Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação
Departamento de Engenharia de Computação e Automação Industrial

Uma Arquitetura de Integração de uma Ferramenta Gráfica a Sistemas de Comunicação Síncrona Aplicada a um Ambiente de Aprendizado Colaborativo

Autor: Paulo Henrique Fisch de Brito
Orientador: Wu-Shin Ting

Trabalho apresentado à Faculdade de Engenharia Elétrica e de Computação da UNICAMP como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica.

Comissão Examinadora

Heloísa Vieira da Rocha

Ivan Luiz Marques Ricarte

Ricardo Ribeiro Gudwin

Campinas, 31 de julho de 2002

Brito, Paulo Henrique Fisch de
B777a Uma arquitetura de integração de uma ferramenta gráfica a sistemas de comunicação síncrona aplicada a um ambiente de aprendizado colaborativo / Paulo Henrique Fisch de Brito. --Campinas, SP: [s.n.], 2002.

Orientador: Wu-Shin Ting.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Ensino a distância. 2. Comunicação Visual. 3. Ensino auxiliado por computador. I. Ting, Wu-Shin. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: An architecture for integration of a graphical tool to synchronous communication systems applied to a collaborative learning environment
Palavras-chave em Inglês: Distance education, Visual Communication, Assisted instruction
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Heloísa Vieira da Rocha , Ivan Luiz Marques Ricarte, Ricardo Ribeiro Gudwin
Data da defesa: 31/07/2002
Programa de Pós Graduação: Engenharia Elétrica

Profa. Dra. Wu Shin-Ting (24.618-2)_____

Profa. Dra. Heloisa Vieira da Rocha :_____

Prof. Dr. Ivan Luiz Marques Ricarte:_____

Prof. Dr. Ricardo Ribeiro Gudwin:_____

Secretária Giane Cristina Sales Geraldo: _____

Coordenador PG Prof. Dr. Ariovaldo Verândio Garcia : _____

Agradecimentos

À minha orientadora por se aventurar junto a mim neste trabalho.

Aos professores Marcelo Firer e Sueli Costa pelas boas idéias e pelo apoio no desenvolvimento do nosso ambiente de colaboração.

Aos professores Antonio Carlos Gilli Martins pelo apoio na condução do experimento.

À professora Otília Terezinha W. Paques pelo espaço cedido em sua matéria para que o experimento fosse realizado.

À Carol que me acompanhou em todos os momentos deste trabalho;

Aos meus pais pelo apoio que sempre me deram

Resumo

Este trabalho trata da problemática envolvida na colaboração entre pessoas mediada pelo computador. Esta área tem crescido bastante principalmente no que se refere ao aprendizado colaborativo. Há uma ênfase, nestes ambientes, na comunicação textual em preterimento do uso de linguagens visuais que existem em diversas áreas do conhecimento. Esta ênfase acontece para que os ambientes atuais possam ser usados em diversas áreas e isso acarreta em um empobrecimento na capacidade de comunicação suportada. Esta tese apresenta uma arquitetura que pode ser usada pelos ambientes de colaboração para que seja possível a integração de aplicações gráficas de colaboração às ferramentas de comunicação genéricas (por exemplo, um bate-papo). Estas aplicações gráficas suportariam alguma linguagem visual de um determinado contexto específico. Com esta integração de ferramentas de comunicação haveria um complemento das linguagens verbal e visual ocasionando uma melhoria na comunicação suportada por estes ambientes. Além disso, é mostrado um ambiente de aprendizado colaborativo, denominado CoLab, que aplica esta arquitetura e um experimento, para testar o CoLab e a arquitetura de integração, foi conduzido e analisado.

Abstract

This work deals with problems involved in computer-mediated communication (CMC). This field has been growing in the context of collaborative learning. Most research efforts has gone into solving textual CMC problems in detriment of the visual communication ones. This is probably because current systems are intended to act in a broad range of fields. On the other side, such systems have poor support to convey information that is ubiquitously represented by visual symbols. Using both textual and visual communication together may allow us to do more cognitive work. Visual symbols are better for providing spatial relationships and details, whereas texts are better for passing abstract concepts. An architecture that supports the integration of collaborative graphic applications into textual generic communication tools is proposed. The graphic application may support visual communication through graphic symbols of specific context. Based on the proposed architecture a collaborative learning system, called CoLab, was developed and an experiment was conducted and analyzed.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Cada vez mais tem crescido o uso de sistemas que permitem que pessoas distantes colaborem entre si através do uso de computadores. Estes sistemas são usados dentro de empresas e dentro do governo mas uma das áreas onde a idéia da colaboração mediada pelo computador mais cresce é na educação. Neste contexto os ambientes de educação a distância (EAD) são os que mais utilizam a idéia de colaboração.

A educação a distância já se tornou uma realidade no Brasil, a cada dia aparecem novos cursos para diversos níveis de estudo existindo até mesmo cursos de pós-graduação pela *internet*. Estes cursos estão sendo criados no setor privado, como a Universidade Virtual Brasileira [1], e também no setor público com a Unirede [2]. Além dessas iniciativas de instituições que utilizam exclusivamente o ensino a distância também encontramos iniciativas em instituições tradicionais de ensino. A maioria, senão todos os sistemas de EAD utilizados nestes cursos, traz ferramentas que permitem a colaboração entre seus diversos tipos de usuários: alunos, monitores e professores (ou facilitadores dependendo do paradigma utilizado [3]). A

1.1 Aprendizado Colaborativo

O aprendizado baseado na interação entre os aprendizes (colaborativo) tem sido bastante estudado e defendido por diversas teorias, seja no campo da pedagogia com o sócio-interacionismo de Vigotsky [4], ou no campo da computação com as teorias de *Computer-Supported Collaborative Learning* (CSCL) como, por exemplo, a *Distributed Cognition* [5] ou a *Situated Cognition* [6]. A idéia principal é a do aluno que constrói o seu conhecimento em colaboração com os outros alunos. Nessa visão o professor não mais assume o papel de detentor do conhecimento. Ele deve, ao invés disto, agir como um “maestro” das interações que ocorrem no ambiente tentando guiar as interações dos alunos para atingir o objetivo do aprendizado [3].

Podemos dividir os sistemas de aprendizado colaborativo em dois grupos: os ambientes genéricos, voltados para qualquer área do conhecimento e os sistemas específicos, voltados apenas para um único domínio, mesmo que seja um domínio amplo. Indiretamente relacionados com ambientes de colaboração apareceram também diversas ferramentas de auxílio à construção de ambientes colaborativos. Veremos agora um pouco sobre estes sistemas, específicos e genéricos, e um pouco sobre estas ferramentas de auxílio.

1.1.1 Sistemas Genéricos

Os sistemas de aprendizado colaborativo do tipo genéricos são os mais comuns e utilizados. A característica principal destes ambientes é que eles têm, como proposta, a possibilidade de suportar cursos nas mais diversas áreas do conhecimento mas sem trazer ferramentas voltadas para alguma área ou contexto específico. Este caráter abrangente é conseguido com o uso de ferramentas de comunicação basicamente textuais, como o bate-papo (comunicação síncrona), o *e-mail* e o fórum de discussão (comunicação assíncrona). Isto não quer dizer que não exista o uso de gráficos e desenhos nestes ambientes. Este uso existe e é usado principalmente na difusão de informação através de hipertextos. Apesar disto, a comunicação direta entre os usuários é feita de forma basicamente textual, não há, por exemplo, um suporte nestes ambientes para o uso de linguagem visual ou para o uso de espaços compartilhados.

Quase todos os ambientes de educação a distância encontram-se nesta categoria. Podemos citar, entre outros, alguns dos que são utilizados no Brasil: o TelEduc [7], [8], o WebCT [9], [10], o AulaNet [11], [12].

O TelEduc (ver seção 2.2.1) foi desenvolvido no Núcleo de Informática Aplicada à Educação (NIED) da Unicamp. Este ambiente foi inicialmente concebido para “o processo de formação de professores para informática educativa”. Atualmente é utilizado para cursos a distância em diversas áreas.

O WebCT (ver seção 2.2.2), acrônimo para *Web Course Tools*, foi desenvolvido pelo grupo de Murraw W. Goldberg, da University of British Columbia. Esta plataforma foi criada para facilitar a criação de cursos na *internet* por professores que não eram da área de informática.

O AulaNet (ver seção 2.2.3) foi desenvolvido no Laboratório de Engenharia de Software (LES) do Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro

(PUC/RJ). Este ambiente tem como objetivos promover a adoção da *internet* como um ambiente educacional, contribuir com mudanças pedagógicas, dando suporte à recriação e encorajar a evolução do conhecimento, tanto para alunos quanto para professores.

1.1.2 Sistemas Específicos

Além dos sistemas de colaboração genéricos, para diversas áreas do conhecimento, existem sistemas que são específicos para determinados contextos, como álgebra [13] e ciências [14]. Estes *softwares* se valem do fato que são desenvolvidos para um contexto específico e trazem ferramentas também específicas para estes contextos.

Uma idéia comum nestes ambientes é o uso de espaços compartilhados. Estes espaços compartilhados são ambientes virtuais onde os usuários interagem entre si e com os objetos existentes. Normalmente estes objetos representam símbolos e/ou conceitos relativos ao contexto ao qual o ambiente se propõe a trabalhar. Segundo Michael Schrage é através do uso de espaços compartilhados que se chega a entendimentos compartilhados sobre o que se está trabalhando [15].

1.1.3 Ferramentas de Auxílio à Construção de Ambientes de Colaboração

Com a rápida disseminação do suporte computacional à colaboração, tanto para trabalho em grupo (*Computer-Supported Collaborative Work*, CSCW) como para a aprendizagem em grupo (*Computer-Supported Collaborative Learning*, CSCL), apareceram algumas ferramentas para auxiliar na prototipagem e construção de ambientes que suportassem trabalho e aprendizagem em grupo [16]. A idéia é facilitar o desenvolvimento de novos ambientes e manutenção de ambientes existentes através de reuso de código. Podemos citar como exemplo, os *toolkits* GroupKit [17], [18] e o Share-Kit [19]. Estes *toolkits* trazem componentes para diversos fins como controle de concorrência, construção de interfaces, comunicação remota, replicação de dados e outros.

1.2 Motivação

Embora as ferramentas de colaboração na maioria dos ambientes de EAD sejam genéricas, estudos na área de *Human-Computer Interface* (HCI) mostraram que é interessante que a interface computacional se aproxime ao contexto da área de conhecimento de seus usuários [20]. Townsend nota que para se construir um sistema colaborativo é necessário analisar o contexto e as necessidades da colaboração para poder então ajustar o sistema de colaboração a este contexto e necessidades [21]. Gutwin e Greenberg verificaram que o uso de comunicação visual acarreta melhorias na usabilidade de trabalho em grupo (*groupware usability*) em sistemas colaborativos [22]. Partindo de um conceito de usabilidade em sistemas individuais: "o grau em que um sistema é efetivo, eficiente e prazeroso de usar, dado um certo conjunto de usuários e atividades", Nielsen define a usabilidade em sistemas colaborativos ao incluir no termo "atividades" a idéia de trabalho em conjunto [23].

Além disso, diversos contextos e áreas se utilizam de linguagens visuais próprias. Estas áreas se utilizam de símbolos específicos no seu processo de comunicação. Podemos citar a Arquitetura com o uso de plantas e maquetes, a Física e a Matemática com o uso de gráficos de funções, tanto em 2D como em 3D, a Música com o uso de partituras e seus símbolos peculiares e diversas outras áreas. Estas áreas se utilizam de modelos de dados que são representados através destes signos gráficos. Estes modelos visuais executam um papel importante na comunicação de idéias. Ao tentar transpor esta comunicação visual para uma linguagem puramente escrita (comunicação textual) ou puramente falada (comunicação via áudio) temos uma perda na qualidade da informação.

Desta problemática surgiu a idéia de dar suporte computacional a uma colaboração que pudesse lançar mão de linguagens visuais para a comunicação de idéias. Esta tarefa não é simples já que cada contexto ou área do conhecimento exige diferentes modelos de dados e linguagens visuais e seria impraticável um sistema que previsse toda e qualquer forma de linguagem visual.

Uma possível solução seria então desenvolver uma arquitetura de sistema que previsse e com isso facilitasse a inclusão de uma aplicação que desse suporte à comunicação de idéias através de uma linguagem visual de contextos determinados. Este sistema deveria prover as ferramentas genéricas de comunicação (que se utilizam de texto, vídeo e áudio) tanto síncronas como assíncronas, e fornecer um modo simples de especializar o sistema para um determinado contexto através da integração de uma aplicação de comunicação visual, permitindo a

colaboração através de linguagens gráficas. O sistema deveria prover também todas as funcionalidades comuns a todas as aplicações colaborativas e gráficas (que permitem a comunicação visual) se estas existirem.

O nosso intuito será o de, além de utilizar as diversas ferramentas genéricas apresentadas nestes ambientes, poder complementá-los com a integração de uma aplicação que permita a colaboração através da comunicação visual às ferramentas existentes de comunicação verbal. Para que os símbolos específicos às diversas áreas do conhecimento complementem a "conversa" verbal tradicional destes ambientes a inclusão desta ferramenta de comunicação visual não deve ser "solta" no ambiente, isto é, não deve ser apenas uma ferramenta que simplesmente utilize estes símbolos mas que consiga fazê-lo de modo integrado às outras ferramentas de comunicação utilizadas. Esta ferramenta deve ser integrada às ferramentas de comunicação textual para que realmente as duas formas de linguagem se complementem. Esta complementação deve ser semelhante à que acontece na colaboração face a face quando junto com a fala e os gestos são utilizados gráficos, desenhos e outras formas de comunicação visual.

1.3 Objetivo

A idéia principal deste trabalho é a integração de ferramentas de comunicação e colaboração. A linguagem verbal, tanto escrita como falada, é rica mas no entanto perde detalhes ao tentar traduzir linguagens visuais de contextos específicos. Desejamos integrar ferramentas que permitam a comunicação por linguagens visuais específicas a determinados contextos às ferramentas de comunicação genéricas normalmente encontradas nos ambientes de colaboração.

Para isso, iremos em primeiro lugar propor um modelo de arquitetura que permita esta integração. Este modelo deverá ser simples para que possa ser implementado sem muitas dificuldades por sistemas de colaboração existentes no mercado. E simples também para os projetistas das aplicações gráficas que irão traduzir uma linguagem visual específica de alguma área do conhecimento.

Como primeiro passo na validação desta arquitetura, temos também como objetivo testar este modelo construindo um ambiente colaborativo de ensino/aprendizagem que se utiliza desta arquitetura e integrando uma aplicação gráfica para colaboração visual neste ambiente. E, por

último, iremos testar o ambiente fazendo um experimento real de modo a realizar algumas avaliações preliminares sobre a sua usabilidade.

1.4 Organização

No Capítulo 2 será feita a descrição de alguns conceitos utilizados neste trabalho e também descreveremos alguns sistemas correlatos ao que iremos apresentar. No Capítulo 3 iremos propor uma arquitetura que permite e facilita a integração de aplicações gráficas específicas a determinados contextos num ambiente provido de ferramentas de comunicação comuns aos sistemas de colaboração. No Capítulo 4 mostraremos um ambiente de colaboração voltado à educação a distância que implementa esta arquitetura e uma aplicação de comunicação visual que será integrada a este ambiente. A seguir, no Capítulo 5 mostraremos um experimento real que foi feito com o ambiente e a sua validação, e então as conclusões do trabalho no Capítulo 6.

Capítulo 2

Conceitos e Sistemas Correlatos

Neste capítulo iremos descrever alguns conceitos utilizados neste trabalho e também alguns sistemas relacionados com o ambiente desenvolvido. Entre os conceitos veremos o que chamamos nesse trabalho de linguagem visual (seção 2.1.1) e também a idéia de usabilidade e de consciência de grupo em interfaces de sistemas colaborativos (seção 2.1.2). Além disso apresentaremos o conceito de padrões de projetos (*Design Patterns*) e os padrões que foram utilizados na implementação do ambiente proposto (seção 2.1.3).

Entre os sistemas relacionados iremos ver três ambientes que suportam o aprendizado colaborativo, o TelEduc (seção 2.2.1), o WebCT (2.2.2) e o AulaNet (2.2.3), e um sistema em que os usuários podem compartilhar diversas ferramentas, de comunicação ou não, mesmo a distância, o NetMeeting (seção 2.2.4).

2.1 Conceitos

2.1.1 Linguagem Visual

É importante notar que o termo linguagem visual usado neste trabalho não se refere a linguagens de programação visual que utilizam recursos gráficos para auxiliar no processo de estruturação do algoritmo ou do fluxo de dados ao invés de especificar estes processos textualmente.

O conceito de linguagem visual que usaremos aqui pode ser definido simplesmente como o uso de imagens com o fim de comunicar idéias e conceitos. Estas imagens podem ser diagramas, mapas, gráficos ou qualquer outro símbolo não lingüístico. O conceito de linguagem visual é usado neste trabalho em contraposição e em complementação à idéia de linguagem verbal. Há

inclusive estudos que afirmam que as pessoas usam sistemas mentais diferentes para representar informações verbais e visuais [24] e que a memória de trabalho usada para representações visuais difere da memória usada para representações verbais [25].

2.1.2 Consciência de Grupo

Ao planejarmos qualquer ambiente, devemos levar em conta quem irá usar o ambiente e com que fim. O objetivo é melhorar a usabilidade do ambiente. Em ambientes individuais podemos definir a usabilidade como sendo “o grau em que um sistema é efetivo, eficiente e prazeroso de usar, dado um certo conjunto de usuários e atividades” [23]. Mas sistemas colaborativos não são como sistemas individuais. Embora as idéias de efetividade, eficiência e prazer continuem válidas para trabalhos em grupo, devemos estender o conceito de usabilidade incluindo no termo “atividades” a idéia de trabalho em conjunto [22].

A questão é: como, na prática, aumentar a usabilidade de um sistema? Gutwin e Greenberg perceberam que para aumentar a usabilidade ou a eficiência de ambiente colaborativo é necessário aumentarmos a consciência dos usuários de que eles estão participando de um trabalho em grupo, de que eles não estão sozinhos no ambiente. Chamaremos esta consciência de consciência de trabalho em grupo ou simplesmente consciência de grupo.

Para que o usuário perceba que está trabalhando em grupo em um determinado ambiente de trabalho este ambiente deve responder de modo transparente e automático questões como “Quem está no ambiente?” e “O que eles estão fazendo?”. Baseados nestas questões Gutwin e Greenberg definiram diversas categorias de consciência de trabalho em grupo. Na Tabela 2.1 podemos ver um resumo destas categorias.

No nosso trabalho a interface do ambiente foi construída com estas categorias em mente. Para isso, sempre que necessário e possível, foram inseridos elementos de interface que respondem estas questões referentes às categorias de consciência de grupo. Quando detalharmos a interface de colaboração (capítulo 4) apontaremos os elementos de auxílio à usabilidade em ambientes colaborativos cujas categorias foram descritas aqui.

Tab. 2.1 - Categorias de Trabalho em Grupo

Categoria	**Questões**
Quem	Quem está no ambiente? Quem está fazendo isto?
O Que/Qual	O que eles estão fazendo? Com qual objeto eles estão trabalhando?
Onde	Onde eles estão trabalhando? Onde eles podem ver?
Quando	Quando isto aconteceu?
Como	Como isto aconteceu?

2.1.3 Padrões de Projeto

Na área de engenharia de *software* existem problemas que aparecem com grande freqüência em aplicações das mais diversas. Um exemplo é a necessidade de se armazenar dados que serão utilizados depois na ordem inversa em que foram guardados, neste caso usa-se a estrutura clássica de dados denominada pilha. Baseado na idéia de que existem diversos problemas gerais recorrentes, apareceu na área de orientação a objetos (a idéia original apareceu na área de Arquitetura [26]), a idéia de padrões de projeto (*Design Patterns*). Estes padrões são definidos como “a descrição de objetos e classes intercomunicantes que são configuradas para solucionar um problema geral num contexto em particular” [27]. Para descrever cada padrão temos necessariamente que descrever o problema em questão e a solução proposta pelo padrão.

Uma das principais vantagens no uso de padrões consagrados, como os descritos em [27], é a criação de um vocabulário comum para nomear as soluções. Assim, entre pessoas que conheçam o mesmo vocabulário de padrões, é suficiente documentar a solução de um sistema ou uma parte deste, com os nomes dos padrões utilizados

Nas próximas seções iremos sintetizar a descrição dos padrões *observer* e *abstract factory*. Estes são os padrões que foram utilizados no projeto do nosso ambiente de colaboração.

