos, a aplicação que receberá os componentes deve possuir certas características como, por exemplo, a capacidade para migração de processos, recursos para a substituição de componentes em tempo de execução e a habilidade para interceptar e reproduzir os eventos gerados pelo usuário.

Os componentes da interface gráfica de usuário que o Flexible JAMM fornece incluem mecanismos de percepção como barras de rolagem multiusuário, visão radar e *telepointers*. Estes componentes propiciam, respectivamente, nomes de usuários em diferentes barras de rolagem, visão em miniatura da área de trabalho comum e ponteiros virtuais representando usuários remotos. Devido às suas funcionalidades e à capacidade de fornecer a percepção de usuários remotos, é comum encontrar estes componentes apenas em sistemas *groupware* especializados.

A principal limitação desta abordagem reside no fato que os componentes oferecidos pelo Flexible JAMM somente suportam os aspectos de percepção durante a colaboração, deixando a cargo do desenvolvedor todo o trabalho restante para implementar os demais aspectos colaborativos. Além disso, somente um pequeno grupo de aplicações existentes possui todos os requisitos necessários para o uso dos componentes do Flexible JAMM. Estas características limitam o uso da abordagem a poucas aplicações, evitando que esta abordagem possua amplo uso e possa ser utilizada de forma sistemática em softwares de diferentes domínios de conhecimento.

2.8.4 Adaptação Transparente

A abordagem chamada Adaptação Transparente foi proposta para ajudar a implementar a colaboração em aplicações comerciais que não possuem o código fonte disponível. Esta abordagem é baseada no compartilhamento da aplicação e no uso de uma Interface de Programa de Aplicação (API) para interceptar as interações locais do usuário, convertê-las em operações abstratas, manipular estas operações por técnicas colaborativas e reproduzir as operações modificadas, por meio da API, nos locais remotos de colaboração [73].

O termo transparente é utilizado por que esta abordagem não requer nenhuma mudança no código fonte da aplicação. Contudo, esta abordagem requer que o fabricante forneça uma API capaz de gerenciar os dados da aplicação e implementar os mecanismos de colaboração.

Diferentemente dos ambientes de compartilhamento de aplicações, os quais não requerem nenhum tratamento específico, a abordagem de Adaptação Transparente requer uma nova camada de software para implementar a arquitetura replicada que fornecerá o compartilhamento de eventos e dados da aplicação. De acordo com Xia et al. [73], a combinação da arquitetura replicada e da abordagem de Adaptação Transparente permite vários benefícios como, por exemplo, uma grande quantidade de interações entre os usuários, um controle de concorrência eficiente e uma melhor interface WYSIWIS relaxada.

Desta maneira, o código fonte da aplicação não é necessário. Contudo, uma API completa, e que forneça o acesso à eventos e dados gerados pela aplicação deve ser fornecida.

Mesmo com a API mais completa, alguns métodos específicos de controle de concorrência e dispositivos de presença são impossíveis de implementar na abordagem de Adaptação Transparente, devido à necessidade da mudança no código fonte da aplicação para implementar personalizações e fornecer controles compartilhados na interface de usuário.

Em comparação com as abordagens de *toolkits*, de sistemas de colaboração transparente e de substituição de componentes, a principal vantagem da abordagem de Adaptação Transparente é a separação da construção de um *groupware* da construção de camadas de *software*, que propicia as suas funcionalidades colaborativas. Portanto, a abordagem de Adaptação Transparente implica no desenvolvimento de uma camada de software para capturar e replicar os eventos gerados pelas aplicações. A Adaptação Transparente tem sido utilizada com sucesso para permitir que algumas aplicações comerciais sejam utilizadas, de forma colaborativa. Contudo, não há relatos de pesquisa que forneçam detalhes sobre como aplicar a abordagem de Adaptação Transparente em aplicações que possuem o código fonte disponível.

2.8.5 Modelo 3C e Engenharia de Groupware

Além das abordagens para o desenvolvimento de software colaborativo, outras abordagens, visando à metodologia, foram desenvolvidas para auxiliar a criação de software colaborativo. Esta seção descreve sumariamente o Modelo 3C, proposto originalmente por Ellis et al. [10], e a Engenharia de Groupware, proposta por Fuks et al. [29].

O modelo 3C de colaboração é baseado na concepção de que para a colaboração entre usuários são necessárias comunicação, coordenação e cooperação entre eles. De acordo com Fuks [30], para colaborarem, os usuários têm que trocar informações (se comunicar), organizar-se (se coordenar) e operar em conjunto num espaço compartilhado (cooperar). É com base na Comunicação, Coordenação e Cooperação que o modelo 3C é definido.

As interações ocorridas durante a comunicação geram compromissos gerenciados pela coordenação, que por sua vez organiza e dispõe as tarefas executadas na cooperação. Durante a cooperação, os usuários têm necessidade de se comunicar para renegociar e para tomar decisões sobre situações não previstas inicialmente. Isso mostra o aspecto cíclico da colaboração. Através da percepção, o usuário informa-se sobre o que está acontecendo, o que outros usuários estão planejando e fazendo, além de adquirir informações necessárias para o seu trabalho. A Figura 2.11 apresenta o diagrama do Modelo 3C, obtido a partir do refinamento do modelo 3C apresentado originalmente por Ellis et al. [10]. Pode-se notar no diagrama o aspecto cíclico da colaboração.

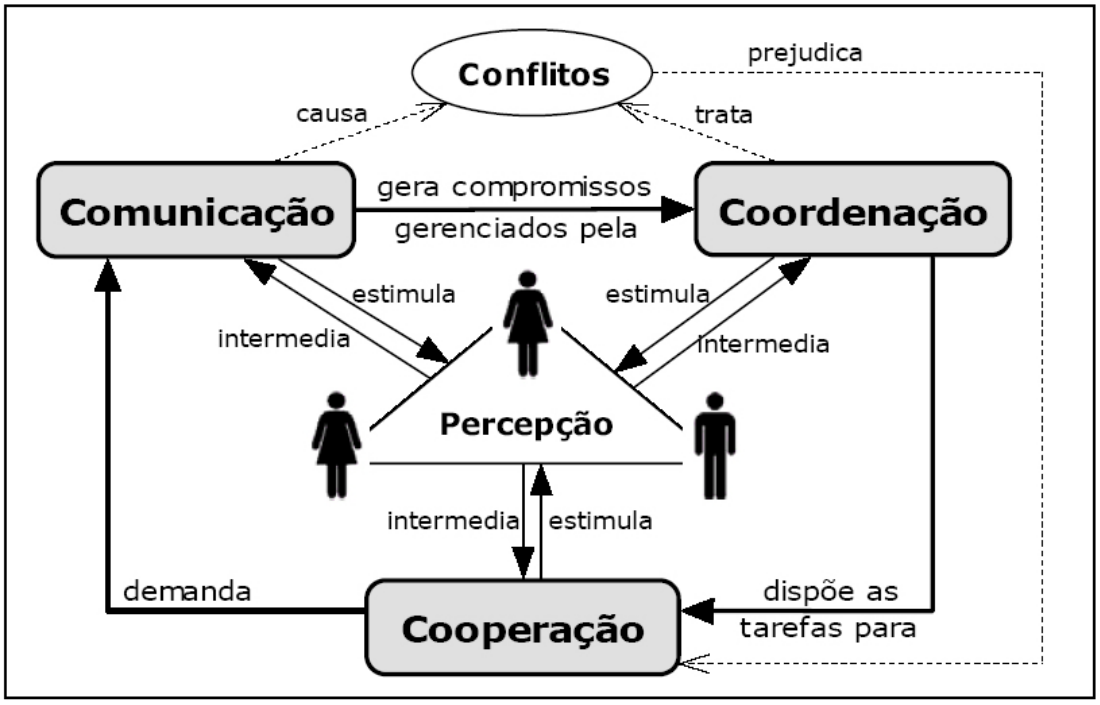

Figura 2.11 - Modelo 3C [30].

O modelo 3C é similar ao modelo Clover [74], descrito na Seção 2.6. O que é chamado de produção no modelo Clover corresponde ao conceito de cooperação no modelo 3C. Diferentemente do modelo Clover, o modelo 3C guia o desenvolvimento de software colaborativo e dá origem a uma arquitetura componentizada.

A partir do Modelo 3C, Fuks et al. [30] propõem a Engenharia de Groupware com o sentido de abordar isoladamente as alternativas para o desenvolvimento de aplicações colaborativas.

A Engenharia de Groupware é definida como um processo sistemático, disciplinado e quantificável para o desenvolvimento e manutenção de sistemas *groupware*. Baseada nos princípios da Engenharia de Software, a Engenharia de Groupware une conceitos e tecnologias das áreas HCI (*Human Computer Interaction)* e CSCW para instrumentar o desenvolvimento e evolução de sistemas *groupware* [30].

A partir do uso do Modelo 3C e dos princípios da Engenharia de Groupware, alguns trabalhos voltados para o desenvolvimento de sistemas *groupware* foram desenvolvidos. Dentre estes trabalhos, destacam-se a abordagem proposta por Gerosa [47] para o desenvolvimento de *groupware* baseado em componentes e a proposta de Pimentel [53], que sugere um processo de desenvolvimento de *groupware* usando o Modelo 3C de Colaboração em diferentes etapas do processo, denominado RUP-3C-Groupware, estendendo o processo unificado RUP (*Rational Unified Process*) para o desenvolvimento de *groupware*.

2.9 Sumário

Este capítulo apresentou os conceitos necessários para contextualizar esta tese. Iniciou-se com uma introdução do conceito de Trabalho Cooperativo Suportado por Computador, passando em seguida para a apresentação da UML e ferramentas CASE. Em seguida, os conceitos quem envolvem o Desenvolvimento Baseado em Componentes, os *frameworks* e os Padrões de projeto foram discutidos no contexto em que eles serão utilizados. O capítulo apresentou ainda uma discussão sobre as principais arquiteturas de software para a criação de aplicações não colaborativas e colaborativas, com exemplos das arquiteturas relacionadas com esta tese. O capítulo termina apresentando as principais abordagens existentes para transformar aplicações não colaborativas em aplicações colaborativas.

No capítulo seguinte, o mapeamento de componentes será apresentado, mostrando-se os requisitos levantados e o modo como eles serão endereçados pelo mapeamento.

Capítulo III

Mapeamento de Componentes

Este capítulo apresenta um mapeamento de componentes de uma aplicação não colaborativa para os componentes de uma aplicação colaborativa. Inicialmente, a especificação de um conjunto mínimo de requisitos colaborativos funcionais de alto nível é apresentada, com o objetivo de identificar quais as funcionalidades colaborativas devem ser implementadas para que uma aplicação não colaborativa apóie a colaboração.

Em seguida, o mapeamento entre os principais componentes de uma aplicação genérica que segue o estilo arquitetural MVC e os componentes de uma aplicação colaborativa é apresentado com o objetivo de atender aos requisitos de colaboração. O uso do mapeamento gera uma aplicação distribuída que utiliza a arquitetura híbrida para a comunicação entre Modelos.

Por fim, o capítulo apresenta uma discussão sobre algumas questões referentes à utilização do mapeamento e as adaptações nos componentes para apoiar a colaboração.

3.1 Requisitos de Colaboração

As aplicações que contêm funcionalidades essenciais de colaboração possuem requisitos específicos que não são contemplados em aplicações não colaborativas. Os pesquisadores da área de CSCW vêm estudando diversos tipos de requisitos para aplicações colaborativas, no decorrer dos últimos 20 anos, desde requisitos relacionados com a interface do usuário até requisitos de estilos arquiteturais. Os requisitos descritos nesta seção são provenientes das áreas de CSCW e encontrados com freqüência em *groupware*.

A especificação de requisitos para uma aplicação colaborativa deve levar em consideração vários aspectos funcionais, sobretudo os aspectos relacionados com a comunicação, colaboração e coordenação. Além destes requisitos, é necessário que a aplicação também atenda às necessidades do domínio em questão como, por exemplo, requisitos característicos de editores de texto, modelagem, desenho etc. Nesta seção, apenas os requisitos de colaboração serão abordados.

Na Engenharia de Groupware descrita na Seção 2.7.4, Fuks et al. [29], citando Tietze [16], propõem a divisão de requisitos de *groupware* em duas categorias: os requisitos de usuário de sistemas *groupware* e os requisitos para desenvolvimento de sistemas *groupware*. Devido ao escopo deste trabalho, apenas os requisitos de usuários serão abordados nesta seção.

Os requisitos de usuário são apresentados por meio de cenários fictícios, envolvendo diferentes domínios de conhecimento, onde usuários dispostos em localidades distintas fazem uso de sistemas *groupware* para realizar suas tarefas. Sob o ponto de vista do usuário, que se utiliza de um sistema *groupware*, Fuks et al. enumeram doze requisitos de usuários, resumidos na Tabela 3.1. Estes requisitos são apresentados com um identificador, um nome e uma breve descrição de seu propósito/função.

Apesar de representarem bem as necessidades dos usuários que se utilizam de sistemas de *groupware*, os requisitos levantados por Fuks et al. são genéricos e nem sempre devem ser implementados. Por exemplo, em uma aplicação de correio eletrônico (e-mail), o requisito Desempenho não precisa ser implementado, pois é comum a latência do sistema, ou seja, o tempo decorrido entre o envio de uma mensagem e o recebimento da mesma, apresentar uma grande variação. Já em sistemas *groupware* que utilizam vídeo conferência, este requisito é crucial para que os usuários possam realizar suas tarefas com sucesso.

Como o objetivo do mapeamento proposto neste trabalho é permitir que aplicações não colaborativas sejam transformadas em *groupware*, faz-se necessário especificar um conjunto mínimo de requisitos colaborativos de usuário. Esses requisitos devem levar em consideração vários fatores como o nível de interação dos usuários, os tipos de objetos a serem compartilhados, os tipos de usuários que utilizam o sistema e as dificuldades de implementação. Com a implementação deste conjunto mínimo de

requisitos em aplicações não colaborativas, os usuários poderão dispor de um software colaborativo para realizar seus trabalhos.

Tabela 3.1 – Requisitos de usuário em um *groupware* [29].

Identificador do Requisito	Nome	Descrição
RU1	Acesso aos objetos compartilhados e às ferramentas de colaboração	Prover fácil acesso aos objetos e às ferramentas de colaboração.
RU2	Auxílio na escolha das ferramentas apropriadas	Saber qual ferramenta é apropriada para executar determinada tarefa em um tipo de objeto.
RU3	Elementos de percepção	Prover informações de percepção do grupo através de elementos de percepção do ambiente.
RU4	Colaboração síncrona e assíncrona	Fornecer serviços de colaboração entre os participantes.
RU5	Acesso ao ambiente independente da estação de trabalho	Independência do ambiente com relação à estação de trabalho.
RU6	Espaço privativo e público e a transição entre eles	Prover recursos para facilitar a atuação individual e coletiva.
RU7	Extensão dinâmica do ambiente	Capacidade do ambiente de incorporar e disponibilizar aos usuários novas ferramentas sem ser reinicializado.
RU8	Sincronização entre ferramentas diferentes	Sincronização e atualização de diferentes ferramentas que operam sobre um mesmo objeto compartilhado.
RU9	Mobilidade	Possibilitar acesso através de dispositivos móveis longe de uma estação de trabalho.
RU10	Agrupamento de ferramentas	Prover recursos para que os usuários agrupem ferramentas.
RU11	Desempenho	Baixa latência do sistema e notificação automática de modificação nos artefatos do ambiente de trabalho.
RU12	Uso das ferramentas de trabalho individual para o coletivo	Prover recursos para que ferramentas de trabalho monousuárias sejam usadas de forma colaborativa.

Para facilitar a escolha dos requisitos que farão parte do conjunto mínimo de requisitos endereçados pelo mapeamento proposto, este trabalho segue os princípios do Modelo 3C apresentado na Seção 2.7.4. Desta maneira, ao menos um requisito relacionado com a Coordenação, à Comunicação e à Cooperação deve ser especificado para o conjunto mínimo de requisitos abordados pelo mapeamento de componentes proposto.

Além de considerar os componentes do Modelo 3C, a escolha dos requisitos mínimos também levou em consideração a análise de várias ferramentas colaborativas, *toolkits*, *frameworks* e componentes que auxiliam o desenvolvimento de aplicações colaborativas. Os requisitos colaborativos que fazem parte do conjunto mínimo de requisitos abordados pelo mapeamento são descritos a seguir e, sempre que possível, sua função será comparada com os requisitos da Tabela 3.1.

O requisito **RU12** destaca-se dentre os demais por apresentar o problema de pesquisa deste trabalho. Prover recursos para que ferramentas de trabalho monousuárias sejam usadas de forma colaborativa é uma tarefa complexa que demanda muitos recursos. Como apresentado no Capítulo 2, algumas abordagens já existem para resolver este problema. Contudo, a solução proposta neste trabalho, um mapeamento de componentes, traz contribuições para um caso particular de ferramentas de trabalho monousuário: as aplicações que seguem o estilo MVC e que possuem código livre. Além do mais, o resultado da aplicação do mapeamento dá origem a uma aplicação distribuída que utiliza uma arquitetura específica, a arquitetura híbrida.

O apoio à comunicação geralmente é citado como um requisito para sistemas colaborativos. Os sistemas que permitem a colaboração síncrona de usuários distantes, em particular, fornecem meios para que os usuários saibam o que os outros usuários pretendem realizar e suas intenções. O canal utilizado pelos usuários para se comunicar e trocar pensamentos e idéias durante a colaboração pode assumir diferentes mídias como texto, áudio e vídeo. Neste contexto, o apoio à comunicação envolve conceitos dos requisitos **RU3**, **RU4, RU6** e **RU11**.

Um dos aspectos mais proeminentes de aplicações colaborativas é a percepção em grupo, que tem como objetivo ajudar os usuários a coordenar os seus trabalhos. Durante o trabalho em grupo, a percepção tem grande importância e se relaciona com a coordenação, comunicação e cooperação.

Nas aplicações colaborativas, a percepção das interações dos usuários com a aplicação permite que um participante fisicamente separado esteja ciente da presença e das ações dos demais participantes da colaboração, facilitando o trabalho em conjunto. A percepção também permite socializar, virtualmente, o ambiente e pode servir para indicar o esforço na interação com a aplicação e com os demais participantes. A percepção se relaciona diretamente com os requisitos **RU3** e **RU12**.

Para apresentar informações de percepção aos membros do grupo, a aplicação deve capturar os relacionamentos entre os usuários e as aplicações com os artefatos atualmente em uso, isto é, o contexto do trabalho colaborativo.

Nos sistemas de colaboração síncrona, os usuários não devem ser responsáveis pela solicitação periódica de novas informações sobre a presença dos usuários e à percepção do status da colaboração. Eles devem ser notificados, automaticamente, quando mudanças relevantes no artefato compartilhado são feitas, ou quando o estado de algum dos usuários do grupo for modificado. Esta notificação se remete diretamente ao requisito **RU11**.

A colaboração deve ser estruturada, de acordo com o contexto colaborativo. Por exemplo, grupos de trabalhos são estabelecidos quando os usuários iniciam sua participação na mesma sessão colaborativa. Outros usuários podem criar novas sessões colaborativas para iniciar novos grupos de trabalho ou podem entrar em uma sessão colaborativa em andamento. O gerenciamento de sessões colaborativas também deve possuir controles de segurança, para que somente usuários autorizados possam colaborar com os demais usuários em uma tarefa ou trabalho. O conceito de sessão colaborativa está intimamente ligado à questão do acesso aos objetos compartilhados, relacionando-se com o requisito **RU1.**

Durante o uso das aplicações não colaborativas, os usuários detêm controle total sobre o documento em uso. Já nas aplicações colaborativas, é necessário implementar alguma técnica de controle de concorrência para coordenar as ações concorrentes e para definir as permissões que os usuários devem possuir sobre partes do documento compartilhado pelo grupo. Como este conceito envolve uma versão do objeto compartilhado privada do usuário, este requisito pode ser associado ao requisito **RU6**, pois uma visão privada implica em um espaço privativo de trabalho.

Uma vez que os principais mecanismos de controle de acesso ao documento compartilhado são baseados na noção de unidade mínima, é recomendado que os desenvolvedores de sistemas *groupware* especifiquem qual parte do artefato representa a unidade mínima de controle e qual mecanismo de controle irá gerenciar os acessos concorrentes a estas unidades mínimas.

Em resumo, os seguintes requisitos colaborativos mínimos devem ser implementados nas aplicações não colaborativas para que elas possam apresentar

aspectos básicos de colaboração síncrona, além dos requisitos específicos do domínio da aplicação: **comunicação**, **percepção em grupo**, **gerenciamento de sessão** e **controle de concorrência**.

Do ponto de vista da Engenharia de Software, outros requisitos adicionais podem ser identificados. Talvez os requisitos mais relevantes, para os usuários, sejam os requisitos de usabilidade, que determinaram se os usuários se sentem confortáveis com as novas funcionalidades colaborativas e se eles as usaram, o que pode determinar o sucesso de um sistema *groupware*.

É desafiador especificar e construir um sistema *groupware* que apóie efetivamente os requisitos mínimos mencionados, especialmente se forem levados em consideração as características e o conjunto completo de funcionalidades específicas de um domínio de conhecimento que as aplicações existentes possuem. Entretanto, mesmo aplicações complexas seguem regras básicas de armazenamento e manipulação de dados, tais como estruturas de dados e operações que manipulam e apresentam os dados para os usuários. Em geral, essas estruturas e aplicações são maleáveis e podem ser modificadas para apoiar novas operações e estruturas necessárias para implementar requisitos colaborativos.

Outras restrições que afetam a aplicação precisam ser consideradas, como os requisitos para segurança, escalabilidade, largura de banda da rede, diversidade de plataformas e integrações com outras aplicações. Para um projeto efetivo de uma aplicação colaborativa, os desenvolvedores devem entender não só os requisitos para o desenvolvimento de sistema *groupware*, mas também todos os outros requisitos funcionais e não funcionais, além das restrições do sistema e de seu ambiente de execução. Além disso, para permitir que o sistema evolua para a colaboração, os desenvolvedores devem considerar, além das necessidades imediatas dos usuários, a antecipação das principais necessidades futuras dos usuários.

3.2 Mapeamento dos Componentes do MVC

A capacidade de adaptação aos novos requisitos e mudanças de funcionamento está se tornando cada vez mais importante nos softwares atuais. Contudo, as modificações e adaptações nem sempre são facilmente implementadas, devido tanto à

maneira pela qual a aplicação foi desenvolvida e estruturada como aos detalhes necessários para implementar as mudanças. Desta maneira, o mapeamento apresentado nesta seção fornece um guia com heurísticas indicando modificações dos componentes da aplicação.

Utilizando este mapeamento, aplicações não colaborativas baseadas no estilo arquitetural MVC podem ser transformadas, permitindo-se a colaboração de acordo com o conjunto mínimo de requisitos especificados. Uma vez que existe uma grande variedade de aplicações baseadas no estilo arquitetural MVC, o mapeamento pode ajudar os desenvolvedores a implementar funcionalidades colaborativas em suas aplicações, expandindo os benefícios do uso de sistemas *groupware*.

De acordo com Schuckman et al. [13], mudar o foco do desenvolvimento de aplicações não colaborativas para aplicações colaborativas requer mais do que compartilhar artefatos comuns e conectar um conjunto de interfaces distribuídas. Deste modo, o mapeamento proposto permite um tratamento uniforme dos requisitos colaborativos mínimos apresentados na Seção 3.1, em um nível alto de abstração, com o objetivo de fornecer um ponto de partida para os desenvolvedores de aplicações, sob o ponto de vista conceitual.

Um dos requisitos para o uso do mapeamento envolve a necessidade de organização dos componentes da aplicação, assim como os meios para manipular estes componentes. A Subseção 2.6.1 apresentou alguns estilos arquiteturais com o objetivo de auxiliar a organização dos componentes de aplicações não colaborativas e colaborativas.

O estilo arquitetural MVC proporciona uma flexibilidade para os desenvolvedores de aplicações, fornecida graças ao nível de acoplamento entre os componentes utilizados na aplicação, e também graças à definição de como os componentes interagem entre si. O emprego do estilo arquitetural MVC pressupõe que os principais componentes da aplicação encontram-se divididos e separados de acordo com a sua funcionalidade, fornecendo os alicerces para o funcionamento da aplicação. Devido a essas características do estilo arquitetural MVC e da sua popularidade em aplicações não colaborativas, este estilo arquitetural é um requisito das aplicações que desejam utilizar o mapeamento.

A obrigatoriedade do estilo arquitetural MVC nas aplicações alvo simplifica o uso do mapeamento e também evita que aplicações imaturas, isto é, aplicações que não possuem um nível de maturidade ideal para a utilização do mapeamento, sejam utilizadas. Além disso, exigir que as aplicações alvo organizem as interações dos seus componentes de acordo com o MVC permite a separação e a identificação de quais componentes e quais interação entre estes componentes devem sofrer modificações. Deste modo, pode-se dizer que o mapeamento foi feito exclusivamente para ser utilizado em aplicações estruturadas de acordo com o estilo arquitetural MVC.

Uma aplicação genérica, que segue o estilo MVC, será utilizada para facilitar a especificação do mapeamento entre os componentes. Esta aplicação genérica contém componentes de acordo com suas funcionalidades e com a parte da arquitetura onde o componente se encontra.

3.2.1 Análise dos Componentes

Partindo de uma aplicação genérica que utiliza o estilo arquitetural MVC, supõe-se que os componentes da aplicação encontram-se distribuídos de uma maneira lógica de acordo com suas funcionalidades.

Para os componentes da Visão, é normal encontrar componentes que gerenciam a visualização dos dados por meio de uma interface de usuário. Os componentes que fazem parte do Controle são responsáveis pela captura das interações do usuário e validação das informações fornecidas ao sistema. Os componentes que pertencem ao Controle também são responsáveis pelo gerenciamento de dados específicos do usuário enquanto ele faz uso do sistema e pela tradução das interações dos usuários em operações que modificam o estado do Modelo. Os componentes do Modelo contêm classes que permitem a persistência de objetos e também classes personalizadas que representam regras de negócio específicas do domínio da aplicação. Existem ainda os componentes de infra-estrutura, como componentes para registros ou depuração, distribuídos em todas as partes do estilo arquitetural MVC.

Deste modo, alguns componentes típicos podem ser associados a partes de uma aplicação genérica que segue o estilo MVC. Esta associação tem como objetivo facilitar o mapeamento dos componentes, de acordo com os requisitos colaborativos mínimos

especificados e também decompor a aplicação em termos das funcionalidades dos seus componentes.

A parte da Visão contém componentes de apresentação de informações por meio de uma interface com o usuário. A parte do Controle contém os componentes responsáveis pela interação com o usuário e também os componentes de controle de sessão de usuário e validação de informações. A parte do Modelo contém componentes responsáveis pelo modelo de dados, lógica da aplicação, regras de negócio e persistência de dados. Sob o ponto de vista do mapeamento, somente estes componentes da aplicação MVC genérica serão considerados.

Em aplicações reais, nem sempre os componentes de uma aplicação MVC são programados pelo desenvolvedor da aplicação. Por exemplo, os componentes responsáveis pelo tratamento dos eventos gerados na aplicação encontram-se localizados na parte do Controle e podem ser implementados pelo uso de um *framework*. Estes componentes geralmente definem como a aplicação vai receber e tratar os eventos da aplicação e do usuário, devendo ser personalizados de acordo com a funcionalidade da aplicação. Contudo, em um nível alto de abstração, estes componentes não deixam de ser classificados como componentes responsáveis pela interação com o usuário e que pertencem à parte do Controle do MVC. É com base nesta classificação de alto nível de abstração que os componentes serão tratados pelo mapeamento.

Veit et al. [55] fazem algumas observações sobre os pontos fracos do modelo MVC, citando o forte acoplamento encontrado entre as partes comuns do Controle e a Visão tornando impraticável o uso de tais partes em outros Modelos. Para resolver esta e outras questões, Veit et al. propõem a aplicação da tecnologia de aspectos na implementação das partes do estilo MVC em vez das técnicas tradicionais da orientação a objeto. O mapeamento proposto sugere uma abordagem diferente para resolver o problema do forte acoplamento entre os componentes, que será vista na próxima seção.

Para criar o mapeamento entre os componentes de uma aplicação MVC genérica, é necessário ter-se como base os requisitos mínimos especificados na Seção 3.1. Esses requisitos abrangem a **comunicação, percepção do grupo, gerenciamento da sessão e um mecanismo de controle de concorrência**, permitindo o apoio mínimo de colaboração entre os usuários. Esta seção se concentrará na descrição de como foi feito o mapeamento entre os componentes de uma aplicação MVC genérica e os

componentes colaborativos, com o objetivo de atender aos requisitos colaborativos mínimos especificados na Seção 3.1.

3.2.2 Mapeando os Componentes

O requisito de comunicação pode agir sob dois aspectos funcionais: a comunicação entre os usuários e a comunicação entre os modelos de dados das aplicações distribuídas. Estes dois aspectos funcionais envolvem, respectivamente, os componentes da partes da Visão e do Modelo da aplicação. Deste modo, o requisito de comunicação é dividido em requisitos de comunicação entre os usuários e requisitos de comunicação do modelo, de acordo com os aspectos funcionais de cada um deles.

Além destes aspectos funcionais, os desenvolvedores que desejarem implementar o mapeamento também devem considerar a infra-estrutura necessária para que as aplicações se comuniquem. Não é comum encontrar a infra-estrutura de comunicação em rede em aplicações não colaborativas, indicando que o desenvolvedor deve implementar a infra-estrutura de comunicação em rede para que as aplicações possam se comunicar através da Internet durante a sessão colaborativa.

A implementação do requisito de comunicação entre usuários não modifica diretamente o estado do Modelo, pois ela é utilizada apenas para apoiar a discussão entre os usuários durante a sessão colaborativa. Para que este requisito seja implementado na aplicação, a interface precisa ser modificada para apoiar a comunicação entre os usuários através da captura e reprodução do que é transmitido de acordo com o canal de comunicação utilizado.

