UMA ABORDAGEM PARA O DESENVOLVIMENTO DE APOIO À PERCEPÇÃO EM AMBIENTES COLABORATIVOS DE DESENVOLVIMENTO DE SOFTWARE

Marco Aurélio Souza Mangan

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profa. Cláudia Maria Lima Werner, D.Sc.

Prof. Marcos Roberto da Silva Borges, Ph.D.

Prof. Jano Moreira de Souza, Ph.D.

Prof. Hugo Fuks, Ph.D.

Profa. Karin Becker, Ph.D.

Profa. Maria da Graça Campos Pimentel, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

FEVEREIRO DE 2006

MANGAN, MARCO AURÉLIO SOUZA

Uma Abordagem para o Desenvolvimento de Apoio à Percepção em Ambientes Colaborativos de Desenvolvimento de Software [Rio de Janeiro] 2006

XI, 213 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2006)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Desenvolvimento de Software

2. Percepção em Trabalho Cooperativo

I. COPPE/UFRJ II. Título ( série )

# Agradecimentos

Gostaria de agradecer a todos que acompanharam ou contribuíram para esta tese. Mesmo correndo o risco de esquecer muitos nomes, aqui vai uma lista desordenada de pessoas que também são responsáveis por esta tese e por este doutor.

Aos orientadores Cláudia e Marcos, meu eterno débito pela muitas horas de trabalho, contribuições e pela paciência.

Aos membros da banca examinadora e aos professores das disciplinas: Jano, Guilherme, Cláudia, Marcos, Renata, Flávia e Marta, meu agradecimento especial pela contribuição à minha formação.

Aos colegas que tive a sorte de trabalhar em conjunto: Leo Murta, Camila, Luciana, Vaninha, Daniele, Renata, Marcos Kalinowski, Bernardo, Diego, Carlos Roberto, Melfry, Isabella e Lopes, agradeço pela ajuda em inúmeras tarefas e pela paciência em escutar meus comentários e tentar ler as anotações nos meus caderninhos.

Aos colegas do grupo de reutilização Márcio Barros, Robson, Aline, Ana Paula, Regina, Regiane, Cristine e Alexandre Dantas, Correa, Gustavo, Hugo, Xavier e Hamilton, meu agradecimento pelas muitas horas de convivência que tornam nosso trabalho menos penoso.

À Carol, Patrícia, Solange e demais funcionários que apóiam nossas atividades, muito obrigado.

Ao Programa de Engenharia de Sistemas e Computação, obrigado pelo apoio financeiro que permitiu a participação em conferências nacionais e internacionais.

À CAPES e ao CNPq, pelo apoio financeiro no âmbito dos projetos PROCAD.

À Pontifícia Universidade Católica do Rio Grande do Sul, por investir na qualificação de seus professores.

Finalmente, gostaria de agradecer aos meus pais, esposa, família e amigos pelo incentivo e pela compreensão das ausências, das alterações de humor e de outras manifestações de comportamento anti-social causadas por esta tese ao longo destes anos.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

# UMA ABORDAGEM PARA O DESENVOLVIMENTO DE APOIO À PERCEPÇÃO EM AMBIENTES COLABORATIVOS DE DESENVOLVIMENTO DE SOFTWARE

Marco Aurélio Souza Mangan

Fevereiro/2006

Orientadores: Cláudia Maria Lima Werner
Marcos Roberto da Silva Borges

Programa: Engenharia de Sistemas e Computação

Esta tese apresenta uma abordagem para o desenvolvimento de apoio à percepção em ambientes colaborativos de desenvolvimento de software que adota a estratégia de alteração de um ambiente existente, com ênfase na construção e reutilização de componentes de software. Um modelo de computação distribuída é aplicado para armazenar e distribuir as informações de percepção coletadas nas aplicações. A integração das aplicações ocorre de forma transparente tanto para o programador do ambiente quanto para o usuário dos serviços de percepção. Foram construídos e avaliados cinco protótipos que serviram tanto como prova de viabilidade quanto para refinar as idéias preliminares sobre a abordagem proposta.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

AN APPROACH TO THE DEVELOPMENT OF AWARENESS SUPPORT IN SOFTWARE DEVELOPMENT COLLABORATIVE ENVIRONMENTS

Marco Aurélio Souza Mangan

February/2006

Advisors: Cláudia Maria Lima Werner
Marcos Roberto da Silva Borges

Department: Computer and System Engineering

This thesis presents an approach to the development of awareness support in software development collaborative environments using the strategy of modification of an already existing development environment, with emphasis on the construction and reuse of software components. A distributed computational model is applied to store and distribute awareness information collected from the applications. Application integration is transparent both to the development environment programmer as to the awareness services user. Five prototypes where built and evaluated in order to demonstrate the feasibility of the approach and refine preliminary ideas.