2.1.3.1 *Observer*

Este padrão define uma dependência de um-para-muitos entre objetos de modo que se um objeto muda o seu estado, todos os seus dependentes são notificados e se atualizam automaticamente.

Muitas vezes quando usamos classes e relacionamentos, aparece o problema de manter consistência de dados entre objetos relacionados. Um efeito colateral disto é um acoplamento maior entre as classes o que restringe a possibilidade de reusabilidade.

Este é um problema comum quando temos diversas apresentações gráficas diferentes para um mesmo conjunto de dados. Por exemplo, podemos ter um conjunto de números que podem ser apresentadas, simultaneamente, como tabela e gráfico de barras. Tanto a tabela quanto o gráfico são dependentes do conjunto de dados e devem ser notificados quando este alterar o seu estado. Este exemplo fala de duas classes mas não há limite no número de dependentes.

O padrão *observer* (**Fig. 2.1**) descreve como estabelecer esta relação de dependência. Os dois objetos chaves são o *subject* e o *observer*. O *subject* pode ter qualquer quantidade de *observers*. Todos os *observers* são notificados quando o *subject* modifica o seu estado e devem sincronizar seu próprio estado com o deste.

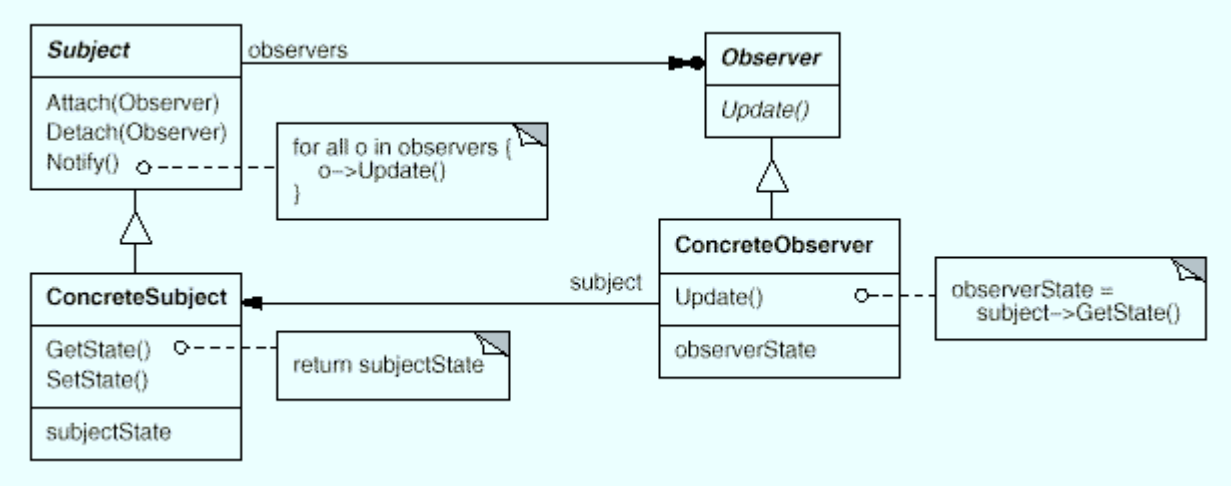

Fig. 2.1 - Padrão *observer*.

Participantes

- ***Subject***: Conhece os seus observadores (qualquer que seja o número destes). Provê uma interface para anexar e remover observadores;
- ***Observer***: Provê a interface de notificação da alteração de estado do *subject*;

- ***ConcreteSubject***: Armazena o estado de interesse dos *ConcreteObservers* e envia a notificação para estes quando da mudança de estado;
- ***ConcreteObserver***: Mantém referência ao *ConcreteSubject*, armazena o estado que deve estar consistente em relação ao do *subject* e implementa a interface de notificação do *observer*.

A aplicação para o caso de apresentação de dados é imediata, desacoplamos um modelo de dados (tabela de números no exemplo) de suas possíveis diferentes visualizações (tabela e gráfico). Um ponto importante a se destacar e relevante neste trabalho é que os observadores (objetos que são avisados da mudança de estado do modelo) podem tanto ser diferentes modos de visualização do modelo numa máquina isolada quanto visualizações em máquinas diferentes e por usuários diferentes num modelo de sistemas distribuídos.

2.1.3.2 *Abstract Factory*

Este padrão provê uma interface para criar uma família de objetos relacionados sem ter que especificar suas classes concretas.

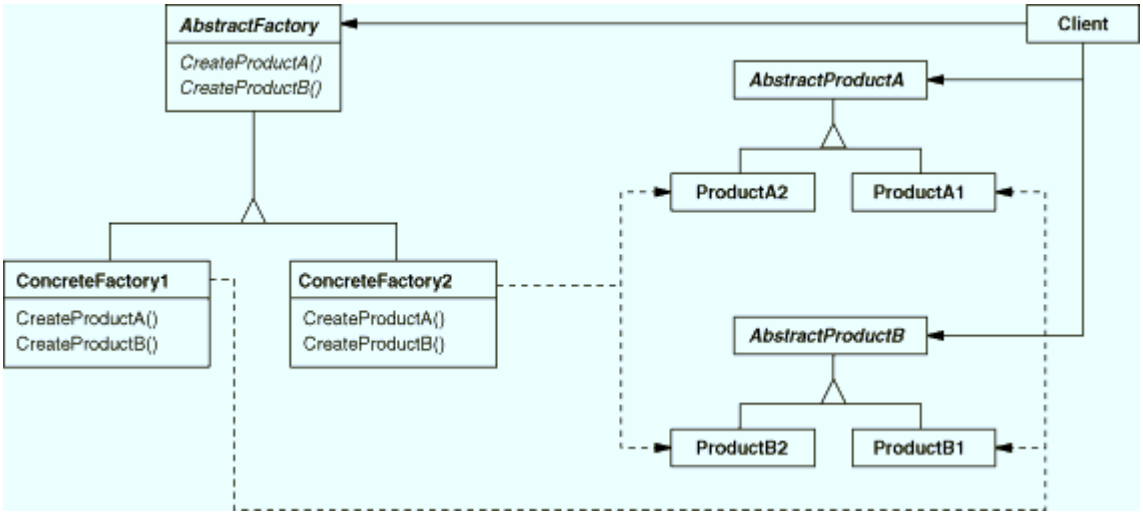

Fig. 2.2 - Padrão *abstract factory.*

Considere um *toolkit* que suporte diversos padrões de aparência (*look-and-feel*). Cada um destes padrões definem a aparência e o comportamento de itens (*widgets*) da interface gráfica

como barra de rolagem, janelas e botões. Para ser portável uma aplicação não pode ter referências "*hard coded"* dos itens de interface de uma aparência específica pois senão não será possível (ou será mais difícil) mudar a aparência.

Este problema pode ser resolvido definindo uma classe abstrata *WidgetFactory* que declara uma interface para a criação de cada item básico de interface gráfica. Deve haver também uma classe abstrata para cada um destes itens. A interface de *WidgetFactory* define uma operação que retorna um novo objeto para cada classe abstrata dos itens de interface gráfica. Os clientes chamam estas operações para obter instâncias dos itens mas não conhecem realmente a classe concreta que os implementam, assim os clientes podem ser independentes ou desacoplados do *look-and-fell* em uso.

O padrão a*bstract factory* (Figura 2.2) descreve as classes e relacionamentos que permitem implementar o comportamento descrito acima. Os objetos chave são o *concreteFactory* e o *concreteProduct* que implementam as interfaces definidas nas classes *AbstractFactory* e *AbstractProduct* e o objeto cliente que é "usuário" dos objetos e classes acima.

Participantes

- ***AbstractFactory***: Define a interface para operações de criação de objetos *abstractProduct*;
- **ConcreteFactory**: Implementa as operações de criação de objetos *concreteProduct*;
- ***AbstractProduct***: Declara a interface para um tipo de objeto produto;
- ***ConcreteProduct***: Define um objeto produto a ser criado pela fábrica concreta correspondente e implementa a interface definida em *AbstractProduct*;
- ***Client***: Usa somente as interfaces declaradas por *AbstractFactory* e *AbstractProduct*.

2.1.3.3 *Proxy*

Este padrão define um objeto para atuar no lugar de outro com o intuito de controlar o acesso ao mesmo.

Muitas vezes é interessante postergar a criação ou uso de um objeto até o momento em que isso é totalmente necessário, pois esta criação e inicialização têm um custo alto. Este custo pode ser, por exemplo, de memória ou de tempo. Um exemplo seria o de um editor de texto que permite a inserção de objetos gráficos que podem ter um tamanho elevado. É custoso criar o

objeto com a imagem embutida mas abrir um documento de texto deveria ser rápido. Seria interessante evitar a criação de todos os objetos dispendiosos logo que um documento é aberto, ainda mais que nem todos são visíveis assim que se abre o arquivo.

Possíveis aplicações incluem o *proxy* remoto, que "esconderia" o fato que um objeto encontra-se em outro local, o *proxy* virtual que permite otimizações quando a criação de um objeto naquele momento pode ser custosa e também o *proxy* de proteção que pode implementar políticas de segurança no acesso a um objeto.

O padrão *proxy* (Figura 2.3) descreve como estabelecer esta relação de dependência. Os dois objetos chaves são o *proxy* e o *realSubject*, estes dois objetos implementam a interface definida pela classe *Subject*. Com os dois objetos implementando a mesma interface fica transparente para o objeto *client* a existência do *proxy*.

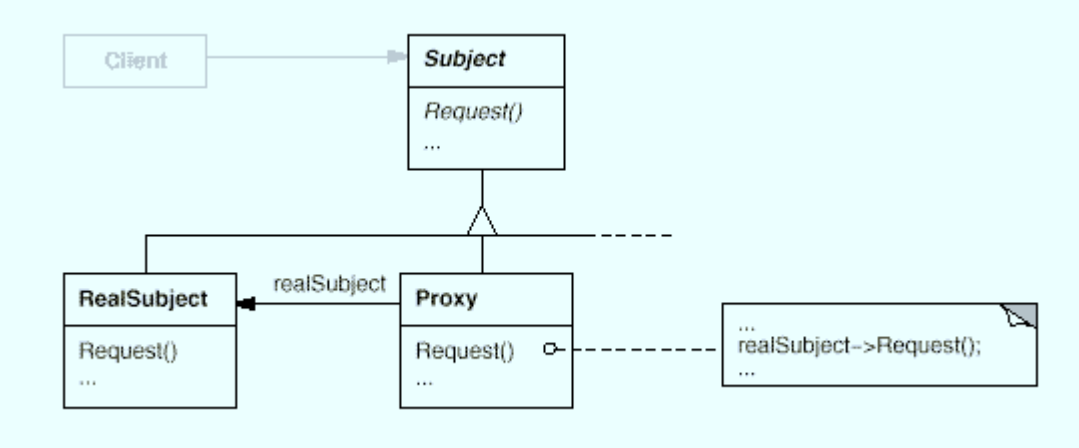

Fig. 2.3 - Padrão *proxy*

Participantes

- ***Proxy***: Controla o acesso ao real*Subject* de forma transparente ao cliente. Pode ser responsável pela sua criação, remoção e por implementar políticas de restrição de acesso, entre outras tarefas;
- ***Subject***: Provê a interface comum entre o *proxy* e o *realSubject*;
- ***RealSubject***: Define o objeto real ao qual o *proxy* representa;

2.2 Ambientes Relacionados

Nesta seção iremos falar de alguns sistemas que têm relação com o que é proposto neste trabalho. Embora nenhum deles possibilite exatamente o mesmo processo de colaboração que estamos propondo, eles oferecem possibilidades de colaboração que estão, de certa forma, englobadas no nosso projeto.

Os três primeiros sistemas, o TelEduc, o WebCT e o AulaNet, assim como o nosso, são voltados para a colaboração no ambiente educativo, isto é, buscam estimular o aprendizado dos participantes através das colaborações entre eles. Diferente do nosso trabalho, não trazem ferramentas visuais específicas ao contexto de nenhuma matéria.

O outro sistema que veremos é o NetMeeting. Este programa, embora não tenha um processo de aprendizado como foco, também traz diversas ferramentas de colaboração, entre elas o *whiteboard* e o *program sharing*. Este último possibilita o compartilhamento de qualquer programa, inclusive programas específicos do contexto de uma determinada matéria ou área do conhecimento.

2.2.1 TelEduc

O TelEduc é um ambiente para a criação, participação e administração de cursos na *internet*. Inicialmente concebido para "o processo de formação de professores para informática educativa", é atualmente utilizado para os mais diversos fins em educação a distância.

Este ambiente dispõe de diversas ferramentas para interação entre os usuários, tanto síncrona quanto assíncronas. Entre as ferramentas que permitem comunicação assíncrona temos:

- **Correio eletrônico**;
- **Fórum de discussões**;
- **Ferramenta perfil**: Permite que o usuário faça uma apresentação de si podendo incluir foto e texto. Auxilia na socialização dos usuários do ambiente;
- **Diário de bordo**: Semelhante aos *blogs*, possibilita que o participante relate um diário de suas atividades, têm finalidade tanto para o acompanhamento do que tem sido feito quanto na sociabilidade entre os usuários;
- M**ural**: Disponibiliza avisos gerais aos participantes.

Como ferramenta de comunicação síncrona existe um bate-papo comum. É possível perceber neste ambiente uma preocupação com a socialização dos participantes. É importante a percepção de que realmente há alguém por atrás de cada nome que aparece, neste sentido é de grande utilidade a foto, encontrada na ferramenta perfil e também os dados pessoais tanto no perfil quanto no diário de bordo.

2.2.2 WebCT

O WebCT foi criado com o objetivo de permitir que educadores que não fossem versados em informática fossem capazes de criar cursos a distância para *internet*.

O autor pode criar o seu curso e adicionar diversas ferramentas, algumas com o intuito de possibilitar a comunicação entre os usuários do ambiente e outras com perfil mais administrativo. Para comunicação síncrona há o sistema de conferência e o *chat*, e para comunicação assíncrona há o *e-mail*. Entre as ferramentas administrativas temos acompanhamento do aluno, auto-avaliação, questionários, distribuição e controle de notas, controle de acesso, geração automática de índices e pesquisa, glossário e calendário do curso.

O WebCT possui ainda uma interface para autoria de cursos que, além de auxiliar na criação de páginas, permite a incorporação das suas ferramentas educacionais. Após a criação de uma página, o autor deve inseri-la no contexto do curso indicando a sua localização relativa no curso, isto é, em relação às outras páginas criadas. A organização das páginas pode ser linear ou hierárquica.

2.2.3 AulaNet

O AulaNet propõe para si três objetivos: promover a adoção da *internet* como um ambiente educacional, contribuir com mudanças pedagógicas dando suporte à recriação e encorajar a evolução do conhecimento, tanto de alunos quanto de professores [28].

Este ambiente foi criado a partir de algumas premissas básicas:

- O curso criado deve possibilitar alta interatividade de modo a encorajar a participação intensa dos alunos no processo de aprendizagem;
- O autor do curso não precisa ser um especialista em *internet*;

- Os recursos disponíveis para a criação do curso deverão incluir os que normalmente estão presentes numa sala de aula além dos que são comumente encontrados em ambientes de *internet*;
- Deve ser possível reutilizar conteúdos já disponíveis em outras mídias digitais.

Este ambiente divide seus usuários, também chamados de atores, em três categorias distintas segundo a sua função:

- **Autor**: cria o curso, incluindo a descrição inicial e todo o seu conteúdo;
- **Estudante**: usuário final do curso;
- **Administrador**: facilita a integração entre o estudante, o instrutor e o curso e lida com todas as questões operacionais do curso.

Neste ambiente disponibiliza para os seus usuários uma ferramenta de comunicação síncrona, o *chat* e uma ferramenta de comunicação assíncrona, o *e-mail*.

2.2.4 NetMeeting

O NetMeeting [29] é um sistema proprietário da Microsoft que permite diversas formas de colaboração como áudio e vídeo conferência, *whiteboard*, bate-papo, transferência de arquivos e a ferramenta *program sharing* que possibilitaria o compartilhamento de qualquer programa instalado no computador. Veremos algumas destas ferramentas a seguir.

O *whiteboard* é uma ferramenta que permite a criação e edição de informações gráficas, com o compartilhamento de uma mesmo tela onde se pode fazer desenhos e inserir textos. Nesta ferramenta, cada usuário tem os apontadores do *mouse* com cores diferentes para que possam ser diferenciados enquanto trabalham, esta característica responde algumas questões das categorias “Quem”, “O Que” e “Onde” (seção 2.1.2).

A ferramenta de compartilhar um programa qualquer (*program sharing*) é provavelmente a mais poderosa deste sistema. Com ela é possível dois ou mais usuários visualizarem e interagirem com uma mesma instância de programa inicializada por um dos usuários. O programa (mais

precisamente a sua interface) atua como o espaço compartilhado onde os usuários podem trabalhar em conjunto.

Teoricamente qualquer programa (disponível para as plataformas que suportam o NetMeeting) pode ser compartilhado. A grande versatilidade desta ferramenta é que o programa a ser compartilhado não necessita ter sido escrito para trabalhar com o NetMeeting e nem mesmo ter sido escrito para ser utilizado num modo colaborativo. Teoricamente, qualquer programa escrito para a plataforma Windows pode ser utilizado dentro desta ferramenta. Isto é possível porque apenas a interface é compartilhada, todo o seu controle e execução ficam contidos numa única máquina da sessão de colaboração, a máquina que iniciou o programa.

Por um outro lado esta mesma versatilidade é responsável também pelos seu ponto fraco no que diz respeito ao suporte à colaboração. Como os programas compartilhados não são implementados para suportar a colaboração, isto é, não trazem suporte às categorias de consciência de trabalho em grupo, a colaboração não será tão efetiva como poderia ser. Além disto, como o programa compartilhado é executado apenas na máquina do usuário que o inicializou (o servidor), e o que é transmitido via rede é apenas a imagem da tela (do servidor para as máquinas clientes) e as ações de mouse e teclado (dos clientes para o servidor) algumas aplicações ficam inviáveis dependendo da largura de banda da rede.

2.3 Considerações Finais

Neste capítulo vimos os conceitos de consciência de grupo e padrões de projeto que serão utilizados nos capítulos subseqüentes. Também vimos ambientes diferentes voltados à colaboração: três utilizados especificamente com fins educativos (TelEduc, WebCT e AulaNet) e um sem propósito educacional podendo ser utilizado em qualquer ambiente que necessite de colaboração remota.

Nos ambientes educativos mostrados, ao contrário do nosso ambiente não há nenhuma forma de comunicação que utilize a linguagem visual que venha a existir no contexto da matéria estudada. Isto é, ao estudar uma matéria específica que se utilize de símbolos e notações próprias, diferentes da textual (por exemplo os gráficos na matemáticas e os desenhos de circuitos na engenharia elétrica), não é possível a comunicação através destes símbolos. Com isso perde-se uma importante capacidade de síntese de conhecimento que estas notações específicas permitem.

Capítulo 3

Uma Proposta de Arquitetura

Neste capítulo será mostrada a arquitetura de um sistema de comunicação e colaboração onde é possível integrar em uma ferramenta comunicação bem difundida (como um bate-papo) uma aplicação que permita uma comunicação através de linguagem visual o que acarreta numa especialização do sistema de comunicação. Mais especificamente, esta aplicação deve permitir o compartilhamento de espaços e modelos de dados que representem uma determinada linguagem visual, representativa de alguma determinada área do conhecimento, e cuja "tradução" para a linguagem textual poderia acarretar em perda de informação ou em aumento de complexidade na comunicação.

Limitaremos a nossa arquitetura de integração ao caso de uma ferramenta de colaboração síncrona por entendermos que este caso é mais complexo e que podemos a partir dele estender a mesma idéia para o caso da colaboração assíncrona.

3.1 Visão de Casos De Uso

Esta arquitetura será proposta com o intuito de facilitar a integração de uma aplicação de colaboração gráfica a um sistema mais amplo de colaboração e comunicação. Por isso, ela terá uma parte do sistema que chamaremos de caixa-branca que será a parte aberta do sistema, a parte que representa a aplicação que ainda será desenvolvida para especializar o ambiente para algum determinado contexto. E uma outra parte que chamaremos de caixa-preta que será a parte do sistema com a qual o desenvolvedor da aplicação gráfica não terá que se preocupar pois já estará pronta e implementada.