Diversos componentes fornecem meios para que os usuários se comuniquem informalmente por meio da própria interface do usuário, como, por exemplo, *chats* eletrônicos ou vídeo conferência. Estes controles podem ser obtidos e incorporados à aplicação para que ela apóie o aspecto funcional da comunicação entre os usuários. Alternativamente, os desenvolvedores podem modificar a interface da aplicação e inserir os controles de usuário necessários para permitir a comunicação. Os componentes de interação e apresentação de dados da aplicação são mapeados para componentes responsáveis pela comunicação entre os usuários. Estes componentes

fornecem controles na interface gráfica proporcionando os meios para a comunicação entre os usuários.

A comunicação entre os modelos das aplicações distribuídas requer uma solução mais detalhada. A implementação dos requisitos que permitem a comunicação dos modelos torna o mapeamento entre os componentes mais complexo, pois é necessário um esforço de desenvolvimento para que as ações e estruturas que armazenam a representação interna do modelo compartilhado possam ser monitoradas e transmitidas.

Este tipo de modificação sugere um mecanismo que monitore as mudanças de estado dos dados internos do modelo da aplicação, geralmente desenvolvido de maneira *ad hoc* devido ao modo e à estrutura de armazenamento dos dados. Por exemplo, se uma aplicação traduz as ações de um usuário sobre um modelo em operações semânticas sobre um grafo implementado por meio de uma lista ligada, será necessário monitorar a comunicação entre a aplicação e as operações efetuadas na lista, para que tais operações sejam comunicadas e eventualmente aplicadas à outra lista ligada de um usuário remoto.

As modificações nos modelos locais e as mudanças que estas modificações propagam nos modelos das aplicações remotas precisam ser sincronizadas para que a consistência entre os diferentes modelos seja mantida. Esta sincronização é de responsabilidade do mecanismo de controle de concorrência escolhido pelo desenvolvedor.

Como a aplicação não colaborativa é carente de mecanismos de controle de concorrência, a implementação deste aspecto funcional fica a cargo do desenvolvedor, que pode utilizar uma solução pronta e personalizá-la ou implementar um mecanismo de controle de concorrência diretamente na aplicação. O mecanismo de controle de concorrência utilizado afeta principalmente como os usuários utilizam o modelo compartilhado, seja com múltiplas versões diferentes ou versões iguais do mesmo modelo.

Os requisitos de comunicação necessários à colaboração, sejam eles requisitos de comunicação entre usuários ou de comunicação de modelos, envolvem os componentes de apresentação de informações e interação com o usuário. O mapeamento indica que ambos os componentes devem ser mapeados para componentes que apresentam informações as usuários e permitem a comunicação entre os modelos.

O requisito de controle de concorrência deve ser implementado nos componentes que fazem parte do Modelo, permitindo o acesso concorrente aos dados por múltiplos usuários remotos. Uma vez que a maioria das operações das aplicações não colaborativas são norteadas por acessos seqüenciais aos dados do modelo, os desenvolvedores precisam adaptar os componentes da parte do Controle e do Modelo para que eles permitam o acesso concorrente aos dados. Deste modo, os componentes que armazenam o modelo de dados específico da aplicação são mapeados para componentes que implementam o mecanismo de controle de concorrência escolhido pelo desenvolvedor.

Como os componentes de validação de dados envolvem diretamente a manipulação dos dados que os usuários fornecem à aplicação, estes componentes também devem ser mapeados para componentes que implementem o controle de concorrência. Isso se deve ao fato de que durante a validação dos dados pode ser necessário obter alguma autorização do controle de concorrência para validar a informação, como, por exemplo, a solicitação de um *token* ou uma trava para o acesso exclusivo do modelo.

O requisito de percepção do grupo está relacionado com a percepção das modificações externas geradas nos modelos locais dos usuários. Em sistemas *groupware*, é comum notificar os usuários das modificações externas em seus modelos por meio de uma atualização na interface do usuário. Este tipo de notificação apresenta aos usuários um novo aspecto funcional: a percepção do grupo. Da mesma forma que a comunicação entre os usuários, este aspecto funcional é diretamente relacionado com a interface da aplicação e com a apresentação de informações, mapeando os componentes de controle e da interface de usuário para os controles de interação multiusuária. Além dos componentes da interface da aplicação, os componentes que gerenciam os dados da sessão também são mapeados para os controles de interação multiusuária.

Os controles de interação multiusuária são uma abordagem comum para apoiar a percepção em grupo. Estes controles, também conhecidos como *awareness widgets*, têm como principal objetivo apoiar a percepção de consciência do grupo durante o uso de um *groupware*, apresentando algumas diferenças dos controles de aplicações com interação monousuária. Contudo, estas diferenças devem ser sutis, para evitar um aumento da carga cognitiva dos usuários, decorrente do aprendizado necessário para se trabalhar com estes novos controles em uma interface já conhecida.

Existem vários controles de interação multiusuária que fornecem a percepção em grupo durante o uso de um *groupware*. Dentre os principais controles, podem-se citar os *telepointers*, as barras de rolagem multiusuário e as visões radar. Estes controles fornecem, respectivamente, cursores que indicam a movimentação do mouse dos usuários, barras de rolagem contendo os nomes e as posições dos usuários em relação ao documento compartilhado e janelas que permitem a visualização em miniatura da área de trabalho.

Em geral, recomenda-se evitar a poluição da interface gráfica com controles que podem sobrecarregar a visão dos usuários com informações da aplicação, pois este tipo de sobrecarga pode gerar uma fadiga mental decorrente do alto esforço cognitivo, gerado tanto pela compreensão das informações apresentadas como pelo acompanhamento das atualizações das interfaces. Como alternativa para evitar esta sobrecarga, Ortega [14] sugere a utilização de um modelo e um *framework* para o desenvolvimento de sistemas *groupware* onde o usuário pode configurar sua interface gráfica de modo a desabilitar ou esconder certas partes da interface gráfica para evitar a quantidade elevada de informações apresentadas.

Outro aspecto da percepção do grupo que deve ser levado em consideração é a notificação da participação dos demais usuários, durante uma sessão colaborativa. Os usuários só podem colaborar uns com os outros, se estiverem na mesma sessão colaborativa, que pode ser definida como uma reunião de usuários que trabalham com um objetivo em comum. Um controle de acesso à sessão precisa ser criado para garantir que somente os usuários autorizados possam acessar as sessão colaborativa. A maioria das aplicações não colaborativas não possui maneiras de autenticar e nem de autorizar os usuários, mas uma aplicação que proporcione a colaboração síncrona entre múltiplos usuários remotos precisa tratar tanto a autenticação como a autorização dos usuários, não apenas para controlar o acesso à sessão, mas também para informar quem está iniciando sua participação na sessão colaborativa.

O componente que proporciona o controle de sessão não precisa ser implementado inteiramente em um único componente. Nas aplicações colaborativas que utilizam arquiteturas distribuídas e híbridas, um servidor é designado para armazenar os dados das sessões antigas e atuais e também para concentrar em um único local o acesso dos usuários que desejam participar de sessões colaborativas. O servidor também deve armazenar a versão mais atual dos modelos das sessões colaborativas ativas, fazendo

com que usuários que entrem na sessão recebam a versão mais atual do modelo compartilhado pelos usuários da mesma sessão.

Dos componentes de aplicação MVC genérica descritos no mapeamento, apenas os componentes que contêm as regras de negócio, lógica da aplicação e persistência de dados não sofrem ações diretas do mapeamento. Apesar de serem componentes essenciais para o funcionamento da aplicação, estes componentes não devem sofrer muitas modificações para que uma aplicação apresente os requisitos mínimos de colaboração. Estes, mesmo não envolvidos diretamente com o mapeamento, geralmente possuem um baixo nível de acoplamento com os demais componentes da aplicação, facilitando a implementação do mecanismo de controle de concorrência e da comunicação do modelo.

Os componentes que representam o modelo de dados e a lógica da aplicação encontram-se na parte do Modelo de uma aplicação que segue o estilo MVC. Estes componentes são criados a partir da modelagem de um processo de software e contêm regras de negócio e aspectos específicos da aplicação. Em muitos casos, estes componentes são considerados o núcleo da aplicação, pois são responsáveis por receber as modificações encaminhadas pelos componentes do Controle e notificar os componentes da Visão sobre modificações no estado do Modelo.

Como descrito na Seção 2.6.1, é pressuposto que componentes de uma aplicação MVC possuam baixo acoplamento e que um mesmo Modelo possa ser utilizado por mais de uma Visão.

Para transformar os componentes do Modelo de uma aplicação de modo a permitir que esta aplicação se torne colaborativa é necessário que os componentes do Modelo sejam fracamente acoplados aos componentes do Controle e da Visão. A habilidade de aceitar modificações no estado do modelo provenientes de outros Controles, que não aqueles locais ao modelo os componentes, é de grande importância para o mapeamento. Se o Modelo possuir a capacidade de se comunicar com componentes que fazem parte do Controle de uma aplicação idêntica, mas que seja executada remotamente, então basta que os componentes do Modelo atualizem sua Visão local para que as modificações remotas sejam apresentadas pelo usuário. Deste modo, os componentes do Modelo devem aceitar tanto modificações cuja origem é dos componentes do Controle local ou de componentes do Controle que sejam executados remotamente.

A modificação do estado de um Modelo proveniente de uma ação gerada por componentes do Controle de uma aplicação remota envolve vários detalhes. A comunicação entre modelos e o controle de concorrência são alguns detalhes apresentados no começo desta seção. Contudo, não há um mapeamento direto entre os componentes, pois eles são específicos e variam de aplicação para aplicação. O que o mapeamento propõe para estes casos é a modificação direta dos componentes de modo a apoiar tanto a modificação local ou remota e a implementação de um mecanismo de controle de concorrência. Deste modo, o mapeamento sugere que o Modelo local das aplicações seja adaptado para receber alterações externas, provenientes de componentes de outro Controle que não o local.

A Tabela 3.2 apresenta um resumo de como os componentes de uma aplicação MVC genérica podem ser mapeados de acordo com o conjunto mínimo de requisitos especificado. A Tabela 3.2 chama-se Tabela de Mapeamento de Componentes e representa uma contribuição deste trabalho, atuando como um guia para os desenvolvedores que desejarem fazer uso do mapeamento de componentes.

Tabela 3.2 – Tabela de Mapeamento de Componentes.

Componente da aplicação não colaborativa	**Parte do MVC**	**Componente de aplicação colaborativa mapeado**	**Requisito colaborativo endereçado**
Componentes de apresentação de informações	Visão	Controles de interface multiusuários, controles de comunicação na interface gráfica	Comunicação entre usuários e percepção em grupo
Componentes de interação com o usuário	Controle	Controles de interface multiusuários, controles de comunicação na interface gráfica	Comunicação entre usuários, percepção em grupo e gerenciamento da sessão
Componentes de controle de dados da sessão	Controle	Mecanismos de controle de concorrência	Percepção em grupo e controle de concorrência
Componentes de validação de dados	Controle	Mecanismo de controle de concorrência	Controle de concorrência
Modelo de dados e lógica da aplicação	Modelo	Modificação direta dos componentes	Comunicação do modelo e controle de concorrência

O mapeamento proposto mantém a estrutura de todos componentes originais da aplicação, permitindo o uso da mesma com ou sem os aspectos colaborativos. Deste

modo, os usuários podem escolher se desejam continuar trabalhando com a aplicação da mesma maneira que sempre fizeram ou utilizar os novos recursos colaborativos.

Em sistemas *groupware* existentes, é comum encontrar a arquitetura de comunicação replicada ou híbrida que mantém os modelos locais dos usuários e conta com um servidor para armazenar uma cópia mestre do modelo. Essas arquiteturas têm como vantagem sobre as demais arquiteturas a eficiência da utilização dos recursos de rede, pois somente os dados relacionados com os eventos do Controle são transmitidos pela rede.

Uma vez que nenhuma mudança estrutural no estilo arquitetural MVC é sugerida pelo mapeamento, o mapeamento requer o uso da arquitetura híbrida para permitir o compartilhamento do modelo e para gerenciar as sessões colaborativas que contêm participantes remotos. A arquitetura híbrida provê recursos para a propagação de alterações e o controle de concorrência, livrando os desenvolvedores destas tarefas.

O servidor da arquitetura híbrida, chamado de Servidor de Colaboração, é baseado na arquitetura Cliente/Servidor e pode ser utilizado para implementar partes do mecanismo de controle de concorrência, assim como todo o controle de autenticação e autorização necessário para o acesso às sessões colaborativas. A Figura 3.1 apresenta a arquitetura híbrida, onde dois usuários se comunicam entre si, por meio de um Servidor de Colaboração, enviando modificações dos seus Modelos Locais para o Modelo Mestre, e recebendo as notificações do Modelo Mestre que deverão ser aplicadas nos seus Modelos Locais.

Figura 3.1 - Dois usuários colaborando em uma Arquitetura Híbrida.

O fluxo de modificações sugeridas pelo mapeamento começa quando um usuário modifica o estado do seu Modelo Local por meio dos componentes do Controle local. Esta modificação é encaminhada e aplicada ao Modelo Mestre armazenado no Servidor de Colaboração, que contêm todos os componentes do Modelo da aplicação. Em seguida, o mecanismo de controle de concorrência, implementado no Servidor de

Colaboração, encaminha esta modificação para os demais Modelos Locais dos outros usuários da mesma sessão colaborativa. Por fim, a porção do mecanismo de controle de concorrência escolhido vai determinar se as modificações recebidas do Modelo Mestre serão aplicadas aos Modelos Locais.

3.3 Utilização do Mapeamento

O objetivo do mapeamento proposto é apresentar um guia genérico baseado em componentes para aplicações baseadas no estilo arquitetural MVC. O mapeamento procura ajudar os desenvolvedores na disciplina de análise e modelagem durante a fase de elaboração de *groupware*. É importante mencionar que o mapeamento proposto é abstrato e conceitual, não apresentando detalhes técnicos de implementação.

Esta seção apresenta uma descrição, em alto nível de abstração, da seqüência de ações que devem ser seguidas para guiar o desenvolvedor durante o uso do mapeamento em uma aplicação não colaborativa. Com a descrição apresentada, pretende-se facilitar o entendimento do mapeamento de componentes proposto e ajudar a tornar sua utilização uma tarefa sistemática. A seqüência de ações para utilizar o mapeamento proposto é agrupada em três passos: projeto das funcionalidades, análise da aplicação e a codificação dos componentes.

Naturalmente, estes passos não são independentes e devem ser considerados em conjunto. Por exemplo, a escolha de uma determinada tecnologia pode depender diretamente da funcionalidade que o desenvolvedor deseja implementar, relacionando o passo de projeto das funcionalidades com a codificação dos componentes.

Do ponto de vista do processo de software, a seqüência de passos recomendada na utilização do mapeamento de componentes pode ser considerada como uma tarefa que envolve tanto as fases iniciais do projeto como as fases de implementação. O fluxo de passos proposto para o uso do mapeamento contêm muitas atividades em comum com as atividades desenvolvidas durante as várias fases de processos de software tradicionais.

O passo que aborda o projeto de funcionalidades é diretamente relacionado com o conjunto de requisitos mínimos especificados. Este conjunto de requisitos vai indicar quais são as funcionalidades colaborativas que a aplicação vai proporcionar por

meio da modificação dos componentes. Empregar técnicas da Engenharia de Requisitos pode auxiliar os desenvolvedores que desejarem implementar o mapeamento e formalizar a produção dos artefatos gerados durante este passo.

A análise da aplicação envolve o estudo da aplicação que será modificada para apoiar as funcionalidades colaborativas. Inicialmente, deve-se verificar se a aplicação segue o estilo arquitetural MVC, pois este estilo arquitetural é um dos requisitos necessários para o uso do mapeamento, conforme descrito na Seção 3.2. Em seguida, os componentes da aplicação devem ser identificados e analisados em detalhes. Neste passo, é importante contar com toda a documentação gerada durante o processo original de desenvolvimento da aplicação, pois, uma vez que os desenvolvedores obtenham a documentação do projeto, eles devem decompor a aplicação em componentes para tornar mais fácil a implementação do mapeamento de componentes de acordo com as funcionalidades especificadas no passo de design das funcionalidades.

No passo da codificação, o desenvolvedor deve modificar o comportamento e funcionamento interno dos componentes com o objetivo de implementar as funcionalidades desejadas. A quantidade de alterações necessárias varia de acordo com cada componente da aplicação, embora muitos componentes possam apresentar estruturas e funcionalidades semelhantes que podem ser reutilizadas.

A implementação do mapeamento proposto levanta muitas questões técnicas sobre como implementar a arquitetura híbrida, o controle de concorrência, a atualização da interface, a comunicação de rede etc. Estas questões devem ser tratadas pelo desenvolvedor no momento da implementação, considerando os recursos disponíveis para resolvê-las.

Em algumas situações, o mapeamento proposto entre os componentes pode ser muito difícil de implementar. Um dos motivos desta dificuldade de implementação está relacionado com organização das aplicações. Muitas aplicações baseadas no estilo arquitetural MVC não são maduras o suficiente para serem promovidas a aplicações colaborativas, tornando necessário que estas aplicações alcancem nível mínimo de qualidade de organização, otimização e usabilidade antes que elas sejam consideradas aptas para receber funcionalidades colaborativas.

É comum encontrar aplicações que não utilizam componentes, de tal modo que todas as funcionalidades que a aplicação provê foram implementadas diretamente na

interface do usuário. Nestes casos, as aplicações devem passar por um processo de organização do código da aplicação em componentes de acordo com o estilo MVC, antes que elas sejam candidatas ao uso do mapeamento proposto.

Mesmo nas aplicações estruturadas de acordo com o estilo MVC outros fatores podem influenciar no uso do mapeamento, evitando que ele seja implementado como descrito. Componentes não maleáveis, ou que não podem ser personalizados, comprometem a utilização do mapeamento, uma vez que é necessário adaptar alguns componentes para a implementação dos requisitos colaborativos.

Nos casos onde os componentes não podem ser manipulados, recomenda-se ao desenvolvedor utilizar uma abordagem mais radical, como a refatoração [50] dos componentes ou a completa substituição dos mesmos por componentes mais suscetíveis a mudanças de comportamento [50]. Estas substituições devem ser feitas com extremo cuidado para evitar comprometer alguma funcionalidade existente da aplicação.

Reutilizar componentes de terceiros pode levar a situações inesperadas, onde se torna necessário reimplementar uma grande quantidade de código. O uso de componentes de terceiros introduz dependências fora do controle dos desenvolvedores do sistema, impondo um esforço continuado de atualização de suas versões e de reconfiguração do sistema além de apresentar um risco maior de incorporar erros. Além disto, há a dificuldade de se encontrar componentes que atendam plenamente às funcionalidades desejadas e que também ofereçam suporte aos requisitos não funcionais da aplicação, como desempenho, segurança, escalabilidade etc.

Como normalmente os componentes são desenvolvidos no estilo caixa preta, revelando o mínimo possível de sua implementação, as maneiras mais comuns de personalizar um componente envolvem a modificação de valores de propriedades, a composição deste componente com outros componentes ou a especialização de determinados comportamentos do componente.

Contudo, substituir componentes ou reescrevê-los nem sempre é a melhor alternativa para implementar os requisitos colaborativos, uma vez que esta estratégia requer um grande esforço de desenvolvimento. Sobrescrever métodos existentes, encapsular operações, adaptar métodos dos componentes e a capacidade de captar os eventos disparados por alguma ação do usuário são alternativas para modificar os componentes sem gastar muito tempo ou esforço de desenvolvimento. As modificações

nos componentes existentes e as remoções de componentes devem ser feitas com cuidado para evitar qualquer dano à estrutura da aplicação e para manter a funcionalidade de todos os requisitos já implementados. É recomendado que o desenvolvedor conheça muito bem a interdependência dos componentes da aplicação e suas interfaces antes de implementar qualquer modificação nos componentes.

Durante a modificação e adaptação dos componentes, o desenvolvedor pode aproveitar a oportunidade de modificar o código fonte para aplicar técnicas de refatoração [50]. Seguindo as técnicas de refatoração, o código fonte da aplicação pode ser melhorado, em termos de organização e estrutura, tornando a codificação de futuras modificações ou correções de erros mais rápida e eficiente.

Alguns aspectos funcionais, como o controle de concorrência, não podem ser mapeados por que não há componentes na aplicação não colaborativa que implementam estes aspectos. Nestes casos, a abordagem mais recomendada é implementar estes aspectos funcionais utilizando novos componentes que forneçam a funcionalidade desejada. Se nenhum componente com a funcionalidade desejada for encontrado ou os componentes encontrados não apresentarem compatibilidade com a aplicação, o desenvolvedor não tem outra escolha a não ser implementar do zero esta funcionalidade, considerando a estrutura da aplicação e a compatibilidade com o código fonte existente.

A implementação do mapeamento, em termos práticos, pode ser composta por modificações *ad hoc* e pelo uso de padrões de projeto. Os padrões de projetos, apesar de serem utilizados diretamente em classes e objetos, possuem princípios básicos que podem ser aplicados aos componentes existentes de uma aplicação. Esta adaptação pode ser facilitada se o desenvolvedor possuir o código fonte dos componentes em vez de apenas a interface.

Por exemplo, para capturar das modificações no modelo local, recomenda-se o uso do padrão *Observer*, pois é comum encontrar o uso deste padrão para proporcionar notificações de eventos entre os componentes do Controle e do Modelo. Com o uso deste padrão, os dados fornecidos pelas interfaces dos componentes podem ser encaminhados até o Servidor de Colaboração. Além do padrão *Observer*, os desenvolvedores também podem fazer uso do padrão *State* para facilitar a identificação de mudanças no estado dos componentes do Modelo local da aplicação e nos componentes do Modelo remoto do Servidor de Colaboração.

Outro padrão de projeto que pode ser amplamente utilizado na implementação do mapeamento de componentes é o padrão *Proxy*. Este padrão tem como objetivo intermediar a comunicação entre alguns objetos e controlar o acesso às funcionalidades dos objetos. A implementação deste padrão proporciona o encapsulamento de todas as operações de um objeto, disponibilizando os meios para que componentes externos façam acessos às operações que forem necessárias na colaboração. Seguindo esta linha de raciocínio, o padrão *Adapter* também pode ser utilizado com a finalidade de simplificar o acesso ao conjunto de funcionalidades dos componentes da aplicação. O padrão *Façade* também pode ser utilizado na implementação do mapeamento, uma vez que ele simplifica a chamada de vários métodos de objetos diferentes, além de fornecer uma interface de funcionamento mais simples que desacopla os subsistemas dos componentes.

Da mesma maneira que as ferramentas CASE apresentam recursos que automatizam a geração de código fonte a partir do modelo, muitos IDEs (*Integrated Development Environment*) automatizam a aplicação de técnicas de refatoração e do uso de padrões de projeto diretamente no código fonte, facilitando o emprego destas técnicas.

Por fim, recomenda-se aos desenvolvedores de aplicações que não implementem todos os requisitos especificados na Seção 3.1 a partir do zero, modificando a aplicação de forma *ad hoc*. Os desenvolvedores que desejam implementar o mapeamento devem procurar reutilizar, estender ou adaptar qualquer código fonte já desenvolvido. Como conseqüência dessas ações, a questão sobre onde as partes reusáveis de uma aplicação se encontram e como estas partes reusáveis podem ser estendidas e adaptadas sem uma profunda reprogramação assume um papel importante.

Uma discussão mais profunda sobre esta questão foge ao escopo deste trabalho, contudo as aplicações desenvolvidas com base nos princípios de reusabilidade, baixo acoplamento e componentização geralmente são estruturadas visando a incorporação de novas funcionalidades para atender novos requisitos, o que facilita a modificação e a extensão da aplicação.

3.4 Exemplo de Uso do Mapeamento

Nesta seção é apresentado um exemplo didático de uso do mapeamento. Para simplificar o exemplo, os componentes do Modelo, da Visão e do Controle de uma aplicação simples são apresentados. Em seguida, é descrito como o mapeamento pode ser aplicado nos componentes da aplicação para implementar requisitos colaborativos.

A aplicação MVC utilizada neste exemplo contém apenas um componente para o Modelo. Este componente implementa um controle de temperatura responsável por encapsular a lógica de conversão entre graus *Farenheit* e graus *Celsius*. O componente possui um par de operações para gravar a temperatura, em graus *Farenheit* e *Celsius*, e outro par de operações para ler a temperatura, retornado o valor da temperatura em graus *Farenheit* ou *Celsius*.

A Visão desta aplicação é composta de um componente chamado *gauge*, que é um desenho de um termômetro inserido em uma janela da interface gráfica. No desenho do termômetro, o usuário pode visualizar a temperatura em graus *Celsius* ou em graus *Farenheit*, por meio de duas escalas diferentes posicionadas nos lados do desenho do termômetro.

Quando o usuário clica em qualquer uma das barras do termômetro, um evento é disparado para o componente do Controle. Este componente do Controle é responsável pelo envio da modificação do valor da temperatura atual ao componente do Modelo. Por sua vez, o componente do Modelo notifica o componente da Visão, que altera as escalas do desenho do termômetro para refletir a nova temperatura atual.

De acordo com o mapeamento, os componentes de apresentação de informações devem ser mapeados para componentes com controles de interface multiusuário e também para controles que permitam a comunicação. Na aplicação de exemplo, pode-se implementar *telepointers* para permitir que mais de um usuário modifique a temperatura atual. A comunicação pode ser implementada por meio de uma mensagem de notificação na barra de título da janela, apresentando o valor antigo e novo da temperatura e também qual usuário efetuou a modificação.

Para implementar o controle de concorrência, o componente do Controle, responsável por enviar a nova temperatura para o Modelo, pode utilizar um controle de concorrência otimista, como a técnica de Transformação Operacional.

Já no Modelo, é necessário implementar toda a infra-estrutura que permite a comunicação. Também é necessário mapear o componente do Modelo para que ele possa receber modificações nas suas propriedades de acordo com a intenção dos usuários remotos. Semáforos podem ser utilizados no componente do Modelo com o objetivo de evitar que mais de um usuário modifique a temperatura ao mesmo tempo.

No que diz respeito à arquitetura, o mapeamento implica na criação de um Servidor de Colaboração para suportar a arquitetura híbrida, uma vez que nenhum componente deve ser removido da aplicação original.

3.5 Sumário

Neste capítulo apresentou-se a especificação de um conjunto mínimo de requisitos colaborativos que uma aplicação não colaborativa deve possuir para apresentar algumas funcionalidades colaborativas. Em seguida, o Mapeamento entre os componentes de uma aplicação genérica que segue o estilo arquitetural MVC e os componentes, controles e mecanismos necessários na implementação do conjunto mínimo de requisitos foi apresentado.

As questões relacionadas com a utilização do mapeamento estão ligadas diretamente à divisão dos componentes da aplicação e de como estes componentes são implementados. Segue-se abordando quais técnicas podem ser utilizadas de acordo com cada situação encontrada, sempre destacando a reutilização.

No próximo capítulo será apresentado como o mapeamento proposto foi utilizado para implementar requisitos colaborativos em uma ferramenta CASE.

Capítulo IV

Aplicação do Mapeamento: Estudo de caso

Neste capítulo descreve-se um estudo de caso onde o mapeamento foi aplicado nos componentes de um software não colaborativo. Desenvolver um protótipo por meio da aplicação do mapeamento teve como objetivo gerar uma prova de conceito do mapeamento de componentes.

Inicialmente apresenta-se como foi feita a escolha da ferramenta que recebeu o mapeamento e se tornou colaborativa. Em seguida, são apresentadas as funcionalidades que o protótipo deve implementar, sempre com base no conjunto mínimo de requisitos. Na seqüência, apresenta-se a organização e estrutura original da ferramenta escolhida e os detalhes técnicos da aplicação do mapeamento. Por fim, uma comparação entre as abordagens do Capítulo 2 a o mapeamento de componentes é apresentada, evidenciando as contribuições da abordagem proposta.

4.1 Escolha da Ferramenta

A escolha da ferramenta utilizada para receber o mapeamento de componentes e gerar um protótipo como prova de conceito levou em consideração vários fatores. O primeiro fator, senão o mais importante, diz respeito à disponibilidade do código fonte da ferramenta alvo, pois sem este recurso seria impossível aplicar o mapeamento e gerar o protótipo. Pela própria definição do Mapeamento proposto, a ferramenta escolhida deve ser possuir componentes estruturados de acordo com o estilo arquitetural MVC. Infelizmente, estes requisitos já filtram uma boa quantidade das aplicações existentes e amplamente utilizadas.

O software escolhido para receber o Mapeamento proposto é uma ferramenta CASE de modelagem. Deste modo, não só os desenvolvedores de software podem ser

beneficiados pelo produto final, como também os educadores da área de Engenharia de Software podem contar com uma ferramenta para auxiliar suas atividades pedagógicas.

Após revisar diversas ferramentas CASE de modelagem de código livre, pode-se verificar que nenhuma delas permite a edição colaborativa de diagramas UML. A ferramenta que mais se destacou na avaliação e que se enquadra nos requisitos para o mapeamento foi a ArgoUML [6], citada na Subseção 2.3.2.

Dentre os principais fatores que levaram a escolha da ArgoUML como ferramenta alvo para o mapeamento, pode-se destacar o seguinte:

- A ArgoUML é uma ferramenta CASE de modelagem madura, possui o código livre, foi escrito em Java e contém seus componentes estruturados de acordo com o estilo arquitetural MVC. Originalmente desenvolvida para um projeto de pesquisa, esta ferramenta atualmente está associada a uma forte comunidade internacional de usuários e desenvolvedores.

- Como toda ferramenta CASE de modelagem, a ArgoUML permite a edição dos principais diagramas UML, contém mecanismos de geração automática de código, a partir do modelo, auxilia a tarefa de engenharia reversa e possui um sistema de críticas dos modelos gerados.

- A ArgoUML atende a vários padrões, a saber: UML (especificação 1.3), XMI, SVG, OCL e outros. Neste aspecto, a ArgoUML se destaca das demais ferramentas CASE de modelagem comerciais.