# Índice

# Índice de Figuras

# Índice de Tabelas

# 1 Introdução

Diversos trabalhos vêm propondo o apoio à percepção aplicado ao desenvolvimento colaborativo de software (FARSHCHIAN, 2001, SCHUMMER e HAAKE, 2001, SARMA *et al.*, 2003). O apoio à percepção torna-se ainda mais importante quando a equipe tem poucas oportunidades para interagir face-a-face. Apesar da sua importância, os ambientes de desenvolvimento com apoio à percepção propostos apresentam deficiências que, em geral, impedem seu uso na prática. Muitas das deficiências encontradas no apoio à percepção nos ambientes colaborativos de desenvolvimento de software são decorrentes da abordagem usada na sua implementação. Quando o apoio à percepção é desenvolvido por meio de uma alteração de um ambiente colaborativo existente, normalmente, o novo ambiente apresenta um número limitado de funções de apoio à tarefa. Quando o ponto de partida é um ambiente de desenvolvimento existente, o apoio implementado fica restrito a esse único ambiente modificado. O tema principal desta tese é a proposta de uma abordagem que permita a integração do apoio à percepção em ambientes de desenvolvimento, com ênfase em cenários de desenvolvimento de software globalizado, que contorne essas duas deficiências citadas.

## *1.1 Motivação*

O desenvolvimento de software é uma atividade colaborativa. Com exceção de sistemas triviais, o software é o resultado do esforço de diversos indivíduos que contribuem com capacidades e perspectivas diferentes. Até mesmo o desenvolvedor que considera que trabalhou sozinho está, de fato, dando continuidade ao trabalho de outros desenvolvedores que desenvolvem as bibliotecas de programação e os demais artefatos que foram em grande parte reutilizados para dar origem ao novo sistema. Em geral, no trabalho profissional, o desenvolvedor é parte de uma equipe de desenvolvimento. Este desenvolvedor consegue perceber com maior facilidade as dependências com os demais participantes. Nem sempre essas relações são formalizadas, mas é possível perceber no

trabalho de uma equipe de desenvolvimento a presença de diversos aspectos do trabalho colaborativo. Os desenvolvedores recebem tarefas específicas, mas costumam influenciar no trabalho dos demais, resolvendo dúvidas, trabalhando em conjunto, ou apenas oferecendo uma segunda opinião.

No final dos anos 90, o desenvolvimento de software alcançou uma escala global. Ou seja, uma equipe de desenvolvimento pode estar alocada a um mesmo projeto, mas distribuída fisicamente em diferentes localidades geográficas. Esta distribuição geográfica pode ser observada em três cenários diferentes. Primeiro, podemos considerar os processos de desenvolvimento de software de grandes empresas multinacionais, como Hewlett-Packard, IBM, Dell (PRIKLADNICKI *et al.*, 2003) e outras (EBERT e NEVE, 2001, BATTIN *et al.*, 2001). Nesses casos, a distribuição do processo de software é uma decorrência da distribuição geográfica da própria organização que o desenvolve.

Em um segundo cenário, esta escala global de produção de software afeta, indiretamente, empresas produtoras de software de pequeno e médio porte. Nesses casos, a dependência com outros processos de software que estão além das fronteiras da organização é que revela a natureza distribuída dos fatores que podem influenciar o sucesso de um processo de software definido localmente. Neste cenário, há uma grande influência da adoção de metodologias de desenvolvimento que pressupõem o desenvolvimento com reutilização de software, como é o caso do desenvolvimento com base em componentes (REPENNING *et al.*, 2001, KOTLARSKY *et al.*, 2003). Neste tipo de desenvolvimento, a existência de reutilização além das fronteiras da organização produtora estabelece dependências entre diversas instâncias de processos de desenvolvimento de software.