Levando-se em conta a parte caixa-branca e a parte caixa-preta a nossa arquitetura terá duas espécies de requisitos. Os requisitos normais de um sistema que representam a visão do usuário e

os requisitos para o desenvolvedor da aplicação colaborativa e gráfica que será o responsável por especializar o ambiente para um determinado contexto, isto é, os requisitos da caixa-branca.

3.1.1 Os Requisitos do Usuário

Antes de desenvolvermos uma arquitetura devemos ter em mente os requisitos desta arquitetura do ponto de vista do usuário, isto é, o problema que ela visa resolver. Mostraremos então dois exemplos práticos de sistemas que poderiam advir desta arquitetura:

- **Exemplo 1**: Dois alunos têm que fazer um trabalho onde devem manipular os parâmetros de uma função para controlar o seu comportamento. Um deles entra no sistema da escola e percebe que o outro aluno já está na "sala virtual" de trabalho do seu grupo. Ele entra nesta sala onde existem integrados um bate-papo e um traçador gráfico de funções. Ambos tem acesso e podem alterar as funções que são desenhadas no traçador (estas funções representam o modelo de dados da aplicação). Ao mesmo tempo que "conversam" no bate-papo usando texto, eles também se comunicam desenhando funções que exprimem as idéias que eles têm acerca da matéria. A visualização e a interação colaborativa com o modelo de dados funciona como linguagem de comunicação visual sobre a área de Matemática do mesmo modo que quando uma pessoa desenha um gráfico num papel para mostrar a uma outra. Ao final eles salvam o trabalho na área do grupo;
- E**xemplo 2**: Dois pesquisadores de biologia molecular abrem no horário combinado o sistema de trabalho colaborativo que os seus centros de pesquisa adquiriram. Enquanto eles conversam por uma áudio-conferência, compartilham a visualização de um modelo de uma proteína. Neste modelo eles podem fazer pequenas modificações na proteína através de manipulação direta, apontar e marcar trechos da proteína para os outros usuários que estão colaborando e modificar a sua visualização sobre a proteína (um *zoom* em determinado trecho da molécula por exemplo). Após discutir por uma hora, um deles salva o trabalho até aquele ponto na sua área particular do sistema enquanto o outro continua o trabalho.

A nossa idéia é que a arquitetura tenha uma parte fixa composta por uma ferramenta de comunicação genérica, isto é, independente do contexto onde deverá ocorrer a colaboração. Um exemplo deste tipo de ferramenta de colaboração é o bate-papo. Esta arquitetura quando implementada terá esta ferramenta genérica na parte "caixa-preta" do sistema. Ela existe (embora possa ser desabilitada) independente do contexto para que se deseja usar o ambiente de colaboração. A aplicação que permite a comunicação gráfica estará integrada a esta ferramenta de comunicação genérica para que as linguagens gráfica e verbal se complementem.

A aplicação gráfica, junto com o seu modelo de dados, fará parte então da "caixa-branca" do sistema. Esta é a parte que poderemos mudar de acordo com o contexto no qual queremos que o sistema atue. Nos exemplos acima, estes contextos seriam a Matemática (exemplo 1) e a Biologia Molecular (exemplo 2).

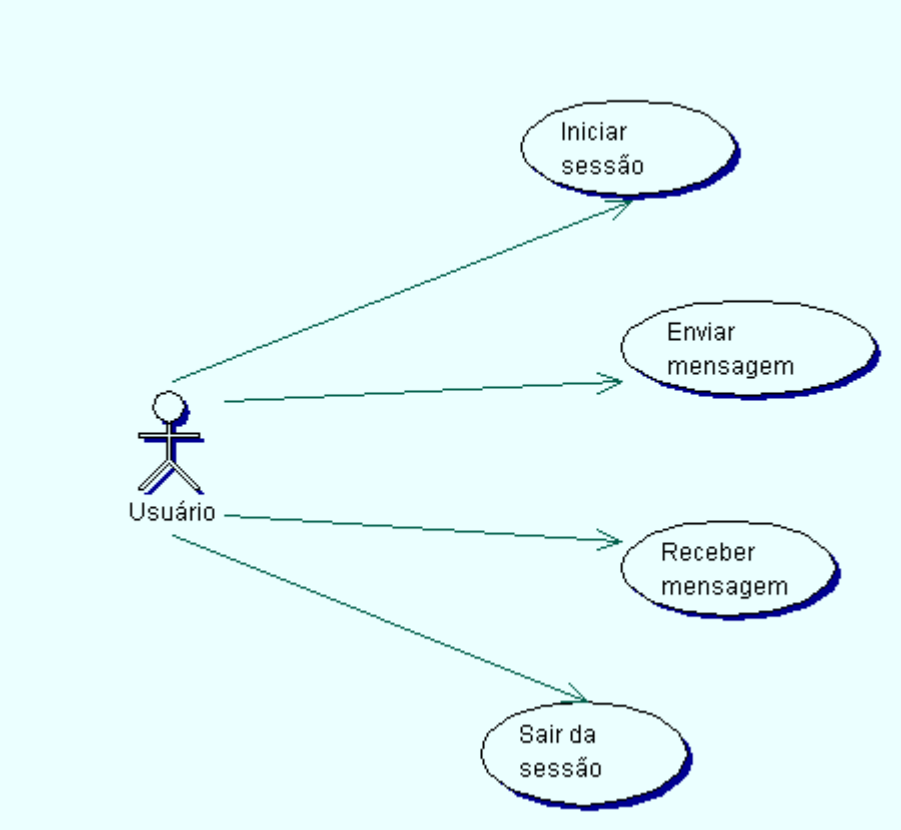

Fig. 3.1 - Casos de uso de uma ferramenta genérica de comunicação síncrona.

Para começarmos a modelar a nossa arquitetura temos que generalizar os seus casos de uso. No caso de uma ferramenta para comunicação síncrona teríamos os seguintes casos de uso (Figura 3.1):

Caso 1 – Iniciar sessão. Início de uma sessão ou acesso a uma sessão já aberta: o usuário escolhe uma das sessões ou das salas de discussão onde, por exemplo, ele tenha combinado com o seu grupo de se "encontrar";

Caso 2 – Enviar mensagem. Envio de uma mensagem aos outros usuários: ao enviar a mensagem esta deve ser prontamente distribuída via rede a todos os outros participantes da sessão (a não ser que seja uma mensagem privada para um único destino);

Caso 3 – Receber mensagem. Recebimento de mensagens dos outros usuários: todas as mensagens enviadas pelos outros usuários chegam a todos os destinos (a não ser que seja uma mensagem privada para um único destino);

Caso 4 – Sair da sessão. Saída da sessão.

Já no caso de uma aplicação que permita que diversos usuários se comuniquem com os outros usuários através de uma determinada linguagem visual, compartilhando e interagindo com um modelo de dados e comunicando-se assim com os outros usuários através de uma determinada linguagem visual, temos ainda os casos de uso (Figura 3.2):

Caso 5 - Abrir modelo de dados. Criação de um novo modelo de dados: este modelo de dados poderá estar armazenado local ou remotamente, mas ao ser aberto deverá ser o mesmo modelo que está sendo visualizado por todos os usuários da "sala";

Casos 6 e 7 – Modificar modelo de dados e perceber a modificação de outros usuários. Modificação do modelo de dados: interagir, com ou sem o uso de manipulação direta, com o modelo de modo a modificá-lo a fim de alcançar um determinado objetivo. A cada modificação do modelo há uma correspondente realimentação visual no aplicativo onde o modelo está sendo visualizado. Isto é essencial para que se tenha a consciência de que o trabalho está sendo feito em grupo;

Caso 7 – Salvar modelo de dados. Armazenamento do modelo de dados: o que poderá ser feito local, no computador do próprio usuário, ou remotamente como num diretório do servidor que poderia ser acessado por qualquer pessoa do grupo.

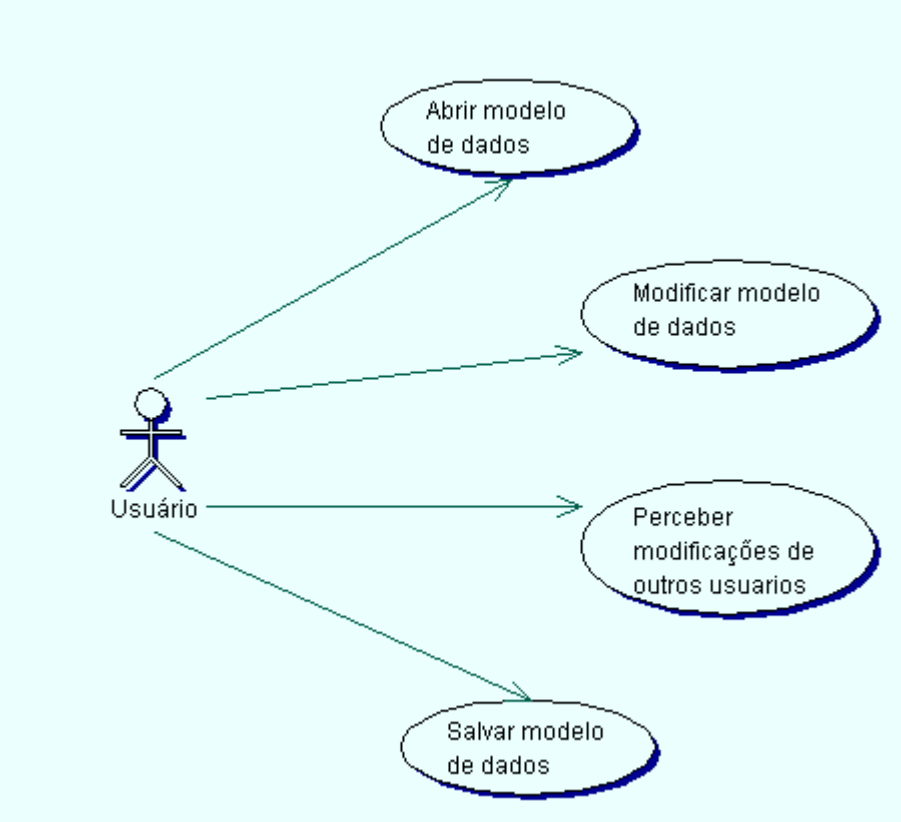

Fig. 3.2 - Casos de uso numa interação com um modelo de dados compartilhado numa aplicação de colaboração gráfica.

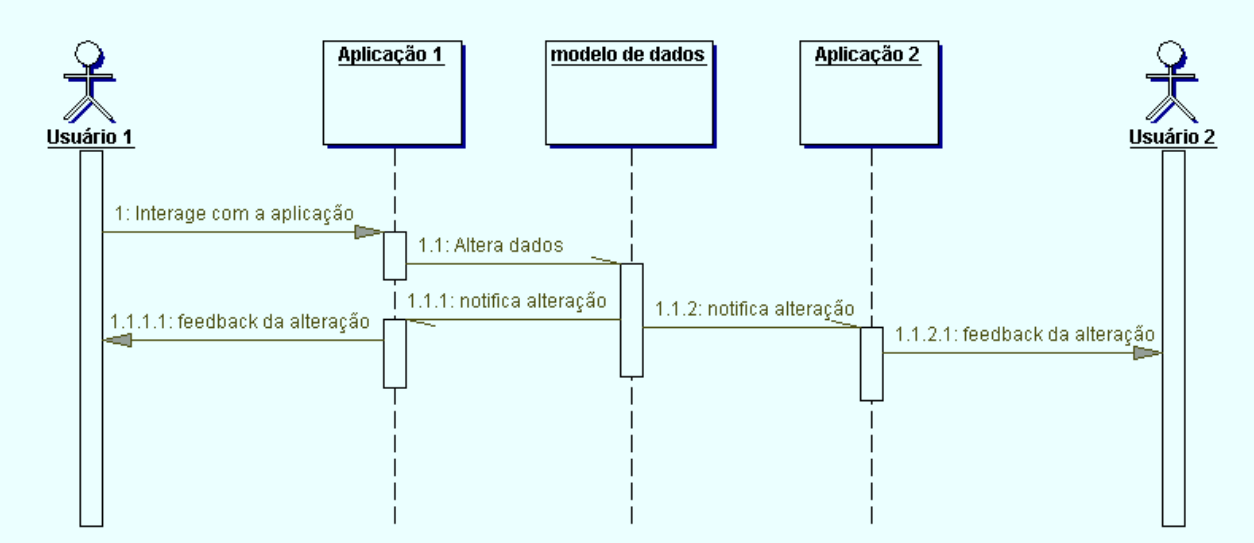

Fig. 3.3 - *Broadcasting* da alteração no modelo de dados.

Para que os usuários tenham realmente a noção de que estão colaborando com outras pessoas “dentro” de uma aplicação gráfica, é importante que eles percebam as modificações efetuadas por estas outras pessoas. Então, as modificações feitas por uma pessoa sobre o modelo

devem ser transmitidas a todos os outros usuários que estejam compartilhando este mesmo modelo de dados. Este comportamento da aplicação, correspondente ao caso 6, é um comportamento dinâmico o qual não é capturável num diagrama de casos de uso mas é representável num diagrama de seqüências como ilustrado na Figura 3.3.

3.1.2 Os Requisitos do Desenvolvedor de Aplicações Gráficas

O projeto da arquitetura deve facilitar ao máximo o trabalho do desenvolvedor das aplicações gráficas de colaboração. Para fazer isso devemos já deixar implementado ao máximo funcionalidades na parte caixa-preta mas ao mesmo tempo tentar não restringir as possibilidades de desenvolvimento dessas aplicações. Para que isso seja feito é necessário tentar abstrair as funções gerais que todas ou pelo menos a maioria (pois na prática é impossível dizer o que é geral para todas) das aplicações possuem.

Para isso iremos na seção 3.2 tentar explicitar claramente que parte do projeto pertence a caixa-preta e que parte seria caixa-branca. Para que realmente fique facilitada a tarefa do desenvolvedor além de um bom projeto é necessário também uma boa documentação deste projeto.

3.2 Visão do Projeto

Como foi visto na seção 3.1 a nossa arquitetura terá uma parte caixa-branca, variante, e uma parte caixa-preta, fixa, que será a ferramenta de comunicação genérica. A pessoa que for desenvolver uma nova aplicação de comunicação gráfica não terá que se importar com a parte caixa-preta mesmo com suas ferramentas de comunicação gráfica estando integradas a essa ferramenta genérica. Veremos agora exatamente que casos de uso se encaixam em cada caixa.

Os casos de uso de 1 a 4, referentes à ferramenta genérica, fazem parte, sem dúvida, da caixa-preta. Eles representam as interações com a ferramenta de comunicação genérica. Os casos de uso referentes à aplicação gráfica deveriam ser todos caixa-branca mas os casos 5 e 7, ler e salvar um modelo, são genéricos o suficiente para entrarem como caixa-preta. O ambiente de colaboração que se utilizar desta arquitetura (ao qual a aplicação gráfica estará integrada) é que

deve se preocupar com a questão da persistência de dados. É ele que deverá saber onde e como salvar os arquivos de cada usuário ou de cada grupo.

Veremos primeiro como fazer para integrar a parte variável do sistema com a parte fixa do sistema. Pelos exemplos vistos no início do capítulo e pelos casos de uso, vemos que a aplicação terá como uma das classes principais o **modelo de dados**. Este modelo deverá ser visualizado e alterado através de uma aplicação de visualização. Uma boa prática em *design* de *software* nos diz que devemos separar a apresentação de um dado da sua codificação [27]. Isso permite uma manutenção independente em cada uma destas partes; assim é possível, por exemplo, modificar o modo como um dado é visualizado sem ter que modificar o modo como um determinado dado é codificado, e vice-versa. Além disso, teremos diversas visualizações (diversos usuários com possivelmente diferentes visões sobre o mesmo modelo) de um único modelo de dados o que representa mais um motivo para separar o modelo de dados da sua aplicação de visualização.

Além da classe que implementará o modelo de dados temos a necessidade de uma classe que irá representar a **aplicação de visualização**. O modelo de dados conterá internamente as informações necessárias para que a aplicação construa a sua visualização. A aplicação por sua vez, através das interações com o usuário, irá requisitar alterações no modelo de dados e deverá também ser informada pelo modelo de dados quando ocorrerem alterações neste para que possa manter sempre atualizada a sua visualização.

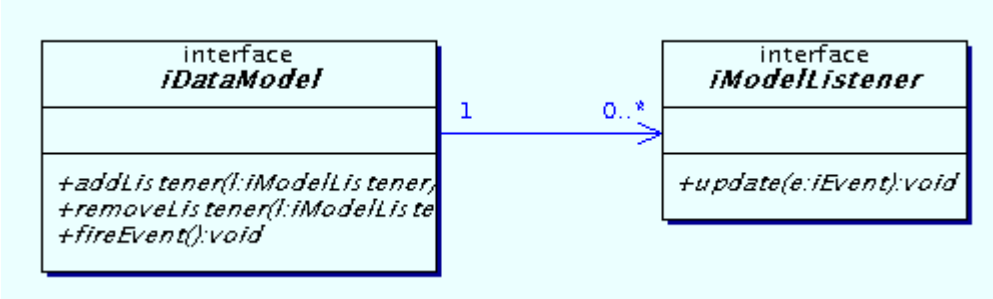

Fig. 3.4 - *Observer* implementando modelo de dados e aplicação de visualização.

O comportamento descrito por estes relacionamentos entre o modelo de dados e a aplicação podem ser implementados usando-se o *design pattern Observer* (seção 2.1.3.1).

Na Figura 3.4 podemos ver como as interfaces *iDataModel* (Subject) e *iModelListener* (*Observer*), que devem ser implementadas respectivamente pelo modelo de dados e pela aplicação de visualização, se relacionam segundo o comportamento descrito pelo padrão. Uma aplicação (implementando *iModelListener*) que queira mostrar e interagir com o modelo de dados

(implementando *iDataModel*) deve se "registrar" neste modelo de dados pelo método *addListener*(). Sempre que a aplicação modifica o modelo de dados, o método *fireEvent*() é chamado e notifica todos os *listeners* que tenham se cadastrado sobre a mudança efetuada.

Qualquer desenvolvedor que queira fazer uma aplicação para tirar proveito de uma linguagem visual utilizando esta arquitetura deverá escrever a sua classe de modelo de dados herdando a interface *iDataModel* e escrever a classe de aplicação de visualização gráfica herdando a interface *iModelListener* segundo o esquema da Figura 3.5. As classes com fundo cinza, nas Figuras **Fig. 3.5**, **Fig. 3.8**, **Fig. 3.9** e **Fig. 3.10**, representam uma aplicação de comunicação gráfica concreta, uma aplicação que seria integrada ao sistema utilizando a arquitetura aqui proposta.

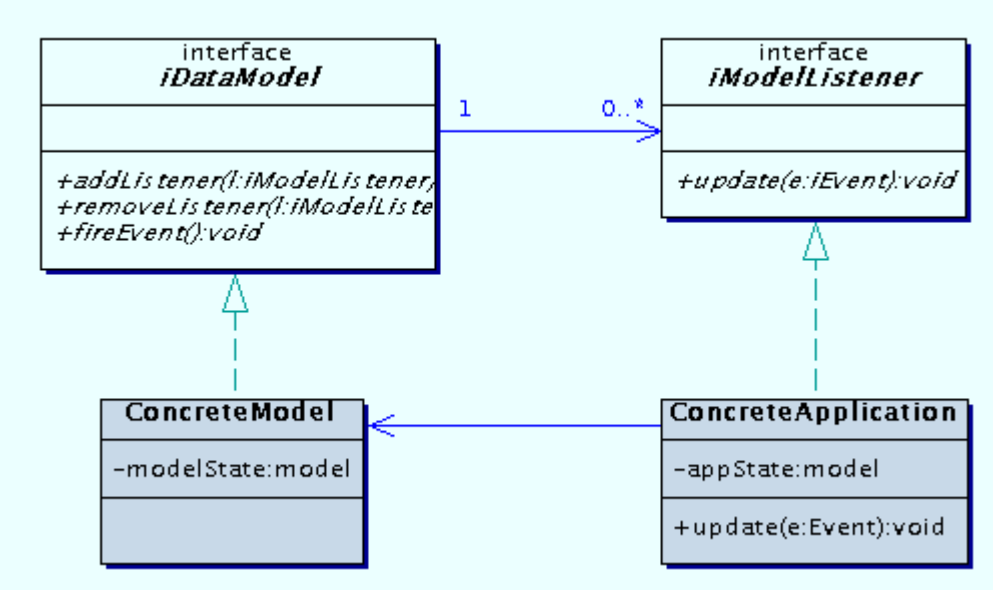

Fig. 3.5 - Classes concretas implementando o modelo de dados e aplicação de visualização.