- Em vez de utilizar um metamodelo proprietário na implementação da especificação UML, a ArgoUML utiliza um metamodelo proprietário fornecido pela empresa NovoSoft.

- A ArgoUML recebeu elogios da crítica especializada, sendo recomendada por várias publicações como uma ferramenta para os desenvolvedores que criam diagramas UML.

4.2 Funcionalidades do Protótipo

Nesta seção são definidas apenas as funcionalidades de um editor colaborativo de diagramas UML sob o ponto de vista da cooperação. Ou seja, os requisitos de

usuário que envolvem a edição de diagramas não serão abordados, uma vez que a ferramenta alvo do mapeamento já deve atender os requisitos de um editor de diagramas UML.

Para facilitar a especificação dos requisitos, uma revisão bibliográfica foi realizada com o objetivo de levantar os requisitos funcionais necessários para apoiar a colaboração em um editor síncrono. A análise das funcionalidades dos editores colaborativos CUTE *(Collaborative UML Technique Editor)* [51] e CO2DE *(Collaborate to Design)* [48] e das ferramentas CASE comerciais *Poseidon for UML* e *Magic Draw* possibilitou a elaboração preliminar de uma lista de requisitos de colaboração.

A partir das funcionalidades dos editores e ferramentas colaborativas e das opiniões dos usuários e especialistas do domínio, diversos diagramas de casos de uso foram construídos, com o objetivo de se identificar situações e cenários de uso da ferramenta CASE de modelagem colaborativa. A Figura 4.1 apresenta dois diagramas casos de uso modelados durante a fase de especificação de requisitos.

O diagrama de casos de uso da Figura 4.1.a apresenta os casos de uso que um usuário pode realizar antes no contexto da participação de uma sessão colaborativa. Estes casos de uso foram utilizados para definir quais seriam as funcionalidades do Servidor de Colaboração. Já o diagrama de casos de uso da Figura 4.1.b apresenta dois atores que representam participantes de uma sessão colaborativa. Pode-se notar pelo diagrama que as operações que os participantes podem realizam envolvem o controle de acesso aos elementos do diagrama e as modificações nos elementos dos diagramas.

O processo de especificação de requisitos e *design* das funcionalidades foi auxiliado por vários diagramas de casos de uso, como os diagramas apresentados na Figura 4.1. Os principais requisitos funcionais, apresentados a seguir, foram especificados a partir dos cenários apresentados pelos casos de uso. Eles servem como base para apoiar a colaboração síncrona durante a modelagem de diagramas UML em uma ferramenta CASE tradicional. São os seguintes:

- Os usuários devem poder iniciar sessões de edição de diagramas colaborativas.
- Os usuários devem poder entrar ou sair de sessões que já estejam em andamento.

- A ferramenta deve permitir o uso simultâneo de vários usuários de forma colaborativa, durante a edição de diagramas UML em um espaço de trabalho compartilhado.
- As ações sobre os elementos do diagrama devem ser coordenadas por um mecanismo de controle de concorrência, que mantenha consistente o estado global do diagrama entre os usuários.
- Interações dos usuários que modifiquem o estado do diagrama devem ser propagadas para os demais usuários através de dispositivos de percepção.
- A ferramenta deve permitir a comunicação dos usuários entre si durante a edição do diagrama, permitindo a coordenação de suas ações.

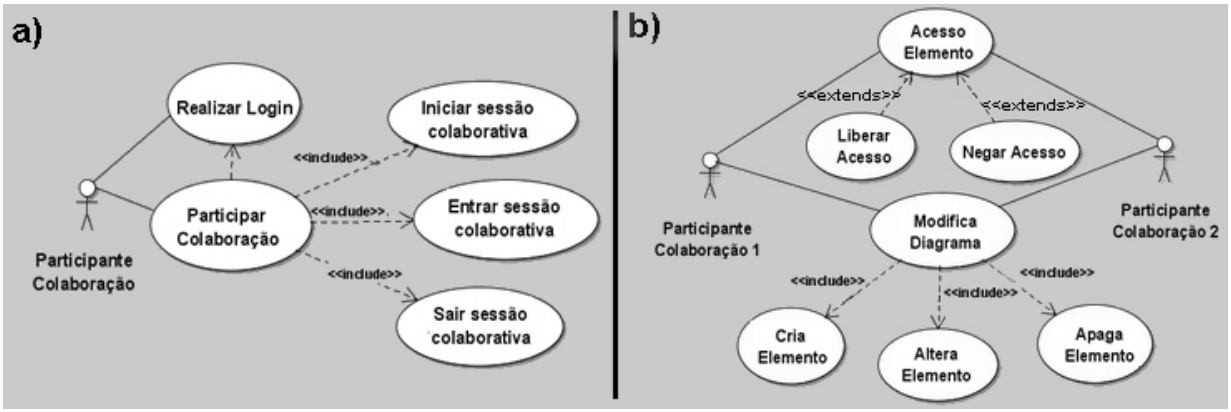

Figura 4.1 - Diagramas de casos de uso: a) Casos de uso de acesso às sessões colaborativas e b) Casos de uso de modificação do modelo.

A colaboração entre usuários, durante a modelagem de algum diagrama da UML, deve levar em consideração também requisitos não funcionais. Estes requisitos impõem alguma restrição ao software como, por exemplo, a necessidade de se implementar um tipo de arquitetura e de se definir um tempo limite de resposta para um determinado processamento. Ou seja, requisitos que não representam valor direto para o negócio do usuário, oriundos de necessidades como segurança, adequação a padrões etc.

Como a prova de conceito adapta uma ferramenta CASE existente, os requisitos não funcionais já implementados devem ser mantidos na ferramenta quando a mesma apoiar a colaboração durante a edição dos diagramas.

4.3 Estrutura Original da ArgoUML

A ArgoUML é uma ferramenta CASE de modelagem que permite a edição de diagramas UML. Nesta seção apresenta-se uma explicação básica sobre o funcionamento interno da ArgoUML, incluindo a estrutura e a arquitetura de seus componentes. Porém, antes de apresentar o funcionamento interno desta ferramenta, é necessário entender como a ArgoUML implementa a especificação 1.3 da UML.

Da mesma maneira que o padrão XMI, a especificação UML 1.3 não inclui nenhuma informação gráfica para representar o diagrama e seus elementos. Devido a esta limitação, cada ferramenta CASE de modelagem que permite a modelagem de diagramas UML deve implementar um formato proprietário para armazenar a informação gráfica.

A ArgoUML trabalha separando as informações dos diagramas UML em dois níveis: o nível do modelo e o nível gráfico.

O nível do modelo inclui todos os dados relacionados com os elementos puros da UML, como classes, casos de uso, interfaces, relacionamentos, propriedades, atributos e operações. Já o nível gráfico é responsável pela representação visual destes elementos de acordo com a interface do usuário, como a localização relativa de cada elemento, seu tamanho e direção dos relacionamentos.

Para trabalhar esta separação em níveis, a ArgoUML possui duas grandes dependências externas. Estas dependências são projetos completamente separados da ArgoUML que foram utilizados pela equipe de desenvolvedores com o objetivo de ganhar tempo na implementação e reutilizar soluções existentes para trabalhar na manipulação dos dois níveis de separação das informações.

As duas dependências externas da ArgoUML são o NSUML e o GEF. Ambas as dependências externas apresentam recursos para o tratamento dos dados no nível de modelo.

A NSUML é uma biblioteca que provê uma implementação do padrão 1.3 do metamodelo da UML, especificado pelo OMG. Criada pela empresa Novosoft, esta biblioteca é utilizada por outros projetos além da ArgoUML. O conteúdo desta biblioteca inclui a definição dos elementos da UML (classificadores, elementos genéricos), a sintaxe abstrata e as regras de boa formação da especificação 1.3 da UML.

Em termos práticos, a ArgoUML utiliza a biblioteca NSUML na representação de elementos e na exportação e importação dos diagramas UML de acordo com o padrão XMI. A representação dos elementos e o armazenamento dos mesmos não são controlados pelas classes da NSUML e sim pelo *framework* GEF.

O GEF (*Graphical Edition Framework)* [27] é um *framework* genérico utilizado na representação e o armazenamento de grafos. Este *framework* permite a construção de aplicações para a edição de grafos utilizando um modelo composto por Nós e Arestas.

Este *framework* por si só é implementado no estilo arquitetural MVC e utiliza a biblioteca gráfica Swing do Java, permitindo que ele atue como uma interface gráfica para estruturas de dados existentes. Ele apóia a manipulação de arquivos com base na utilização do padrão PGML (*Portable Graphics Markup Language*), utilizado para o armazenamento de grafos. Entretanto, a ArgoUML possui a funcionalidade de exportar os dados tanto no formato SVG como no formato PGML, sendo este último o formato utilizado neste trabalho para a troca de informações do modelo entre os clientes e o servidor durante a colaboração.

O principal objetivo da utilização do GEF na ArgoUML é representar o modelo da UML. Como o GEF trabalha internamente com grafos, alguns elementos da UML, como classes, interfaces ou atores, são abstraídos como nós do grafo. Já os elementos que representam ligações, como associações, generalizações e realizações, são armazenados como arestas do grafo. O GEF também fornece outras funcionalidades, como o gerenciamento do editor e o tratamento de eventos disparados pelos usuários. Outras aplicações utilizam o GEF para facilitar a construção de editores de diversos tipos de diagramas como, por exemplo, o *Poseidon for UML*.

A ArgoUML estende o GEF adaptando suas funcionalidades para manipular os diagramas UML. Para ilustrar como a ArgoUML estende o GEF, a Figura 4.2 apresenta um diagrama de classes parcial contendo algumas das classes do GEF e da ArgoUML responsáveis pela criação dos elementos de modelagem Interface, Classe e Pacote de um diagrama de classes. As classes apresentadas no diagrama da Figura 4.2 possuem mais especializações, implementações de interfaces e generalizações que não foram apresentadas para não poluir o diagrama e tornar didático o exemplo de extensão do *framework* GEF pela ArgoUML.

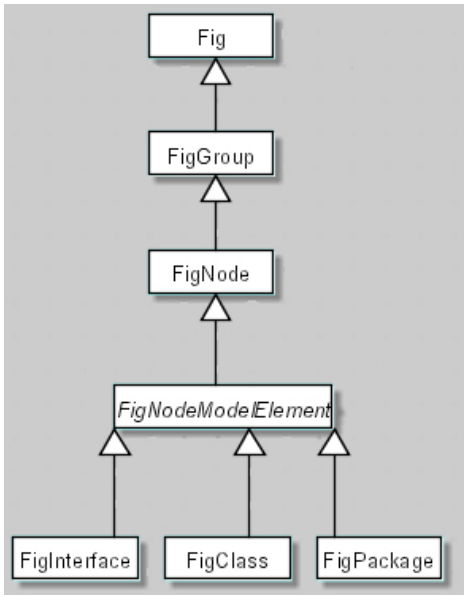

Figura 4.2 - Diagramas de classe parcial do GEF e da ArgoUML.

As classes Fig, FigGroup e FigNode fazem parte do GEF e são responsáveis pela representação interna dos elementos representados como nós no grafo pelo GEF. A classe abstrata FigNodeModelElement e as classes concretas FigInterface, FigClass e FigPackage pertencem ao ArgoUML.

Apesar da ArgoUML de utilizar as classes do *framework* GEF por meio da especialização, nota-se que a classe abstrata FigNodeModelElement, que pertence ao ArgoUML, é uma especialização da classe concreta FigNode. Este tipo de prática, uma classe abstrata especializar uma classe concreta, não é considerada uma boa prática de programação e deve ser evitada, principalmente quando se faz uso de um *framework* complexo como o GEF.

Nenhuma das classes apresentadas na Figura 4.2 contém informações sobre o desenho gráfico destes elementos, tornando-se responsáveis apenas pelo armazenamento dos elementos. A ArgoUML possui uma estrutura de herança semelhante à apresentada na Figura 4.2 para armazenar as relações entre elementos como, por exemplo, a herança ou a associação entre classes.

A ferramenta ArgoUML possui sua estrutura organizada em diversos subsistemas organizados de acordo com o estilo arquitetural MVC. Cada subsistema na ArgoUML possui um nome e é implementado como um único diretório/pacote do Java. Os subsistemas possuem uma classe que segue o padrão *Façade* e uma interface chamada *Plug-in* que permite a utilização das classes contidas dentro do componente.

De acordo com o *Cookbook* [6] da ArgoUML, uma classe *Façade* é normalmente uma parte da API ou uma versão simplificada da API. Para cada subsistema X na ArgoUML que utiliza o subsistema Y, o desenvolvedor de X precisa decidir, em tempo de projeto, se ele deseja utilizar a API de Y quando utilizar o subsistema Y, por meio da declaração de todo o pacote na instrução *import* do Java, ou utilizar a classe *Façade* de Y, colocando apenas o nome da classe *Façade* na sua cláusula *import* em todos os arquivos do subsistema X que fazem uso do subsistema Y.

As partes do Modelo, do Controle e da Visão são implementadas na ArgoUML por meio de seus subsistemas. Além do uso do *framework* GEF, cujos componentes fornecem a base para a construção da ferramenta, a ArgoUML é divida em vários subsistemas. Os subsistemas mais importantes para este trabalho incluem os subsistemas de baixo nível, os subsistemas do Modelo e os subsistemas da Visão/Controle.

É comum encontrar os subsistemas da Visão e do Controle agrupados nas aplicações que seguem o estilo arquitetural, pois os componentes destes subsistemas se relacionam diretamente. Apesar de contar com controles da interface gráfica independentes, a maioria dos componentes dos subsistemas de Controle se relacionam diretamente com os elementos da interface gráfica da ArgoUML. Na própria documentação oficial da ArgoUML, o *Cookbook* [6], estes dois subsistemas são tratados como um, raramente discriminando qual subsistema cada componentes pertence.

Os subsistemas de baixo nível são os subsistemas responsáveis pela infra-estrutura e podem ser utilizados por qualquer outro subsistema. Os subsistemas considerados de baixo nível fornecem o apoio à Internacionalização (*Internationalization)*, o acesso às classes do JRE (*Java Runtime Environment*), Swing e recursos de depuração por meio de registros (*Logging*).

Os subsistemas do Modelo contêm todos os componentes necessários para se manipular dos elementos do diagrama criados pelos usuários. Muitas das classes contidas nestes subsistemas generalizam as classes contidas no GEF, fornecendo recursos específicos, como a manipulação de alguns elementos da interface gráfica.

Os subsistemas responsáveis pelos componentes da Visão/Controle contêm tanto a implementação dos elementos da interface gráfica da ArgoUML como os componentes responsáveis pelo tratamento de eventos dos usuários. Dentre os principais

subsistemas da Visão e do Controle, pode-se destacar os seguintes: o subsistema *Diagram*, responsável pela visualização gráfica do modelo e de seus elementos; o subsistema *Explorer*, que contém uma visualização em forma de árvore do diagrama; e o subsistema *Reverse Engineering*, responsável pela geração de diagramas a partir do código fonte.

Apesar de ser organizado em dependências externas, subsistemas e componentes, a ArgoUML é uma ferramenta complexa. A versão do *framework* GEF, denominada 0.10.4, contém aproximadamente 250 classes. A ArgoUML em si contêm aproximadamente 1.100 classes na versão denominada vPRE 0.16.1. Na lista de discussão internacional sobre o desenvolvimento, descobre-se que cerca de 200 desenvolvedores espalhados pelo mundo contribuem com o desenvolvimento da ArgoUML.

Antes do início da implementação, um estudo foi conduzido no código da ferramenta e na documentação disponível do projeto. Este estudo teve como objetivo a compreensão da estrutura, arquitetura, funcionamento e organização do código fonte, tanto do GEF como da ArgoUML, visando identificar como o mapeamento proposto seria implementado nesta ferramenta. Este estudo levou aproximadamente quatro meses, contando com o esforço de um desenvolvedor, de nível pleno, trabalhando, em média, seis horas por dia.

É importante deixar claro que a ArgoUML não é uma aplicação MVC ‘pura’, ou seja, não é fácil identificar quais são os componentes do Modelo, da Visão e do Controle. Os componentes da Visão e do Controle, em particular, são tratados como componentes que pertencem aos subsistemas da Visão e do Controle ao mesmo tempo, o que dificulta a identificação das responsabilidades e das interações dos componentes. Além disso, o *framework* GEF também não discrimina quais componentes pertencem à quais partes do MVC, uma vez que este *framework* também contém componentes para o Modelo, o Controle e a Visão.

A modificação de uma versão específica da ArgoUML levantou uma questão relevante sobre a continuidade dos requisitos implementados. Deste modo, as versões tanto do GEF como da ArgoUML foram congeladas e modificadas em paralelo ao desenvolvimento oficial da ferramenta, pois a implementação do mapeamento tem como objetivo produzir um protótipo como prova de conceito e não um produto pronto para ser utilizado em larga escala. Entretanto, os principais desenvolvedores da lista

oficial da ArgoUML se mostraram interessados no resultado final, a ponto de incluírem no cronograma de modificações do projeto possíveis alterações, a serem implementados a longo prazo, que incorporem resultados deste trabalho.

Para efeitos de comparação, a Figura 4.3 apresenta a interface gráfica do usuário original da ArgoUML, onde os principais elementos da interface foram enumerados. Os elementos enumerados são apresentados a seguir.

- Item 1 - barra de menu. Nesta barra de menu as principais operações podem ser realizadas como, por exemplo, salvar um arquivo, criar um novo diagrama ou gerar o código fonte para um modelo;
- Item 2 - barra de ferramentas. Esta barra de ferramentas apresenta um atalho para os principais comandos do menu, como salvar ou imprimir um diagrama;
- Item 3 - painel de navegação. Este painel apresenta os diagramas criados pelo projeto atual e seus respectivos elementos;
- Item 4 - barra de ferramentas do diagrama. Esta barra possui botões que variam de acordo com o diagrama editado. Os botões nesta barra permitem a criação de novos elementos no diagrama editado no momento;
- Item 5 - painel do diagrama. É neste painel que os elementos do diagrama são manipulados pelo usuário;
- Item 6 - painel de prioridades. Neste painel as tarefas a serem feitas no diagrama (*TODO tasks*) são listadas de acordo com suas prioridades; e
- Item 7 - painel de propriedades. Este painel possui abas, cujos nomes variam de acordo com o diagrama ou elemento do diagrama que está selecionado. Cada aba contém controles específicos para a manipulação dos valores das propriedades.

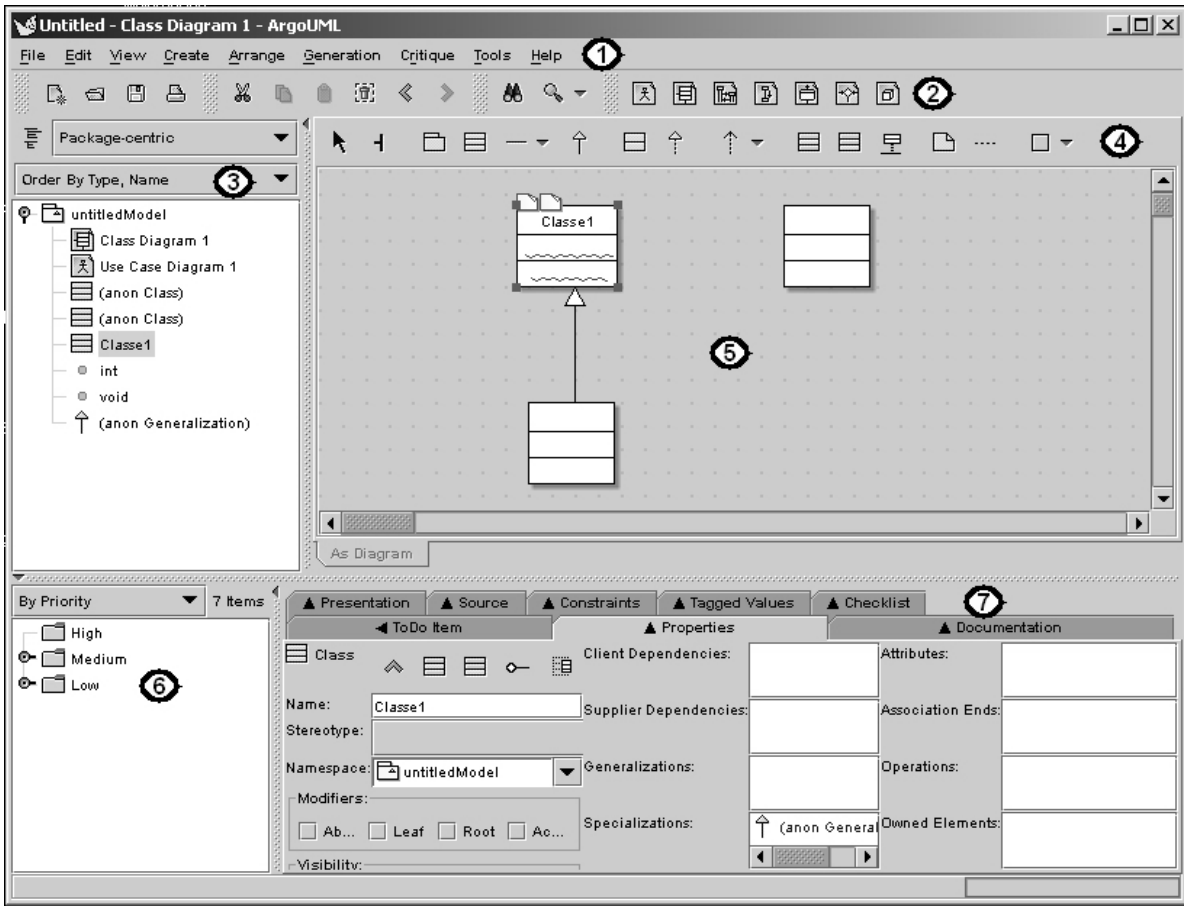

Figura 4.3 – Interface gráfica do usuário original da ArgoUML.

4.4 Aplicação do Mapeamento

Nesta seção, apresenta-se o modo como cada uma das funcionalidades especificadas na Seção 4.2 foi implementada na ArgoUML. O tipo de *groupware* implementado nesta pesquisa apresenta características distribuídas e síncronas, pois o grupo de usuários interage apenas com a ferramenta CASE de modelagem. O protótipo obtido como resultado da implementação do mapeamento foi batizado de CoArgoUML (*Collaborative ArgoUML*).

Devido ao forte acoplamento dos componentes do GEF aos componentes da ArgoUML, notou-se que, para implementar o compartilhamento dos elementos, uma grande quantidade de modificações deveria ser implementada diretamente nas classes do GEF, pois apenas a extensão das classes do GEF por meio da herança não é suficiente para implementar a mudança de comportamento das classes.

Outro fator que influenciou a modificação do GEF está relacionado com a abstração do tipo de elemento manipulado pelo *framework*. Para o GEF não importa o tipo de elemento utilizado no diagrama, desde que os elementos do modelo possam ser armazenados em um grafo. Esta característica simplificou o uso da ferramenta e forneceu mais funcionalidades, pois, em vez de se concentrar em implementar requisitos colaborativos em apenas um tipo de diagrama, a modificação do GEF agregará os requisitos colaborativos a todos os diagramas UML com que a ArgoUML trabalha.

Sob o ponto de vista do mapeamento, a dependência externa NSUML não sofreu nenhuma modificação. Deste modo, a implementação do mapeamento envolve apenas o *framework* GEF e a ArgoUML propriamente dita. Apesar de ser um *framework* e permitir personalizações, grande parte das funcionalidades colaborativas exigiram modificações pontuais diretas em alguns componentes do GEF.

As modificações no GEF e na ArgoUML serão apresentadas nesta seção pela ordem nas quais elas foram implementadas. A aplicação completa do mapeamento demandou a modificação de 40 classes na ArgoUML e 19 classes no GEF. Foram criadas oito novas classes para o ArgoUML e quatro novas classes para o GEF.

Para incluir acesso às novas funcionalidades, três arquivos de configuração foram modificados. O Servidor de Colaboração demandou a criação de duas novas classes. Todo o trabalho de modificação manual do GEF e da ArgoUML, além da criação do servidor de colaboração, levou três meses de trabalho, contando com o esforço de um desenvolvedor, de nível pleno, que trabalhou, em média, seis horas por dia.

Por restrições de espaço, apenas as principais modificações e criações que se relacionam com os requisitos e que foram implementadas no protótipo serão discutidas nas próximas subseções. O código fonte completo do CoArgoUML e do Servidor de colaboração se encontra no CD que acompanha este trabalho.

4.4.1 Comunicação entre os Clientes e o Servidor

De acordo com o mapeamento, os componentes do GEF responsáveis pelo modelo de dados e lógica da aplicação devem sofrer modificações diretas para implementar a comunicação do modelo.

O requisito de comunicação entre usuários deve ser implementado na ArgoUML, pois tanto nos componentes de apresentação de informações como nos componentes responsáveis pela interação do usuário compõem a interface gráfica. Deste modo, é necessário criar uma canal de comunicação entre o GEF e Servidor e outro canal de comunicação entre a ArgoUML e o Servidor.

Para separar os fluxos de dados de naturezas diferentes, optou-se pela criação de uma camada de comunicação cliente para tanto para o GEF como para a ArgoUML. A Figura 4.4 mostra como é feita a comunicação entre a parte cliente do GEF com o Servidor de Colaboração e a parte cliente da ArgoUML com o Servidor de Colaboração. No ambiente representado pela Figura 4.4, dois usuários utilizam o protótipo CoArgoUML e o Servidor de Colaboração para participar de uma sessão colaborativa de modelagem.

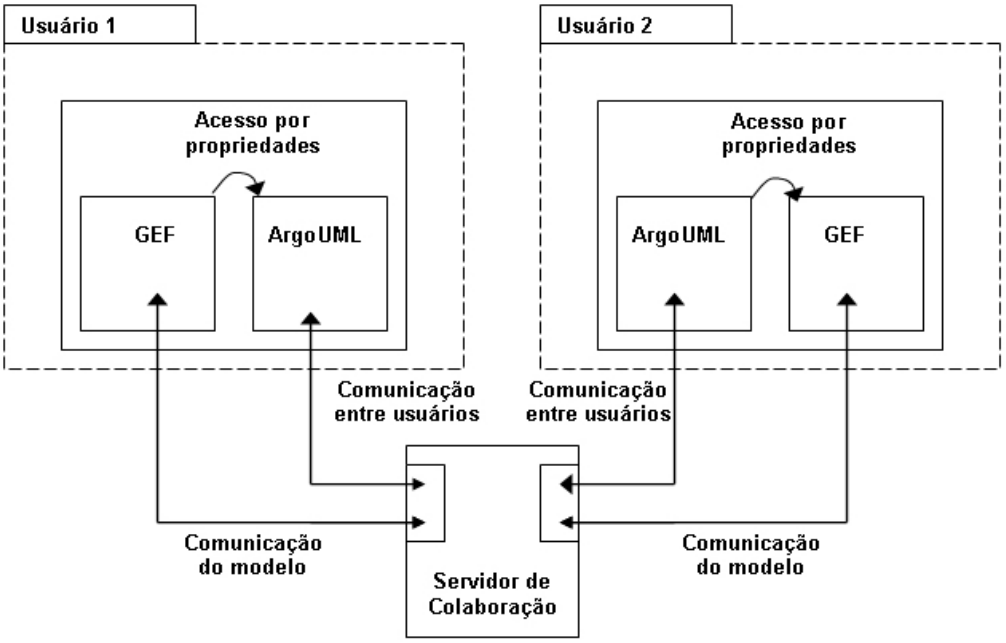

Figura 4.4 – Conexões do protótipo com o Servidor de Colaboração.

Em uma mesma instância da ArgoUML, a comunicação entre os objetos criados pelo GEF e os objetos da ArgoUML é feita por meio de propriedades das classes implementadas em métodos públicos. Os dois canais de comunicação com o servidor foram implementados utilizando a classe *Socket* do Java para estabelecer uma conexão utilizando o protocolo TCP/IP através da Internet.

A comunicação entre os clientes e o Servidor de Colaboração é feita através do protocolo TCP/IP. As classes clientes envolvidas diretamente neste processo são, na

parte do GEF, ClienteConectaGEF e ClienteRebeceGEF, para enviar e receber as informações do Servidor de Colaboração. Na parte da ArgoUML as classes se chamam ClienteConectaArgo e ClienteRebeceArgo. Estes dois pares de classes possuem toda a infra-estrutura de comunicação por meio de *Sockets* e requerem uma especialização da classe Thread padrão do Java.

Uma das dificuldades envolvidas na utilização de *Sockets* em Java como canal de comunicação entre aplicações é a necessidade de transmitir dados serializáveis pela máquina virtual Java. Uma classe é definida como serializável em Java se a mesma implementar a interface Serializable. Para permitir que subclasses de uma classe não serializável sejam serializáveis, duas condições devem ser atendidas: a classe não serializável deve possuir um construtor padrão, para que as subclasses possam inicializar o estado da mesma, e as subclasses devem se responsabilizar pela transferência dos objetos. A implementação da transferência de objetos é feita por meio dos métodos writeObject e readObject da classe Socket.

Para envio de dados serializados, duas classes, uma do GEF e outra da ArgoUML, foram utilizadas para enviar e receber dados do Servidor de Colaboração. Um objeto da classe ArrayList é criado quando o método *EnviaEvento* das classes ClienteConectaGEF ou ClienteConectaArgo é acionado. Este método recebe como parâmetro o evento enviado, do tipo *String*, e também um segundo objeto contendo os dados a serem enviados, do tipo *Object*. O nome do evento identifica qual é o significado da transferência de dados, que pode ser uma requisição de trava (*lock*), uma notificação da movimentação do ponto de usuário, ou a criação de um novo elemento. O segundo parâmetro traz dados relacionados com o evento em si como, por exemplo, a nova posição de um elemento no diagrama.