Em um terceiro cenário, temos o desenvolvimento de software aberto (*open source development*). Neste cenário, um grupo de voluntários colabora para o aprimoramento contínuo de produtos de software cujo código fonte e demais artefatos relacionados encontram-se disponíveis, principalmente, através da Internet (MOCKUS *et al.*, 2000).

Em cada um destes três cenários, podemos perceber que as equipes de desenvolvimento de software acabam por estabelecer dependências com outras equipes e indivíduos. Estas dependências podem ocorrer, respectivamente, de forma explícita,

implícita, ou na forma de participação voluntária. Em um caso concreto, provavelmente uma combinação dos três cenários possa ser constatada.

Desta forma, nota-se a necessidade de estudar e apoiar a realização desses processos de desenvolvimento de software distribuídos geograficamente. A distribuição geográfica aporta vantagens ao processo de desenvolvimento de software, pois possibilita, por exemplo (BUXMANN e KÖNIG, 2000, HERBSLEB *et al.*, 2001):

a) selecionar participantes e recursos de um conjunto total maior do que estaria disponível em uma área geográfica limitada;

b) explorar condições locais que favoreçam aspectos econômicos, oferta de mão-de-obra qualificada ou outra tendência natural local que seja apropriada às necessidades do processo;

c) distribuir instalações físicas de forma estratégica, para diminuir os riscos em caso de desastres ou cenários econômicos desfavoráveis.

De fato, estas vantagens são similares àquelas que beneficiam processos globalizados ou distribuídos em outras indústrias (BUXMANN e KÖNIG, 2000). Entretanto, existem também desvantagens. A distância geográfica entre os participantes impõe diversas barreiras inesperadas quando a diversidade de idiomas, valores e conceitos dos participantes é colocada em contato. Contornar estas barreiras significa equacionar problemas culturais (EBERT e NEVE, 2001, AUGIER e VENDELO, 1999).

Todo esse contexto nos leva à necessidade de apoiar esses desenvolvedores de software que enfrentam, também, dificuldades relacionadas com a sua característica de distribuição, relatadas em estudos empíricos (HERBSLEB *et al.*, 2001, HERBSLEB e MOCHUS, 2003). Nesta tese, o apoio à percepção estudado visa reduzir os problemas relacionados com a separação no tempo e no espaço, em particular, a dificuldade de comunicação e de percepção entre os participantes.

Existem diversas formas de apoiar o trabalho dessas equipes. Dentre elas, tornam-se freqüentes as propostas que integram funcionalidades existentes em ferramentas de trabalho em grupo e ambientes de desenvolvimento de software. Infelizmente, o desenvolvimento desses novos sistemas não é trivial. Em geral, o número de funcionalidades existente em um ambiente de desenvolvimento de software é grande e a estas estão sendo adicionadas as funcionalidades necessárias para o grupo. Isto torna significativo o problema técnico da criação e manutenção da infra-estrutura

necessária para que o desenvolvimento globalizado ocorra.

## 1.2 Objetivos

O tema desta tese é o problema do apoio ao desenvolvimento distribuído de software pela perspectiva do desenvolvedor da infra-estrutura de apoio. Nossa intenção é tentar minimizar problemas relacionados com a tecnologia e a infra-estrutura. Em específico, se busca otimizar a relação entre os benefícios desta infra-estrutura colaborativa e o custo e o esforço necessários para sua realização, configuração e operação.

Esta tese apresenta uma abordagem que visa estudar e apresentar soluções envolvendo alguns dos aspectos da construção desses ambientes colaborativos. Em particular, os problemas de (a) obter informações que indiquem o surgimento ou a formação de duplas ou equipes pequenas de participantes de equipes distribuídas que tenham interesse em manter atividades colaborativas e (b) oferecer o apoio necessário para esta colaboração.

São três as principais contribuições desta tese: (a) a integração de serviços de colaboração, por meio de uma arquitetura de software com base em componentes de software, (b) uma proposta de desenvolvimento com o uso da arquitetura e (c) o uso e observação dessa arquitetura com a proposta de uma metodologia para avaliação desses componentes.

A primeira contribuição, portanto, é a integração de serviços de colaboração sugeridos em diferentes trabalhos da literatura como sendo necessários a estas equipes em situação de distribuição. Os efeitos desses serviços de colaboração combinados podem vir a minimizar as dificuldades decorrentes da diminuição das capacidades de comunicação, percepção e coordenação introduzidas pela situação de distribuição.