Agora que já temos pelo menos uma interface que representa o modelo de dados (*iDataModel*) e uma que representa a aplicação de visualização colaborativa (*iModelListener*), temos que ver como integrar esta parte variável do sistema (caixa-branca) no resto da arquitetura (a caixa-preta).

A nossa arquitetura será implementada usando-se um modelo cliente-servidor. Tanto a ferramenta genérica quanto a aplicação gráfica terão um componente servidor responsável pela centralização e difusão (*multicasting*) das informações e um componente cliente que será responsável pela interação com o usuário. Veremos agora um modelo de classes para a

ferramenta de comunicação genérica. Normalmente num sistema onde muitas pessoas podem colaborar, como numa turma de alunos de um ambiente de educação a distância, faz-se uma divisão da área de colaboração em sessões diferentes (semelhante às salas de um bate-papo). Assim vários grupos podem colaborar com independência uns dos outros. Na Figura 3.6 vemos um modelo que representa as classes de uma ferramenta genérica deste tipo.

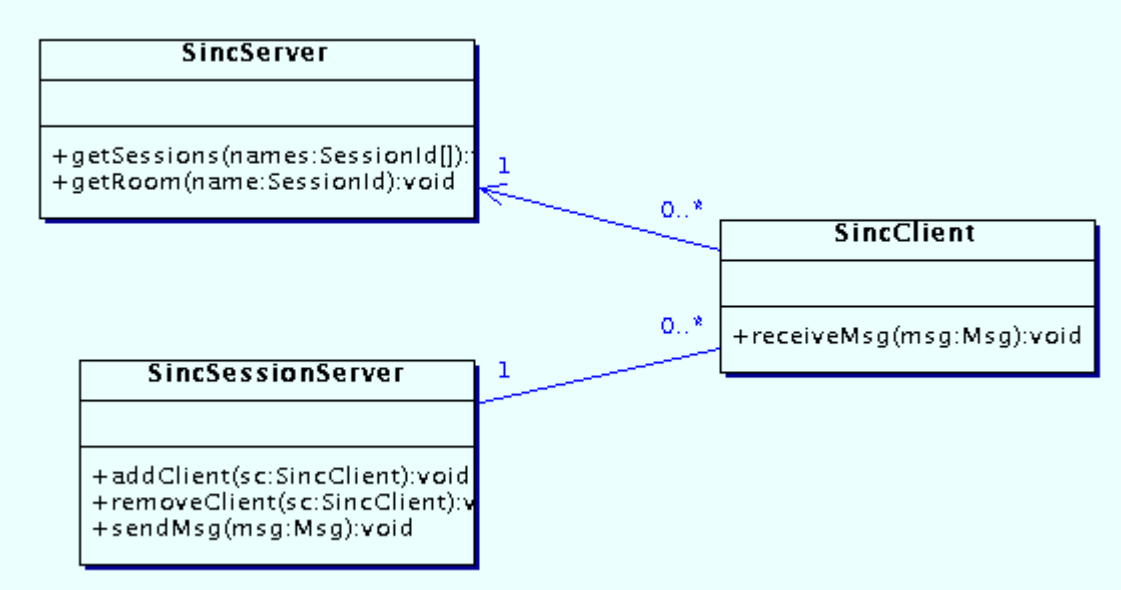

Fig. 3.6 - Classes implementando um modelo cliente-servidor para uma ferramenta de comunicação genérica.

O módulo cliente (classe SincClient) da ferramenta solicita as sessões (salas) existentes para a classe SincServer no servidor fazendo uma chamada ao método *getSessions():names[]*. Esta lista de sessões existentes é apresentada ao usuário que irá escolher uma delas. Neste instante começam os passos que serão interessantes mais para frente para o propósito da integração das ferramentas de comunicação (tanto a genérica verbal quanto a gráfica).

Em primeiro lugar vamos supor que este usuário é o primeiro a solicitar a entrada nesta sala ou sessão e que esta será inicializada no instante em que esta solicitação for feita. Na Figura 3.7 vemos as interações que ocorrem a partir deste instante. O módulo cliente (o objeto sincClient) chama o método sincServer. *getSession(name)*:SincSessionServer para conseguir uma referência e conectar-se ao servidor da sessão de comunicação. O objeto sincServer verifica que esta sessão não está inicializada, pois é a primeira vez que é feita uma solicitação por ela, e cria o objeto sincSessionServer como uma instância de SincSessionServer. Após fazer isto devolve ao módulo cliente a referência a este objeto. O cliente agora irá inicializar realmente a conexão à sessão de

comunicação. Ele passa uma mensagem para *sincSessionServer* pedindo o seu cadastramento na sessão através do método *addClient* (SincClient). A partir deste momento toda mensagem recebida pelo servidor da sessão através do método *sendMsg(msg)* é repassada, a cada cliente cadastrado, através do método *receiveMSG(msg)* do cliente.

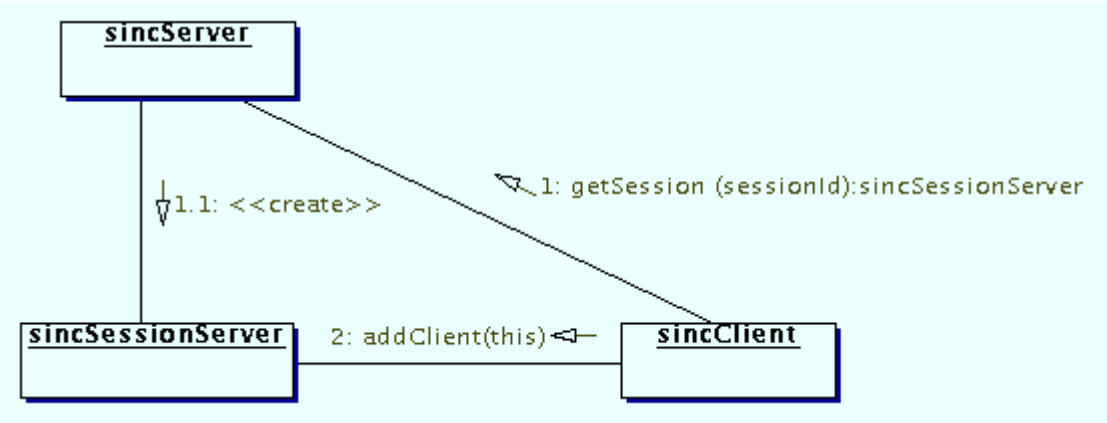

Fig. 3.7 - Diagrama de colaboração mostrando a conexão do cliente com o servidor da ferramenta de comunicação genérica.

Este modelo de uma ferramenta genérica de comunicação é simplificado. Ele poderia ser composto por diversas outras classes. Por exemplo, poderia haver uma classe que seria a responsável pela autenticação do cliente no servidor. Apesar disto, as interações descritas por este modelo são gerais e estarão presentes mesmo em modelos mais complexos. O que mais nos interessa neste modelo de interação é identificar em que instante e por quais classes serão inicializados o servidor e o cliente da aplicação gráfica para que possamos generalizar este procedimento no sistema o que irá permitir a troca da ferramenta gráfica sem maiores esforços.

Como a aplicação gráfica irá se integrar à ferramenta de comunicação genérica para que as linguagens (verbal e gráfica) representadas por elas se complementem iremos delegar para as classes SincSessionServer e SincClient a responsabilidade de instanciar, respectivamente, o servidor e o cliente da aplicação gráfica de colaboração. Na Figura 3.8 podemos ver o instante em que são criados o servidor e o cliente da aplicação. O servidor da ferramenta genérica ao ser criado irá criar também o servidor da aplicação gráfica (interação 1.1 e 1.1.1). O cliente da ferramenta genérica (*sincCliente*) ao se conectar ao servidor da sessão recebe como valor de retorno o endereço do servidor da aplicação gráfica que está integrado àquela sessão e usa este

endereço para criar o cliente da aplicação gráfica (interação 2 e 3) que por sua vez irá usar este endereço para se conectar ao seu servidor.

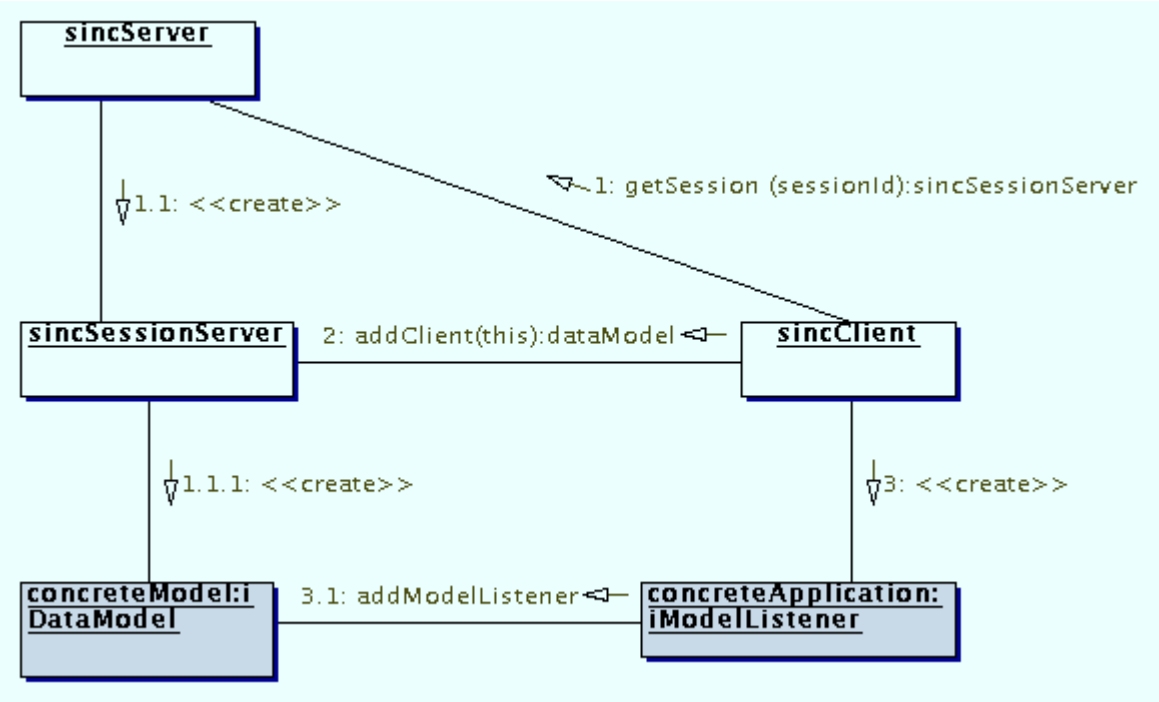

Fig. 3.8 - Diagrama de colaboração mostrando como a ferramenta de comunicação inicializa a aplicação gráfica.

Agora temos mais um problema para resolver. Vimos que o servidor e o cliente da ferramenta genérica de colaboração têm que inicializar, respectivamente, o cliente e o servidor da aplicação gráfica de colaboração. Mas só são conhecidas suas interfaces, e não as classes concretas que implementam a aplicação real, pois estas são a parte variante do sistema e ainda serão escritas por outros desenvolvedores. Para resolver este problema, o de instanciar uma classe quando se conhece apenas a sua interface usaremos o *design pattern* descrito no Capítulo 1.

Na Figura 3.9 vemos um exemplo de como seria a *Abstract Factory* no lado do servidor, que iria instanciar o modelo de dados concreto (*ConcreteModel*). A ferramenta genérica de comunicação, representada nesta figura pela classe SincSessionServer irá solicitar para ma classe que implementa a classe abstrata *DataModelFactory* que crie uma instância do modelo de dados *iDataModel*, isto é, de um objeto que representa e armazena o modelo de dados. Mas quem irá responder ao pedido será a classe concreta ConcreteModelFactory e que irá retornar uma classe concreta (ConcreteModel) que implementa a interface *iDataModel*.

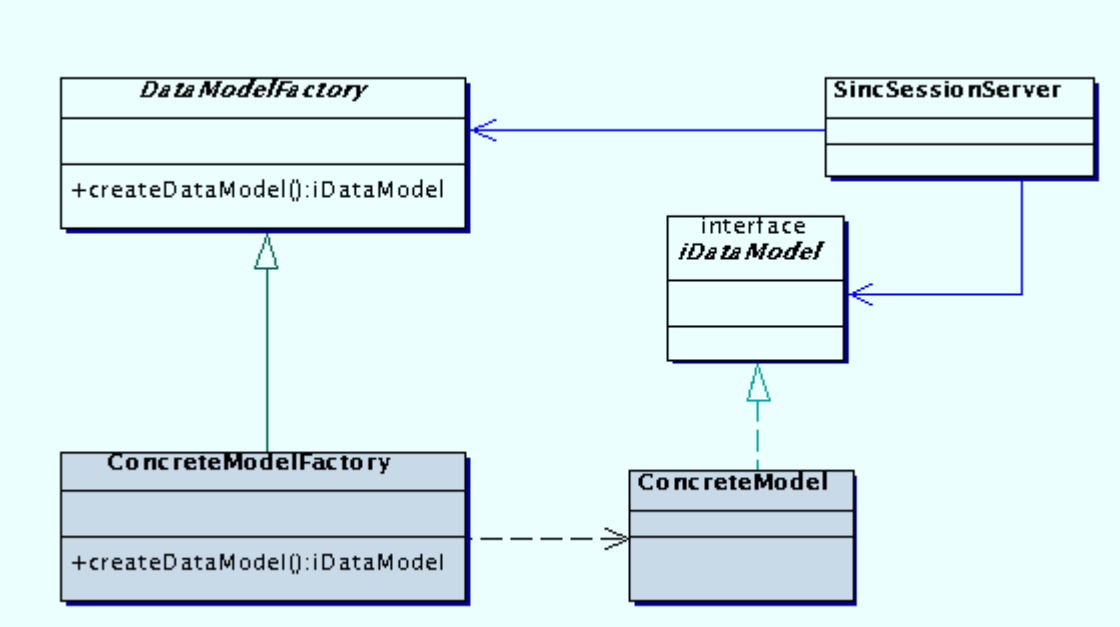

Fig. 3.9 - Classes concretas implementando o modelo de dados e aplicação de visualização.

Agora falta apenas vermos como fazer para que a ferramenta genérica (que está na caixa-branca) conseguisse uma referência à fábrica concreta da aplicação gráfica (caixa-preta). Esta associação entre a ferramenta de comunicação genérica com a fábrica correta (e conseqüentemente com a aplicação gráfica correta) pode acontecer de duas formas:

- **Associação estática**: Um exemplo onde poderíamos ter uma associação estática seria no caso de um sistema que tivesse apenas uma aplicação gráfica colaborativa para qualquer grupo ou usuário. Para este caso bastaria sobrescrever o método que retorna a fábrica de aplicação gráfica para sempre retornar a mesma fábrica. Neste exemplo teríamos um ambiente de colaboração novo para cada aplicação de comunicação gráfica nova;
- **Associação dinâmica**: Um exemplo onde teríamos uma associação dinâmica seria num ambiente de educação a distância. Nestes sistemas geralmente temos diversas turmas de diversos cursos diferentes onde provavelmente teríamos diferentes aplicações gráficas colaborativas co-existindo e seria interessante que sempre pudéssemos acrescentar novas aplicações gráficas para atender novos contextos. Neste caso teríamos que ter uma forma de fazermos uma associação dinâmica entre a aplicação gráfica colaborativa e a sessão de comunicação genérica. Uma forma de

se conseguir esta associação dinâmica seria a utilização de tecnologia de componentes “plugáveis” como, por exemplo, *JavaBeans* [30].

Por haver esta variação de opções de associação nós iremos deixar esta classe em aberto, ela teria que ser implementada de modo diferente para cada caso. No Capítulo 4 veremos como ela foi implementada no nosso ambiente.

No diagrama de classes da Figura 3.10 vemos esta classe e todas as outras no modelo completo da nossa arquitetura. As classes SessionModelAssociation e SessionAppAssociation, que fazem a associação entre a fábrica da aplicação gráfica e a ferramenta genérica, precisam ter suas estratégias definidas e implementadas de acordo com as características do ambiente de colaboração que aplicar esta arquitetura.

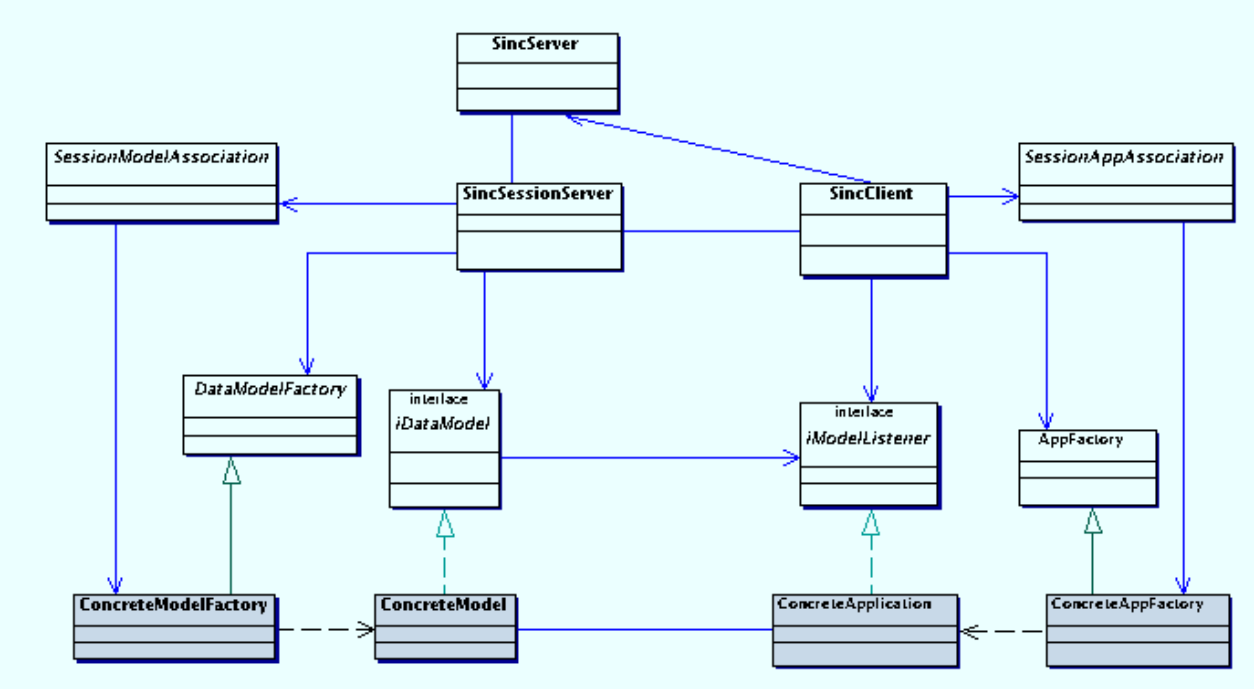

Fig. 3.10 - Modelo completo da arquitetura proposta.

3.3 Considerações Finais

O objetivo deste capítulo era propor uma arquitetura que servisse de modelo para a integração de uma aplicação de comunicação gráfica numa ferramenta de comunicação genérica

(como um bate-papo ou uma áudio/vídeo-conferência). A arquitetura foi proposta utilizando-se um modelo cliente-servidor.

O modelo foi proposto tendo em vista a simplicidade e, por conseqüência, uma maior abrangência das aplicações gráficas que poderiam ser construídas e integradas aos sistemas que utilizem esta arquitetura. Duas hipóteses são levantadas aqui. A primeira hipótese, sobre a simplicidade da arquitetura, é que não será complexo desenvolver uma aplicação gráfica e integrá-la a um sistema que implemente esta arquitetura; veremos como fazer isto no Capítulo 1. Já a segunda hipótese é que acreditamos também que quem já tenha um sistema de colaboração implementado e queira adaptá-lo para incorporar esta arquitetura também não terá grandes dificuldades. Esta segunda hipótese não será abordada nesta tese, ficando como trabalho futuro e sendo discutida no Capítulo 6.