Para o envio e recebimento dos dados entre a ferramenta e o Servidor de Colaboração, os métodos writeObject e readObject da classe Socket são utilizados. Tanto as classes que enviam dados, ClienteConectaGEF ou ClienteConectaArgo, como as classes que recebem dados, ClienteRebeceGEF ou ClienteRebeceArgo, mantêm a conexão constante com o Servidor de Colaboração e verificam se há algum dado a ser enviado ou recebido periodicamente por meio de um laço.

Ao receber a mensagem, o servidor realiza o mesmo processo, porém de maneira inversa, com apenas uma modificação: em vez de enviar a mensagem para um único destino, envia uma mensagem para cada um dos clientes conectados no momento.

Uma limitação do protótipo construído diz respeito à quantidade de usuários suportada. Devido à complexidade de implementação dos *telepointers*, e também a fatores relacionados ao desempenho, a percepção de múltiplos participantes e à interface gráfica, optou-se por restringir a utilização simultânea do protótipo à apenas dois usuários. Este é o motivo pela qual o experimento descrito no Capítulo 5 limita-se à coleta de dados durante a utilização simultânea do protótipo por apenas dois usuários numa sessão colaborativa.

4.4.2 Implementação do Servidor de Colaboração

O Servidor de Colaboração proposto para auxiliar a troca de dados entre os participantes de uma mesma sessão colaborativa possui várias responsabilidades. Além de replicar os elementos do diagrama para todos os usuários de uma mesma sessão colaborativa, o Servidor de Colaboração ainda é responsável pelo encaminhamento das mensagens de comunicação, pelo armazenamento da versão mais atualizada do modelo, pela autenticação dos usuários e por parte do mecanismo de controle de concorrência distribuído.

O Servidor de Colaboração é implementado por meio de duas classes. A classe principal chama-se ServCollab e é responsável pela infra-estrutura de comunicação com os clientes e pelo gerenciamento das sessões colaborativas.

A segunda classe do Servidor de Colaboração chama-se ServCollabT. Esta classe herda da classe Thread do Java e é responsável pelo envio e recebimento de dados, além de autenticar os usuários e armazenar a versão mais atualizada do modelo para cada sessão colaborativa. A classe ServCollabT também é responsável pelo controle de concorrência distribuído, implementado no Servidor de Colaboração e descrita com mais detalhes na Subseção 4.4.4, onde os possíveis níveis de trava que o CoArgoUML permite são apresentados.

Um dos requisitos de colaboração a serem implementados está relacionado com o gerenciamento de segurança da sessão colaborativa. Como o Servidor de

Colaboração também é um protótipo, apenas um controle simples de usuário/senha foi implementado.

4.4.3 Edição de Elementos

A edição de elementos em um diagrama na ArgoUML, isto é, uma inclusão, exclusão ou alteração de alguma propriedade do elemento, envolve várias classes da ArgoUML e do GEF. Para exemplificar as classes envolvidas, considera-se a inclusão de um novo elemento em um diagrama.

Para inserir um elemento em um diagrama, o usuário deve clicar com o botão esquerdo do mouse no ícone do elemento da barra de ferramentas do diagrama. Esta ação modifica o modo do editor, ou seja, uma nova instância da classe *ModePlace* é criada e inserida na tabela *hash* da classe *ModeManager*, uma vez que ambas as classes pertencem ao GEF. Em seguida, o usuário clica no painel do diagrama para indicar que ele deseja criar o elemento que dispara, nesta ordem, os eventos *MousePressed*, *MouseClicked* e *MouseReleased* da classe *Editor* do GEF. Estes eventos, por sua vez, disparam chamadas a eventos com o mesmo nome para cada um dos modos que estiverem registrados no diagrama editado.

No caso da inclusão de um novo elemento, os eventos *MousePressed*, *MouseClicked* e *MouseReleased* da instância da classe *ModePlace* serão disparados. Em seguida, estes eventos fazem chamadas aos objetos da ArgoUML que herdam da classe *Diagram* do GEF, que por sua vez acessam o objeto da classe *Layer* para, finalmente, realizar a criação de um objeto da classe *Fig* do GEF, de acordo com a hierarquia de classes da Figura 4.2. A modificação e a exclusão de elementos em um diagrama seguem o mesmo raciocínio, porém manipulam classes diferentes.

Sob o ponto de vista do mapeamento, implementar a replicação dos elementos do modelo requer a modificação de algumas classes do GEF e da ArgoUML. A principal classe modificada para a replicação de elementos foi a classe Editor do GEF, pois é ela que faz o tratamento direto dos eventos gerados pelo mouse do usuário. As modificações na classe Editor incluem requisições, liberações e modificações de travas nos elementos antes que a ação que os disparou seja executada por meio dos métodos de comunicação em rede da classe ClienteConectaGEF.

Deste modo, o controle de concorrência pode cancelar um evento do usuário caso não tenha a autorização de manipulação concedida pela trava. Versões modificadas dos métodos *MousePressed, MouseClicked* e *MouseReleased* da classe Editor foram implementadas, aproveitando-se o código já pronto, pois as versões remotas reproduzem fielmente nos demais participantes da mesma sessão colaborativa a manipulação local nos elementos do diagrama. As modificações nos métodos *MousePressed, MouseClicked* e *MouseReleased* da classe Editor envolvem a verificação da existência da trava para um determinado elemento do diagrama e o cancelamento da ação que disparou o evento caso o usuário não possua a trava no elemento.

4.4.4 Mecanismo de Controle de Concorrência

Neste trabalho, optou-se pela utilização de travas como mecanismo para o controle de concorrência dos acessos aos elementos contidos no diagrama UML editado colaborativamente.

Para a aplicação das travas durante a edição colaborativa, cada elemento de modelagem foi considerado um objeto ou unidade. Dessa forma, a granularidade de compartilhamento e trava é o elemento de modelagem. Um estado de um diagrama de estados, um ator de um diagrama de caso de uso e um relacionamento que representa uma herança entre duas classes em um diagrama de classes são considerados exemplos de elementos de modelagem.

Todos os elementos de modelagem contidos no diagrama possuem um conjunto de propriedades cujos valores podem ser modificados pelos usuários durante a edição colaborativa, desde que os usuários consigam selecionar este elemento. Em termos práticos, é por meio do controle da seleção dos elementos que a trava libera ou não o acesso à manipulação das propriedades de um elemento no diagrama. Ao se criar um novo elemento no diagrama, coloca-se o usuário que o criou como dono da seleção inicial do mesmo, pois uma nova trava é automaticamente solicitada e concedida pelo Servidor de Colaboração. De maneira análoga à realizada pela maioria dos editores gráficos, para retirar a seleção de um objeto basta selecionar outro, ou colocar o mouse numa região vazia do editor e apertar o botão esquerdo, o que dispara uma liberação de todas as travas em poder do usuário naquele instante.

A geração de travas nas dependências dos elementos foi dividida em quatro níveis, para facilitar a edição colaborativa e evitar travas desnecessárias. Cada nível indicará ao algoritmo quais travas devem ser gerados no momento da seleção do elemento. As definições dos quatro níveis de travas são mostradas na Tabela 4.1.

Tabela 4.1 - Níveis de travas do mecanismo de controle de concorrência.

Nível	Descrição
1	A trava é solicitada apenas no elemento
2	A trava é solicitada no elemento e nas suas relações diretas
3	A trava é solicitada no elemento e na estrutura alcançável por ele
4	A trava é solicitada em todos os elementos do diagrama

No nível de trava 1 o usuário solicita uma trava apenas no elemento que deseja modificar. É comum este tipo de trava ocorrer durante o início da modelagem, onde existem poucos elementos no diagrama.

No nível de trava 2 o usuário solicita travas no elemento que deseja modificar e também em todas suas relações diretas. Aplica-se este nível de trava nos casos onde o elemento e suas relações devem ser modificados, como na mudança de posição do elemento no diagrama.

No nível de trava 3 o usuário solicita travas no elemento que deseja modificar e também em todas as relações e elementos alcançáveis a partir do elemento a ser modificado. Este nível é aplicado em diagramas onde a estrutura alcançável a partir do elemento pode sofrer alguma modificação.

O nível de trava 4 solicita travas em todos os elementos do diagrama. Este tipo de trava é utilizada em casos em que um determinado usuário deseja modificar todos os elementos, como a mudança do valor de uma propriedade comum a todos os elementos do diagrama.

A escolha do nível de trava depende do usuário e não do contexto e dos elementos envolvidos. Antes da identificação de quais relações o elemento possui, o algoritmo verifica qual é o nível de trava atual, obtém todas as relações do elemento no diagrama e gera as travas de acordo com o nível atual. A seguir um exemplo mostra como o algoritmo funciona.

Uma trava é solicitada o momento da criação de um novo elemento no diagrama. Como este elemento ainda não aparece nos modelos dos demais participantes, a trava é solicitada somente no elemento.

Se o usuário deseja criar uma relação entre elementos, é necessário verificar se estes elementos já possuem alguma trava. O algoritmo verifica as relações do elemento, de acordo com o nível atual, e solicita as travas nas relações e nos elementos antes de sua criação. Se os elementos envolvidos já possuírem alguma trava, o algoritmo cancela a ação que criou a relação.

Caso o usuário faça uma trava em um elemento existente no diagrama, o algoritmo verifica se ele possui relações, de acordo com o nível de trava atual. Se o elemento não as possui, somente uma trava no elemento é solicitada. Caso ele possua relações, o algoritmo verifica qual nível de trava utilizado no momento, para então enviar todas as solicitações de trava em outros elementos de acordo com o nível de trava atual.

Para facilitar a visualização de quais travas em quais elementos o usuário possui, a cada nova trava concedida o elemento é preenchido com a cor verde. A partir deste momento, o usuário pode realizar as operações que desejar e, após a liberação da trava devido à retirada da seleção deste elemento, o elemento volta à sua cor de preenchimento normal e as modificações são enviadas para o servidor de colaboração, responsável por notificar os participantes da mesma sessão colaborativa e atualização do seu modelo local. Quando um usuário recebe uma trava em um elemento, este mesmo elemento recebe a cor de preenchimento vermelha nos diagramas dos demais usuários, indicando que este elemento já está com um trava concedida a outro usuário.

Deste modo, os demais usuários podem visualizar facilmente os objetos disponíveis para receber novas travas e quais objetos contêm travas. As cores de preenchimento verde e vermelho foram escolhidas por analogia com um semáforo de trânsito, onde a cor verde indica que o motorista de um veículo pode seguir em frente e a cor vermelha indica ao motorista do veículo que ele deve aguardar a cor verde para prosseguir. Para os usuários do protótipo, o elemento que estiver preenchido com a cor verde pode ser modificado e o elemento que estiver preenchido com a cor vermelha não pode sofrer modificações.

Além da modificação da cor do elemento de acordo com as travas, a interface do CoArgoUML possui uma janela que mostra quais elementos contêm travas e quem são os proprietários destas travas.

Uma aba no painel de propriedades da interface do CoArgoUML permite a modificação do nível de trava atual. Esta aba está localizada no painel de propriedades da interface da ArgoUML e é codificada na classe TabLck, uma nova classe que herda da classe da TabText da ArgoUML. Esta aba contém quatro botões que permitem ao usuário trocar o nível de trava no momento que desejar. Um dos botões está sempre preenchido com a cor verde para indicar qual é o nível de trava atual. A Figura 4.5 apresenta a aba que permite a mudança de nível de trava. Pode ser visto pela Figura 4.5 que o nível de trava atual é o nível 3.

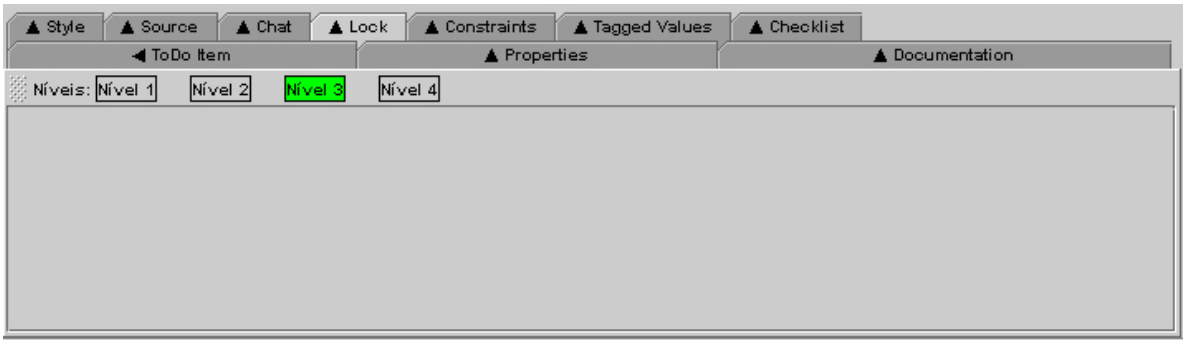

Figura 4.5 – Aba que permite a modificação do nível de trava.

4.4.5 Implementação dos Dispositivos de Presença

Para atender aos requisitos de comunicação entre usuários e percepção em grupo, três dispositivos de presença foram implementados no protótipo CoArgoUML: a janela de *chat*, a janela de travas e os *telepointers.* Além destes dispositivos de presença, o protótipo também conta com uma área de trabalho compartilhada, implementada no painel de diagrama que replica e recebe as modificações efetuadas por usuários remotos.

O *chat* é uma ferramenta simples utilizada na comunicação informal dos usuários durante a sessão colaborativa. Ele permite a interação dos participantes por meio da troca de mensagens não estruturas. No protótipo CoArgoUML, a comunicação entre os participantes, assim como a comunicação do modelo, é enviada para o Servidor de Colaboração, que contêm referências a todos os participantes da mesma sessão de colaboração.

A implementação do *chat* no protótipo consistiu na inclusão de uma nova aba no painel de propriedades da ArgoUML. Esta nova aba contém uma caixa de texto para que o usuário digite a mensagem, um botão para enviar a mensagem e uma lista que apresenta as mensagens recebidas. Para criar esta aba na interface da ArgoUML uma nova classe chamada TabCht foi criada por meio da herança da classe TabText. Desta maneira, as modificações necessárias nesta classe incluem a criação dos novos controles e o envio e recebimento das mensagens por meio dos métodos das classes ClienteConectaArgo e ClienteRebeceArgo.

A janela de travas é um dispositivo de presença utilizado para notificar os usuários da sessão colaborativa informando quais elementos possuem travas e seus proprietários. Esta janela tem finalidade informativa, pois não permite nenhum tipo de interação com o usuário. A janela fornece uma tabela com duas colunas: a primeira mostra o nome da classe do elemento que está travado e a segunda mostra o nome do proprietário da trava. A implementação desta janela utiliza a classe TabViewLocks, que também herda da classe TabText. Esta classe recebe as notificações de aquisições e liberações de travas diretamente da classe ClienteRebeceArgo, responsável por atualizar os dados da tabela apresentada nesta janela.

Um dos tipos de dispositivos de presença mais conhecidos e usados em sistemas *groupware* são os *telepointers*. Estes dispositivos assumem a forma de cursores identificados de maneira diferente para cada participante. Os *telepointers* permitem comunicar a localização, o movimento e até mesmo o foco de atenção dos usuários dentro do espaço de trabalho compartilhado.

Um *telepointer* é definido como um ponteiro virtual contendo a posição do mouse de todos os outros usuários conectados ao servidor num determinado instante. Os *telepointers* são representados por um sinal de “+” junto com o nome do usuário e sua cor.

Para captar a movimentação do mouse de um participante foi necessário monitorar a posição do ponteiro do usuário utilizando o evento *MouseMoved* da classe Editor do GEF. Desta maneira, a cada movimentação do mouse do usuário uma atualização da posição é enviada para o Servidor de Colaboração, que se encarrega de encaminhar esta modificação para os demais participantes. Para evitar uma sobrecarga da rede e da aplicação nesta tarefa, o envio da notificação de movimentação do ponteiro

do usuário ocorre apenas se o usuário moveu seu ponteiro para uma nova localização distante de, no mínimo, dez *pixels* da última posição conhecida.

A classe FigPointer, que é uma especialização da classe Fig do GEF, implementa os *telepointers* dos usuários. Desta maneira, o ponteiro é considerado mais um elemento do diagrama e, sempre que há necessidade de se mudar a localização do ponteiro, pode-se utilizar os métodos *SetLocation*, da classe FigPointer, e o método *Add*, da classe *Editor*.

A Figura 4.6 apresenta a interface gráfica do CoArgoUML, destacando os novos dispositivos de presença descritos a seguir. Na Figura 4.6, os usuários hirata e mauro modelam um diagrama de classes durante uma sessão colaborativa. Porém, somente a interface gráfica do usuário mauro é apresentada na Figura 4.6.

O item A da Figura 4.6 mostra a aba que contém a ferramenta de *chat* utilizada na comunicação entre os usuários durante a modelagem. O item B apresenta o *telepointer* do usuário hirata, indicando que ele está com o seu ponteiro perto da classe sem nome, cuja cor é vermelha devido à trava que o usuário hirata mantém nesta classe.

O item C da Figura 4.6 mostra a janela de travas, indicando que o usuário atual é o usuário mauro. Nesta Figura pode-se notar que existe um elemento do diagrama, identificado por FigClass, cuja trava está atribuída ao usuário mauro, além do elemento identificado por FigInterface, cuja trava está atribuída ao usuário hirata. O item D da Figura 4.6 apresenta a interface chamada <<interface>>, preenchida com a cor verde, pois o usuário mauro é proprietário de uma trava nesta interface.

Na janela de travas, indicada pelo item C da Figura 4.6, optou-se por apresentar o tipo de elemento e sua localização em relação ao painel do diagrama quando o nome do mesmo ainda não foi criado. Desta maneira, os usuários podem identificar mais facilmente quais são os elementos sem nome que possuem travas.

4.5 Discussão sobre a Aplicação do Mapeamento

Nesta seção discute-se a aplicação do Mapeamento de Componentes na ferramenta CASE ArgoUML.

Aplicar o mapeamento em uma aplicação complexa como a ArgoUML exigiu do autor um profundo conhecimento do código fonte da ArgoUML e também do *framework* GEF. Este conhecimento deve-se à grande quantidade de funcionalidades e requisitos implementados, assim como a complexidade apresentada na ArgoUML e no *framework* GEF.

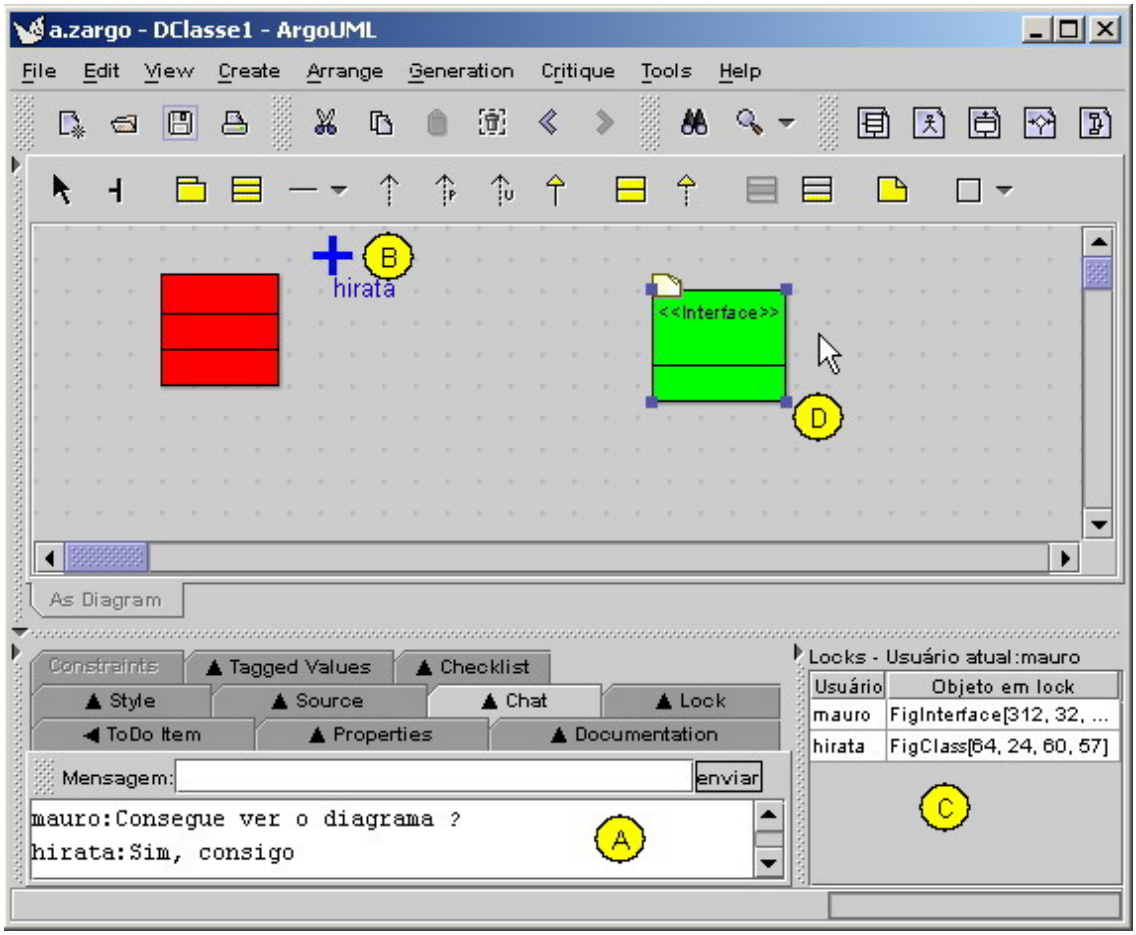

Figura 4.6 – Interface gráfica do CoArgoUML com os dispositivos de presença destacados.

Obter informações sobre como modificar a ArgoUML evolveu mais da metade do tempo total de implementação gasto pelo autor. Apesar de contar com a ajuda dos desenvolvedores oficiais da ferramenta, por meio da troca de mensagens na lista de discussão oficial do projeto, a documentação pobre aliada à falta de padronização do código fonte representou um grande dificuldade para a implementação do mapeamento.

O uso do mapeamento na ArgoUML envolveu a modificação dos componentes de subsistemas complexos. Em particular, os componentes dos subsistemas da Visão/Controle receberam uma atenção especial por serem altamente acoplados. Além

disso, identificar qual é a sua responsabilidade e qual é a sua função de alguns componentes da Visão/Controle representou um grande desafio, consumindo esforços e recursos adicionais do autor.

A experiência adquirida pelo autor durante a modificação de uma ferramenta complexa com a ArgoUML não só pode ser reutilizada em outras aplicações existentes, mas também representa um esforço da comunidade acadêmica em modificar aplicações existentes para suportar funcionalidades colaborativas. Em relação ao mercado de ferramentas CASE, o suporte à colaboração em uma ferramenta CASE de código livre representa um diferencial, permitindo que a ArgoUML se destaque em relação às seus concorrentes.

4.6 Comparação de Abordagens

Uma vez que o mapeamento de componentes da aplicação foi apresentado, é necessário comparar suas características com as abordagens existentes que proporcionam o uso colaborativo em aplicações não colaborativas. Esta comparação tem como objetivo auxiliar os desenvolvedores na escolha da abordagem a ser utilizada, além de fornecer uma contextualização do uso do mapeamento proposto, permitindo ao desenvolvedor compreender melhor qual tipo de abordagem é mais recomendada na implementação dos requisitos colaborativos desejados, de acordo com seu contexto.

Para comparar as abordagens existentes com o mapeamento de componentes, é necessário estabelecer alguns critérios comparativos. Os critérios escolhidos nesta comparação entre abordagens concentram-se principalmente na aplicabilidade das abordagens, baseado nas experiências de uso das abordagens relatadas em trabalhos relacionados e no estudo experimental das abordagens conduzido pelo autor deste trabalho.

Um critério a ser considerado antes da adoção de alguma abordagem leva em consideração a necessidade do código fonte. Este critério está diretamente relacionado com o modelo de licença que a aplicação segue, modelo este que pode ser dividido em dois tipos: aplicações abertas, isto é, aplicações que possuem uma licença onde o código fonte é disponibilizado; e aplicações proprietárias, onde a licença não permite que o código fonte seja disponibilizado publicamente. Na comparação de abordagens

apresentada nesta seção, quatro valores auto-explicativos serão especificados para o critério de disponibilidade do código fonte, a saber: requer o código fonte; não requer o código fonte; requer API do sistema operacional; e requer API da aplicação.

Outro critério importante a ser considerado na comparação das abordagens são os requisitos tecnológicos relacionados com a implementação. Estes requisitos influenciam diretamente o uso da aplicação a partir do momento que a abordagem for implementada. Por exemplo, a abordagem de *toolkits* requer uma linguagem de programação específica para que o *toolkit* possa ser utilizado, o que pode causar problemas de portabilidade decorrentes da necessidade deste requisito. O critério Requisito Tecnológico, utilizado na comparação das abordagens realizada nesta seção, pode assumir os seguintes valores: 'Depende da linguagem do *toolkit*', 'Requer arquitetura centralizada', 'Requer camada de software específico para cada aplicação', 'Depende da linguagem do Componente' e 'Requer aplicação no estilo arquitetural MVC'.

É fundamental compreender os contextos nas quais as abordagens se encontram inseridas, antes de considerar sua utilização. Enquanto algumas abordagens demandam a modificação direta do código fonte, outras devem ser consideradas apenas para novas aplicações. No critério do contexto, a abordagem pode possuir o valor "Criar novas aplicações colaborativas", para indicar o contexto de construção de novas aplicações por meio da abordagem, ou o valor "Promover colaboração entre aplicações", para indicar que a abordagem é utilizada para promover a colaboração em uma aplicação já existente. Nota-se que uma abordagem pode possuir os dois valores para este critério, como é o caso da abordagem de Substituição de Componentes.

A transformação de uma aplicação requer não apenas recursos computacionais e métodos para sua implementação, mas também fatores que podem tornar mais fácil ou mais difícil o uso de alguma abordagem. Para auxiliar a comparação entre a dificuldade de uso das abordagens, o critério 'Dificuldade de implementação' foi escolhido. Os valores discretos deste critério representam a dificuldade de utilização prática da abordagem, que pode ser classificada como "Baixa", "Média" e "Alta".

A classificação das abordagens envolve um estudo comparativo informal, com base no uso de cada uma das abordagens e levando em consideração a dificuldade encontrada pelo autor para colocar em prática um protótipo funcional utilizando cada uma das abordagens. Este protótipo funcional contou com: uma funcionalidade para a

comunicação, uma funcionalidade para a coordenação e uma funcionalidade para a cooperação.

Além da experiência prática coletada durante a utilização das abordagens, uma revisão bibliográfica foi conduzida para levantar dados sobre o grau de dificuldade de implementação de cada uma das abordagens. É importante notar que a classificação da dificuldade de utilização de uma abordagem na comparação descrita nesta seção é subjetiva. Idealmente, deve-se coletar métricas e medidas para uma comparação objetiva deste critério. Contudo, devido à falta de recursos disponíveis durante a elaboração deste trabalho, esta coleta de medidas não foi endereçada.

O mapeamento proposto foi comparado com as abordagens para desenvolvimento colaborativo apresentadas na Seção 2.7, isto é, as abordagens *toolkit*, a primeira geração de Sistemas de Colaboração Transparente, as abordagens ICT, ICT2 e ACT da segunda geração de Sistemas de Colaboração, a abordagem de Adaptação Transparente e a Substituição de Componentes. A Tabela 4.2 apresenta a comparação de abordagens de acordo com os critérios descritos.

De acordo com os dados da Tabela 4.2, o Mapeamento de Componentes deve ser utilizado no contexto das aplicações que possuem código fonte disponível, ou ao menos o código fonte dos componentes que constituem a aplicação. Devido a este valor para este critério, esperara-se que a maioria das aplicações candidatas para o uso do Mapeamento sejam as aplicações chamadas de aplicações de código livre, onde um grupo de desenvolvedores disponibiliza livremente o acesso ao código fonte da aplicação. Também é importante notar que, em comparação com as outras abordagens, o Mapeamento de Componentes é o único que requer não só a estruturação da aplicação em componentes, mas também uma organização lógica dos mesmos, pois o estilo arquitetural MVC é um dos requisitos para o uso do mapeamento.

Deste modo, a principal contribuição esperada pelo Mapeamento de Componentes é a proposta de que com esta abordagem softwares não colaborativos possam apresentar recursos colaborativos e, possivelmente, possibilitar a reutilização de aplicações existentes em um contexto colaborativo. Esta contribuição é relevante, pois possibilita associar em uma mesma solução as ferramentas já existentes, adequadas à tarefa e provavelmente familiares aos participantes, com as funcionalidades colaborativas necessárias para realizar esta tarefa em ambientes distribuídos colaborativos.

Tabela 4.2 – Comparações entre abordagens.

Abordagem	**Disponibilidade do Código Fonte da Aplicação**	**Requisito Tecnológico**	**Contexto**	**Dificuldade de Implementação**
Toolkits	Não requer o código fonte	Depende da linguagem do *toolkit*	Criar novas aplicações colaborativas	Média
Sistemas de Colaboração Transparente - 1º geração	Não requer o código fonte	Requer arquitetura centralizada	Promover colaboração entre aplicações	Baixa
Sistemas de Colaboração Transparente - 2º geração	Requer API do Sistema Operacional	Requer camada de software específico para cada aplicação	Promover colaboração entre aplicações	Alta
Adaptação Transparente	Requer API da aplicação	Requer camada de software específico para cada aplicação	Promover colaboração entre aplicações	Alta
Substituição de Componentes	Requer o código fonte da aplicação	Depende da linguagem do Componente	Criar novas aplicações colaborativas e Promover colaboração entre aplicações	Média
Mapeamento de Componentes	Requer o código fonte dos componentes da aplicação	Requer aplicação no estilo arquitetural MVC	Promover colaboração entre aplicações	Média

A abordagem ‘*Toolkits*’ é utilizada para a construção de novas aplicações colaborativas e não permite a promoção de colaboração entre aplicações já existentes e, por isso, não pode ser comparada diretamente com a abordagem ‘Mapeamento de Componentes’. Contudo, para as situações onde uma nova aplicação colaborativa deve ser construída, não é exagero admitir que a abordagem ‘*Toolkits*’ é a melhor opção disponível.