Os serviços de colaboração considerados neste trabalho pertencem a duas categorias. Alguns, de uso mais amplo, encontram-se propostos e citados na literatura de Trabalho Cooperativo/Colaborativo Apoiado por Computador (*Computer Supported Cooperative Work* - CSCW) (BORGHOFF e SCHLICHTER, 2000). Alguns dos mecanismos considerados exploram: informação síncrona, teleapontadores, janelas de radar (GUTWIN e GREENBERG, 1999); e outros, informação assíncrona, *gauges* de

percepção (KANTOR e REDMILES, 2001); indicadores de atividade (BORGES e PINO, 1999) e ritmos de trabalho (BEGOLE *et al.*, 2002). Outros serviços são parte integrante de ambientes colaborativos específicos para Engenharia de Software: *Tukan* (SCHÜMMER e HAAKE, 2001), MILOS (MAURER e MARTEL, 2002), *Serendipity* (GRUNDY *et al.*, 2000), *Palantír* (SARMA *et al.*, 2003) e gSEE (FIELDING *et al.*, 1998). Entretanto, ainda são raros os ambientes colaborativos que apresentam esses serviços combinados para a edição de diagramas e documentos estruturados utilizados na etapa de concepção e projeto de software, como é o caso do ambiente que foi construído com a abordagem descrita nesta tese. A maioria dos ambientes trata da colaboração durante a edição de código fonte de programas e de documentos não estruturados.

A segunda contribuição é a proposta de como esses serviços podem ser desenvolvidos e integrados na melhoria de ambientes de desenvolvimento de software pré-existentes, no projeto de novos ambientes e, na reutilização de partes de sua implementação. Esta contribuição é relevante, pois possibilita associar em uma mesma solução (a) as ferramentas já existentes, adequadas à tarefa e provavelmente familiares aos participantes, com (b) os serviços colaborativos necessários para realizar esta tarefa em condições de distribuição. A associação destas ferramentas se torna possível através da aplicação de uma variação da técnica de compartilhamento de aplicações chamada de colaboração transparente (BEGOLE *et al.*, 1999, LI e LI, 2002). Um sistema de colaboração transparente (*collaboration transparency system*) permite o compartilhamento síncrono de aplicações monousuário legadas por meio de alterações em seu ambiente de execução (BEGOLE, 1998). Nesta tese, se considera também a aplicação do compartilhamento assíncrono de dados coletados nas aplicações *desktop* e repositórios utilizados no desenvolvimento de software. Esta parte central da abordagem é descrita por meio de uma arquitetura de software (Arquitetura de Colaboração Transparente - ACT) e de instruções para sua utilização na construção de novos sistemas e na possível extensão de sistemas existentes. Espera-se que as diversas decisões de projeto contidas nessa arquitetura contribuam para guiar o desenvolvimento de novos ambientes e aumentar a oferta de apoio à percepção disponível ao desenvolvimento de software.

Por fim, a terceira contribuição é decorrente do uso e da observação desses

ambientes que combinam serviços colaborativos e ferramentas de desenvolvimento em cenários de avaliação. Três tipos de avaliações foram realizados: (a) verificar a viabilidade e limitações da construção do ambiente, (b) avaliar a dificuldade de criar cenários apropriados para uma tarefa específica e (c) avaliar o impacto do apoio oferecido por este ambiente na atividade dos participantes.

Assim, em linhas gerais, esta tese apresenta uma abordagem para a criação de ambientes colaborativos, voltados para atividades de desenvolvimento de software, dentro da técnica de transparência. Esta abordagem permite realizar ambientes que combinam mecanismos de percepção, colaboração, coordenação e memória de grupo de acordo com as necessidades de uma tarefa específica. Além disso, providências são tomadas para que o próprio ambiente ofereça algum apoio durante a observação de dados qualitativos e quantitativos do processo e dos resultados desta colaboração.