Capítulo 4

Uma Aplicação: CoLab

Neste capítulo apresentaremos um ambiente de aprendizado colaborativo, chamado CoLab, que aplica a arquitetura proposta no Capítulo 3. Foi escolhida a área da Educação por ter sido esta uma das motivações deste trabalho. Apesar disto é importante frisar que a arquitetura proposta poderia ser utilizada para outras áreas, que não a Educação, que envolvam trabalho colaborativo e onde a linguagem visual atue de forma decisiva na comunicação.

Em conjunto com os professores Marcelo Firer e Sueli Costa, do Instituto de Matemática, Estatística e Computação Científica (IMECC) da Unicamp foi escolhida a área de funções de uma variável como o assunto a ser abordado no processo de aprendizagem. Uma ferramenta amplamente utilizada para ilustrar os conceitos neste contexto é o traçador gráfico de funções ou *plotter* gráfico. Portanto, como aplicação gráfica (a parte caixa-branca) do nosso ambiente, desenvolvemos um traçador gráfico colaborativo.

O desenvolvimento do CoLab se deu num processo iterativo junto a estes professores. Foram feitas diversas reuniões até se chegar num ambiente e num traçador de funções que fossem satisfatórios aos objetivos traçados para o curso. Estas reuniões faziam parte de um ciclo onde primeiro eram implementadas algumas funcionalidades que então eram mostradas aos professores nas reuniões. Estes por sua vez davam suas sugestões e opiniões que acarretavam em um novo ciclo com modificações, inserções e também remoções de funcionalidades.

Sendo o CoLab um ambiente de colaboração, uma das idéias que nortearam o desenvolvimento de sua interface e da interface da aplicação gráfica foi a de aumentar a consciência de que os usuários estão trabalhando em grupo, ou seja, uma maior percepção de que outras pessoas estão interagindo no mesmo ambiente de trabalho. Aumentando a consciência de que o trabalho está sendo feito em grupo aumentar-se-ia também a eficiência da comunicação entre os usuários [22].

Na seção 4.1 mostraremos uma visão geral do sistema. Na seção 4.2 iremos falar sobre os módulos fixos do sistema, isto é, os módulos caixa-preta. Na seção 4.3 falaremos nos módulos variantes, isto é, os módulos caixa-branca, a nossa aplicação gráfica. E por fim, na seção 4.4, os comentários finais.

4.1 O Ambiente

Para implementar nosso ambiente queríamos uma linguagem de programação que nos permitisse escrevê-lo de modo a minimizar o esforço e o tempo de desenvolvimento. Além deste fator nossa escolha se baseou em alguns critérios pré-definidos:

1. Facilidade para criar processos de comunicação via rede, entre as classes dos módulos cliente e servidor do nosso ambiente;
2. Possibilidade de encontrar componentes prontos na *internet* que pudessem ser reusados no nosso ambiente com o objetivo de reduzir o tempo de implementação;
3. Uma linguagem que fosse orientada a objetos pois o projeto da nossa arquitetura foi feito usando-se modelagem orientada a objetos;
4. Por fim gostaríamos que o nosso ambiente fosse multi-plataforma.

Escolhemos então a linguagem Java. Utilizamos a versão 1.2.2 do *Java Development Kit* última versão disponível na época em que este trabalho foi desenvolvido. Para o nosso primeiro critério, comunicação entre máquinas, Java conta com um pacote (*package*) apenas para comunicação de dados entre computadores diferentes, o pacote *remote method invocation* (rmi). Usando este pacote, é fácil fazer com que uma classe em um computador invoque um método em outra classe que se encontra em outro computador de forma absolutamente transparente (para a classe que está invocando o método), como se a classe invocada estivesse no mesmo computador. O trabalho fica por conta de se conseguir a "referência" a esta classe remota. Este trabalho também é facilmente executado usando-se um servidor de nomes (*rmiregistry*) que também existe no pacote rmi.

Para o segundo critério, o reuso de código, a linguagem Java também oferece facilidades. Há vários repositórios na *internet* de componentes prontos, os quais podem ser baixados e

utilizados. Há tanto componentes “fechados”, onde não se tem acesso ao código fonte quanto componentes “abertos” onde temos acesso ao código fonte. No nosso ambiente, como veremos mais adiante, nós usamos tanto componentes “fechados” como “abertos”, além dos que nós mesmos programamos. Como Java é uma linguagem orientada a objetos não tivemos problemas quanto ao terceiro critério. Os usuários de um ambiente de colaboração, seja voltado ao ensino ou não, podem estar utilizando diversos sistemas operacionais, como Windows, Linux e Unix, por isso o quarto requisito era justamente que o ambiente fosse multi-plataforma e Java atende a este requisito pois os programas rodam numa máquina virtual independente da máquina e do sistema operacional em que está instalada.

4.1.1 Um Ambiente Modular

Os ambientes voltados à aprendizagem em colaboração geralmente são compostos por módulos funcionais que atuam de modo mais ou menos independentes entre si, isto é, a utilização ou inclusão de um destes módulos não implica, necessariamente, na utilização ou inclusão de outros módulos. Em alguns ambientes é possível inclusive, para o facilitador ou professor do curso, decidir quais módulos estarão presentes para os alunos e quais módulos estarão de fora.

Pesquisando-se alguns dos ambientes existentes de aprendizagem colaborativa é possível verificar que eles possuem um conjunto semelhante de ferramentas básicas:

- Um navegador de *internet*, que normalmente é utilizado como veículo básico de divulgação do conteúdo do curso, além de outras informações;
- Uma ou mais ferramentas de comunicação síncrona, geralmente um bate-papo usado para reuniões *on-line*;
- Uma ou mais ferramentas de comunicação assíncrona. Normalmente encontramos uma ferramenta de *e-mail* e por vezes fórum de notícias.

Além destas ferramentas, voltadas para a comunicação de informações, entre aluno-professor e entre aluno-aluno, encontramos também ferramentas específicas para o professor que irá ministrar o curso colaborativo:

- Ferramentas que permitem controlar o ambiente do curso, como, por exemplo, a que permite especificar que módulos estarão presentes e que módulos estarão de fora do ambiente;
- Ferramentas que apresentam estatísticas sobre a utilização do ambiente por parte dos alunos ou aprendizes. Estas ferramentas podem, por exemplo, mostrar a quantidade de interações no bate-papo e entre que usuários ocorreram estas interações, quantidade de páginas visitadas pelos alunos ou tempo de acesso a cada página.

Devido ao tempo disponível para a implementação do nosso ambiente tivemos que nos decidir por algumas destas ferramentas apresentadas e deixamos outras de lado. Além disto incluímos outras que não são tão comuns a estes ambientes mas que achamos que seriam úteis. Apesar de acharmos importantes as ferramentas especificamente voltadas para o professor foi uma decisão de projeto nossa não incluí-las devido ao já citado tempo disponível para implementação e por acharmos que elas eram prescindíveis na avaliação da viabilidade da arquitetura proposta.

Os módulos que incluímos no nosso ambiente são:

1. Um navegador de *internet*;
2. Um módulo gerenciador de grupo, com uma interface gráfica com informações sobre os usuários que se encontram *on-line*, isto é, conectados ao ambiente num determinado instante;
3. Uma ferramenta de *e-mail*;
4. Um fórum de notícias;
5. Um bate-papo;
6. Um editor de relatórios;
7. Um *plotter* colaborativo;
8. Um *plotter* individual.

4.1.2 Organização

Em concordância com a arquitetura proposta, o CoLab utiliza o modelo cliente-servidor. Dividimos então nossas classes nos pacotes, *server* e *client*. Além desta divisão há também a divisão das classes nos módulos que elas implementam. Na **Fig. 4.1** podemos ver como ficou a divisão lógica das classes em pacotes. O pacote *chat* com as classes do bate-papo, o pacote *tool* com as classes abstratas da aplicação de colaboração visual, o pacote *plotter* com as classes do traçador gráfico de funções, o pacote *mail* com as classes da ferramenta de *e-mail*, o pacote *forum* com as classes do fórum de notícias, o pacote *group* com as classes do gerenciador de grupo e o pacote *gui* (*graphical user interface*) com as classes da interface do usuário. Os relacionamentos mostrados entre os pacotes *chat*, *tool* e *plotter* representam os relacionamentos entre as classes deste pacote mostrados na Figura 3.9 e serão mais bem explicados quando mostrarmos mais detalhadamente cada módulo.

Como para o nosso experimento só haveria uma aplicação gráfica colaborativa fizemos uma associação estática entre o bate-papo e a fábrica da nossa aplicação gráfica. Há uma classe denominada *ModulesOwner* que contém um método que retorna a fábrica do *plotter*. Por questão de homogeneidade e organização agrupamos a criação de todos os módulos nesta classe. Todos os módulos são “acessíveis” por este ponto em comum. Esta classe tem dois métodos específicos para cada módulo:

1. Verificar a existência de um módulo;
2. Retornar uma referência para este módulo, criando-o se necessário.

A única exceção são os módulos dos traçadores (*plotter* colaborativo e individual). Assim um módulo que necessite uma informação sobre outro ou enviar uma mensagem para outro módulo pode usar o método do item 2 para conseguir uma referência para este outro módulo. Há uma classe ModulesOwner no servidor e uma no cliente.

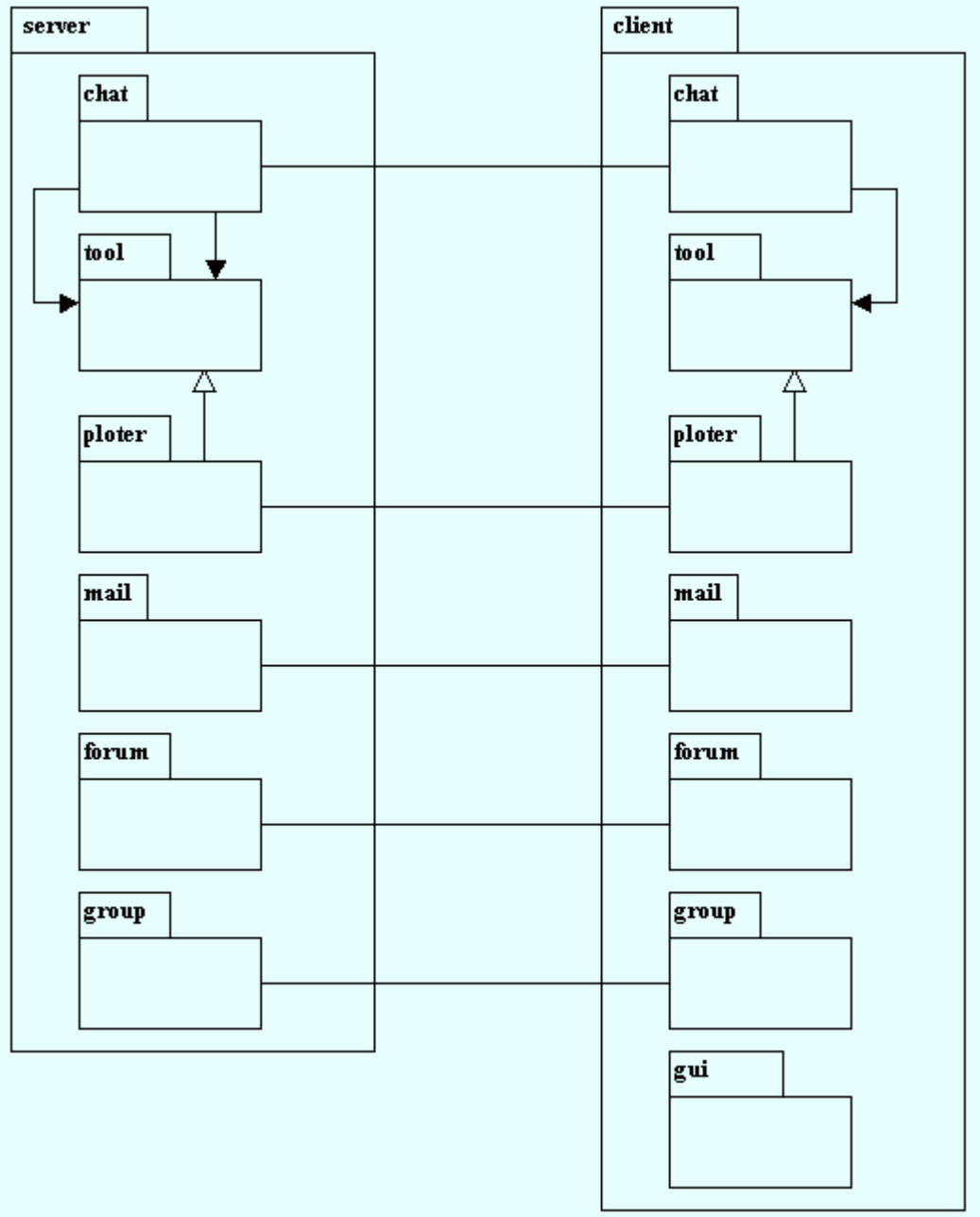

Fig. 4.1 - Divisão das classes do CoLab em pacotes

No caso dos *plotters*, por eles constituírem a parte variante do sistema, isto é, a parte caixa-branca, não é retornada a classe que os implementa mas sim a classe que implementa a sua fábrica de criação. A fábrica do *plotter* é retornada através do método *ModulesOwner.getToolFactory()*. Como já visto temos classes para os mesmos módulos no servidor e no cliente. Com isso, há a *server.ModulesOwner* e a *client.ModulesOwner*. Estas

classes são usadas apenas internamente nestes pacotes. Para que os clientes consigam uma referência para seus servidores foi usado um servidor de nomes.

Nas próximas seções iremos detalhar melhor os módulos, incluindo o objetivo de cada um. Podemos dividir os módulos em dois grupos distintos, os que têm relação com a implementação da arquitetura proposta no Capítulo 3 (os módulos do bate-papo e do *plotter*) e os que não têm. Na seção 4.2 mostraremos a "visão de implementação" dos módulos que se relacionam com a arquitetura proposta e na seção 4.3 uma visão mais geral dos outros módulos.

4.2 Módulos Fixos ou Caixa-Preta

4.2.1 Módulo Navegador

A janela principal do ambiente é composta por dois módulos (Figura 4.2). O primeiro deles é um navegador de *internet*. A função do navegador é a de disponibilizar o conteúdo do curso. Usamos esta ferramenta para disponibilizar um tutorial sobre o próprio ambiente CoLab, o objetivo do curso e um cronograma de como ele seria dado. O curso dado, que foi baseado em exercícios, também teve suas aulas disponibilizadas através deste navegador. As páginas com as tarefas de cada aula continham diversos *links* para outras páginas com as explicações teóricas que poderiam facilitar na resolução dos exercícios.

Um dos nossos critérios de desenvolvimento era tentar minimizar o tempo de desenvolvimento através de reuso de código. Por isso e pelo fato do navegador não estar envolvido na implementação da nossa arquitetura, nós optamos por usar um componente disponível na *internet*. Encontramos dois componentes prontos que utilizam *Javabeans*, a tecnologia de componentes da linguagem Java e que poderiam facilmente ser acoplados ao CoLab. Abaixo os prós e contras de cada um:

- **HotJava –** Era desenvolvido pelo próprio fabricante da linguagem Java. Dispunha de código-fonte dependendo da versão que se quisesse utilizar. Contra ele havia o problema de que este projeto estava sendo descontinuado e, além disto, ele utiliza uma tecnologia de interface gráfica (AWT) mais antiga e com algumas incompatibilidades com a que nós usamos no nosso projeto (SWING);

- **ICEbrowser –** Oferecia um suporte melhor aos padrões utilizados em páginas da *internet*. É de uso livre para fins não lucrativos mas é um *software* comercial.

Optamos pelo ICEbrowser por causa das deficiências do HotJava. A inclusão deste componente no ambiente não teve maiores dificuldades.

Fig. 4.2 - Janela principal do ambiente.

4.2.2 Módulo Gerenciador de Grupo

O segundo módulo que encontramos na janela principal (Figura 4.2) é o módulo gerenciador de grupos (Figura 4.3. O objetivo deste módulo é facilitar que os usuários, tanto os alunos como os professores e monitores, se "encontrem" no ambiente. Com este módulo, o usuário pode descobrir: todos os outros usuários que estão interagindo no ambiente naquele instante, a categoria de cada usuário (no nosso experimento: aluno, monitor e professor), a que grupo de trabalho pertence cada usuário, e ainda em que sala de bate-papo cada usuário se encontra.

Segundo as categorias descritas por [22], esta janela fornece informações da categoria **Quem** que seria uma das formas de se aumentar a consciência de trabalho em grupo.

Fig. 4.3 - Janela com as informações sobre os usuários conectados ao ambiente.

Assim que o aluno entra no ambiente, ele pode facilmente identificar se há outros integrantes do seu grupo atuando naquele instante no CoLab e com isso terminar o trabalho que eles têm que fazer juntos. Ou ainda se não há um professor para tirar uma dúvida, ou ainda verificar que não há ninguém e ele decidir fazer uma parte do trabalho sozinho.

4.2.3 Módulo Relatório

Faria parte do curso que demos como experimento, a elaboração de relatórios por parte dos alunos sobre as tarefas que eles realizariam no curso. Tínhamos a opção de deixar que eles usassem um editor fora do ambiente ou então nós teríamos que implementar um editor de texto, mesmo que simplificado, dentro do CoLab. Cada abordagem apresenta suas vantagens e desvantagens.

Pedindo que os alunos usassem um editor fora do nosso ambiente nos pouparia um bom tempo de implementação que poderia ser utilizado em outras tarefas. Mas em compensação diversas tarefas ficariam dificultadas. O compartilhamento dos dados fica dificultado, com o editor dentro do CoLab o aluno só precisa gravar o relatório na área de trabalho do seu grupo para compartilhar este relatório. Além disso, ele não precisa ficar se “movendo” entre CoLab e o editor para transferir dados (por exemplo, inserir uma figura no relatório).

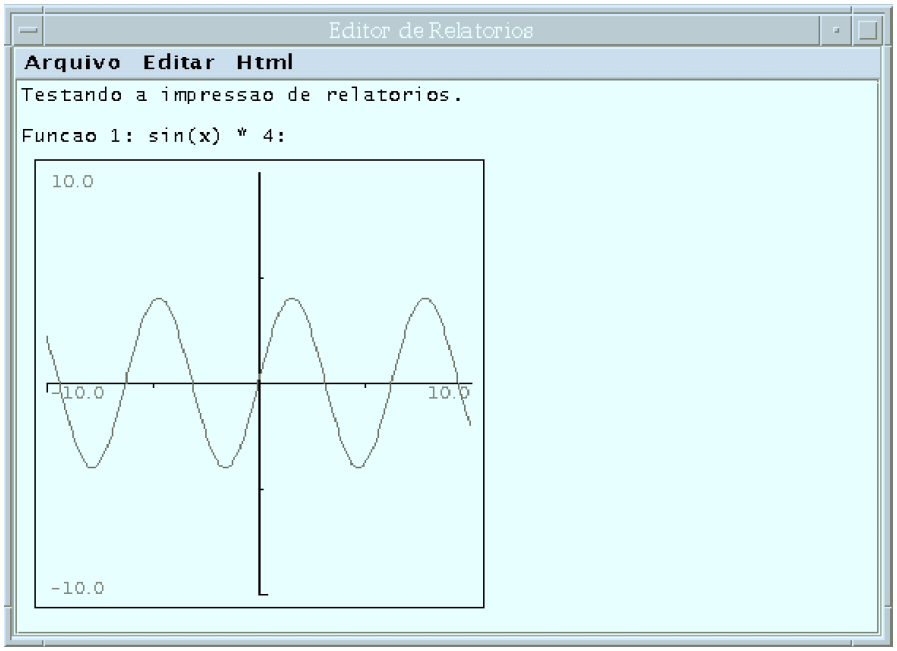

Fig. 4.4 - Exemplo de um relatório simples.