Comparando a abordagem ‘Mapeamento de Componentes’ com a abordagem ‘Sistemas de Colaboração Transparente - 1º geração’ pode-se verificar que a

dificuldade de implementação desta última é menor do que a primeira. Porém, como apresentado na Subseção 2.8.2, a primeira geração de sistemas de colaboração transparente não modifica a aplicação e, segundo alguns autores, este tipo de abordagem não suporta a colaboração adequadamente.

A abordagem ‘Sistemas de Colaboração Transparente - 2º geração’ não requer o código fonte da aplicação, em oposição à abordagem ‘Mapeamento de Componentes’. Contudo, para utilizar esta abordagem é necessário manipular uma API complexa e extensa, a API do sistema operacional. Deste modo, o desenvolvedor deve conhecer as principais operações que o usuário pode fazer na aplicação e também detalhes específicos do sistema operacional que não dizem respeito à aplicação, aumentando a dificuldade de implementação. Devido a esta característica, o desenvolvedor se distancia do foco principal, que é tornar a aplicação colaborativa, ao se aprofundar na programação da API do sistema operacional. Além disso, o compartilhamento fornecido por esta abordagem se limita à replicação de eventos.

A próxima abordagem listada na tabela, a abordagem ‘Adaptação Transparente’, possui um requisito tecnológico mais raro de se encontrar do que o código fonte: uma API da aplicação. Este tipo de requisito limita o uso desta abordagem à aplicações bem documentadas e que possuam uma API que permita a interceptação e reprodução de eventos, apresentado as mesmas limitações da segunda geração de sistemas de colaboração transparente. O desenvolvedor que optar por utilizar esta abordagem deve conhecer profundamente tanto à aplicação com sua API, uma vez que é por meio desta última que todas as funcionalidades colaborativas serão implementadas. Com base na experiência do autor no uso desta abordagem, a alteração direta do código fonte da aplicação pode requerer menos recursos do que a utilização da API da aplicação.

A abordagem ‘Substituição de Componentes’ é abordagem mais próxima da abordagem ‘Mapeamento de Componentes’, em relação aos critérios utilizados nesta comparação. Contudo, o Mapeamento de Componentes se destaca em relação à Substituição de Componentes por não requerer recursos técnicos específicos, como uma determinada linguagem de programação ou ambiente de desenvolvimento, durante a implementação da abordagem. É interessante notar que, em certas aplicações, as abordagens ‘Substituição de Componentes’ e ‘Mapeamento de Componente’ podem ser utilizadas em conjunto, especialmente quando o mapeamento indicar que algum

componente não pode ser modificado e deve ser substituído. Contudo, para que o desenvolvedor utilize estas duas abordagens em conjunto, é necessário que todos os requisitos de ambas as abordagens sejam levados em consideração.

4.7 Sumário

Neste capítulo apresentou-se como o mapeamento de componentes proposto foi utilizado na criação de uma prova de conceito do mapeamento. A ferramenta ArgoUML, escolhida para receber o protótipo, é uma ferramenta CASE de modelagem de código livre e que segue o estilo arquitetural MVC. As funcionalidades colaborativas desejadas para o protótipo foram selecionadas de acordo com o estudo de algumas ferramentas colaborativas atuais, como o *Poseidon for UML*. Neste capítulo mostrou-se como os componentes da ArgoUML são organizados em subsistemas e como o *framework* GEF interage com funcionamento da ferramenta. Em seguida, as modificações na ferramenta foram apresentadas por meio de descrições em alto nível de abstração e sem entrar em detalhes específicos que envolvem o código fonte.

Por fim, apresentou-se uma comparação entre as principais abordagens encontradas na bibliografia e a abordagem deste trabalho, contextualizando e evidenciando as contribuições da proposta.

Capítulo V

Experimento Controlado

Neste capítulo descreve-se a avaliação de usabilidade através de um experimento controlado. Com o objetivo de avaliar a usabilidade do produto final do mapeamento, um experimento controlado foi conduzido levando em consideração o contexto da modelagem colaborativa de diagramas UML.

No capítulo apresenta-se o *CSCW Lab*, a metodologia de avaliação utilizada durante o experimento, e a análise dos dados. Em seguida, são apresentados os detalhes dos usuários e do ambiente utilizado no experimento. O capítulo é finalizado com algumas observações sobre a realização do experimento.

5.1 Contexto do Experimento

A elaboração de diagramas UML, assim como vários artefatos gerados durante o desenvolvimento de software, é utilizada para ajudar a modelar, documentar e explicar o problema e a solução, além de facilitar a comunicação entre os integrantes do projeto de construção de software. Uma afirmação que se pode fazer sobre esta atividade é que, quanto mais idéias e contribuições para o modelo surgirem, maiores são as chances de se atingir um resultado final de qualidade. Portanto, deve-se buscar situações onde se estimule o maior número de interações possível.

A criação de uma ferramenta para auxiliar a colaboração remota na modelagem de diagramas UML é discutida sob vários aspectos, assim como a inserção de técnicas colaborativas no processo de desenvolvimento de aplicações colaborativas. Estudos iniciais mostram que as principais vantagens envolvem ganhos de produtividade e melhorias na

qualidade dos artefatos produzidos. Com base nestes estudos iniciais, um experimento controlado foi conduzido para procurar evidências que de o uso de ferramenta colaborativa para apoio à modelagem de diagramas UML aumente a satisfação dos usuários durante a tarefa de modelagem.

O objetivo do experimento é verificar a satisfação dos usuários durante uma modelagem de diagramas UML colaborativa. Esta satisfação pode ajudar a validar a implementação dos requisitos especificados para o protótipo.

Além de procurar por evidências da satisfação dos usuários durante a modelagem colaborativa, o experimento controlado também tem como objetivo coletar dados para analisar o esforço dos usuários e a usabilidade do protótipo construído. Desta maneira, pode-se verificar empiricamente se o mapeamento proposto pode ser aplicado com sucesso a uma ferramenta de interação monousuária, indicando que o uso da versão colaborativa da ferramenta pode trazer benefícios para seus usuários.

É importante destacar que uma avaliação formal do mapeamento proposto não é abordada neste trabalho. Do mesmo modo que as avaliações formais de *frameworks*, a avaliação formal do mapeamento demanda o uso controlado desta abordagem em diferentes aplicações de diferentes contextos. Devido a necessidade de recursos específicos e à falta de pesquisas na área de avaliações formais de abordagens como o mapeamento e *frameworks*, este tipo de avaliação não será considerada neste trabalho.

O experimento controlado apresentado neste capítulo constou da observação tanto das interações e o comportamento dos grupos dos usuários, como também do modo pela qual a aplicação forneceu apoio ao processo de elaboração de diagramas desde a idéia conceitual até a versão final do diagrama. O experimento foi conduzido para prover conclusões estatísticas significativas sobre o seguinte:

- Satisfação dos requisitos colaborativos
- Produtividade geral e qualidade do trabalho colaborativo
- Efeitos do perfil de usuários no trabalho colaborativo
- Satisfação e confiança dos usuários

A hipótese do experimento é a seguinte: o suporte colaborativo aplicado à modelagem de diagramas UML vai tornar mais satisfatória para os usuários a tarefa de modelagem de diagramas. A investigação de vantagens e desvantagens da edição colaborativa de diagramas UML, por meio da análise dos dados coletados durante o experimento descrito neste capítulo, vai fornecer a base para a comprovação, ou não, da hipótese.

5.2 Metodologia Utilizada

A metodologia utilizada para o experimento segue as sugestões propostas pelo *CSCW Lab* de Araújo et al. [67]. A abordagem do *CSCW Lab* é uma proposta que sugere o uso de metodologias de avaliação no contexto de um grupo de pesquisa de *groupware*. O seu principal objeto é definir um método para a avaliação de sistemas *groupware* que contenha passos para conduzir uma avaliação, além de heurísticas para o uso de técnicas e instrumentos disponíveis.

De acordo com o *CSCW Lab*, uma avaliação de *groupware* deve se concentrar em uma ou mais dimensões de um total de quatro dimensões, a saber: Contexto do Grupo, Usabilidade, Colaboração e Impacto Cultural. A Figura 5.1 apresenta como estas dimensões se relacionam.

O objetivo da dimensão Contexto do Grupo refere-se à heterogeneidade do grupo a ser avaliado. Advoga-se a dificuldade de encontrar um grupo ideal para conduzir avaliações devido às singularidades de cada um dos participantes do grupo. Como estratégia, o *CSCW Lab* propõe a identificação de variáveis, métricas e instrumentos para medir o contexto do grupo levando em consideração os fatores psicológicos, etnográficos e sociológicos dos participantes.

A dimensão Usabilidade envolve o uso do *groupware* e o nível de aceitação por parte dos usuários. Avaliar a usabilidade, de acordo com o *CSCW Lab*, envolve a avaliação de como o usuário utilizou o *groupware* e de como o grupo interagiu com ele.

A Colaboração como dimensão no *CSCW Lab* tem como objetivo verificar se o *groupware* conseguiu atingir seu objetivo principal, isto é, promover a colaboração. Como estratégia para esta verificação, o *CSCW Lab* propõe o uso de Modelos de Colaboração para identificar níveis de colaboração em domínios específicos e a análise das características de cada nível.

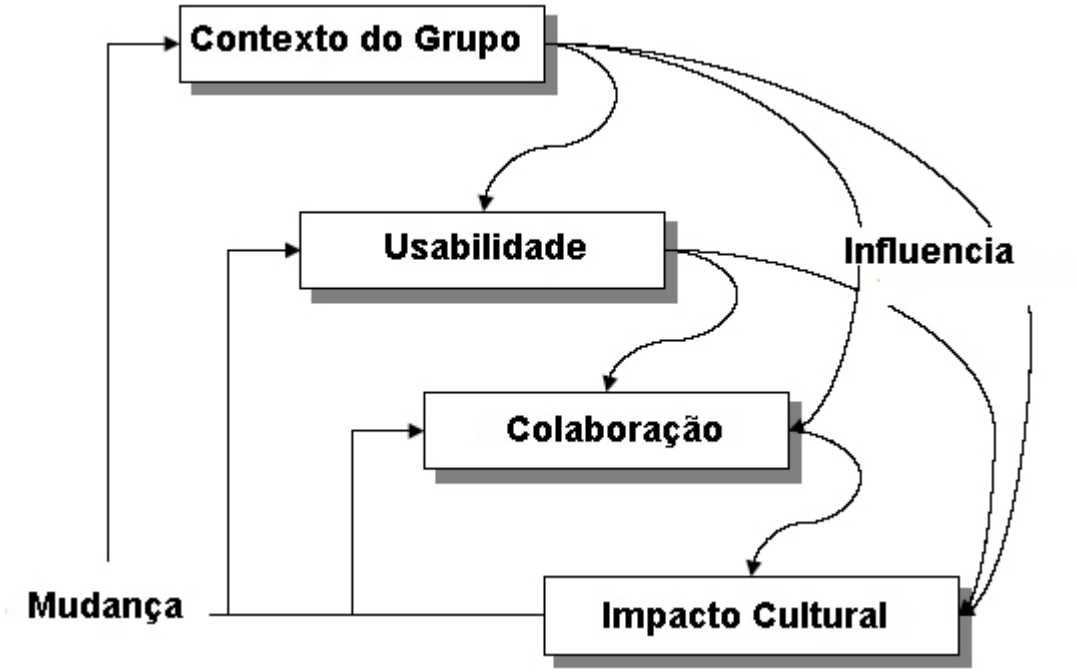

Figura 5.1 – Relacionamento das dimensões do *CSCW Lab*. Traduzido de [67].

O Impacto Cultural está diretamente relacionado com a maneira pela qual o grupo trabalha, como é a união do grupo, as maneiras esperadas e inesperadas que o *groupware* foi utilizado, entre outros fatores. O Impacto Cultural afeta tanto o nível individual como os níveis de grupo e organizacionais em termos de impressões, percepções, satisfações, comprometimento, aprendizado e mudança de atitude.

Do ponto de vista de metodologia de ensino, durante o aprendizado dos grupos no experimento, a metodologia utilizada foi a resolução de problemas. Esta metodologia pode constituir tanto um conteúdo educativo como um modo de conceber as atividades educativas. O ensino baseado na resolução de problemas supõe fomentar nos alunos o domínio de procedimentos para dar respostas a situações distintas e mutáveis.

O uso do protótipo descrito no Capítulo 4 foi aplicado em um experimento executado em um ambiente controlado, simulando-se a situação de colaboração. Neste contexto, a avaliação do experimento é classificada como somativa-quantitativa em um experimento controlado.

5.3 Participantes do Experimento

Para avaliar a eficiência da modelagem colaborativa, o experimento controlado foi conduzido com 14 sujeitos, sendo que a maioria deles estudantes do curso de pós-graduação em Ciência da Computação do ITA (Instituto Tecnológico de Aeronáutica). Os estudantes afirmaram possuir algum conhecimento prévio de modelagem de diagramas UML e foram divididos em 7 pares de forma aleatória, sem repetições, para que trabalhassem em duplas durante a criação colaborativa de diagramas de casos de uso e de classes. Determinou-se a quantidade de participantes do experimento com base nas pesquisas de avaliação de *groupware* encontradas na literatura, como o estudo de dispositivos de presença apresentado por Gutwin e Greenberg [9], e na quantidade de recursos disponíveis.

Os usuários que participaram da colaboração não sofreram nenhuma restrição sobre quais ações poderiam ser executadas na ferramenta de modelagem. Isto é, eles possuíam total liberdade para criar, arrastar, apagar e modificar as propriedades dos elementos no diagrama em que trabalhavam em conjunto. A única restrição imposta durante a modelagem em conjunto é relacionada com o uso concorrente das travas durante a modelagem. Os estudantes não apresentaram nenhum papel predefinido durante a modelagem em conjunto, evitando, desta maneira, qualquer tipo de prioridade, discriminação ou desigualdade nas operações dos usuários.

Uma sessão do experimento, que durou cerca de duas horas, foi conduzida para cada um dos 7 pares. Todas as sessões do experimento realizaram-se entre novembro de 2005 e janeiro de 2006 em duas salas do ITA. Uma carta convite foi enviada a todos os usuários interessados em participar do experimento, descrevendo sumariamente como o mesmo funcionaria, sem entrar em detalhes específicos sobre as tarefas. Nesta carta,

descreveu-se de forma breve sobre o que se tratava o experimento, sem entrar em detalhes sobre as tarefas a serem executadas. Para motivar a participação dos usuários, dois brindes foram sorteados entre todos os participantes.

5.4 Ambiente do Experimento

Um ambiente físico adequado foi montado para que o experimento fosse realizado, contando com dois cenários: o Cenário A e o Cenário B. A diferença entre os cenários está no mecanismo de comunicação utilizado.

O ambiente utilizado durante a experiência foi composto por duas salas separadas geograficamente, cada uma contendo um computador equipado com o protótipo CoArgoUML e uma conexão de rede. Além dos computadores que representam a estação de trabalho dos usuários, cada sala contou com um monitor que observou o comportamento dos usuários e uma câmera de vídeo que fez a gravação da face dos usuários enquanto os mesmos participavam da experiência. Foram utilizadas as letras A e B para facilitar a identificação de cada usuário de cada par. Este ambiente está ilustrado na Figura 5.2, que representa o Cenário A.

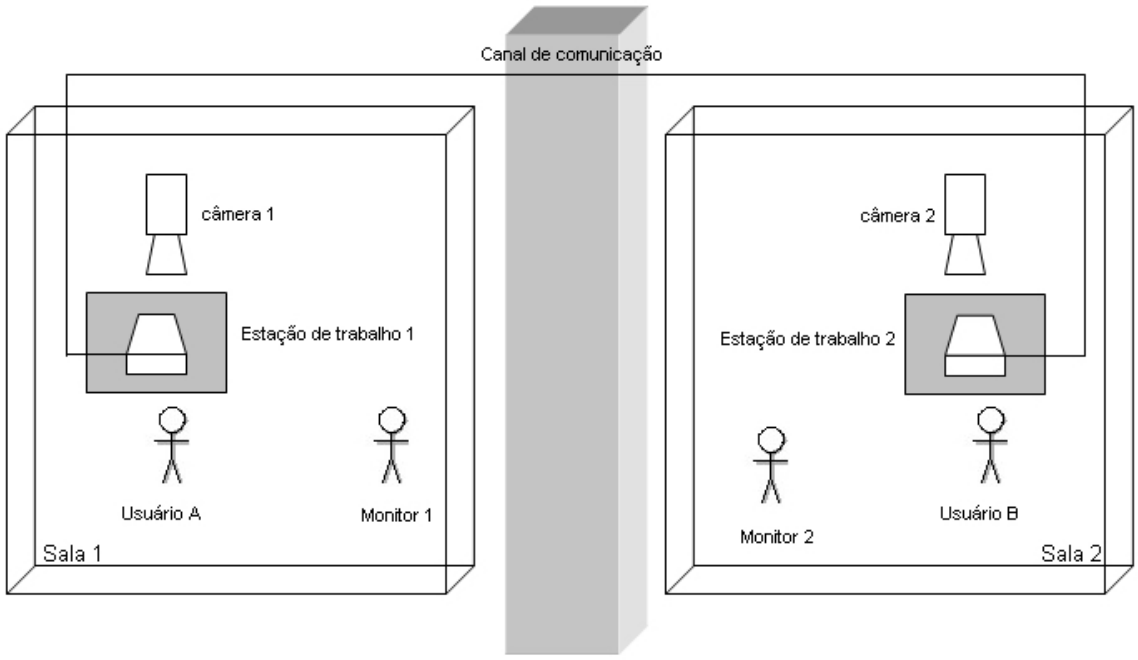

Figura 5.2 - Cenário A da experiência, onde a comunicação foi feita por meio do *chat*.

O Cenário B possui a mesma estrutura do Cenário A, porém, neste caso, os usuários contaram com fones de ouvido com um microfone acoplado para permitir uma áudio conferência proporcionada por um software de VOIP (Voz Sobre IP). Deste modo, no Cenário B os usuários utilizaram a ferramenta de comunicação integrada na aplicação, o *chat* textual, ou a áudio conferência proporcionada pela ferramenta de VOIP. A Figura 5.3 apresenta o Cenário B.

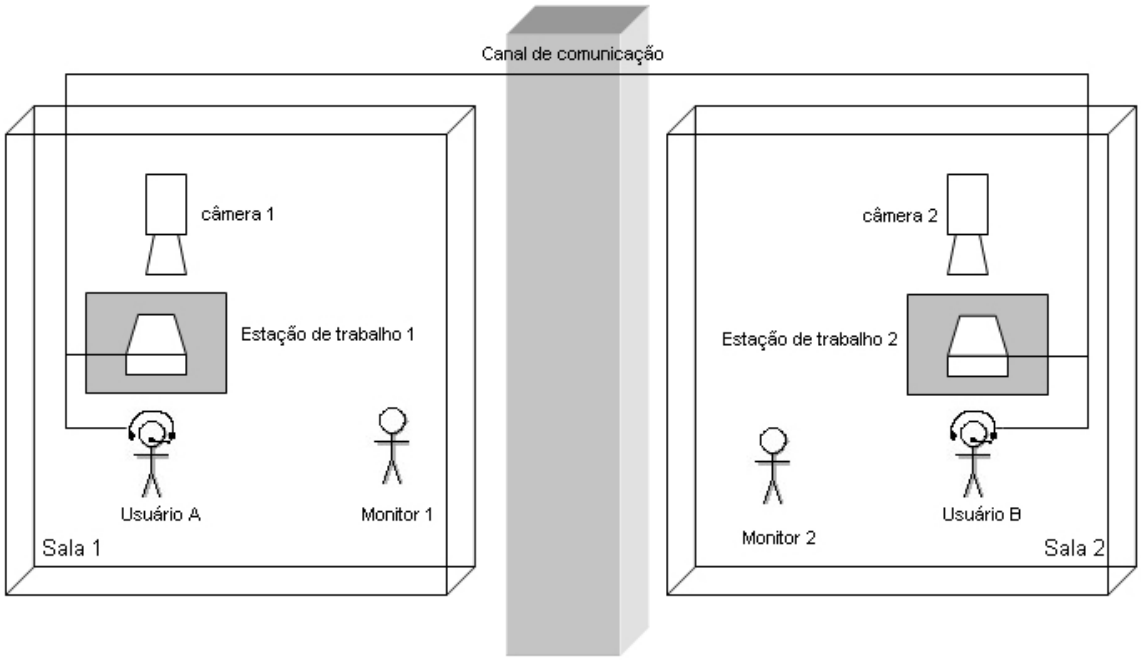

Figura 5.3 - Cenário B da experiência, onde a comunicação foi feita por meio de áudio conferência.

Os participantes foram divididos em dois conjuntos: no primeiro conjunto os grupos 1, 2 e 3 utilizaram um *chat* como canal de comunicação. No segundo conjunto os grupos 4, 5, 6 e 7 utilizaram um *software* de áudio conferência para se comunicar. A divisão dos pares de usuários em dois conjuntos tem como objetivo identificar a influência de diferentes canais de comunicação durante a colaboração. A importância do uso de diferentes canais de comunicação não é o principal objetivo do experimento, porém coletar dados sobre a utilização destes canais de comunicação fornece dados relevantes sobre como o conteúdo da comunicação e a forma que ela foi conduzida.

Para coletar os dados gerados durante a experiência quatro métodos diferentes foram utilizados:

- A observação do participante durante a sessão colaborativa. Durante as sessões colaborativas os monitores analisaram o comportamento dos usuários através das anotações pessoais sobre o comportamento dos usuários. Ao término do experimento, as gravações dos comportamentos dos usuários foram analisadas com o objetivo de coletar dados adicionais;

- Medidas das métricas de software coletadas pela ferramenta. O CoArgoUML gerou várias medidas durante o experimento. Os mecanismos utilizados na captura destas medidas incluíram o registro da conversa do bate papo, a quantidade de travas por usuário, a quantidade de elementos criados e excluídos no diagrama por usuário e tempo da sessão colaborativa durante a execução de cada tarefa;

- Respostas dos questionários. Antes do início do experimento, os usuários receberam um questionário com perguntas que ajudaram os monitores a identificar o perfil de cada usuário. Após o término de cada tarefa, os usuários foram instruídos a preencher um pequeno questionário contendo questões sobre o esforço mental. Ao término do experimento um questionário final foi apresentado aos usuários, com o objetivo de medir vários aspectos gerais do experimento. Os questionários de esforço mental foram elaborados com base no estudo apresentado por Gutwin e Greenberg [9].

- Entrevistas com os usuários. Após o término do experimento uma pequena entrevista foi feita com cada usuário. Esta entrevista, conduzida pelos monitores, teve como objetivo captar alguma sensação que dificilmente é identificada pelos questionários, pela observação do comportamento dos usuários e pelos registros da ferramenta.

Antes de iniciar as tarefas monitoradas no experimento, cada participante preencheu um pequeno formulário sobre seu perfil. Em seguida, três tutoriais que ensinam os principais componentes da interface gráfica do CoArgoUML foram encaminhados para cada um dos usuários. Estes tutoriais explicaram passo a passo como montar um diagrama

simples de casos de uso e de classes a partir de duas situações apresentadas. O principal objetivo destes tutoriais foi familiarizar o usuário com a interface da ferramenta e com a modelagem de casos de uso e de classes no CoArgoUML a partir de um cenário fictício apresentado na forma de um texto.

Durante a execução dos tutoriais, os usuários não colaboraram, porém aprenderam a utilizar os canais de comunicação, o ambiente compartilhado e o controle de concorrência por meio das travas. Os monitores esperaram que cada membro do par terminasse os três tutorais, de modo que a tarefas colaborativas começaram somente após o par terminar os três tutoriais. Todos os documentos utilizados durante a experiência, assim as respostas dos questionários e os registros coletados durante a experiência, foram gravados no CD que acompanha este trabalho.

Os pares iniciaram a execução das tarefas após terminarem os tutoriais. Para cada tarefa uma nova sessão colaborativa foi iniciada e cada um dos participantes recebeu um documento explicando o que deveria ser feito e qual seria sua denominação naquela tarefa, ou seja, se o participante seria o usuário A ou o usuário B. Após a entrega do texto, os monitores iniciaram a contagem do tempo da sessão colaborativa que terminou somente quando os dois participantes sinalizaram, simultaneamente ou não, o término da modelagem para aquela tarefa.

Cada tarefa apresentada aos usuários continha uma pequena introdução indicando aos usuários que a modelagem não precisava ser completa, ou seja, somente os principais elementos do diagrama deveriam ser modelados. As propriedades, atributos, métodos e demais características dos elementos de modelagem não precisavam ser especificados. A exceção a regra é o uso do nome dos elementos no diagrama. Deste modo, a experiência enfatizou a capacidade geral de modelagem dos usuários, evitando o uso de detalhes auxiliares do modelo. Também foi indicado aos usuários que em caso de qualquer dúvida eles poderiam fazer perguntas para os monitores.

As tarefas apresentadas aos usuários envolvera a criação de diagramas UML a partir de um cenário fictício fornecido a ambos os usuários. Estas tarefas apresentaram um tempo limite de 30 minutos, com 10 minutos de tolerância, administrado pelos monitores.

A tarefa 1 consistia em modelar um diagrama de classes a partir de um texto que descrevia, com poucos detalhes, como funciona uma partida de futebol. Este tema foi escolhido por ser de fácil compreensão e por, supostamente, ser familiar a todos os usuários que participaram do experimento. Os membros do par receberam a denominação de usuário A e usuário B para poderem se identificar durante a sessão colaborativa. Os usuários receberam exatamente a mesma descrição do cenário da tarefa 1, apresentado a seguir:

"Imagine-se tendo que explicar a um cidadão dos E.U.A. o que é futebol. Esta explicação precisa ser feita através de um diagrama de classes da UML, que você e seu parceiro devem modelar de acordo com a descrição a seguir. Para evitar criar um modelo muito grande, procure modelar somente o que estiver especificado na descrição a seguir.

O jogo de futebol é realizado por duas equipes de jogadores. Cada equipe é composta por 11 jogadores com diferentes funções: goleiro, defesa, meio de campo, laterais e atacantes. Somente um goleiro pode existir em cada equipe.

O jogo é realizado num campo com medidas regulares (em comprimento e largura) e tem duas balizas colocadas em extremos opostos do campo. Ganha o jogo a equipe que marcar mais gols na baliza do adversário durante dois tempos de quarenta e cinco minutos cada. No jogo existe apenas uma bola que também apresenta características regulares (diâmetro, peso etc). O jogo é mediado por 3 árbitros: um é o árbitro principal e os outros dois são auxiliares."

Na segunda tarefa, uma pequena modificação foi acrescentada: definiu-se aleatoriamente que um dos membros do par seria o usuário A e o outro seria o usuário B. O usuário A desta vez receberia uma parte de cenário a ser modelo e o usuário B receberia outra parte. Deste modo, as duas partes do cenário apresentadas aos usuários contêm informações complementares sobre o cenário fictício.

Na tarefa 2, os usuários deveriam modelar um diagrama de casos de uso de acordo com as informações complementares que lhes foram apresentadas. No documento com a tarefa 2 apresentado aos usuários foi indicado explicitamente que cada um deles tinha apenas parte do cenário e que eles deveriam coordenar qual papel cada um desempenharia durante a modelagem do diagrama de classes.

Mais uma vez, o cenário fictício escolhido apresenta uma situação que, supostamente, todos os usuários já conhecem, além de apresentar características reais de

um cenário de um caso de uso: a descrição do funcionamento de um consultório odontológico. Nesta colaboração, esperava-se que uma grande quantidade de informações sobre o cenário fosse trocada, uma vez que os usuários deveriam se comunicar para trocar as informações complementares do cenário.

Para ambos os usuários participantes da tarefa 2, o seguinte trecho comum foi apresentado:

"Um dentista quer automatizar o atendimento aos pacientes de seu consultório.

Quando um paciente deseja marcar uma consulta, é verificada a agenda do dentista e oferecido o primeiro horário disponível (data e hora), de acordo com o que o paciente deseja. Se o paciente concordar com o horário, é registrado na agenda o nome do paciente e horário combinado.

Os pacientes já cadastrados têm a ficha de consulta preenchida automaticamente. Os pacientes novos devem fornecer seus dados de cadastro: RG, endereço, telefone, data nascimento, profissão. A consulta consiste de dois tipos de serviços: de limpeza e restauração, ou exames para diagnóstico. Na realização da consulta, o dentista faz o registro do serviço efetuado em detalhes e, se necessário, o paciente marca uma nova consulta. O dentista pode pesquisar as fichas de seus pacientes por nome ou data de consultas. Diariamente a agenda é impressa com 2 dias de antecedência para que os pacientes confirmem a consulta. Também pode ser impressa a agenda do dia ou da semana."

O trecho exclusivo apresentado para o usuário A na tarefa 2 foi o seguinte:

"Um dentista quer automatizar o atendimento aos pacientes de seu consultório.

Se o paciente concordar com o horário da consulta é registrado na agenda o nome do paciente e horário combinado. A consulta consiste de dois tipos de serviços: de limpeza e restauração, ou exames para diagnóstico. Na realização da consulta, o dentista faz o registro do serviço efetuado em detalhes e, se necessário, o paciente marca uma nova consulta. Diariamente, é agenda de consultas com dois dias de antecedência é impressa para que os pacientes confirmem a consulta."

O trecho exclusivo apresentado para o usuário B na tarefa 2 foi o seguinte:

"Um dentista quer automatizar o atendimento aos pacientes de seu consultório. Quando um paciente deseja marcar uma consulta, é verificada a agenda do dentista e oferecido o primeiro horário disponível (data e hora), de acordo com o que o paciente deseja. Os pacientes já cadastrados

têm a ficha de consulta preenchida automaticamente. Os pacientes novos devem fornecer seus dados de cadastro: RG, endereço, telefone, data nascimento, profissão. O dentista pode pesquisar as fichas de seus pacientes por nome ou data de consultas. As fichas e a agenda do dia ou da semana podem ser impressas."

Por fim, a terceira tarefa apresentou aos usuários uma situação onde os usuários deveriam modificar um diagrama de classes pronto e que deveria sofrer algumas modificações. O cenário da tarefa 3 descreve o funcionamento de uma locadora de DVDs, no que diz respeito aos preços e gêneros de filmes com os quais esta locadora trabalha. Da mesma forma que na tarefa 2, cada usuário recebeu apenas uma parte do cenário a ser modelado, porém desta vez as informações a serem inseridas no diagrama são distintas e não complementares.

Os usuários também receberam a indicação de que cada um receberia informações diferentes e que deveriam coordenar suas ações durante a modelagem do diagrama de classes. Nesta colaboração, esperava-se que houvesse uma coordenação prévia das modificações a serem implementadas no diagrama inicial, uma vez que cada usuário possuía informações diferentes e o mecanismo de travas pressupõe uma seqüência de ações serializadas. Na tarefa 3 houve uma troca de usuários. Ou seja, o usuário nomeado A na tarefa 2 recebeu o nome de usuário B na tarefa 3 e vice-versa.

Para ambos os usuários participantes da tarefa 3 o seguinte trecho comum foi apresentado: "Um sistema que faz o controle de locações de DVDs foi modelado utilizando um diagrama de classes da UML. Uma parte do modelo pode ser visto na Figura 1:

Figura 1 - Diagrama de Classes incompleto.

Os responsáveis pela modelagem do sistema foram desligados da equipe antes de terminar a modelagem do diagrama. Você e seu par devem terminar o diagrama de classes acima adicionando novas informações ao modelo."

O trecho exclusivo apresentado para o usuário A na tarefa 3 foi o seguinte:

“O sistema deve fazer incluir uma classificação referente ao gênero dos filmes que você deve modelar no diagrama de classes. Em relação ao gênero, os filmes são classificados como: Aventura, Comédia, Documentário, Drama, Ficção.

Modele esta classificação no diagrama de classes atual. Você e seu par podem incluir, alterar ou remover quaisquer elementos no diagrama.”

O trecho exclusivo apresentado para o usuário B na tarefa 3 foi o seguinte:

“O sistema deve fazer incluir uma classificação referente ao preço de locação dos filmes que você deve modelar no diagrama de classes. Em relação ao preço de locação, os filmes são classificados como: Preço Regular, Preço de Lançamento, Preço Promocional.

Modele esta classificação no diagrama de classes atual. Você e seu par podem incluir, alterar ou remover quaisquer elementos no diagrama.”

Para auxiliar a compreensão da ordem dos eventos ocorridos durante as sessões colaborativas do experimento, a Tabela 5.1 apresenta a ordem correta dos eventos e a descrição dos mesmos. De todos os eventos do experimento, apenas os eventos 9, 10, 11, 12, 13 e 14 foram gravados pela câmera de vídeo.

5.5 Observações Gerais

As sessões do experimento não apresentaram algumas dificuldades aos monitores. Esta seção tem como objetivo descrever algumas observações gerais que ocorreram durante as sessões do experimento.

Inicialmente, uma lista de usuários candidatos foi elaborada para pré-selecionar quem seria chamado para participar da experiência. Devido à compromissos pessoais dos candidatos os monitores encontraram dificuldades para encontrar um horário comum na agenda dos membros de cada par. Estas dificuldades geraram pequenos atrasos nas datas originais das sessões colaborativas que não afetaram os dados coletados.

Outro detalhe a cerca dos usuários que não pôde ser evitado pelos monitores foi a familiaridade dos usuários uns com os outros. O experimento foi planejado de modo que os usuários não soubessem com que estivem colaborando, porém devido à proximidade e ao

fato que todos os usuários pertencerem à mesma instituição de ensino alguns usuários já conheciam seus parceiros.

Tabela 5.1 – Ordem correta dos eventos em cada sessão do experimento.

Ordem	**Descrição do evento**
1	Documento de boas vindas
2	Questionário pré-experimento
3	Tutorial número 1
4	Avaliação de esforço do Tutorial número 1
5	Tutorial número 2
6	Avaliação de esforço do Tutorial número 2
7	Tutorial número 3
8	Avaliação de esforço do Tutorial número 3
9	Tarefa número 1
10	Avaliação de esforço da tarefa número 1
11	Tarefa número 2
12	Avaliação de esforço da tarefa número 2
13	Tarefa número 3
14	Avaliação de esforço da tarefa número 3
15	Questionário pós-experimento
16	Entrevista com o Monitor

Uma das sessões do experimento apresentou problemas técnicos relacionados com o armazenamento dos registros, inviabilizando a coleta de dados. Os dados coletados nesta sessão do experimento em particular foram descartados completamente pelos monitores e não constam nas análises do Capítulo 6. O autor deste trabalho assumiu o papel de monitor nos experimentos, agindo apenas como observador e respondendo a dúvidas dos usuários sobre o uso do protótipo.

O ambiente do experimento contou com o protótipo CoArgoUML, descrito no Capítulo 4. Entretanto, nas duas primeiras sessões, o protótipo apresentou instabilidades, gerando travamentos e um comportamento inconsistente com a sua especificação. Estas instabilidades causaram travamos na aplicação, não permitindo aos usuários terminarem suas tarefas.

Nestas situações, os monitores reiniciaram a tarefa, sem nenhuma perda de trabalho dos usuários, e descontaram o tempo decorrido com o ajuste do ambiente do tempo

total gasto na tarefa. Nas demais sessões do experimento a ferramenta foi corrigida e nenhum problema que interrompesse a sessão colaborativa foi detectado.

Os monitores observaram ainda que, durante o período de treinamento onde os usuários seguiram os passos dos tutoriais, os usuários não estavam familiarizados com a colaboração durante a modelagem e nem com o uso do mecanismo de travas. O tutorial 3, em particular, foi modificado após a primeira sessão colaborativa para ser mais claro, curto e didático.

5.6 Sumário

Neste capítulo apresentam-se os detalhes do experimento conduzido para avaliar a usabilidade do protótipo construído.

O objetivo do experimento é verificar a satisfação dos usuários durante uma modelagem de diagramas UML colaborativa. Esta satisfação pode ajudar a validar a implementação dos requisitos especificados para o protótipo.

A hipótese do experimento estabelece que o suporte colaborativo aplicado à modelagem de diagramas UML vai tornar mais satisfatória para os usuários a tarefa de modelagem de diagramas. A investigação de vantagens e desvantagens da edição colaborativa de diagramas UML, por meio da análise dos dados coletados durante o experimento descrito neste capítulo, vai fornecer a base para a comprovação, ou não, da hipótese.

A metodologia do *CSCW Lab*, que apresenta regras para a elaboração de experiências voltadas para a avaliação de sistemas *groupware* baseado em dimensões utilizada durante a elaboração do experimento, foi descrita sumariamente.

Em seguida, os participantes do experimento foram descritos assim como todos os detalhes de cada passo do ambiente, tutorais e tarefas que compuseram o experimento. Por fim, algumas observações sobre o experimento são apresentadas com o objetivo de descrever o nível do impacto de certos eventos nos resultados.

No próximo capítulo, será feita uma análise dos dados coletados durante o experimento.

Capítulo VI

Análise dos Dados

Neste capítulo descreve-se a análise dos dados coletados durante o experimento descrito no capítulo anterior. Os dados coletados são totalizados e apresentados com o objetivo de comprovar a hipótese do experimento. Após a apresentação dos dados mais relevantes, o capítulo termina apresentado algumas observações sobre as análises dos dados.

6.1 Dados Coletados

Após o término da experiência, o processo de coleta, limpeza, refinamento e acerto de dados foi iniciado. Dentre as principais fontes de dados, os dados coletados por meio dos questionários receberam a maior quantidade de análises apresentadas neste capítulo.

A partir dos dados do questionário de perfil, também conhecido como questionário pré-experimento, obtiveram-se as informações sobre o perfil dos usuários. Por exemplo, a maioria dos usuários disse já ter trabalhado com uma ferramenta de modelagem e se diz experiente na elaboração de diagramas UML, porém algumas análises qualitativas indicaram o contrário.

O questionário de avaliação de esforço, preenchido pelos usuários, após cada tutorial e cada tarefa, foi utilizado para coletar dados sobre o esforço cognitivo dos usuários. Com base no estudo apresentado por Gutwin e Greenberg [9], as respostas deste questionário podem apresentar resultados surpreendentes em algumas ocasiões. Neste trabalho, somente os resultados dos questionários de esforço das tarefas serão analisados, descartando as respostas obtidas pelos questionários preenchidos após os tutoriais.

O último questionário, chamado questionário pós-experimento ou de avaliação final, apresentou um resumo geral da experiência, contendo algumas questões de múltipla escolha e questões dissertativas. Este questionário também contou com perguntas sobre esforço cognitivo, porém desta vez o questionário perguntou sobre o esforço cognitivo geral gasto em todas as tarefas.

O uso de questionários na coleta de dados durante a experiência apresentou dados empíricos quantitativos de vários fatores. Contudo, estes dados devem ser complementados na análise pelas outras formas de coleta de dados utilizadas durante o experimento.

A marcação do tempo que os pares gastaram em cada tarefa precisou ser revisada de acordo com a gravação em vídeo, pois havia uma pequena discrepância entre os registros internos da ferramenta, a marcação dos monitores e o tempo da gravação. Após analisar cuidadosamente escolheu-se utilizar o tempo da gravação, pois estes dados representam corretamente o tempo decorrido.

A gravação do experimento forneceu dados relacionados ao comportamento dos usuários por meio de suas expressões faciais, vocais e gestuais. Apesar de conter muitas informações, os dados coletados pela gravação requerem uma análise de um especialista em comportamento humano.

Os registros da ferramenta apresentaram a maior quantidade de dados quantitativos de todos os métodos coletados. Além de conter toda a descrição da conversa por meio do *chat*, as manipulação de elementos durante as sessões colaborativas foram armazenadas nos registros.

Por fim, os diagramas produzidos durante as tarefas foram encaminhados a dois especialistas em modelagem de dados com o objetivo de analisar quantitativamente os diagramas dos usuários, por meio de notas para critérios específicos. Nenhuma informação pessoal que pudesse identificar os participantes foi apresentada aos avaliadores.

Devido à limitação de espaço e recursos, apenas análises superficiais foram conduzidas a partir dos dados coletados. Estas análises, mesmo superficiais, apresentam bons resultados e ajudaram a comprovar a hipótese do experimento, como será apresentado na próxima subseção.

6.2 Análise e Interpretação dos Dados

A análise dos dados foi iniciada a partir dos dados coletados pelo questionário de perfil. Os alunos que participaram do experimento eram estudantes do curso de pós-graduação com idades entre 23 e 34 anos, cuja média de idade está por volta de 26 anos com desvio padrão igual a 2,27, incluindo 6 homens e 6 mulheres.

Por se tratar de uma amostra relativamente jovem, pode-se especular sobre a formação acadêmica da amostra, mais especificamente sobre o conhecimento necessário à modelagem de diagramas UML. Como a UML tornou-se popular em meados de 1998, é razoável concluir que a maioria destes estudantes teve alguma instrução sobre esta modelagem durante o período que cursaram a faculdade, evidenciando o pré-requisito de conhecimento da UML, considerado um fator exclusivo na participação no experimento. Outros dados relevantes obtidos a partir do questionário de perfil foram os seguintes:

- 92,86% dos usuários disseram utilizar o computador para trabalhar diariamente;
- 92,86% dos usuários disseram utilizar o computador para acessar a Internet diariamente;
- 71,43% dos usuários disseram conhecer/participar de jogos eletrônicos com interação multiusuária;
- 100% dos usuários disseram conhecer/participar de bate papos eletrônicos;
- 85,71% dos usuários disseram conhecer a modelagem de diagramas UML;
- 71,43% dos usuários disseram conhecer um pouco sobre ferramentas de apoio ao trabalho colaborativo; e
- Ao menos 78,5% dos usuários disseram conhecer um pouco ou mais sobre programação em pares.

Com base nestes dados, ficou claro que os participantes possuem contato com as tecnologias que permitem a colaboração no seu dia a dia. O valor de 100% na questão sobre uso/participação de bate papos eletrônicos é especialmente interessante,

pois os usuários tiveram acesso a esta tecnologia de comunicação durante o experimento, explicando o por quê a comunicação não foi indicada como um problema pelo questionário pós-experimento. Também é curioso notar o conhecimento da programação em pares na maioria dos usuários, uma vez que esta técnica é implementada na XP (*Extreme Programming*), uma metodologia relativamenta nova, se comparada com as outras metodologias de desenvolvimento de software.

Quatro gráficos foram elaborados para analisar os dados coletados pelos questionários de avaliação de esforço. Estes gráficos de linha relacionam a percepção de esforço médio de todos os usuários com a tarefa realizada, da mesma maneira que Gutwin e Greenberg [9]. As seguintes perguntas constaram na avaliação de esforço:

1) O quão difícil foi completar a tarefa 1?

2) O quanto de esforço a tarefa 1 necessitou?

3) O quão difícil foi se concentrar para executar a tarefa 1?

4) O quão difícil foi a discussão com o seu par durante a tarefa 1?

5) O quão difícil foi utilizar os níveis de trava durante a tarefa 1?

6) Baseado nas informações fornecidas pela ferramenta, o quão difícil foi perceber as intenções de seu parceiro durante a tarefa 1?

Os gráficos das Figuras 6.1, 6.2 e 6.3 apresentam a comparação de dificuldade média percebida entre cada uma das tarefas do experimento.

Analisando os dados do gráfico da Figura 6.1, pode-se notar uma grande variação entre a questão Q1 e Q2, provavelmente relacionada com o fato que os usuários não indicaram muita dificuldade na tarefa 1, pois eles se sentiram confortáveis em modelar uma situação conhecida. Contudo, a quantidade elevada de elementos no cenário (jogador, goleiro, time etc) apresenta indícios de que a tarefa foi trabalhosa, de acordo com o as respostas dos usuários na questão Q2. O valor da questão Q5, em especial, pode ser explicado pela baixa adesão do uso dos níveis de trava, pois a maioria dos usuários não alterou o nível 1 de trava, escolhido como nível padrão do protótipo.

A tarefa 2 apresentou um resultado levemente inferior ao resultado das questões da tarefa 1. As respostas nesta tarefa, na média, apresentaram valores mais próximos indicando que os usuários já estavam se acostumando com a dificuldade, esforço, concentração, discussão e outros recursos do protótipo. O valor das respostas

na questão Q3 da tarefa 2 apresentou um fato curioso: tanto na média como em todos os valores os usuários indicaram o valor mediano (representado pelo número 3) para o esforço da tarefa 3. Isso pode ser explicado pelo fato que a maioria dos usuários se sente mais confortável e mais confiante na modelagem de diagramas casos de uso do que nos diagramas de classe.

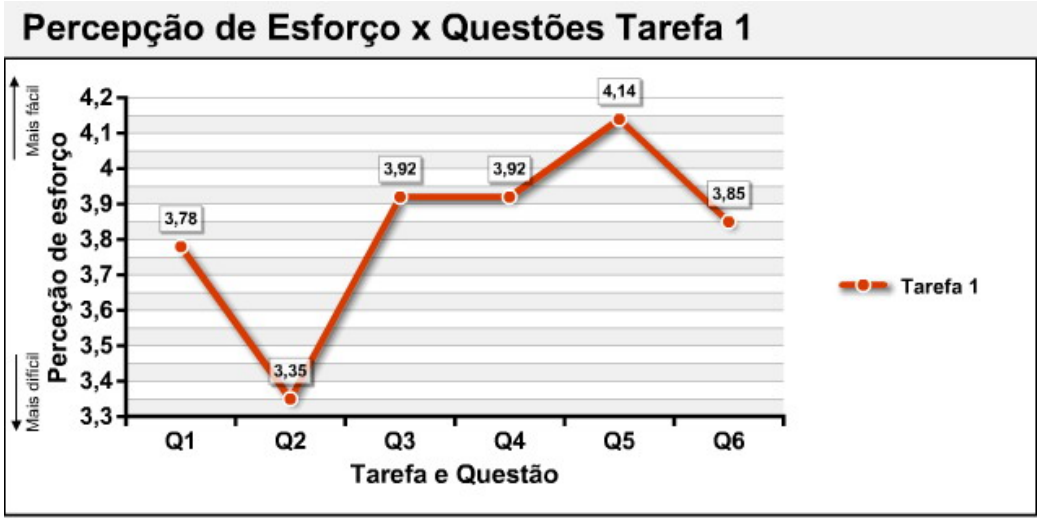

Figura 6.1 - Gráfico comparando a percepção de esforço média na tarefa 1.

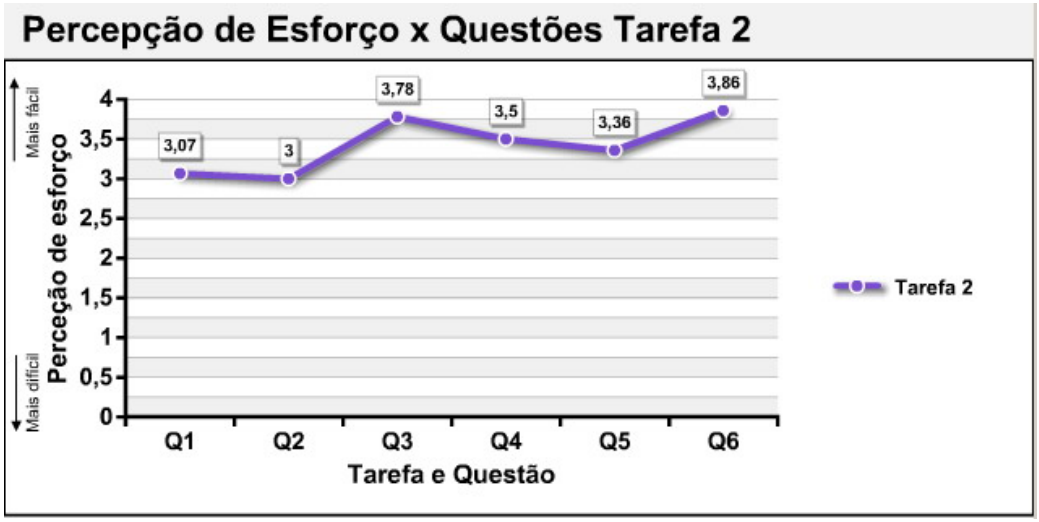

Figura 6.2 – Gráfico comparando a percepção de esforço média na tarefa 2.

A tarefa 3 apresentou o maior índice de esforço se comparado à questão Q4, que infere sobre a discussão com o parceiro. Provavelmente este valor é relacionado com como a tarefa 3 foi apresentada, permitindo que os usuários trabalhassem em paralelo, o que diminuiu a comunicação e também evitou conflitos e enganos. Outra observação sobre da tarefa 3 é que, de acordo com as respostas de todas as questões, os

usuários se sentiram mais confiantes e experientes no uso do protótipo, pois esta era a segunda vez que eles trabalharam na modelagem de um diagrama de classes colaborativamente.

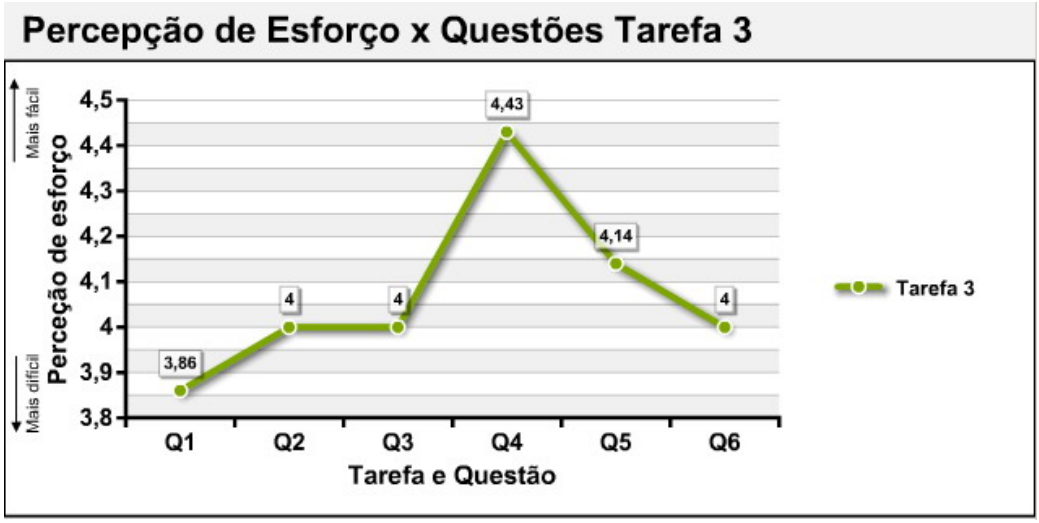

Figura 6.3 – Gráfico comparando a percepção de esforço média na tarefa 3.

O questionário pós-experimento também perguntou aos participantes do experimento questões relacionadas com o esforço. Porém, desta vez os usuários deveriam levar em consideração o esforço percebido em todas as tarefas. A Figura 6.4 apresenta o gráfico de linha comparando as respostas médias das questões de avaliação de esforço apresentadas nas Figuras 6.1, 6.2 e 6.3 comparando-se com os valores das médias das respostas das questões do questionário pós-experimento.

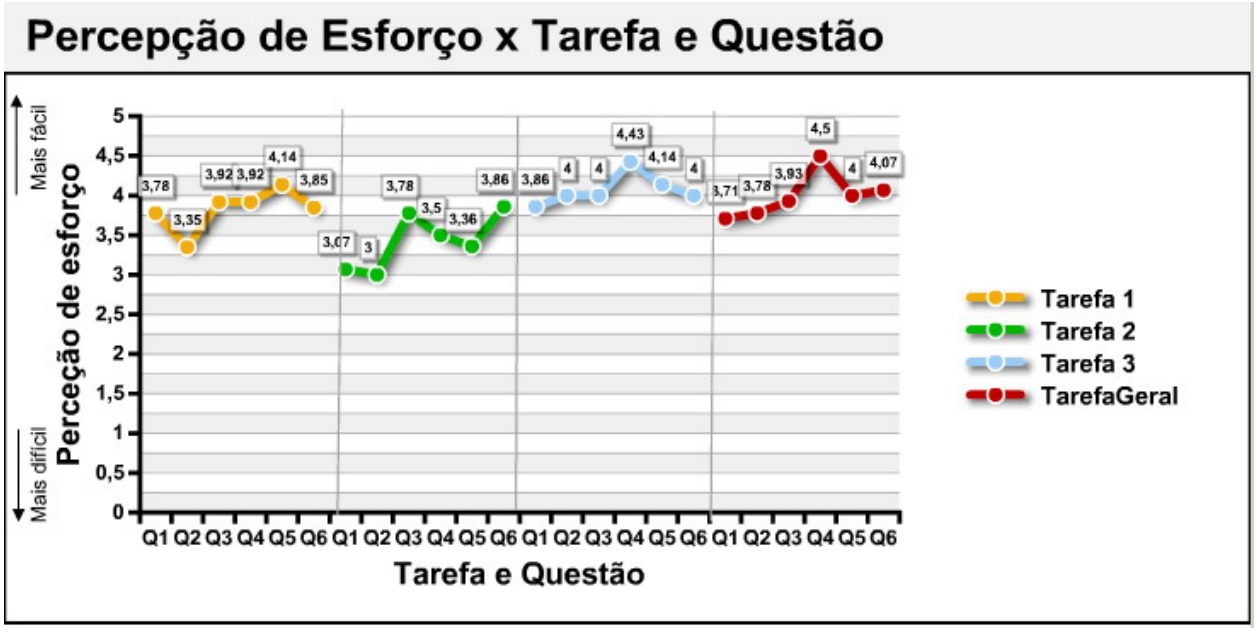

Figura 6.4 – Gráfico comparando a percepção de esforço média das tarefas.

A análise da média de esforço indicado pelos usuários segue a tendência das médias das tarefas 1, 2 e 3. Nota-se que os usuários indicaram os menores valores, indicando mais dificuldades, para as questões Q1 e Q2, que questionam sobre a dificuldade e o esforço, respectivamente. A questão Q4 foi a questão cujo valor foi maior, na média, indicando que não houve muita discussão durante a colaboração, o que pode ser comprovado pelo ambiente cordial encontrado nos registros de comunicação e na gravação de vídeo.

A partir do questionário pós-experimento, pode-se identificar que 92,86% dos usuários afirmar que, em algum momento da experiência, se sentiram como se estivessem colaborando diretamente com o parceiro, como se o parceiro estivesse fisicamente ao seu lado e não em outra sala.

Este dado fortalece a idéia de que os usuários se concentraram no que estavam fazendo e que os mecanismo de percepção remota empregados atenderam aos requisitos. É importante notar que 50% dos usuários responderam estar ciente o tempo todo do seu parceiro, evidenciando a sensação de presença. As opiniões sobre o grau de entendimento dos usuários também apresentam resultados interessantes, com 50% dos usuários indicando que o grau de entendimento foi “Muito Bom” e 46% indicando o valor “Bom”. As demais respostas para esta questão indicaram que o grau de entendimento foi “Mais ou menos”.

Quanto à satisfação dos usuários, a maioria (79% dos usuários) indicou que percebeu melhoras na usabilidade do diagrama obtido na modelagem colaborativa e prefere modelar diagramas em conjunto. Em outra questão, 43% dos usuários indicaram que a oportunidade de modelagem colaborativa de diagramas UML, oferecida pelo CoArgoUML, é Excelente, ao passo que 35% dos usuários indicaram a resposta “Muito Boa”. Os demais usuários indicaram a resposta “Boa”. Estas respostas indicam que, pelo menos sob o ponto de vista dos usuários, esta experiência agradou e é desejada.

Para confirmar a satisfação dos usuários, a questão 12 do questionário pós-experimento permitiu aos usuários apresentar comentários gerais sobre o experimento. Quase todos os comentários apresentaram opiniões a favor tanto do experimento como da colaboração durante a modelagem de diagramas UML. Eis algumas opiniões consideradas relevantes:

- “O experimento foi excelente, adorei participar. Primeira vez que faço uso de uma ferramentas colaborativa que funciona! É muito bom trabalhar em grupo pois o parceiro tem sempre uma visão diferente da nossa!”.
- “Bastante interessante o experimento. Saber como avaliar a pesquisa é fundamental e é através de experiências como estas que aprendemos. Parabéns”.
- “O Experimento se mostrou inovador e original, adequando-se a um ambiente corporativo e ganhando espaço no campo da inovação tecnológica. Destaca-se como uma motivação a mais no cenário globalizado e moderno. Parabéns”.
- “Muito bom, mas deveria se aplicado informalmente, como se fosse uma demonstração de um produto. Dificuldade devido a alguns nós que não apareceram e nem os objetos criados pelo parceiro (as vezes)”.
- “O experimento é interessante, utiliza recursos de tecnologia interativa que auxiliam a comunicação entre os parceiros e prioriza a colaboração. Os tutoriais são claros mas o questionário de tarefas não é tão adequado, pois a pessoa avaliada tem uma sensação de automatização das respostas”.

As mensagens que foram gravadas pela ferramenta durante o uso da ferramenta *chat* foram classificadas de acordo com o que os usuários desejaram expressar com a mensagem. Esta classificação é útil para identificar quais recursos foram mais utilizados pelos usuários, assim como quais recursos não foram usados. A Tabela 6.1 apresenta uma classificação inicial do tipo de mensagem seguida de alguns exemplos.

Uma análise dos dados coletados durante o experimento foi utilizada para o desenvolvimento de uma métrica com o foco na avaliação da eficácia. Esta métrica, apresentada por Pichiliani e Hirata [56], utilizou os dados do experimento descrito no Capítulo 5. Esta nova métrica tem como objetivo apresentar uma alternativa de avaliação da evolução da eficácia do aprendizado de grupos de alunos, onde uma ferramenta colaborativa foi empregada para auxiliar o aprendizado da UML.

Tabela 6.1 – Classificação das mensagens trocadas no *chat*.

Classificação da mensagem	Exemplo
Troca de informações visuais	"Usuário A:tem outras 4 classes sem nome. O q elas representam?" "Usuário B:classes sem nomes? quais, onde aponta pra eu ver"
Divisão de tarefas	"Usuário A:Eu vou fazer as ligações depois vc verifica e analisa se precisa mudar, OK?" "Usuário B:Ok... e você, por favor, analisa o que eu fiz, ok?"
Solicitação de opinião ou sugestão	"Usuário A:O que acha?" "Usuário B:Concorda com essas ligações?"
Comunicação de concepção	"Usuário A:Eu sinto que falta um ator que ficou oculto... A secretária.... pois o paciente não vai interagir direto com o sistema para marcar hotário, fazer a ficha, etc..." "Usuário B:acho legal criarmos uma classe generica (abstrata) para Arbitro e outra especifica para arbitro principal e auxiliares"

6.3 Observações

Apesar de não se analisar completamente os dados, com base na satisfação dos usuários e dos resultados apresentados pelas análises superficiais, pode-se afirmar que a hipótese pode ser confirmada devido à satisfação dos usuários e da comunicação fornecida pelos mecanismos de presença remota. Os resultados obtidos pelo experimento confirmam as expectativas iniciais dos monitores, que esperavam que a maioria dos usuários apreciasse a colaboração durante a edição de diagramas UML. Devido aos resultados positivos apresentados pelas análises dos gráficos, pode-se perceber houve uma grande satisfação daqueles que empregaram esta tecnologia.

O índice quantitativo apresentado neste trabalho pode ser utilizado na avaliação do desempenho do aprendizado dos grupos de alunos que fazem uso da modelagem de diagramas UML de forma colaborativa.

6.4 Sumário

Neste capítulo apresentam-se algumas análises dos dados coletados durante o experimento descrito no Capítulo 5. Após uma breve revisão sobre as formas de coleta dos dados durante o experimento, o capítulo apresentou os principais fatos relevantes obtidos a partir do questionário perfil. Em seguida, os dados que mediram a quantidade de esforço cognitivo foram analisados por meio de gráficos que levaram em consideração a médias das respostas dos usuários por cada tarefa.