As desvantagens de não se integrar as ferramentas que serão utilizadas num mesmo ambiente, embora contornáveis na maioria das vezes, adicionam uma carga cognitiva desnecessária ao usuário [31]. Optamos então pela implementação do editor de relatórios dentro do ambiente. A Figura 4.4 mostra um relatório que pode ser gerado dentro do ambiente. Além de edição de texto propriamente dita, foi decidido conjuntamente nas reuniões de discussão com os professores que seria interessante os seguintes itens para o editor de relatórios:

1. A inserção de figuras com os gráficos apresentados no *plotter*;
2. A possibilidade de impressão dos relatórios.

Para a inserção de figuras, decidimos por criar um mecanismo que seria mais facilmente implementado. Como não se tem conhecimento prévio sobre a aplicação gráfica colaborativa, não se pode simplesmente abrir um arquivo da aplicação e inserir no relatório (embora soubéssemos que seria implementado um *plotter*, este poderia ser trocado no futuro por outra aplicação gráfica já que este é o objetivo da arquitetura proposta). Além disso, implementar um recurso de *copy and paste* tradicional não é trivial para tipos genéricos.

Resolvemos então incluir na interface da classe principal que implementa o editor de relatórios (classe *Notepad*) um método que possibilita a inserção de componentes visualizáveis

no editor. Criamos então o método *Notepad.insertComponents (JComponent[])*. Para a transferência de imagens do *plotter* para o editor, há uma função no menu do *plotter* chamada **Gerar Pré-Relatório**. Esta função cria componentes com imagens das funções que estão desenhadas no *plotter* (uma com todas as funções na mesma imagem e uma para cada função sozinha) e as submete para o relatório utilizando o método *insertComponents (JComponent[])*. Para conseguir uma referência para o editor de relatório basta usar a classe *ModulesOwner* comentada na seção 4.1.1.

Já a impressão de relatórios deu mais problemas. Pelo fato de Java ser uma linguagem onde os programas executam dentro de uma máquina virtual alguns tipos de acessos ao hardware não são triviais. Na versão utilizada (JDK 1.2.2) o pacote de impressão fornecido (*java.awt.print*) é bastante simplificado. Não existe uma forma de descobrir as impressoras existentes nem a capacidade de cada impressora. Para imprimir algum objeto este deve ser capaz de se desenhar num determinado local, isto é, de colorir ponto a ponto uma região 2D para que o resultado final seja o desenho do objeto. Esta tarefa de desenho é simples, pois as classes que implementam a interface do editor de relatórios já possuem esta funcionalidade (a capacidade de se desenhar numa região 2D). O problema é que, ao ser impresso, mesmo um relatório simples como o da Figura 4.4, fica com um tamanho em torno de 17Mb o que inviabiliza a sua impressão na maioria das impressoras disponíveis hoje no mercado. A nossa solução para contornar este problema foi converter o arquivo de impressão gerado para um arquivo do tipo gif, com isso o tamanho cai para somente 5Kb possibilitando assim a impressão.

4.2.4 Módulo *E-mail*

Assim como o editor de textos havia a possibilidade de se utilizar programas de *e-mail* disponíveis no mercado inclusive como *software* livre. Apesar disto resolvemos colocar o módulo de *e-mail* “dentro” do ambiente. Esta decisão foi tomada por causa das seguintes razões:

- Em primeiro lugar para permitir uma futura integração desta ferramenta de comunicação com a aplicação gráfica colaborativa. Neste caso seria uma integração entre ferramentas de comunicação assíncronas, diferentemente de como estamos fazendo com o bate-papo. Por exemplo, poderíamos enviar um modelo de dados (que poderia ser um conjunto de funções do nosso *plotter*), anexo ao *e-mail*. No

momento de abertura do *e-mail* o modelo também seria aberto automaticamente mas dentro da aplicação gráfica (rodando em modo assíncrono);

- Em segundo lugar o *e-mail* foi implementado dentro do ambiente para que o professor (ou facilitador) possa ter um maior controle das interações que ocorrem entre os alunos. Através de estatísticas, por exemplo, seria possível determinar que embora um determinado grupo X não tenha efetuado nenhuma sessão de bate-papo na realização de uma determinada tarefa este mesmo grupo X trocou diversas mensagens de *e-mail*. Isto poderia dar uma dica ao professor que eles resolveram também em conjunto a tarefa, só que de modo assíncrona.

A primeira razão listada diz respeito à apresentação, implementada na parte cliente do módulo cliente do *e-mail*. Já a segunda diz respeito a certo controle dos e-mails enviados por todos os usuários, o que seria implementado no módulo servidor do *e-mail*. Por isso tanto o servidor como o cliente do e-mail foram implementados "dentro" do nosso ambiente.

4.2.5 Módulo Fórum

O fórum de notícias é um meio de comunicação assíncrono diferente do *e-mail*. As mensagens são compartilhadas por todos os usuários e são ordenadas não por ordem de chegada mas por "*threads*" de discussão. Normalmente uma mensagem que tenha sido enviada como resposta a outra irá aparecer logo abaixo e identada à direita da mensagem original.

Pelos mesmos motivos citados no módulo de *e-mail* resolvemos implementar esta ferramenta "dentro" do CoLab, ao invés de buscar uma solução pronta. Além do comportamento padrão de uma fórum de notícias resolvemos incluir uma funcionalidade não muito comum a este tipo de ferramenta. Acrescentamos aqui a capacidade de um autor de um curso no ambiente poder definir exatamente que conjunto de assuntos que poderão ser usados para definir a mensagem, isto é, no campo do assunto ao invés do aluno escrever um texto qualquer ele deve escolher um assunto de uma lista predefinida. Fuks notou que a categorização das mensagens pode ser útil para "organizar a discussão, aprofundar o debate dos temas e levar os participantes a refletir sobre suas mensagens, resultando numa melhoria no trabalho do grupo e no aprendizado" [32].

4.2.6 Módulo Bate-Papo

O módulo de bate-papo foi o módulo de comunicação verbal (textual) ao qual integramos a nossa aplicação gráfica colaborativa, no nosso caso o traçador de funções matemáticas. Ele é composto por dois *frames* principais (Figura 4.5). Um onde aparecem os nomes de todos os usuários que estão na sala do bate-papo e também onde se escolhe para quem mensagem será dirigida clicando-se no nome da pessoa. Além disso, há a opção de se escolher o destinatário TODOS para que todo mundo receba a mensagem. Normalmente nos bate-papos a lista de nomes aparece dentro de um *combo-box* onde só é possível visualizar o destinatário atual das mensagens. Para que se possa visualizar toda a lista de pessoas na sala, e escolher um destinatário, é preciso apertar o botão do *combo-box*. Achamos que seria melhor implementar usando uma lista pois assim é possível a todo instante ver todos as pessoas que se encontram na sala no momento.

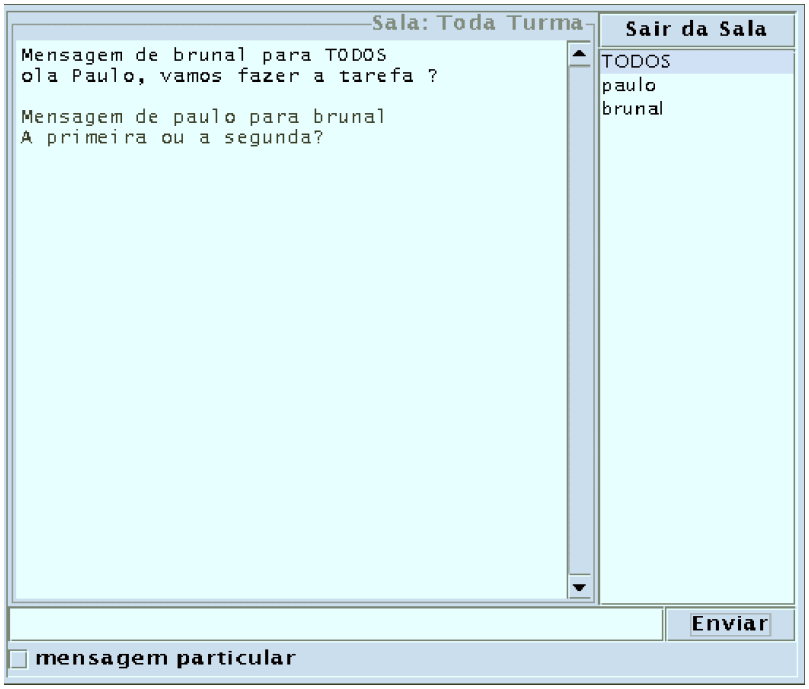

Fig. 4.5 - Módulo de bate-papo.

No outro *frame* deste módulo aparecem as mensagens enviadas por todos os usuários da sala. Há uma única modificação nesta parte da interface do nosso bate-papo em relação ao que normalmente vemos nas ferramentas de bate-papo disponíveis na *internet*. Quando a mensagem

tem como destinatário um usuário específico, ao invés da sala toda, a mensagem aparece na tela deste usuário com uma cor destacada. Assim um aluno percebe facilmente que há uma mensagem dirigida para ele sem ter que ler todos os cabeçalhos das mensagens.

4.3 Módulos Variantes ou Caixa-Branca

Os módulos do CoLab que propõem uma forma de integrar uma ferramenta de comunicação gráfica com uma de comunicação textual, são o *plotter* colaborativo, que é a parte caixa-branca que será descrita aqui, e o bate-papo. Achamos que seria interessante colocar também um *plotter* não-colaborativo. Utilizamos o mesmo modelo de dados e criamos outro módulo de interface, que é o *plotter* individual. Adicionamos mais um método na fábrica de criação do *plotter* colaborativo no cliente para que fosse possível a instanciação deste *plotter* individual.

Como já dissemos, cada um desses módulos foi implementado dentro de um pacote diferente. Há um pacote com as classes do bate-papo, o pacote "*chat*". Um pacote com as interfaces "pais" abstratas das aplicações de comunicação visual, o pacote "*tool*". E, por último, há um pacote com as classes do *plotter*, a nossa ferramenta concreta de comunicação visual, o pacote "*plotter*". Nas figuras 4.6 e 4.7 vemos estes pacotes já com as classes propostas pela arquitetura do Capítulo 3 (Figura 3.5).

4.3.1 O *Plotter*

O *plotter* foi desenvolvido objetivando um curso sobre manipulação de funções através do controle dos parâmetros e composição. Este curso foi desenvolvido inicialmente para aulas presenciais pela professora Sueli Costa (IMECC) e depois adaptado para o ambiente CoLab.

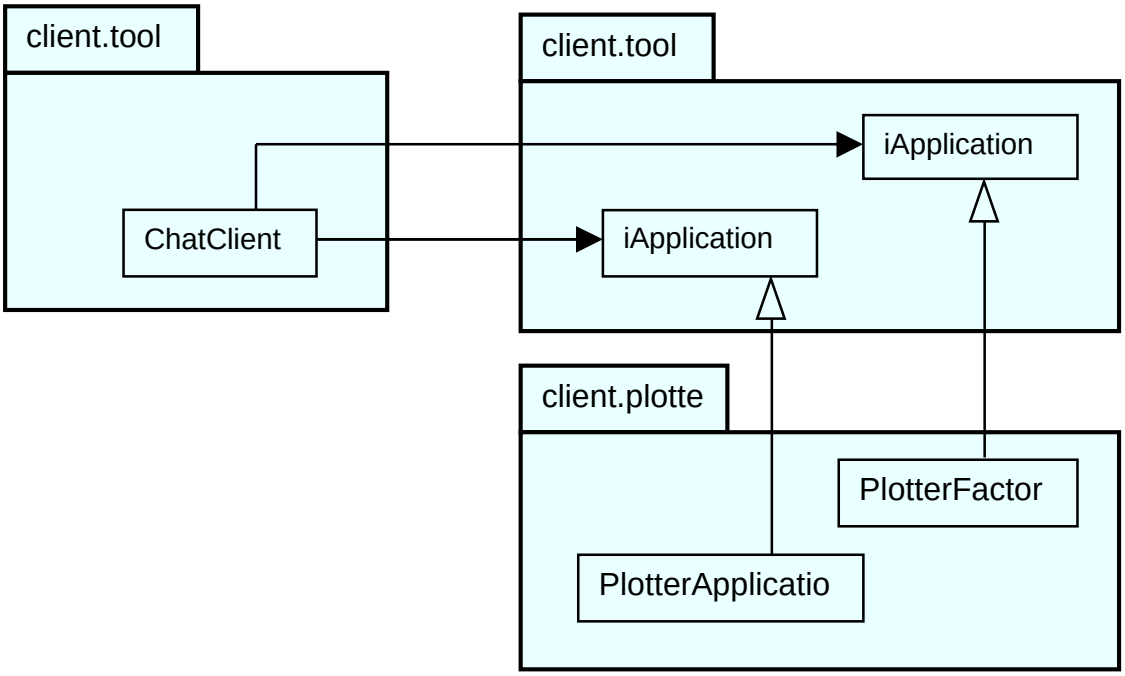

Fig. 4.6 - Pacotes e classes que implementam a nossa arquitetura na parte cliente.

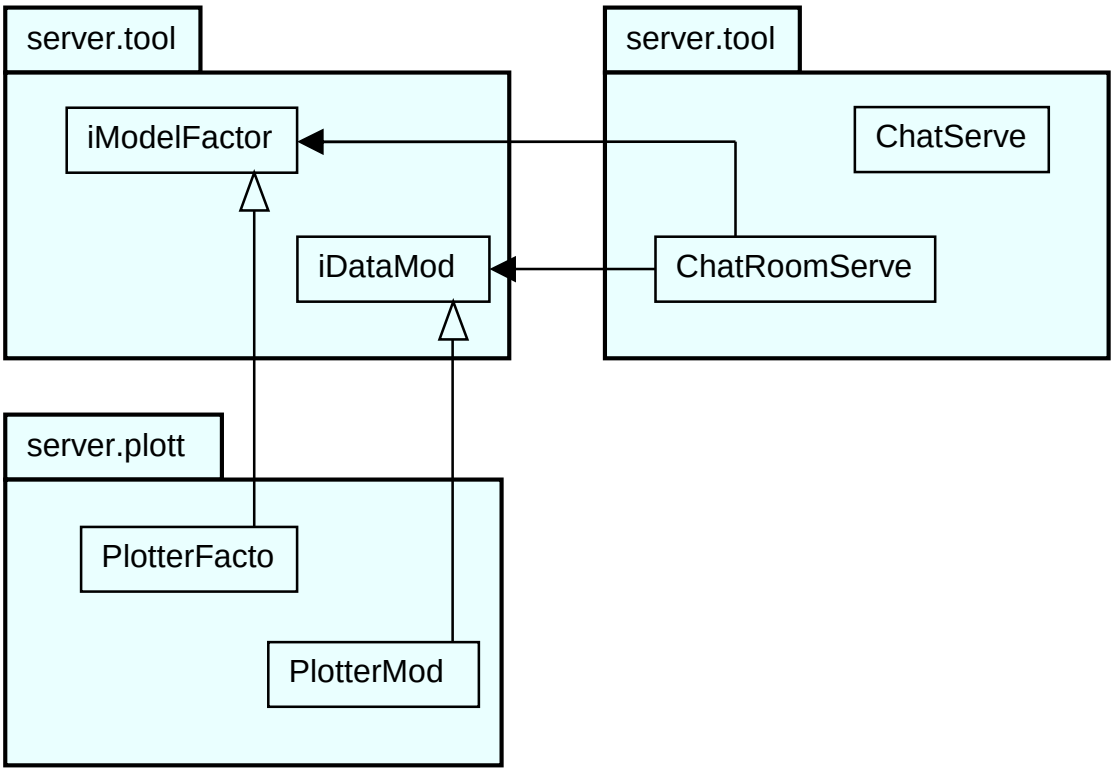

Fig. 4.7 - Pacotes e classes que implementam a nossa arquitetura na parte servidor.

Para diminuir o tempo de desenvolvimento procuramos por um *plotter* pronto na *internet*. Achamos um componente que implementava um conjunto de funções suficiente para os nossos propósitos. Só que a interface deste *plotter* não atendia aos nossos requisitos: só era possível traçar uma função por vez e nós queríamos que fosse possível traçar diversas funções ao mesmo tempo. Além disso, não haviam classes totalmente separadas que implementavam o modelo de dados e interface de apresentação, e isto era necessário para integrar esta ferramenta no nosso ambiente. Mesmo tendo que mudar várias funcionalidades do *plotter* aproveitamos boa parte do código pronto, principalmente a parte que faz a análise léxica e sintática das funções.

Os passos, em linhas gerais, que nós seguimos para adaptar este *plotter* para ser integrado no CoLab serve como uma "receita" para se adaptar e integrar outras ferramentas gráficas para a colaboração visual em outras áreas específicas:

1. Separamos o modelo de dados (o conjunto de funções que seriam desenhadas no *plotter*) numa classe, e a apresentação gráfica em outra. Fizemos com que este modelo implementasse a interface genérica de modelo de dados proposta pela nossa arquitetura (interface *iDataModel*, Figura 3.5);
2. Depois adaptamos a interface de apresentação para as nossas necessidades e fizemos com que ela implementasse a interface *iModelListener* (Figura 3.5);
3. Escrevemos uma classe do tipo fábrica (padrão *Abstract Factory*) para instanciar a classe de apresentação, no cliente, e uma para instanciar o modelo de dados, no servidor;
4. Sobrescrevemos o método *getToolFactory()* das classes *ModulesOwner* (uma no cliente e outra no servidor) para que retornasse a fábrica do *plotter*.

4.3.2 O Controle de Concorrência do Modelo de Dados

Um dos problemas relacionados ao modelo era o controle de concorrência. Queríamos que todos os usuários que estivessem na sala de bate-papo visualizassem as mesmas funções e pudessem inserir e remover funções no *plotter*. Para isso foi preciso implementar um controle de concorrência para acesso ao modelo de dados. Não queríamos que, por exemplo, um usuário apagasse uma função em que outro estivesse trabalhando. Este controle de concorrência deveria

fazer uma associação usuário-função e não permitir que uma determinada função que estivesse associada a um determinado usuário fosse alterada por outro.

Para implementar este controle de concorrência tínhamos duas opções, implementar o controle de concorrência no próprio modelo de dados ou então numa outra classe que ficaria "entre" o modelo de dados e a classe de apresentação do *plotter*, responsável pela interação com o usuário. Como teríamos também um *plotter* não-colaborativo que não necessitaria de controle de concorrência decidimos por implementar a concorrência numa outra classe a parte. Isto também nos permitiria alterar a estratégia de controle de concorrência sem ter que alterar a classe de modelo de dados.

Esta classe de controle de concorrência foi implementada usando-se o *design pattern Proxy*. Este padrão é utilizado, entre outras coisas, quando se deseja usar uma classe que irá controlar o acesso a uma outra [27]. Na Figura 4.8 vemos como ficam os relacionamentos entre as classes. A classe *ProxyModel* primeiro verifica se o cliente pode acessar o método requisitado. Se a permissão for negada ela retorna uma exceção, senão, ela delega a requisição para a classe *RealPlotterModel* e depois recebe a resposta desta classe e a repassa para o cliente.

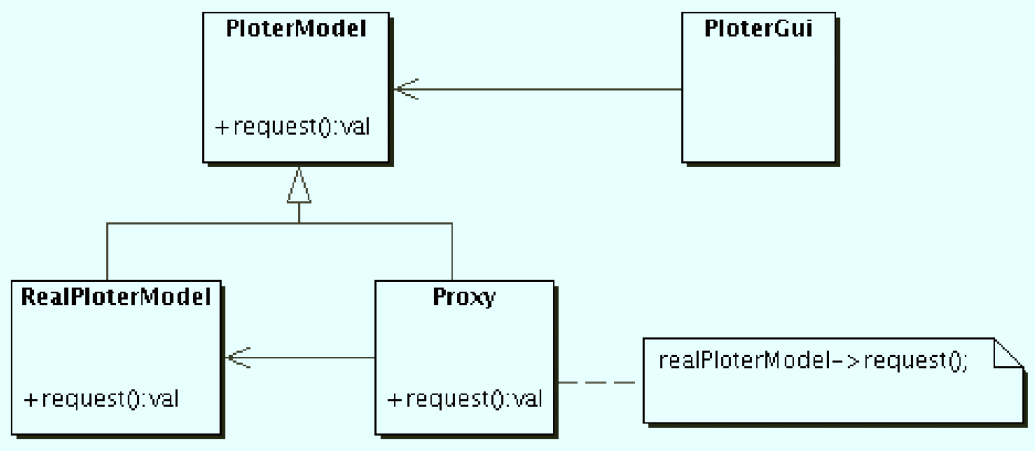

Fig. 4.8 - Padrão *Proxy* implementando o controle de concorrência.

A fábrica do modelo de dados do nosso *plotter* no servidor fica responsável por instanciar a classe *ProxyModel* no instante em que é requisitado um novo modelo de dados (classe *RealPlotterModel*) a ela. A referência para o modelo de dados que a fábrica retorna é na verdade uma referência para a classe *ProxyModel* instanciada e este *proxy* é quem tem a referência para o

modelo de dados real. Assim fica transparente para o cliente que ele está se comunicando apenas indiretamente com o modelo de dados através do *proxy*.

4.3.3 O *Plotter* Colaborativo

Para a apresentação do *plotter* colaborativo, isto é, para a sua interface foi implementada a classe *SharedPlotterPanel* (equivalente à *ConcreteApplication* na Figura 3.5) que representa a aplicação de visualização gráfica. Na Figura 4.9 podemos ver a interface desta aplicação junto com o bate-papo.

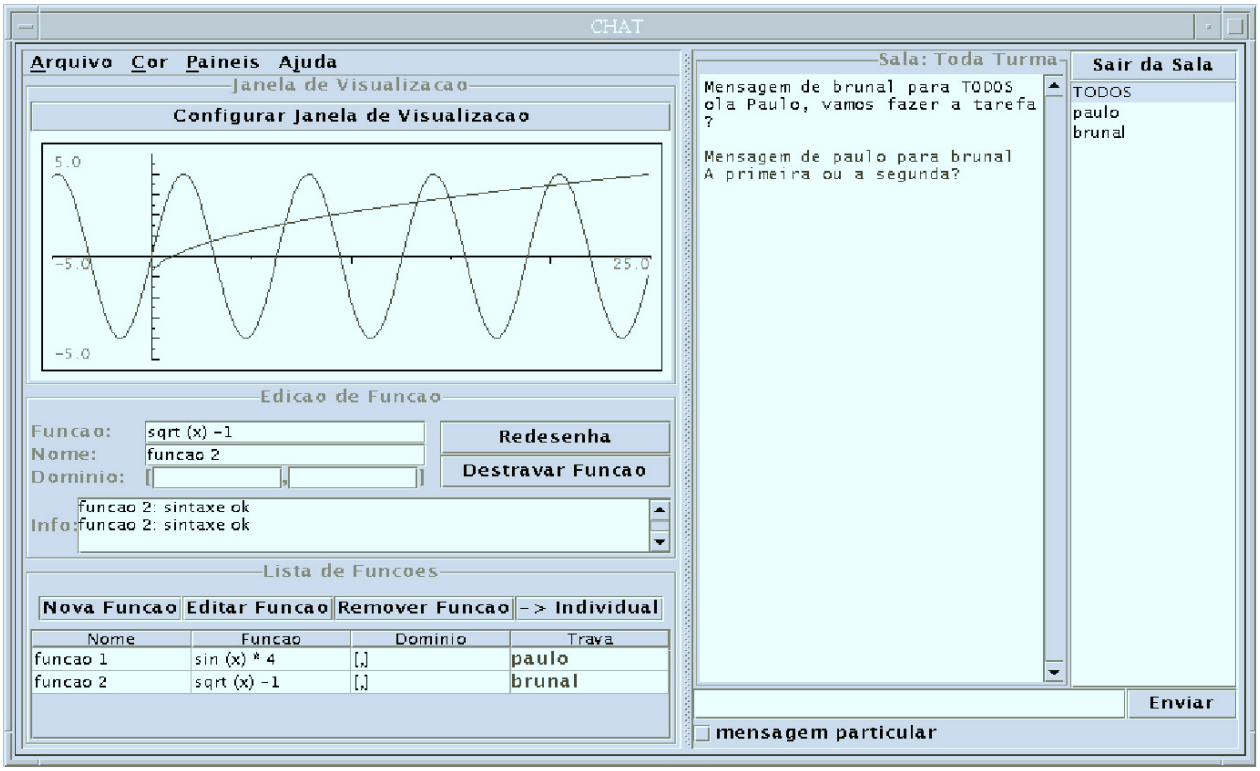

Fig. 4.9 – O *plotter* colaborativo.

O objetivo desta ferramenta é que os alunos possam traçar funções que todos visualizem. A partir desta visualização colaborativa eles poderiam discutir verbalmente (no bate-papo) sobre os exercícios propostos. A linguagem visual traduzida na visualização das funções deveria complementar a linguagem verbal utilizada no bate-papo assim como o uso de um lápis e um

papel para desenhar funções pode complementar uma conversa sobre o comportamento de funções.

Utilizamos de alguns recursos na nossa interface para que os usuários tivessem uma maior consciência de que o trabalho está sendo feito em grupo. Usando as categorias e elementos descritos por [22] que proporcionariam ou aumentariam a consciência de trabalho em grupo criamos algumas dicas visuais. Abaixo listamos as categorias que são utilizadas nesta aplicação para aumentar a consciência do trabalho em grupo:

- **Quem** - Esta categoria representa as dicas visuais para que o usuário saiba quem está trabalhando no espaço colaborativo. Esta dica visual é proporcionada pelo *frame* da janela do bate-papo onde é possível ver todos os usuários que estão na sala e, conseqüentemente, estão também interagindo na aplicação gráfica colaborativa que está integrada ao bate-papo;
- **Onde** - Esta categoria representa as dicas visuais sobre onde cada usuário está agindo no espaço de trabalho compartilhado. Esta categoria é traduzida nesta aplicação para qual função cada usuário está manipulando em cada instante. Cada usuário pode manipular uma (ou nenhuma) função por vez. Esta informação é mostrada na tabela que lista as funções que estão desenhadas. Há um campo nesta tabela com o nome de quem está manipulando cada função. Ao entrar no bate-papo e no traçador pede ao usuário para escolher uma cor que irá lhe caracterizar. Esta cor é então utilizada na linha da tabela que mostra a função que o usuário está trabalhando e também é utilizada para traçar a função no traçador gráfico. Na Figura 4.9 é possível facilmente verificar o que está fazendo o usuário Paulo que foi a última a enviar uma mensagem.

4.3.4 O *Plotter* Individual

Além da aplicação gráfica em modo colaborativo o CoLab espera que também seja implementada a mesma aplicação em modo não colaborativo. Isto não traria muitas dificuldades para o programador da aplicação pois o modelo de dados pode ser o mesmo e a interface de visualização seria igual ou então simplificada em relação à colaborativa.

Na Figura 4.10 podemos ver como ficou o traçador de funções não colaborativo. Como envolve apenas um usuário as cores foram usadas com outro propósito. Aqui, para melhorar a visualização o usuário pode escolher a cor em que cada função será desenhada. Além disso, foi implementada a funcionalidade de se transferir um conjunto de funções de um traçador para o outro, colaborativo e não colaborativo, caso os dois estejam abertos. Assim um usuário pode trabalhar um pouco no seu traçador não colaborativo e depois transferir para o de visualização colaborativa para mostrar o que fez.

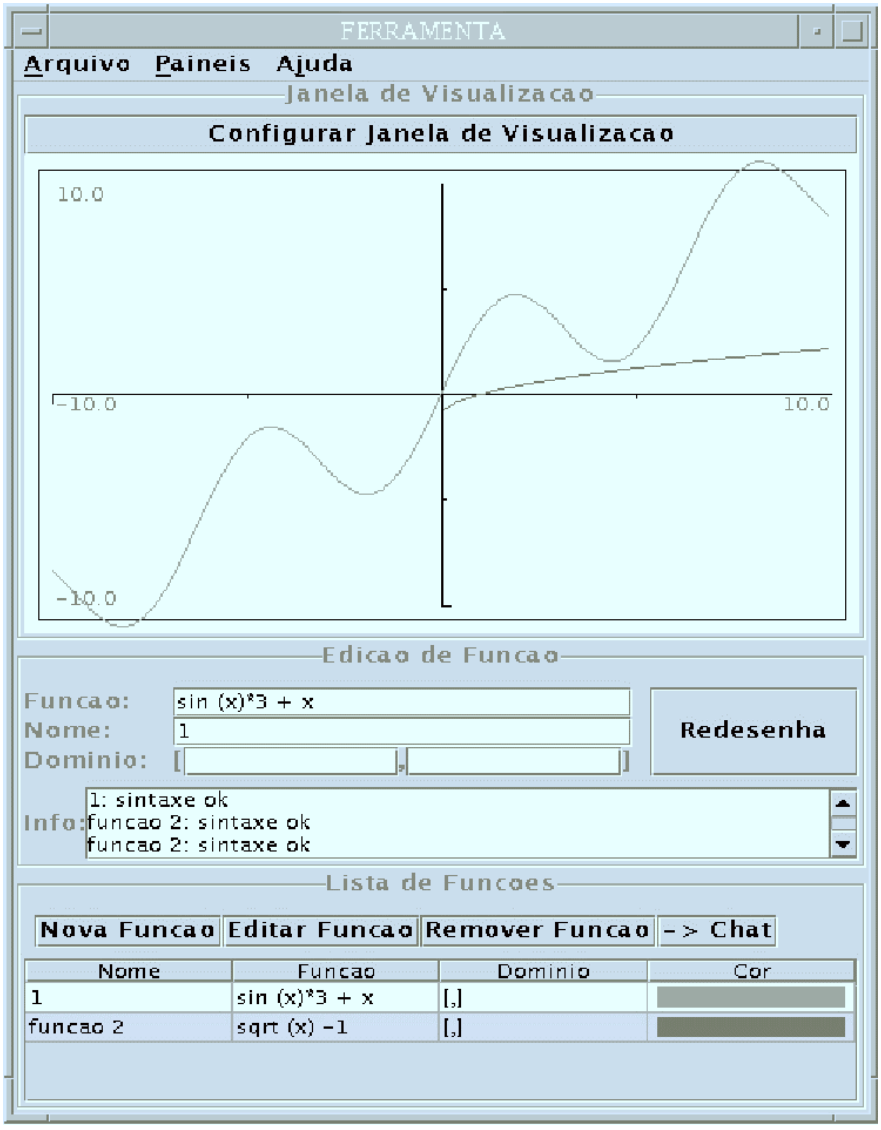

Fig. 4.10 - *Plotter* individual.

Na Figura 4.10 podemos ver a interface do *plotter* não colaborativo. Neste *plotter* as cores são usadas com outra finalidade. Clicando na coluna **cor** na tabela com a lista das funções o usuário pode trocar a cor em que as funções aparecem. Assim é possível uma melhor organização e fica mais fácil "achar" uma função tanto na lista como no desenho.

Este *plotter* individual foi concebido para que seja possível trabalhar com funções sem ter que entrar no bate-papo. Mas pode ser interessante para uma pessoa que esteja no bate-papo usá-lo para, momentaneamente, trabalhar sozinha numa função e depois publicá-la para os outros. Para isso foi desenvolvido um mecanismo para facilitar este chaveamento de funções entre os *plotters*. No painel com a lista de funções (Figura 4.10) há o botão [**chat**]. Clicando-se neste botão a função selecionada é enviada para o *plotter* colaborativo, se este estiver aberto. No *plotter* colaborativo há o botão [**individual**] que faz o contrário, envia uma função do *plotter* colaborativo para o individual.

4.4 Considerações Finais

Neste capítulo foi mostrado o ambiente CoLab. Este ambiente aplica a arquitetura proposta no Capítulo 3, onde se espera que através da integração de ferramentas de comunicação possamos ter um aumento na eficiência da comunicação entre os usuários.

O CoLab foi implementado num modelo cliente-servidor e com diversas ferramentas separadas em módulos. Os módulos foram classificados em caixa-preta e caixa-branca. Os módulos caixa-preta são aqueles que não sofreriam alterações no caso do aproveitamento do ambiente para outras aplicações. Este aproveitamento se daria com o desenvolvimento de novas ferramentas gráficas que substituiriam ou complementariam o módulo caixa-branca, no caso o traçador de funções.

O ambiente mostrado neste capítulo também apresenta algumas características com o intuito de suportar melhor a aprendizagem colaborativa, tais como as "dicas" de interface para aumentar a consciência do trabalho em grupo [22] e o chaveamento entre os traçadores colaborativos e não colaborativos.

Capítulo 5

Experimento

Fizemos um experimento com o objetivo de testar o ambiente e a arquitetura proposta. Nossa idéia principal era ver como o sistema CoLab se comportaria numa ambiente real de uso. O experimento não teve como objetivo verificar se o uso integrado de comunicação visual com verbal traria um suporte melhor ao aprendizado colaborativo. O que foi verificado foi se o sistema realmente possibilitaria uma colaboração onde haveriam duas formas de comunicação se complementando, a verbal (o bate-papo) e a visual, com os recursos gráficos do traçador de funções.

Para executar o experimento, demos um curso que fez parte da matéria de Laboratório de Cálculo de uma turma de primeiro ano de Licenciatura em Matemática da Unicamp. Este curso foi elaborado a partir de um outro curso que já havia sido ministrado mas sem o auxílio do computador.

Nas próximas seções iremos ver primeiro uma descrição do curso que foi planejado incluindo o ambiente físico em que este seria dado e a divisão dos alunos (seção 5.1). Depois, na seção 5.2, veremos o perfil da turma que iria participar do experimento. Na seção 5.3 veremos a descrição do que ocorreu na experiência. E finalmente na seção 5.4 as considerações finais.

5.1 O Curso

O curso foi elaborado para o aprendizado sobre a modificação do comportamento de funções através da manipulação de parâmetros e através da composição de funções. Este curso deveria ter duração de três aulas e foi inserido na disciplina de Laboratório de Cálculo de uma turma de Licenciatura de Matemática.

5.1.1 O Conteúdo do Curso

Este curso foi desenvolvido e primeiramente administrado em sala de aula (sem o uso de computadores) pela Professora Sueli Costa. Devido à estrutura do curso e a seu conteúdo achamos que seria possível utilizá-lo como experimento do sistema. O curso foi então adaptado, com a ajuda da Professora Sueli e do Professor Marcelo, para ser ministrado no ambiente CoLab.

O curso foi planejado para que o aprendizado ocorresse através da resolução de exercícios. Dividimos então o curso em três aulas:

1. Na primeira aula seriam feitas experiências com funções simples. Basicamente foi trabalhada a função **y = sen(x)**. Foi pedido que eles variassem os parâmetros **A**, **B**, **C** e **D**, com a função na forma **y = Asen(Bx + C) + D**, um parâmetro de cada vez. Visualizando o resultado da manipulação destes parâmetros, isto é, da variação destes parâmetros, eles deveriam perceber como o comportamento da função variava de acordo com conceitos como periodicidade e período, amplitude, freqüência, máximos e mínimos, positivo e negativo, simetria e regiões de crescimento e decrescimento;
2. Na segunda aula, pedimos que eles trabalhassem com os mesmos conceitos mas desta vez utilizando funções mais complexas. Pedimos que eles trabalhassem com funções compostas como **y = f(x) * g(x)** e **y = f(g(x)),** onde **f(x)** e **g(x)** são por sua vez funções como **x*x**, **sen(x)**, **sqrt(x)** e **1/x**;
3. Na terceira e última aula, os alunos deveriam utilizar o conhecimento adquirido para desenhar uma máscara no *plotter*. Apenas utilizando funções matemáticas, eles deveriam, através da manipulação dos parâmetros, conseguir o desenho de um rosto ou de uma máscara na tela do *plotter*.

Normalmente os cursos de funções partem de uma visão sintética do assunto, onde são estudados os conceitos teóricos sobre o comportamento de funções em termos dos seus parâmetros. Por exemplo, estudando o comportamento da função **y = sen (A*x)** o professor, normalmente com o uso de gráficos, “passa” o conhecimento de que o parâmetro **A** irá controlar a freqüência e o período da função.

Já a idéia principal por trás do nosso curso é que os alunos partam da visão analítica para uma visão sintética. Através da observação e da comparação dos gráficos gerados com parâmetros diferentes para as funções, eles compreendam os conceitos e sejam capazes de sintetizá-los, construindo funções que, no último exercício, lhes permitirão elaborar o desenho desejado (no caso a careta). Queremos com isso, tirar o máximo de proveito do uso da ferramenta gráfica neste ambiente de ensino já que a capacidade de análise comparativa das funções só é possível neste contexto através do seu gráfico no *plotter*.

5.1.2 O Ambiente Físico do Experimento

Os alunos de Licenciatura de Matemática tinham dois laboratórios de computação à sua disposição para a matéria de Laboratório de Cálculo. Isso nos permitiria dispor dois alunos em cada máquina. Decidimos criar grupos de quatro alunos sendo que eles ficariam separados, dois em cada computador, e estes computadores ficariam em salas diferentes. Queríamos que qualquer comunicação entre cada par do mesmo grupo ocorresse apenas através do sistema, essa comunicação seria predominantemente síncrona, através do bate-papo e do *plotter* colaborativo. Na **Fig. 5.1** vemos um esquema de como os alunos estariam separados. Em cada uma das salas ficaria um monitor para auxiliar os grupos.

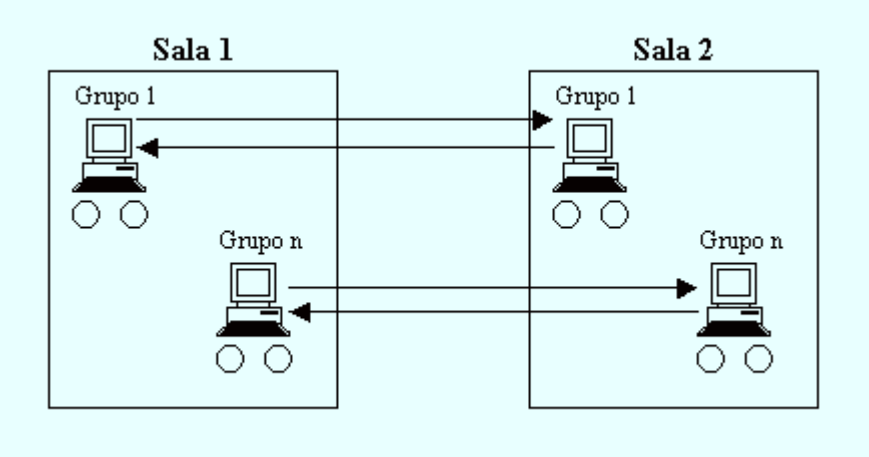

Fig. 5.1 - Ambiente idealizado para o experimento.

Embora este tenha sido o ambiente físico idealizado, uma semana antes do experimento ocorreu um imprevisto e um dos laboratórios não pode ser utilizado. Com isso foi necessário

modificar o esquema planejado. Tínhamos que o ideal seria o menor número possível de alunos por máquina, para que a comunicação entre eles ocorresse somente pelo CoLab. Como este experimento faria parte de uma matéria normal dos alunos envolvidos, não poderíamos dispensar nenhum aluno para termos um número ideal de alunos por máquina, todos tinham que participar do curso. Com isso tivemos que colocar três alunos em cada computador e todos na mesma sala. Na **Fig. 5.2** vemos o esquema da disposição física dos alunos na única sala disponível. O que tentamos fazer foi que alunos de um mesmo grupo não ocupassem computadores próximos um do outro. Como veremos na seção 5.3 este problema com a sala modificou o comportamento esperado do experimento com o CoLab.

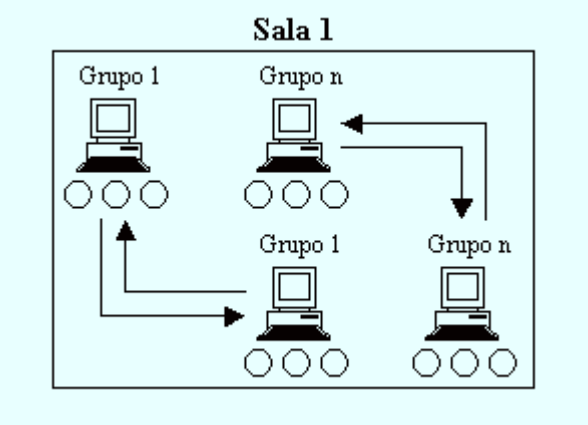

Fig. 5.2 - Ambiente real do experimento.

5.2 O Perfil dos Alunos

A turma que realizou o experimento foi uma turma de primeiro ano de Licenciatura em Matemática que contava com 39 alunos. Por ser um curso para a formação de professores e pelo fato de vários alunos desta turma já trabalharem realmente como professores de matemática esperávamos que eles pudessem analisar o ambiente não apenas com a visão de alunos usando uma ferramenta colaborativa, mas esperávamos também uma realimentação (*feedback*) dos alunos que já eram professores sobre a viabilidade de uso deste tipo de ferramenta colaborativa num ambiente de ensino.