Neste capítulo apresentou-se também a definição de uma métrica quantitativa relacionada com a evolução da eficácia, seguida da explicação das variáveis observadas e dos valores desta métrica para cada par do experimento. Por fim, o capítulo discutiu algumas observações gerais sobre as análises preliminares apresentadas com o objetivo de fornecer uma base para a afirmação da hipótese do experimento proposto.

Capítulo VII

Conclusões e Trabalhos Futuros

Este capítulo apresenta inicialmente as principais limitações da abordagem proposta neste trabalho, o Mapeamento de Componentes, e também as limitações do protótipo construído como prova de conceito. Em seguida, alguns trabalhos futuros são sugeridos decorrentes das questões em aberto deixadas pela abordagem sugerida. Por fim, o capítulo apresenta as conclusões e implicações da abordagem proposta neste trabalho.

7.1 Limitações do Mapeamento e do Protótipo

No que diz respeito à utilização do mapeamento proposto como abordagem na implementação de requisitos colaborativos, a seguinte lista de limitações é apresentada:

- A utilização do mapeamento requer acesso ao código fonte da aplicação alvo, o que pode ser considerado uma limitação à sua aplicabilidade;
- O mapeamento depende da organização interna dos componentes da aplicação alvo. Esta organização é um ponto importante que pode limitar severamente a viabilidade do uso do mapeamento;
- O conhecimento necessário da aplicação alvo pode requerer um alto investimento em termos de recursos humanos. Este tipo de conhecimento é crucial para as modificações necessárias na aplicação do mapeamento; e
- Uma limitação do mapeamento diz respeito à restrição da utilização da arquitetura híbrida após a utilização do mapeamento na aplicação alvo, uma vez que o Modelo, a Visão e o Controle não devem alterados.

A seguir é apresentada uma lista de características do protótipo construído, o CoArgoUML, consideradas como secundárias em relação ao objetivo principal da pesquisa:

- O número máximo de usuários que podem se conectar ao sistema está limitado a dois devido à complexidade técnica necessária para suportar mais do que dois usuários;
- Não existe nenhum tipo de segurança implementado pelo sistema, uso de criptografia nas mensagens trocadas entre as partes, ou mecanismos que implementem restrições das ações, de acordo com papéis predefinidos;
- O sistema de trava é considerado um mecanismo de controle de concorrência pessimista, que pode inibir a criatividade dos usuários e serializar as operações em um mesmo elemento do diagrama;
- A comunicação entre os usuários através do diálogo proporcionado pelo *chat* e pela áudio conferência foi feita apenas por emulação de *broadcast*, não permitindo a comunicação de um sub-grupo através de *multicasting*; e
- O protótipo não possuiu nenhum procedimento capaz de proporcionar uma maior robustez, durante a ocorrência de falhas de comunicação inerentes aos sistemas distribuídos ou ao próprio hardware utilizado.

No que diz respeito ao experimento, pode-se listar como limitações: 1) A quantidade de usuários que participaram do experimento; 2) A número de diagramas modelados; 3) Uma análise mais profunda dos dados coletados durante o experimento; e 4) A falta de análises referentes à utilização do mapeamento em outras aplicações não colaborativas de diferentes domínios de conhecimento.

7.2 Trabalhos Futuros

Além das limitações apresentadas, algumas sugestões são feitas a seguir para facilitar e ampliar o uso da abordagem proposta:

- Uma análise mais profunda, isto é, uma análise de todas as formas de interação dos componentes, dos componentes encontrados nas aplicações não colaborativas;

- Um estudo de viabilidade sobre a automação parcial ou completa do mapeamento, com o objetivo de diminuir a intervenção manual no código fonte;
- A investigação de uma forma para utilizar o mapeamento proposto em aplicações compostas por nenhum ou poucos componentes;
- A utilização de conceitos de Programação Orientada à Aspectos com o objetivo de tentar facilitar, tanto a descoberta dos locais que devem ser modificados na implementação dos requisitos colaborativos, como a implementação em si;
- A integração do Servidor de Colaboração com outras ferramentas CASE que exportem seus modelos, de acordo com o padrão XMI, alcançando algum nível de interoperabilidade entre aplicações CASE de modelagem; e
- A implementação de novos requisitos colaborativos ao protótipo apresentado, assim como a replicação de informações mais granulares como métodos, propriedades e atributos dos elementos dos diagramas.

7.3 Conclusões

O objetivo principal desta pesquisa foi apresentar um mapeamento que permite a extensão de aplicações não colaborativas para apoiar a colaboração síncrona, utilizando a Internet como rede de interconexão. Para aplicar este mapeamento, foi fundamental a aplicação não colaborativa seguir o estilo arquitetural MVC, um estilo arquitetural utilizado para construir aplicações que separa os dados da aplicação (o modelo); do código de tratamento de dados (o controlador), da interface de visualização (a visão). O mapeamento foi feito entre os componentes originais da aplicação e os componentes colaborativos com base nos requisitos especificados.

Como prova de conceito do mapeamento, uma aplicação alvo não colaborativa baseada no estilo arquitetural MVC foi modificada para proporcionar a colaboração síncrona. Esta aplicação é uma ferramenta CASE de modelagem chamada ArgoUML, que foi modificada por meio da aplicação do mapeamento proposto. A nova versão do

ArgoUML, com as novas funcionalidade colaborativas, foi batizada de CoArgoUML (*Collaborative ArgoUML*).

O protótipo foi utilizado em um experimento com o objetivo de verificar a satisfação dos usuários durante a edição colaborativa de diagramas UML. O experimento consiste em verificar se o suporte colaborativo aplicado à modelagem de diagramas UML torna mais satisfatória, para os usuários, a tarefa de modelagem de diagramas.

A análise dos dados coletados durante as sessões colaborativas do experimento forneceu uma base empírica para validar a hipótese do experimento, além de coletar dados suficientes na elaboração de uma nova métrica quantitativa relacionada à eficácia de aprendizado.

Dessa forma, através dos resultados obtidos pelas opiniões dos usuários e pela análise dos dados coletados durante o experimento, pôde-se concluir que o protótipo de ferramenta CASE colaborativa satisfez o conjunto mínimo de requisitos colaborativos apresentados na Seção 3.1.

Apesar dos trabalhos desenvolvidos até hoje na área de CSCW para propiciar a criação de aplicações colaborativas, como *toolkits*, Adaptação Transparente e Substituição de Componentes, a complexidade de desenvolvimento envolvida neste processo ainda é grande. Não apenas pelos motivos apresentados, mas também pelo fato de que as abordagens existentes ainda requerem uma grande quantidade de esforço de desenvolvimento e não são suficientemente adequadas para utilização nas aplicações existentes.

A utilização de uma aplicação não colaborativa, por vários usuários e de maneira simultânea, cria novos desafios a serem resolvidos como: o controle de sincronia; a consistência das informações e de mecanismos que facilitem a percepção; e a consciência entre as atividades dos participantes.

Baseado neste contexto, o mapeamento proposto neste trabalho fornece um guia que tem como objetivo ajudar os desenvolvedores a implementar requisitos colaborativos em seus projetos. Como a pesquisa para o desenvolvimento de aplicações colaborativas ainda é recente, supõem-se o mapeamento apresentado neste trabalho apresente à comunidade de desenvolvedores uma contribuição significativa. Deste

modo, as implicações decorrentes do uso do mapeamento podem gerar novas oportunidades para criar projetos de sistemas *groupware*.

O mapeamento proposto neste trabalho abre um precedente no que diz respeito à modificação de uma aplicação baseada no estilo arquitetural MVC para apoiar aspectos de colaboração síncrona.

As principais conclusões obtidas neste trabalho são listadas a seguir:

- Com o mapeamento apresentado neste trabalho, concluí-se que uma aplicação de código livre, que é baseada no estilo arquitetural MVC, pode ser modificada para suportar requisitos colaborativos;
- O mapeamento proposto implicou na modificação de aproximadamente 5% do código fonte da aplicação alvo escolhida, tornando-a uma aplicação com suporte para múltiplos usuários;
- Implementar requisitos colaborativos em uma aplicação complexa e com um desenvolvimento distribuído entre várias pessoas de países diferentes é viável; e
- A utilização de uma ferramenta CASE de modelagem colaborativa agrada aos usuários e torna a tarefa de modelagem mais fácil.

Uma implicação do mapeamento proposto envolve o encorajamento à comunidade de desenvolvedores, para que eles modifiquem suas aplicações e aumentem a quantidade de sistemas *groupware* disponíveis para os usuários. Além deste encorajamento, estimular o desenvolvimento novas aplicações que contem com requisitos colaborativos representa um iniciativa rumo ao uso futuro de aplicações colaborativas em diferentes domínios do conhecimento.

Referências Bibliográficas

[1] THE UIMS TOOL DEVLOPERS' WORKSHOP. **A metamodel for runtime architecture of an interactive system: the UIMS tool developers workshop.** *ACM SIGCHI Bulletin*, vol.41, n.1, p.32-37, 1992.

[2] SOUZA, Adriana Silveira de. **Um Estudo Sobre Trabalho Cooperativo Suportado Por Computador (CSCW):** Trabalho individual. Universidade Federal do Rio Grande do Sul, Porto Alegre, 1996.

[3] SCHMIDT, Albercht; SPECKER, Alexander; PARTSH, Gerhard; WEBER, Michael; HÖCK, Siegfield. **An Agent-Based Telecooperation Framework.** Disponível em: <http://cia.informatik.uni-ulm.de/papers/cobuild98.pdf >. Acesso em: 8 fev. 2007.

[4] CHABERT, Annie; GROSSMAN, Ed; JACKSON, Larry S.; PIETROWIZ, Stephen R.; SEGUIN, Chris. **Java Object-Sharing in Habanero**. *Communications of ACM*, vol.41, nro.6, p.69-76, 1998.

[5] SILVA FILHO, Antonio Mendes da. **Arquitetura de Software**. 1.ed. São Paulo: Editora Campus, 2002.

[6] TIGRIS.ORG. **ArgoUML**. Disponível em: <http://argouml.tigris.org>. Acesso em: 8 fev. 2007.

[7] PRAKASH, Atul; KNISTER, Michael J. **A framework for undoing actions in collaborative systems**. *ACM Transactions on Computer-Human Interaction*, vol.4., p.295-330, 1994.

[8] FERREIRA, Aurélio B. H. **Novo dicionário Aurélio da língua portuguesa**. 1. ed. Rio de Janeiro: Nova Fronteira, 1986.

[9] GUTWIN, Carl; GREENBERG, Saul. **A Descriptive Framework of Workspace Awareness for Real-Time Groupware**. *Computer-Supported Cooperative Work*, vol.11, n.3, p.411- 446, 2002.

[10] ELLIS, Clarence; GIBBS, S. J; REIN, G. L. **Groupware: some issues and experiences**. *Communications of the ACM*, v.34, n.1, p.38-58, jan., 1991.

[11] ELLIS, Clarence; GIBBS, S. J. **Concurrency control in groupware system**. In: Proceedings of the 1989 ACM SIGMOD International Conference on Management of data (SIGMOD'89), San Diego, Califórnia, E.U.A., p.399-407, 1989.

[12] SZYPERSKI, Clemens. **Component Software: Beyond Object-Oriented Programming**. 2. ed. Nova York: Addison-Wesley/ACM Press, 2002.

[13] SCHUCKMANN, Christian; KIRCHNER, Lutz; SCHÜMMER Jan; HAAKE, Jorg. M. **Designing object-oriented synchronous groupware with COAST**. In: Proceedings of the 3rd ACM Conf. on Computer Supported Cooperative Work (CSCW'96), Nova York, E.U.A., p.30-38, 1996.

[14] ORTEGA, Christian Marcus Rauh. **Modelo e Framework de Implementação de Aplicações Colaborativas**. 120f. 2000. Tese (Mestre em Informática) - Instituto Tecnológico de Aeronáutica. São José dos Campos.

[15] GARNKEL, Daniel; WELTI, Bruce C; YIP, Thomas W. **SharedX: A tool for real-time collaboration**. HP Journal, vol.45, n.2, p.23-36, abril, 1994.

[16] TIETZE, Daniel A. **A Framework for Developing Component-based Cooperative Applications**. 2001. Dissertação de PhD em Ciência da Computação - Univerdade Technischen Darmstadt, Alemanha.

[17] SUTHERS, Daniel D. **Architectures for Computer Supported Collaborative Learning**. In: Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT 2001), New Orleans, Lousiana, E.U.A., p.6-8, 2001.

[18] SUTHERS, Daniel D; JONES, Dan. **An Architecture for Intelligent Collaborative Educational Systems**. In: Proceedings of the 8th World Conference on Artificial Intelligence in Education (AIED'97), Kobe, Japão, p.55-62, 1997.

[19] FIRESMITH, Donald G. **Frameworks: the golden path to object Nirvana**. Journal of Object-Oriented Programming, v.6, n.6, p.6-8, out. 1993.

[20] LI, Du; LI, Rui. **Transparent sharing: interoperation of heterogeneous single-user applications**. In: Proceedings of the 8th ACM Conf. on Computer Supported Cooperative Work (CSCW'02), New Orleans, Lousiana, E.U.A., p.246-255, 2002.

[21] ORFALY, Edwards Hankly. **The Essential Distributed Objects Survival Guide.** 1 ed. Nova York: John Wiley& Sons, 1995. Cap.12, p.221-238.

[22] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1.ed. Nova York: Addison Wesley, 1994.

[23] SANTORO, Flavia. M; BORGES, Marcos Roberto da Silva; SANTOS, Neide. **Ambientes de aprendizagem do futuro: teoria e tecnologia para cooperação**. Mini-curso do XIII Simpósio Brasileiro de Informática na Educação, São Leopoldo, Rio Grande do Sul, 2002.

[24] BUSCHMANN, Frank; MEUNIER, Regine; ROHNERT, Hans; SOMMERLAD, Peter; STAL, Michael. **Patterns Oriented Software Architecture: A System of Patterns**. 2. ed. Chichester, Reuno Unido: John Wiley & Sons Ltd, 1996.

[25] TEWISSEN, Frank; BALOIAN, Nelson A.; HOPPE, Heinz Ulrich; REIMBERG, Erich. **"MatchMaker": Synchronising Objects in Replicated Software-Architectures**. In: Proceedings of the 6th Internationa Workshop on Groupware (CRIWG 200), Madeira, Portugal, p.18-20, 2000.

[26] CALVARY, Gaëlle; COUTAZ Joëlle; NIGAY, Laurence. **From single-user architectural design to PAC*: A generic software architecture model for CSCW**. In: Proceedings of the SIGCHI Conference on Human factors in computing systems, p.242-249, 1997.

[27] TIGRIS.ORG. **GEF**. Disponível em: <http://gef.tigris.org>. Acesso em: 8 fev. 2007.

[28] BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **The Unified Modeling Language User Guide**. 1. ed. São Paulo: Addison-Wesley, 1998.

[29] FUKS, Hugo; RAPOSO, Alberto Barbosa; GEROSA, Marco Aurélio. **Engenharia de Groupware: Desenvolvimento de Aplicações Colaborativas.** XXI Jornada de Atualização em Informática, Anais do XXII Congresso da Sociedade Brasileira de Computação, v.2, Cap.3, p.89-128, 2002.

[30] FUKS, Hugo; RAPOSO, Alberto Barbosa; GEROSA, Marco Aurélio. **Do Modelo de Colaboração 3C à Engenharia de Groupware**. IX Simpósio Brasileiro

de Sistemas Multimídia e Web - WebMidia 2003 (Trilha especial: Trabalho Cooperativo Assistido por Computado), Salvador, BA, p.225-252, 2003.

[31] ABDEL-WAHAB, Hussein; FEIT, Mark. A. **XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration**. In: Proceedings of IEEE Tricomm, Chapel Hill, Carolina do Norte, E.U.A., p.159-167, 1991.

[32] GIMENES, Itana Maria de Souza, HUZITA, Elisa Hatsue Moriya. **Desenvolvimento Baseado em Componentes: Conceitos e Técnicas**. 1. ed. Rio de Janeiro: Ciência Moderna, 2005.

[33] BEGOLE, James C. A; ROSSON, Mary Beth; SHAFFER, Clifford A. **Flexible collaboration transparency: supporting worker independence in replicated application sharing systems**. *ACM Transactions on Computer-Human Interaction*, p.95-132, 1999.

[34] HILL, Jason; GUTWIN, Carl. **The MAUI Toolkit: Groupware Widgets for Group Awareness**. *Computer Supported Cooperative Work*, vol.13, p.539-571, 2004.

[35] NUNAMAKER, Jay F; DENNIS, Alan R; VALACICH, Joseph S; VOGEL, Douglas R; GEORGE, Joey F. **Electronic Meeting Systems to Support Group Work: Theory;Practice at Arizona**, *Communication of the ACM*, vol.34, n.7, p.40-61, 1991.

[36] LU, Jiajun; LI, Rui; LI, Du. **A State Difference Based Approach to Sharing SemiHeterogeneous SingleUser Editors**. Sixth International Workshop on Collaborative Editing Systems (IWCES6) in CSCW'04, Chicago, Illinois, E.U.A., 2004.

[37] STAFFORD, Judith A; WOLF, Alexander L. **Architecture-Level Dependence Analysis for Software Systems**. International Journal of Software Engineering and Knowledge Engineering, vol.11, no.4, p.431-451, 2001.

[38] COUTAZ , Joëlle. **PAC, an object oriented model for dialog design**. In: Proceedings of the Second IFIP Conference on Human-Computer Interaction, Stuttgart, Alemanha, p.431-436, 1987.

[39] GRUDIN, Jonathan, POLROCK, Steven. **CSCW, groupware and workflow: experiences, state of art, future trends**. In: Proceedings of Conference on Human Factors in Computing Systems (CHI '99), Pittsburgh, Pennsylvania, E.U.A., p.120-121, 1999.

[40] GRUDIN, Jonhatan. **Computer Supported Collaborative Work: History and Focus**, IEEE Computer, vol.27, n.5, p.19-26, 1994.

[41] BOULILA, Naoufel; DUTOIT, Allen H; BRÜEGGE, Bernd. **D-Meeting: an Object-Oriented Framework for Supporting Distributed Modelling of Software.** International Workshop on Global Software Development, International Conference on Software Engineering. Portland, Oregon, E.U.A., 2003.

[42] WILLIAMS, Neil; BLAIR, Gordan S; COULSON, Geoff; DAVIES, Nigel; RODDEN, Tom. **The Impact of Distributed Multimedia systems on CSCW**. In: Computer Support for Co-operative Work, John Wiley and Sons, p.147-169, 1994.

[43] MICROSOFT. **NetMeeting Home.** Disponível em: <http://www.microsoft.com/windows/netmeeting/ >. Acesso em: 8 fev. 2007.

[44] PINHEIRO, Manuele Kirsch. **Mecanismo de Suporte à Percepção em Ambiente Cooperativos**. 167f. 2001. Tese (Mestre em Ciência da Computação) - Universidade Federal do Rio Grande do Sul, Porto Alegre.

[45] MANGAN, Marco Aurélio Souza. **Uma abordagem para o desenvolvimento de apoio à percepção em ambientes colaborativos de software**. 2006. Tese (Doutorado em Engenharia de Sistemas e Computação) - Universidade Federal do Rio de Janeiro, Rio de Janeiro.

[46] CAMPO, Marcelo; TRICE, Roberto T. **O uso de técnicas visuais e a vegacionais para compreensão de frameworks orientados a objetos**. Anais do IX Simpósio Brasileiro de Engenharia de Software, Recife, Pernanbuco, p.175-190, 1995.

[47] GEROSA, Marco Aurélio. **Desenvolvimento de Groupware Componentizado com Base no Modelo 3C de Colaboração**. 272f. 2006. Tese (Doutorado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

[48] BORGES, Marcos Roberto da Silva; MEIRE, Alexandre Pereira; PINO, José Alberto. **An Interface for supporting versioning in a cooperative editor.** In: Proceedings of the 10th International Conference on Human-Computer Interaction, Crete, Grécia, p.849-853, 2003.

[49] LOZANO, Marta. **Consistency Maintenance Framework For Collaborative Software Modelling Tools**. 94f. 2003. Tese (Mestre em Ciência da Computação) – Universidade de Dublin, Escócia.

[50] FOWLER, Martin. **Refactoring: Improving the Design of the Existing Code**. 1. ed. São Paulo: Addison-Wesley, 2000.

[51] DIAS, Márcio de Souza; XEXÉO, Gerado. **Editor Cooperativo para Diagramação de Software OO**. Anais do XI Simpósio Brasileiro de Engenharia de Software, Fortaleza, Ceará, p.499-502, 1997.

[52] DIAS, Márcio de Souza. **COPSE: Um ambiente de suporte ao projeto cooperativo de software**. 1998. Tese (Mestre em Ciência da Computação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro.

[53] PIMENTEL, Mariano. **RUP-3C-Groupware: um processo de desenvolvimento de groupware baseado no Modelo 3C de Colaboração**. 178f. 2006. Tese (Doutorado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

[54] STEFIK, Mark J.; BOBROW, Daniel G.; FOSTER, Gregg; LANNING, Stan; TATAR, Deborah. **WYSIWIS Revised: Early Experiences with Multiuser Interfaces**. *ACM Transactions on Office Information Systems*, vol.5., p.147-167, 1987.

[55] VEIT, Matthias; HERRMANN, Stephan. **Model-View-Controller and Object Teams: A Perfect Match of Paradigms.** In: Proceedings of the 2nd international conference on Aspect-oriented software development (AOSD 2003), Boston, Massachusetts, E.U.A., p.140-149, 2003.

[56] PICHILIANI, Mauro C., HIRATA, Celso M. **Usando a modelagem colaborativa no aprendizado da UML**. Anais do XXVI Congresso da Sociedade Brasileira de Computação, XII Workshop de Informática na Escola (WIE), Campo Grande, Mato Grosso, p.309-317, 2006.

[57] BEAUDOUIN-LAFON, Michel. **Computer Supported Co-Operative Work**. 1. ed. Nova York: John Wiley & Sons, 1999.

[58] GRAHAM, Nicholas; URNES, Tore; NEJABI, Roy. **Efficient distributed implementation of semi-replicated synchronous groupware**. In: Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'96), Washington, E.U.A., p.1-10, 1996.

[59] MOLLI, Pascal; SKAF-MOLLI, Hala; OSTER, Gérald. **SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments**. In: Proceedings of the 7th Intenational Conference on Computer Supported Cooperative Work in Design (CSCWD 2002), Rio de Janeiro, p.80-84, 2002.

[60] DOURISH, Paul; EDWARDS, W. Keith. **A Tale of Two Toolkits: Relating Infrastructure and Use in Flexible CSCW Toolkits**. *Computer Supported Cooperative Work*, vol.9, n.1, p.33-51, 2000.

[61] DEWAN, Prasun, CHOUDHARY, Rajiv. **Flexible user interface coupling in collaborative systems**. In: Proceedings of the ACM Conference on Human Factors in Computing Systems (ACM CHI'91), Louisiana, E.U.A. p.41-48, 1991.

[62] DEWAN, Prasun. **Multiuser architectures**. In: Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, p.247-270, 1995.

[63] PRIKLADNICKI, Rafael. **MuNDDoS Um modelo de Referência para o Desenvolvimento Distribuído de Software**. 2003. Tese (Doutorado em Informática) - Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre.

[64] HILL, Ralph D. **The Abstraction-Link-View Paradigm: Using Constraints to Connect User Interfaces to Applications**. In: Proceedings of the ACM Conference on Human Factors in Computing Systems (ACM CHI'91), California, E.U.A., p.335-342, 1992.

[65] JOHNSON, Ralph E.; FOOTE, Brian. **Designing reusable classes.** Journal of Object-Oriented Programming, vol.1, n.2, p.22-35, 1988.

[66] WIRFS-BROCKS, Rebecca; JOHNSON, Ralph E. **Surveying Current Research in Object-Oriented Design**. *Communications of the ACM*, vol.33, n.9, p.124, 1990.

[67] ARAÚJO, Renata M.; SANTORO, Flávia M.; BORGES, Marcos R. S. **The CSCW Lab for Groupware Evaluation.** In: Proceedings of the 8th International Workshop on Groupware: Design, Implementation and Use (CRIWG 2002), La Serena, Chile, 2002.

[68] BENTLEY, Richard; HORSTMANN, Thilo; SIKKEL, Klass; TREVOR, Jonathan. **Supporting Cooperative Information Sharing with the World Wide Web: The BSCW Shared Worksapce System**. Disponível em:

< http://www.w3.org/Conferences/WWW4/Papers/151/ >. Acesso em: 8 fev. 2007.

[69] TAYLOR, Richard N.; MEDVIDOVIC, Nenad; ANDERSON, Kenneth. M; WHITEHEAD, E. James Jr; ROBBINS, Jason E. **A component-and message-based architectural style for GUI Software**. In: Proceedings of the 17th International Conference on Software Engineering, Seattle, Washington, E.U.A., p.295-304, 1995.

[70] PRESSMAN, Roger S. **ENGENHARIA DE SOFTWARE**. 1.ed, São Paulo: Makron Books, 1995.

[71] GREENBERG, Saul; ROSEMAN, Mark. **Groupware toolkits for synchronous work**. In: BEAUDOUIN-LAFON, M. (ed.). Computer-Supported Cooperative Work: trends in software series. Nova York: John Wiley & Sons, 1998.

[72] LUKOSCH, Stephan; SCHÜMMER, Till. **Patterns for Managing Shared Objects in Groupware Systems**. In: Proceedings of the 9th European Conference on Pattern Languages and Programs, Irsee, Alemanha, p.333-378, 2004.

[73] XIA, Steven; SUN, David; SUN, Chengzheng; CHEN, David; SHEN, Haifeng. **Leveraging Singleuser Applications for Multiuser Collaboration: the CoWord Approach**. In: Proceedings of the 9th ACM Conference on Computer Supported Cooperative Work (CSCW'04), Chicago, Illinois, E.U.A., p.162-171, 2004.

[74] LAURILLAU, Yann; NIGAY, Laurence. **Clover Architecture for Groupware**. In: Proceedings of the 8th ACM Conference on Computer Supported Cooperative Work (CSCW'02), New Orleans, Lousiana, E.U.A., p.236-245, 2002.

Apêndice A

Questionários da Experiência

Este Apêndice apresenta todos os questionários utilizados no experimento para a coleta de dados. O CD que acompanha este trabalho contém todos os questionários utilizados no experimento, assim como suas respostas.

A1. Questionário de Perfil

Os usuários participantes da experiência preencheram o questionário de perfil antes de começar os tutoriais. Este questionário tem o objetivo de identificar características do participante, como idade, conhecimento anterior e freqüência de uso da Internet.

QUESTIONÁRIO DE PERFIL

Questionário de Perfil do Usuário

Nome:.....

E-mail:.....

Idade: ..... Sexo: Masculino [ ] Feminino [ ] Data do experimento: ...../...../.....

1) Ocupação (se estudante, indique graduação ou pós-graduação e área de estudo)

.....

2) Preenche com um X a opção que indica a freqüência com que você usa computadores.

Para trabalhar?	Para se divertir?	Para acessar a Internet?
[ ] Nunca	[ ] Nunca	[ ] Nunca
[ ] Uma vez por mês	[ ] Uma vez por mês	[ ] Uma vez por mês
[ ] Uma vez por semana	[ ] Uma vez por semana	[ ] Uma vez por semana
[ ] Várias vezes por semana	[ ] Várias vezes por semana	[ ] Várias vezes por semana
[ ] Diariamente	[ ] Diariamente	[ ] Diariamente

3) A seguir você encontrará uma série de afirmações a respeito do seu conhecimento prévio de algumas tecnologias. Indique quais afirmações se aplicam a você e quais não se aplicam. Caso se aplique a você, faça um círculo em volta da palavra SIM. Caso não se aplique a você, faça um círculo em volta da palavra NÃO.

Você já teve oportunidade de utilizar algum sistema de realidade virtual? SIM NÃO

Você já teve oportunidade de participar de uma partida de algum jogo eletrônico multi-player? SIM NÃO

Você já teve oportunidade de participar de bate-papos (chats) eletrônicos? SIM NÃO

Você já teve oportunidade de participar de bate-papos (chats) eletrônicos? SIM NÃO

Você já teve oportunidade de utilizar alguma ferramenta de ensino a distância (e-learning)? SIM NÃO

Você já teve oportunidade de utilizar alguma ferramenta de modelagem de diagramas da UML? SIM NÃO

Você já teve experiência com a ferramenta chamada ArgoUML? SIM NÃO

Você já teve oportunidade de utilizar algum sistema de VOIP (Voz sobre IP)? SIM NÃO

4) Preencha o seu grau de familiaridade sobre os itens abaixo colocando um X nos valores da escala:

Item	Desconheço	Conheço um pouco	Conheço o suficiente	Conheço muito	Sou especialista
Modelagem Orientada à Objetos	☐	☐	☐	☐	☐
Programação Orientada à Objetos	☐	☐	☐	☐	☐
Modelagem de diagramas da UML	☐	☐	☐	☐	☐
Processo de desenvolvimento de software	☐	☐	☐	☐	☐
Ferramentas de apoio ao trabalho colaborativo	☐	☐	☐	☐	☐
Programação em pares (pair-programming)	☐	☐	☐	☐	☐
Locks (técnica de controle de concorrência)	☐	☐	☐	☐	☐
Software de controle de versão (SourceSafe,CVS, etc)	☐	☐	☐	☐	☐
Gerência de Projetos de Software	☐	☐	☐	☐	☐

A2. Questionário de Avaliação de Esforço

Após o término de cada tutorial e de cada tarefa os usuários responderam a um questionário com perguntas sobre o esforço exigido para completar a tarefa. O questionário de avaliação de esforço é reproduzido abaixo.

QUESTIONÁRIO DE AVALIAÇÃO DE ESFORÇO

Nome:.....

Este questionário tem como objetivo avalizar o esforço entre os tutoriais e as tarefas. A seguir você encontrará uma série de perguntas sobre o esforço mental gasto nas tarefas. Faça um círculo em volta do número que melhor representa a sua avaliação do esforço necessário para completar as tarefas.