Outra característica relevante é que nesta disciplina de Laboratório de Cálculo I, os alunos já estavam fazendo experiências com distintos *softwares* dedicados à Matemática. Embora

nenhum deles fosse voltado para o trabalho em grupo achamos que isto também poderia aumentar a visão crítica deles sobre o CoLab.

5.3 Descrição da Experiência

No primeiro dia da experiência ocorreram alguns problemas com o *software* CoLab. Estes problemas foram decorrentes da falta de uma metodologia adequada de testes do *software*. Eles só apareceram quando houvesse várias pessoas utilizando o bate-papo com o *plotter* colaborativo e após um determinado tempo de uso. Os testes que foram efetuados antes de experimento foram sempre de curta duração e com poucas pessoas utilizando o sistema. Estes erros não impediram totalmente o uso do sistema, mas a maioria dos grupos teve que sair e retornar ao programa diversas vezes. Estas interrupções atrapalharam o andamento das tarefas do primeiro dia de aula, principalmente a partir da metade da aula. Enquanto no começo da aula os alunos estavam achando “divertido” utilizar o sistema, ao final da aula eles já estavam, em sua maioria, cansados e reclamando do comportamento do ambiente. Para a segunda e terceira aulas estes problemas já estavam todos solucionados.

Além deste problema, houve o problema do espaço físico da sala. As tarefas que foram passadas para eles pediam explicitamente que eles fizessem os trabalhos entre todos os integrantes do grupo, com conversas através do bate-papo, entre os que se encontravam em computadores diferentes, para a resolução do exercício. Mas, o que pudemos perceber, foi que quando eles queriam discutir alguma coisa o faziam entre os três alunos (em alguns casos dois) que estavam sentados juntos no mesmo computador. A conversa face a face é muito mais fácil e rápida do que a comunicação no bate-papo.

A princípio, achamos que o fato dos alunos não usarem o bate-papo junto com o *plotter* para comunicação impossibilitaria a verificação da utilidade da integração de ferramentas de comunicação verbal e gráfica. Mas pudemos observar que na resolução dos exercícios eles discutiam entre si (os alunos que estavam sentados no mesmo computador) verbalmente e para demonstrar melhor suas idéias utilizavam o *plotter* para complementar com a linguagem visual as idéias que estavam expondo com a fala. Pudemos perceber então que havia um ganho na comunicação com o uso de mídias diferentes de comunicação.

5.3.1 Avaliação

A primeira avaliação do experimento foi feita com o último exercício pedido. Para fazer uma careta com o uso de funções matemáticas, eles deveriam ter adquirido a capacidade de síntese de funções através das atividades de análise das aulas anteriores. Na **Fig. 5.3** vemos algumas das caretas geradas pelos grupos e podemos perceber que este objetivo foi atingido.

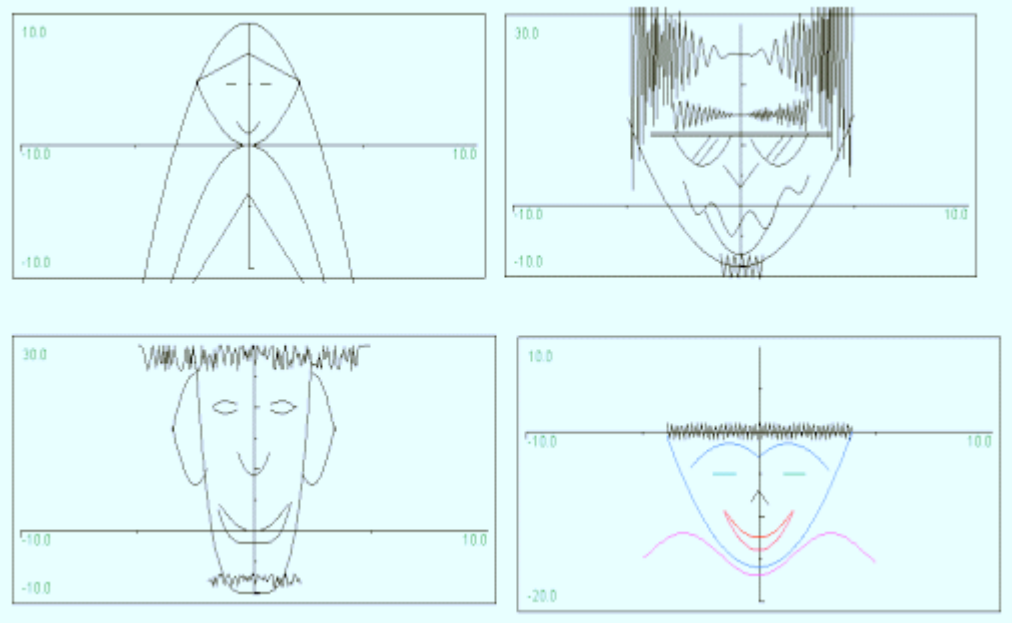

Fig. 5.3 - Caretas feitas pelos alunos na última atividade do curso.

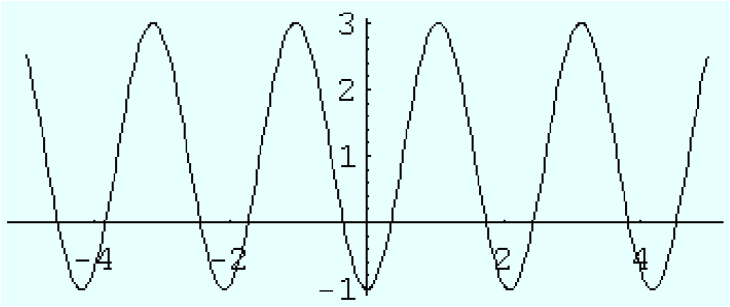

Fig. 5.4 - Gráfico da primeira questão do teste aplicado.

Para termos mais dados para análise do experimento, pedimos que, uma aula antes deles começarem o curso no CoLab, fizessem um teste que iria medir o conhecimento prévio deles sobre o comportamento de funções em razão de seus parâmetros. Uma aula após os três dias de

atividades no CoLab repetimos este teste (mudando alguns valores). Embora este tipo de avaliação não forneça subsídios conclusivos para verificar que o uso do ambiente proporciona uma melhoria em termos de aprendizado pode-se tirar algumas observações a partir da resposta dos alunos. O teste era composto por duas questões. Abaixo vemos a primeira questão do teste:

> O gráfico da Fig. 5.4 representa uma função da forma **y = Asen(Bx + C) + D**. O que você pode dizer (*) sobre os valores de A, B, C e D?
>
> (*) As afirmações podem ser, por exemplo, da forma A=4, ou A maior que 4, ou A positivo, etc.

No primeiro teste tivemos a presença de 34 alunos e no segundo 32. Considerando cada uma das quatro variáveis (A, B, C e D) como um item de igual valor comparativo e o número de alunos em cada teste, tivemos 136 (34 alunos) respostas para o primeiro teste (antes do curso com o CoLab) e 128 respostas para o segundo teste (depois do curso com o CoLab) na primeira questão.

Das análises que fizemos, os resultados que deram diferenças mais significativas entre um teste e outro foram:

- Das 134 respostas esperadas no primeiro teste 34% ou 46 itens foram em branco enquanto que no segundo teste apenas 6% ou 8 itens. Podemos dizer que o experimento deu subsídios para que os alunos ao menos tentassem resolver as questões. Seria interessante repetir esta análise com um teste efetuado uns seis meses depois deste experimento para verificar se ele serviu para fixar realmente o conteúdo;
- Verificamos o número de respostas conceituais para cada item da primeira questão. Consideramos que uma resposta conceitual quando o aluno respondia o que a variável significava no comportamento da função, isto é, se a variável representava a amplitude, o período (ou freqüência) e a translação horizontal e vertical. Enquanto no primeiro teste apenas 6% das respostas foram do tipo conceitual com 50% de acertos), no segundo teste tivemos 26% de respostas conceituais (com 82% de acertos).

A segunda questão do teste pedia que os alunos relacionassem uma lista com sete funções com sete gráficos diferentes. Nesta questão não houve diferença significativa entre os dois testes. No primeiro teste, antes do curso, houve uma média de acertos de 3,5 (de um máximo de 7,0) para funções relacionadas corretamente enquanto no segundo teste a média foi 3,75.

Por fim, além dos testes, passamos um questionário com perguntas gerais sobre o CoLab. Entre outras coisas perguntamos sobre os aspectos positivos e negativos do CoLab e perguntamos a opinião deles sobre o experimento com o CoLab ter atingido o objetivo de ensinar o "efeito da variação de parâmetros e composição em funções de uma variável". Abaixo algumas questões do questionário com algumas de suas respectivas respostas.

Na sua opinião quais foram os aspectos negativos do CoLab?

Vários alunos reclamaram da dificuldade de realizar o trabalho devido aos "travamentos" do programa:

> *As eventuais falhas no programa, algumas vezes impossibilitando de utilizar o CoLab*. (Roberto).

> *Programa pesado (trava muito).* (Carlos Eduardo).

Além dos erros existentes no sistema e que ocasionaram problemas no primeiro dia de uso, algumas máquinas com pouca memória (16Mb) tiveram problemas em rodar o CoLab. Este problema acontece porque aplicativos com interface gráfica interativa implementados com Java requerem uma quantidade maior de memória. Nos nossos testes o CoLab executou sem problemas em máquinas com 32Mb de memória.

Diversos alunos deram sugestões quanto à interface do programa, principalmente quanto à interface do *plotter*:

> *Pela quantidade de funções, às vezes, editadas, tornava-se difícil visualizar os gráficos. Sugestão: para não perder informações, seria interessante um*

> *tipo de caixa de seleção para escolher quais funções você quer analisar naquele momento (sem precisar deletá-las).* (Elaine).

A interface foi desenvolvida com o auxílio dos professores de Matemática mas é imprescindível coletar informações com os reais usuários do sistema, neste caso os alunos. No desenvolvimento de novas aplicações gráficas a serem integradas ao CoLab poder-se-ia tentar o *design* centrado no usuário [20] para conseguir-se uma interface mais amigável.

Na sua opinião quais foram os aspectos positivos do CoLab?

Apesar de terem trabalhado pouco utilizando a troca de informações via computador eles acharam este ponto positivo:

> *A facilidade de trocar informações*. (Elaine).

> *Em grupo de trabalho e possibilidade de verificar o que os demais integrantes estão fazendo.* (Patrícia).

> *A comunicação com outras máquinas, a possibilidade de socializar os gráficos e informações.* (Giancarlo).

Provavelmente eles fizeram estas afirmações ao comparar o CoLab com os diversos outros *softwares* que eles usaram nesta matéria, como por exemplo, MPP, SLogo, Cabri Geometric e Mathematica.

Apesar de alguns alunos darem sugestões quanto à interface, diversos outros a elogiaram principalmente por possuir recursos que os outros programas que eles estavam acostumados a utilizar não possuíam:

> *Um aspecto positivo eu senti quando fizemos uma atividade para comparar gráficos de funções. Eu posso colocar várias funções ao mesmo tempo e comparar*. (Janile).

A possibilidade de escolher o domínio da função. (Emília).

A maioria dos programas que eles utilizavam não oferecia uma interface amigável que possibilitasse alterar de modo eficiente as funções a serem traçadas. Outro ponto positivo destacado por um aluno, numa conversa, foi o fato do editor de relatórios ser integrado ao sistema, pois eles tinham dificuldades de gerar relatórios com gráficos feitos em outros programas.

5.4 Considerações Finais

Embora o experimento não tenha saído como o esperado acreditamos que houve a integração de meios de comunicação. A conversa face a face substituiu a comunicação verbal do bate-papo mas não substituiu a comunicação visual possibilitada pelos recursos gráficos do *plotter*.

Avaliamos como satisfatório o experimento realizado no sentido de testar a arquitetura do *software* num ambiente próximo ao real. Tecnicamente ela se mostrou funcional e viável.

Capítulo 6

Considerações Finais e Trabalhos Futuros

O objetivo primeiro deste trabalho era explorar a potencialidade de recursos gráficos e comunicação visual em ambientes colaborativos. Para isso foi necessário propor uma arquitetura que permitisse a integração de ferramentas de comunicação através da linguagem visual e da linguagem verbal. As ferramentas de comunicação verbal, principalmente as textuais, são comuns nos ambientes de colaboração diferentemente das ferramentas que possibilitassem a comunicação visual. Apesar disto, a comunicação visual é extremamente utilizada em contextos específicos a determinadas áreas do conhecimento.

Propusemos então uma arquitetura que permitia esta integração (Capítulo 3) e para testar a viabilidade desta arquitetura construímos então um ambiente de aprendizado colaborativo, o CoLab, e uma aplicação de colaboração visual específica para o ensino de funções de uma variável (Capítulo 4). Além disso, fizemos um experimento para verificar como se comportaria a arquitetura e se a aplicação gráfica teria a utilidade planejada de complementar a linguagem verbal com o uso da linguagem visual (Capítulo 5).

Através dos resultados obtidos acreditamos que a arquitetura realmente possibilitou a integração de diferentes formas de comunicação. O experimento realizado tinha como propósito validar o ambiente construído, e por conseguinte a arquitetura proposta, e também verificar se seria realmente útil a complementação das linguagens verbal (genérica) e visual (específica a um contexto). Apesar do experimento não ter saído como o planejado verificamos durante o experimento que os alunos utilizavam o traçador gráfico justamente com este propósito, o de complementar as suas idéias e opiniões as quais eram expressadas verbalmente. Mas como não foi utilizado a ferramenta de comunicação verbal do ambiente, o bate-papo, acreditamos que seria interessante fazermos novos experimentos, preferencialmente a distância, para avaliarmos melhor esta questão.

6.1 Trabalhos Futuros

Uma das nossas intenções com a arquitetura proposta é que ela fosse simples para que ambientes de colaboração já construídos pudessem ser adaptados para utilizá-la. Dada esta hipótese, presente na seção 3.3, seria interessante, como trabalho futuro, tentar aplicar esta arquitetura em algum outro ambiente de colaboração existente. Uma possibilidade seria utilizar o TelEduc para este fim. O fato dele ser distribuído como *software* livre sob a licença GPL (*General Public License*) seria um facilitador. Além de testar esta hipótese, nós também estaríamos aplicando esta arquitetura de integração num ambiente mais "completo" que o CoLab, já que o TelEduc contém diversas outras ferramentas interessantes para um ambiente de aprendizagem colaborativa, o que nos daria um produto mais elaborado para futuros experimentos.

Como uma segunda idéia para um trabalho futuro, gostaríamos de verificar, num experimento educacional, se a integração de diferentes formas de comunicação traria proveitos à aprendizagem num ambiente colaborativo. Afinal, uma das motivações deste trabalho foi perceber que os sistemas atuais de aprendizagem colaborativa negligenciam outras formas de colaboração que não a verbal. Este experimento poderia utilizar o *plotter* construído durante este tese ou alguma outra ferramenta gráfica de colaboração. Seria interessante que este experimento fosse conduzido num curso a distância para que não se repita o problema que tivemos neste trabalho quanto ao espaço físico. Deverá ser pesquisado qual a melhor metodologia a ser aplicada na condução e validação deste experimento.

E, por último, estender esta arquitetura para integrar também ferramentas de comunicação assíncrona, as quais também são de suma importância para a colaboração.

Referências Bibliográficas

[1] UVB. Universidade virtual brasileira. http://www.iuvb.edu.br/br. (Acesso em 20 de março de 2002).

[2] Unirede. http://www.unirede.br/. (Acesso em 20 de março de 2002).

[3] A. Jones e N. Mercer. Theories of learning and information technology. In Peter Scrimshaw, editor, Language, Classrooms & Computers. Routledge, 1993.

[4] L. S. Vygotsky. Pensamento e Linguagem. Martins Fontes, 1991.

[5] G. Salomon, D. Perkins e T. Globerson. Partners in cognition: Extending human intelligence with intelligent technologies. Educational Researcher, 20(3):2–9, 1991.

[6] Jy Wana Daphne Lin Hsiao. Cscl theories. http://www.edb.utexas.edu/csclstudent/Dhsiao/theories.html. (Acesso em 20 de março de 2002).

[7] Teleduc. http://hera.nied.unicamp.br/teleduc/. (Acesso em 20 de março de 2002).

[8] Heloisa Rocha. O ambiente TelEduc para educação a distância baseada na web: Princípios, funcionalidades e perspectivas de desenvolvimento. In M. C. MORAES, editor, Educação a distância: Fundamentos e práticas. Unicamp/NIED, 2002.

[9] WebCT. http://www.webct.com. (Acesso em 20 de março de 2002).

[10] M. Goldberg, S. Salari e P. Swoboda. World Wide Web - Course tool: An environment for building WWW-based courses. Computer Networks and ISDN Systems, Vol.28, n.7, 1996.

[11] Aulanet. http://www.aulanet.com.br. (Acesso em 20 de março de 2002).

[12] H. Fuks, M. A. Gerosa e C. J. P. Lucena. Sobre o desenvolvimento e aplicação de cursos totalmente a distância na *internet*. Revista Brasileira de Informática na Educação, (9), 2001.

[13] M. K. Singley, P. G. Fairweather e S. Swerling. Team tutoring systems: Reifying roles in problem solving. In Proceedings of the Computer Support for Collaborative Learning (CSCL). Mahwah, NJ: Lawrence Erlbaum Associates, December 1999.

[14] C. Colwell, E. Scanlon e M. Cooper. Using remote laboratories to extend access to science and engineering. Computers & Education, 38(1-3):65–76, January-April 2002.

[15] Michael Schrage. No More Teams! Mastering the Dynamics of Creative Collaboration. Currency Doubleday, 1995.

[16] Klaas Sikkel. Groupware technology and software reuse. European Reuse Workshop (ERW'98), Madrid, November 1998. position papers vol. II: 45-48.

[17] M. Roseman e S. Greenberg. Building real-time groupware with groupkit. ACM Transactions on Computer-Human Interaction, 3(1):66–106, March 1996.

[18] S. Greenberg e M. Roseman. Groupware toolkits for synchronous work. In Michel Beaudouin-Lafon, editor, Trends in CSCW. John Wiley & Sons Ltd, 1996.

[19] P. Jahn. Getting started with share-kit. ftp://ftp.cs.tu-berlin.de/pub/local/kbs/share-kit/share-kit.tar.gz. (Acesso em 20 de março de 2002).

[20] Donald Norman. Cognitive engineering. In Donald Norman and Stephen Draper, editors, User Centered System Design: New Perspectives on Human-Computer Interaction, pages 31–61. Hillsdale, NJ: Erlbaum Associates, 1986.

[21] A. Townsend, A. Hendrickson e S. DeMarie. Meeting the virtual work imperative. Communications of the ACM, 45(1):23–26, 2002.

[22] C. Gutwin e S. Greenberg. The effects of workspace awareness support on the usability of real-time distributed groupware. ACM Transactions on Computer-Human Interaction, 6(3):243–281, September 1999.

[23] J. Nielsen. Usability Engineering. Academic Press Prof., Inc., San Diego, CA, 1993.

[24] Paivio, A. Mental Representation: A Dual Code Approach. Oxford University Press, 1986.

[25] R. Mayer e V. Sims. For Whom Is a Picture Worth a Thousand Words? Extensions of a Dual-Coding Theory of Multimedia Learning. Journal of Educational Psychology, 86(3): 389-401, 1994.

[26] Christopher Alexander et al. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977.

[27] Erich Gamma et al. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[28] C. Lucena et al. AulaNet – an environment for the development and maintenance of courses on the Web. Proceedings of the International Conference on Engineering in Education, 1998.

[29] Netmeeting. http://www.microsoft.com/windows/netmeeting/default.asp. (Acesso em 20 de março de 2002).

[30] Joseph O'Neil. JavaBeans Programming from the Ground Up. Osborne/Mc-Graw Hill, 1998.

[31] M. Mandviwalla e S. Khan. Collaborative object workspaces (cows): exploring the integration of collaboration technology. Decision Support Systems, 27(3):241–254, 1999.

[32] H. Fuks, M. A. Gerosa e C. J. P. Lucena. Usando a categorização e estruturação de mensagens textuais em cursos pelo ambiente aulanet. Revista Brasileira de Informática na Educação, (10), 2002.