TUTORIAL 1

a) O quão difícil foi completar o tutorial 1?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

b) O quanto de esforço o tutorial 1 necessitou?

Muito esforço . . . 1 2 3 4 5 . . . Pouco esforço

c) O quão difícil foi se concentrar para executar o tutorial 1?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

TUTORIAL 2

a) O quão difícil foi completar o tutorial 2?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

b) O quanto de esforço o tutorial 2 necessitou?

Muito esforço . . . 1 2 3 4 5 . . . Pouco esforço

c) O quão difícil foi se concentrar para executar o tutorial 2?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

TUTORIAL 3

a) O quão difícil foi completar o tutorial 3?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

b) O quanto de esforço o tutorial 3 necessitou?

Muito esforço . . . 1 2 3 4 5 . . . Pouco esforço

c) O quão difícil foi se concentrar para executar o tutorial 3?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

TAREFA 1

a) O quão difícil foi completar a tarefa 1?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

b) O quanto de esforço a tarefa 1 necessitou?

Muito esforço . . . 1 2 3 4 5 . . . Pouco esforço

c) O quão difícil foi se concentrar para executar a tarefa 1?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

d) O quão difícil foi a discussão com o seu par durante a tarefa 1?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

e) O quão difícil foi utilizar os níveis de lock durante a tarefa 1?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

f) Baseado nas informações fornecidas pela ferramenta, o quão difícil foi perceber as intenções de seu parceiro durante a tarefa 1?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

TAREFA 2

a) Qual foi o seu papel na Tarefa? A ( ) B ( )

b) O quão difícil foi completar a tarefa 2?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

c) O quanto de esforço a tarefa 2 necessitou?

Muito esforço . . . 1 2 3 4 5 . . . Pouco esforço

d) O quão difícil foi se concentrar para executar a tarefa 2?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

e) O quão difícil foi a discussão com o seu par durante a tarefa 2?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

f) O quão difícil foi utilizar os níveis de lock durante a tarefa 2?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

g) Baseado nas informações fornecidas pela ferramenta, o quão difícil foi perceber as intenções de seu parceiro durante a tarefa 2?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

TAREFA 3

a) Qual foi o seu papel na Tarefa? A ( ) B ( )

b) O quão difícil foi completar a tarefa 3?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

c) O quanto de esforço a tarefa 3 necessitou?

Muito esforço . . . 1 2 3 4 5 . . . Pouco esforço

d) O quão difícil foi se concentrar para executar a tarefa 3?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

e) O quão difícil foi a discussão com o seu par durante a tarefa 3?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

f) O quão difícil foi utilizar os níveis de lock durante a tarefa 3?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

g) Baseado nas informações fornecidas pela ferramenta, o quão difícil foi perceber as intenções de seu parceiro durante a tarefa 3?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

A3. Questionário de Avaliação Final

Após o término todas as tarefas os usuários responderam a um questionário final quer continha perguntas sobre como foi a colaboração, esforço geral necessário nas tarefas e comentários gerais com sobre a experiência. O questionário de avaliação final é reproduzido abaixo.

QUESTIONÁRIO DE AVALIAÇÃO FINAL

Nome:.....

1) A questão abaixo serve para ajudar a avaliar a sensação de presença durante a sessão colaborativa. Responda colocando um S para Sim ou N para Não na lacuna.

a) Em algum momento, durante o experimento, você se sentiu como se você estive colaborando diretamente com seu parceiro, como se ele fisicamente do seu lado e não em outra sala?
Resposta: [ ]

2) Durante os momentos da experiência em que não houve interação entre você e seu parceiro como você classifica a sensação de presença remota?

a) Estava ciente do meu parceiro o tempo todo.
b) Concentrei-me em outra ação, mas tinha consciência da presença do meu parceiro.
c) Concentrei-me em outra ação e por breves momentos esqueci do meu parceiro.
d) Esqueci completamente do meu parceiro.

3) Qual o grau de entendimento entre você e seu parceiro durante a execução das tarefas?

a) Muito bom.
b) Bom.
c) Mais ou menos.
d) Fraco.
e) Muito fraco.

4) Lembre-se da última vez que você executou alguma tarefa do cotidiano em colaboração com alguém (como mover um móvel, praticar algum esporte ou tocar uma música). Como você classifica a dificuldade na coordenação com seu parceiro em uma tarefa do cotidiano e no experimento colaborativo executado?

a) A dificuldade foi a mesma.
b) A coordenação foi mais difícil no experimento.
c) A coordenação foi mais difícil na tarefa do cotidiano.
d) Não me lembro.

5) Escolha a alternativa que melhor expressa a satisfação de utilizar uma ferramenta colaborativa para modelar diagramas da UML:

a) Percebi uma melhora na modelagem e prefiro modelar diagramas em conjunto.
b) Notei ganho, mas prefiro modelar sozinho os diagramas.
c) Não notei nenhum ganho ou perda significativa durante a modelagem.
d) Não notei melhora e prefiro trabalhar sozinho na modelagem.
e) Prefiro modelar os diagramas sem o auxílio de ferramentas.

6) Em sua opinião, qual é a utilidade do uso dos níveis de locks para controlar a concorrência ao acesso dos elementos de modelagem durante a edição colaborativa? Escolha somente uma alternativa.

a) Aplica-se totalmente.

b) Aplica-se.
c) Não faz diferença.
d) Não se aplica.
e) Definitivamente não se aplica.

7) Baseado no uso da ferramenta durante a experiência classifique a utilidade dos elementos apresentados na Figura 1, de acordo com a escala de notas abaixo:

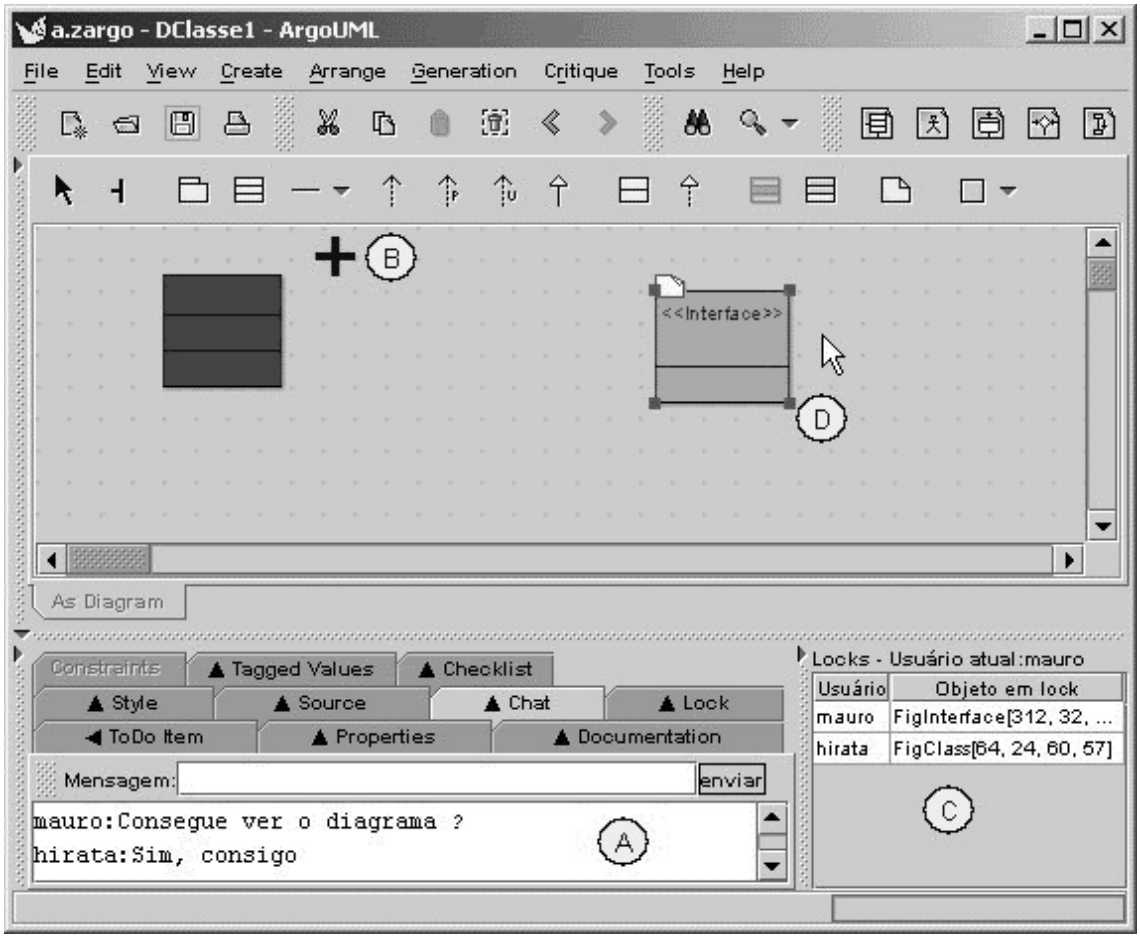

Figura 1. Elementos da ferramenta utilizados na colaboração.

Escala de notas: (1) Excelente, (2) Bom, (3) Razoável, (4) Ruim, (5) Péssimo, (6) Não utilizado.

Elemento	Nota
A) Janela de bate-papo	☐
B) Telepointer	☐
C) Janela de Locks	☐
D) Controle de Locks	☐
E) VOIP (Voz sobre IP)	☐

8) A seguir você encontrará uma série de perguntas sobre o esforço mental gasto em todas as tarefas. Faça um círculo em volta do número que melhor representa a sua avaliação do esforço necessário para completar as tarefas.

a) O quão difícil foi completar as tarefas?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

b) O quanto de esforço as tarefas necessitaram?

Muito esforço . . . 1 2 3 4 5 . . . Pouco esforço

c) O quão difícil foi se concentrar para executar as tarefas?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

d) O quão difícil foi a discussão com o seu par durante as tarefas?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

e) O quão difícil foi utilizar os níveis de lock durante as tarefas?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

f) Baseado nas informações fornecidas pela ferramenta, o quão difícil foi perceber as intenções de seu parceiro durante as tarefas?

Muito difícil . . . 1 2 3 4 5 . . . Muito fácil

9) Baseado na experiência, coloque um X na coluna Adequado ou Inadequado para os itens abaixo:

Item	Adequado	Inadequado
Hardware utilizado	☐	☐
Software utilizado	☐	☐
Ambiente do experimento	☐	☐
Monitor(es)	☐	☐
Primeiro Tutorial	☐	☐
Segundo Tutorial	☐	☐
Primeira Tarefa	☐	☐
Segunda Tarefa	☐	☐
Terceira Tarefa	☐	☐
Parceiro	☐	☐
Questionários	☐	☐

10) Em geral, a idéia de modelar colaborativamente diagramas da UML é:

a) Excelente.
b) Muito Boa.
c) Boa.

d) Razoável.
e) Ruim.
f) Muito ruim.
g) Péssima.

Para concluir, gostaríamos de fazer algumas perguntas finais para melhor caracterizar os respondentes desta pesquisa.

11) Escreva algum comentário final sobre o seu parceiro durante a sessão colaborativa.

.....

.....

.....

.....

.....

.....

.....

12) Escreva algum comentário geral sobre o experimento como um todo.

.....

.....

.....

.....

.....

.....

.....

Apêndice B

Registros do protótipo

Este Apêndice apresenta um exemplo dos registros de modificação nos diagramas e de comunicação entre os usuários gerados pelo protótipo CoArgoUML durante a realização do experimento. O CD que acompanha este trabalho contém todos os registros utilizados pelo protótipo para armazenar as interações dos usuários com a aplicação.

B1. Registro das Modificações nos Diagramas

O protótipo CoArgoUML foi modificado para armazenar em um arquivo texto chamado ObjetosCliente.txt, individual para cada estação, todas as inclusões e exclusões de elementos nos diagramas durante as tarefas das sessões colaborativas realizadas no experimento. As mudanças no nível de trava também foram armazenadas por este registro. A Figura B1 mostra um trecho do arquivo de registro que contém algumas entradas que indicam a criação e exclusão de elementos no diagrama de classes e também as mudanças no nível de trava.

Pode-se notar que o nome completado da classe que representa o elemento assim como a sua localização relativa no diagrama é armazenada neste registro. A data completa incluindo a hora também é armazenada junto com o nome do usuário que gerou a entrada no registro.

```
09/11/2005-14:57:26 Elemento:
org.argouml.uml.diagram.static_structure.ui.FigClass[112, 72,
60, 57] criado por A

09/11/2005-15:18:24 Elemento:
org.argouml.uml.diagram.static_structure.ui.FigClass[112, 144,
60, 57] criado por B

09/11/2005-15:19:14 Elemento:
org.argouml.uml.diagram.static_structure.ui.FigClass[344, 160,
60, 57] criado por B

09/11/2005-15:27:54 Usuario: B mudou para o Nivel 1 de lock.

09/11/2005-15:28:01 Elemento:
org.argouml.uml.diagram.ui.FigGeneralization[172, 201, 284, 87]
removido por B

09/11/2005-15:28:03 Elemento:
org.argouml.uml.diagram.ui.FigGeneralization[172, 201, 188, 95]
removido por B

09/11/2005-15:28:06 Elemento:
org.argouml.uml.diagram.ui.FigGeneralization[172, 201, 55, 87]
removido por B

09/11/2005-15:28:08 Elemento:
org.argouml.uml.diagram.ui.FigGeneralization[127, 201, 50, 87]
removido por B
```

Figura B1 - Trecho do arquivo de registro contendo a manipulação dos elementos.

B2. Registro das Comunicações

No Cenário A do experimento, onde os usuários utilizaram apenas o *chat* integrado ao protótipo para se comunicarem, todas as conversas foram armazenadas em um arquivo chamado LogChat.txt, que também é individual para cada estação. A comunicação por áudio conferência do Cenário B foi transcrita posteriormente a partir da gravação em vídeo dos usuários. A Figura B2 mostra um trecho do registro que contém o diálogo entre os usuários durante a modelagem colaborativa. Os nomes dos usuários foram omitidos para preservar as suas identidades.

09/11/2005-15:07:13 B:Olá XXX... e aí, vamos decidir quais serão as classes?

09/11/2005-15:11:00 A:Olá YYY. Vamos sim... estava aqui pensando e demorei pra ver sua mensagem...

09/11/2005-15:11:48 B:tá ok. Então... por onde você acha que devemos começar?

09/11/2005-15:12:24 A:Creio que deveremos ter as classes: Goleiro, Defesa, Meio de Campo, Laterais e Atacantes... O que vc acha?

09/11/2005-15:13:41 B:estou pensando aqui... estou com dúvida em relação à representação de uma equipe... o que você acha?

09/11/2005-15:14:14 A:essas classes seriam uma generalização de time A e time B que por sua vez seria uma generalização de TIME

09/11/2005-15:14:38 A:o que acha?

09/11/2005-15:14:54 B:perfeito! concordo! vamos ao trabalho!!!

09/11/2005-15:15:08 A:Vamos Lá....

09/11/2005-15:15:34 B:Ah... acredito que podemos dividir o trabalho... o que você acha?

09/11/2005-15:16:52 B:acho que falei isso tarde demais...

Figura B2 - Trecho do arquivo de registro contendo o diálogo dos usuários.

Apêndice C

Exemplos de Diagramas Obtidos no Experimento

Neste apêndice alguns dos diagramas produzidos colaborativamente durante o experimento serão reproduzidos. Estes diagramas contêm o resultado final da sessão de modelagem dos pares de usuários. No CD que acompanha este trabalho todos os diagramas gerados podem ser encontrados.

C1. Diagramas da Primeira Tarefa

A tarefa 1 consistia em modelar um diagrama de classes a partir de um texto que descrevia, com poucos detalhes, como funciona uma partida de futebol. Este tema foi escolhido por ser de fácil compreensão e por, supostamente, ser familiar a todos os usuários que participaram do experimento. As Figuras C1 e C2 são exemplos dos diagramas de classe produzidos colaborativamente durante a tarefa 1.

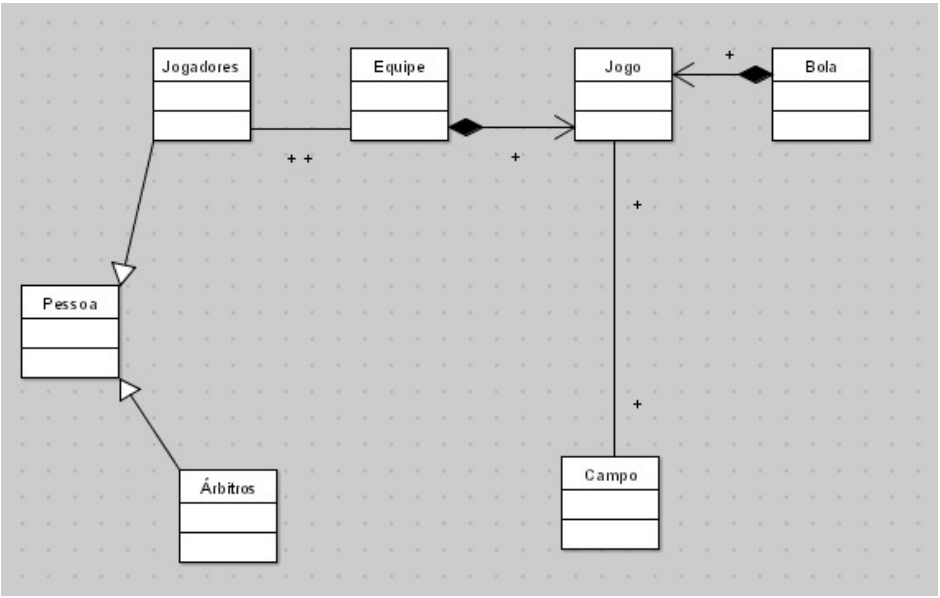

Figura C1 – Diagrama de classes produzido pelo par 1 durante a tarefa 1.

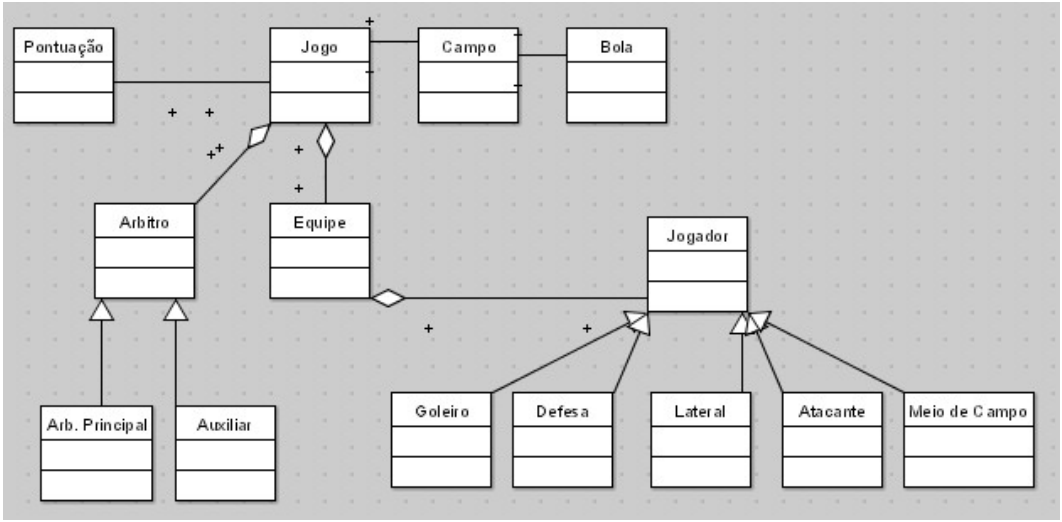

Figura C2 – Diagrama de classes produzido pelo par 3 durante a tarefa 1.

C2. Diagramas da Segunda Tarefa

Na tarefa 2 os usuários deveriam modelar um diagrama de casos de uso de acordo com as informações complementares que lhes foram apresentadas. O cenário fictício escolhido descreve do funcionamento de um consultório odontológico. As Figuras C3 e C4 são exemplos dos diagramas de casos de uso produzidos colaborativamente durante a tarefa 2.

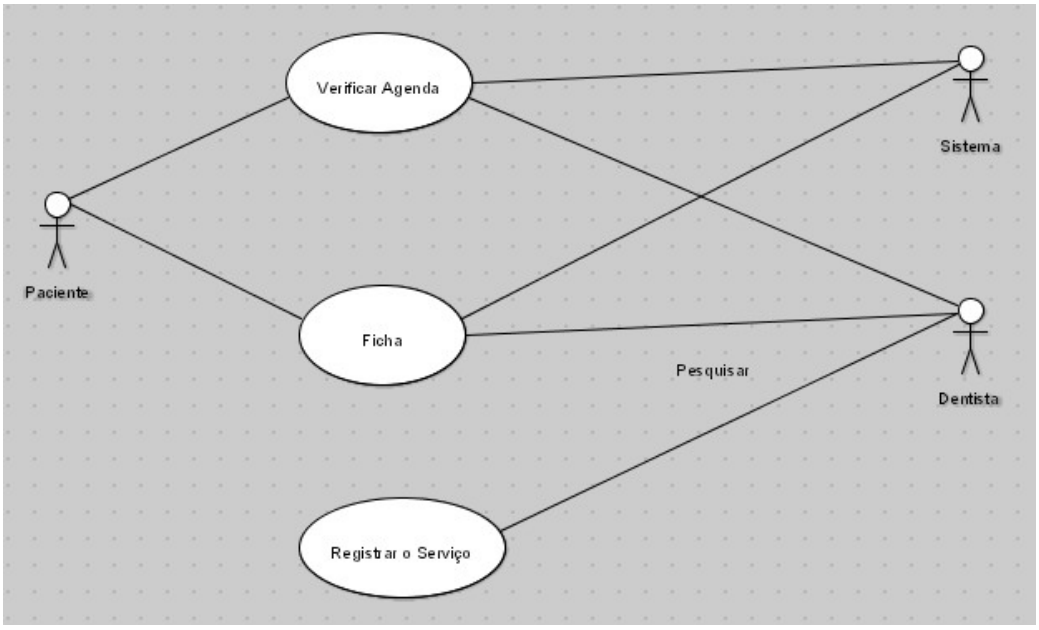

Figura C3 – Diagrama de casos de uso produzido pelo par 4 durante a tarefa 2.

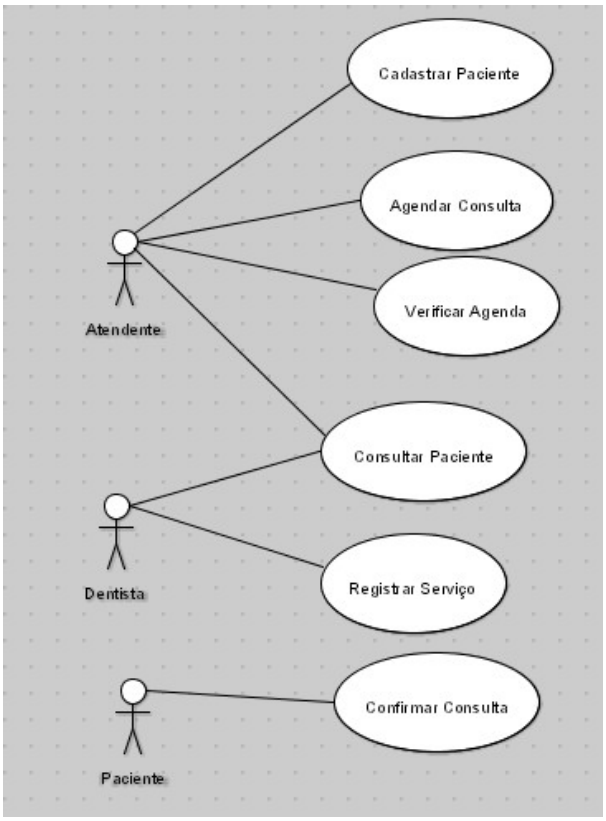

Figura C4 – Diagrama de casos de uso produzido pelo par 3 durante a tarefa 2.

C3. Diagramas da Terceira Tarefa

A tarefa 3 apresentou aos usuários uma situação onde os usuários deveriam modificar um diagrama de classes que já estava pronto e que deveria sofrer algumas modificações. O cenário da tarefa 3 descreve o funcionamento de uma locadora de DVD, no que diz respeito aos preços e gêneros de filmes com os quais esta locadora trabalha. As Figuras C5, C6 e C7 são exemplos dos diagramas produzidos colaborativamente durante a tarefa 2.

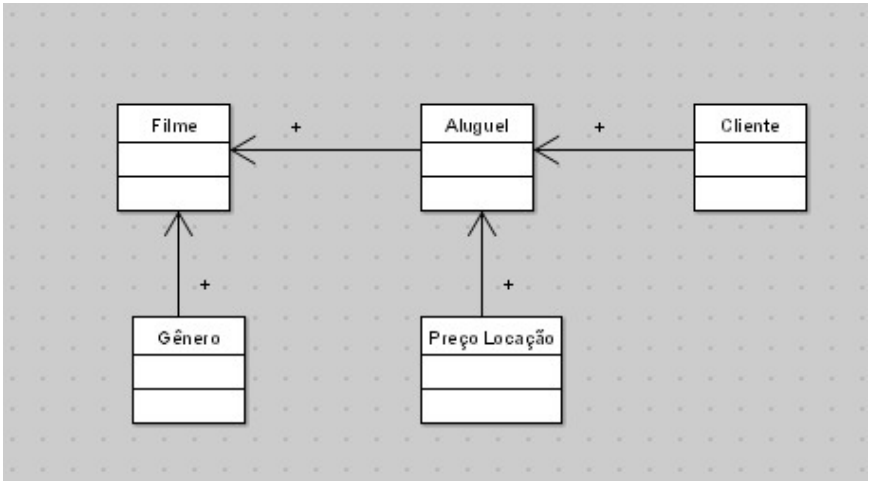

Figura C5 – Diagrama de classes produzido pelo par 1 durante a tarefa 3.

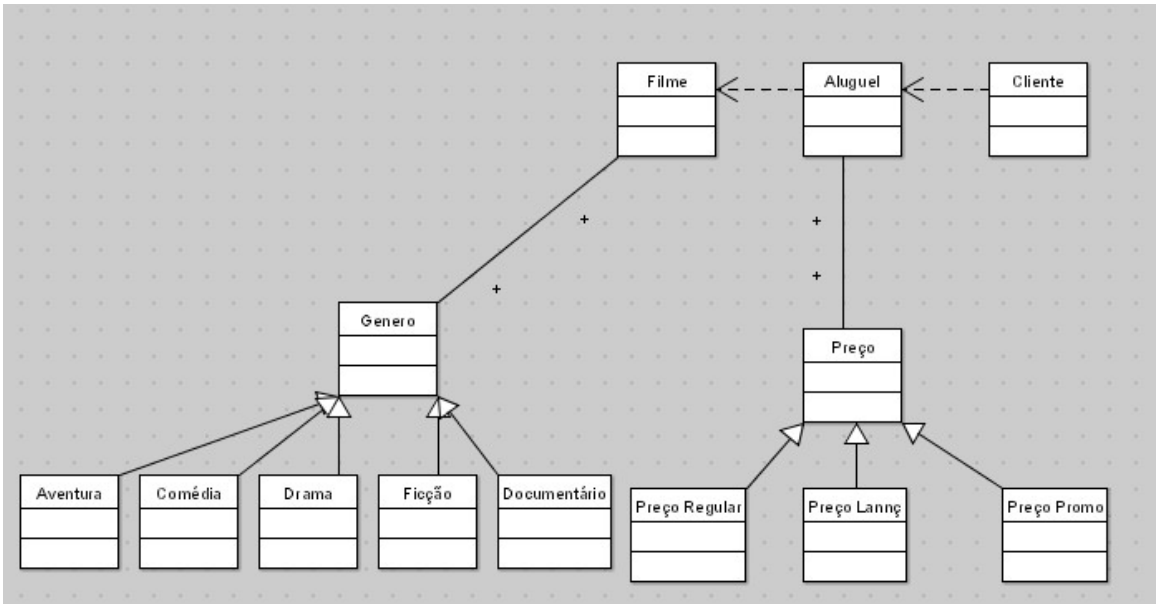

Figura C6 – Diagrama de classes produzido pelo par 3 durante a tarefa 3.

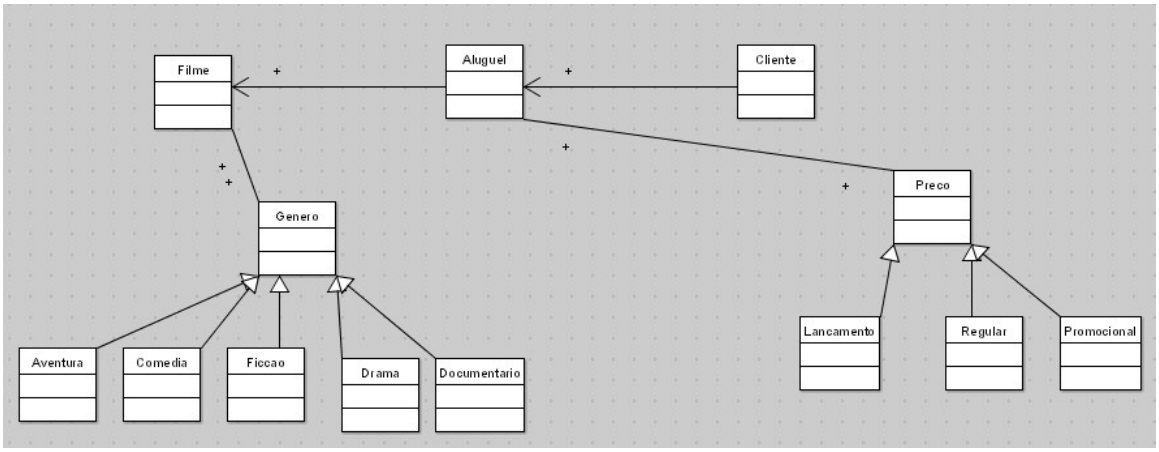

Figura C7 – Diagrama de classes produzido pelo par 2 durante a tarefa 3.