A investigação dos aspectos colaborativos de um processo de trabalho oferece a oportunidade de revolucionar a maneira como o trabalho é realizado. Em especial, a aplicação de *groupware* pode ser um instrumento de transformação de uma organização (LLOYD e WHITEHEAD, 1996). Inclusive, a colaboração pode ser utilizada para a reformulação de um processo de trabalho já existente. No contexto de Engenharia de Software, por exemplo, Boehm *et al*. (2001) relatam o sucesso de uma metodologia inovadora para a elicitação de requisitos conhecida como WinWin. Entretanto, nesta tese, adota-se a opção mais modesta de incentivar a construção de ambientes colaborativos que ofereçam apoio para a forma de trabalho convencional que faz parte do dia-a-dia do desenvolvimento, ampliada pela noção de percepção. Ao invés de uma revolução nos processos de trabalho, espera-se que ocorra uma evolução natural e gradual apoiada na introdução de *groupware* (WULF, 1999).

## 1.3 O Contexto de Engenharia de Software nesta Tese

A motivação apresentada tenta estabelecer a demanda pelo tipo de apoio para colaboração que está sendo proposto dentro do contexto do processo técnico do desenvolvimento de software.

Entretanto, muito do que será tratado neste trabalho poderia ser aplicado para apoiar o trabalho colaborativo em diferentes ramos da ciência e da indústria. De fato, a

dificuldade técnica da proposta e as soluções apresentadas poderiam ser similares, mesmo sem o contexto da aplicação em Engenharia de Software.

Existem características que tornam o desenvolvimento de software um contexto adequado a este trabalho. Em primeiro lugar, o fato de utilizarmos o contexto de Engenharia de Software estabelece um conjunto de requisitos e restrições para o apoio à colaboração que sugerem uma solução mais elaborada, ao mesmo tempo em que nos oferece a possibilidade de estabelecer objetivos, métricas e outros indicadores de qualidade e eficiência no cumprimento das tarefas com e sem o apoio à percepção.

Engenheiros e desenvolvedores de software são participantes familiarizados com o uso de tecnologia. Além disso, uma parcela considerável do trabalho desses já é realizada com auxílio do computador, com o uso intensivo de editores gráficos e textuais e ferramentas de transformação de artefatos. Além disso, podemos contar com a presença de alguns entusiastas da tecnologia e de pessoas que não tem maiores restrições quanto ao uso do computador no ambiente de trabalho.

Outra característica do desenvolvimento de software é a diversidade de ferramentas já disponíveis, envolvidas em um processo de evolução e criação constante. Diversas propostas existem para criar conjuntos de ferramentas interoperáveis neste tipo de ambiente. A existência prévia de ferramentas que apóiam o trabalho no domínio de engenharia de software sugere que estas sejam aproveitadas e reutilizadas para a evolução do ambiente de desenvolvimento em direção ao apoio à colaboração.

A concepção da arquitetura de colaboração transparente prevê a possibilidade de adaptação e extensão do apoio à colaboração para outros domínios de aplicação. Mas, neste trabalho, estas extensões permanecem apenas como um potencial para o desenvolvimento de trabalhos futuros. Essa limitação do escopo do trabalho existe para controlar a complexidade do trabalho atual e guiar a seleção da literatura necessária.

Este trabalho encontra-se no contexto do projeto *OdysseyShare*, do grupo de Reutilização do Programa de Engenharia de Sistemas e Computação, nesta Universidade. O projeto desenvolve o ambiente *OdysseyShare* (WERNER *et al.*, 2002, WERNER *et al.*, 2003), um ambiente colaborativo voltado para o desenvolvimento de componentes de software. O ambiente *OdysseyShare* é uma alteração do ambiente *Odyssey* (WERNER *et al.*, 2000) contendo três extensões: uma máquina de processos (MURTA, 2002), um repositório de componentes (SOUZA *et al.*, 2001) e o apoio à

percepção, este último desenvolvido durante esta tese. Essas extensões são complementares, visto que a máquina de processos representa o planejamento e coordenação de atividades e o repositório de componentes incentiva a troca e reutilização de componentes de software em equipes que se encontram integradas pela Internet. O apoio à colaboração no *OdysseyShare* foi desenvolvido com base na abordagem apresentada nesta tese.

## 1.4 Organização desta Tese

O restante desta monografia está organizado em sete capítulos e um anexo. Os próximos dois capítulos apresentam, respectivamente, o estado da prática e da arte relacionados com aspectos da colaboração no processo de Engenharia de Software. Neste trabalho, apresenta-se uma análise crítica de algumas práticas e ferramentas que visam apoiar algum grau de colaboração entre os participantes (Capítulo 2). A Engenharia de Software Colaborativa é uma comunidade de interesse que está se formando a partir das experiências coletadas com o uso das tecnologias de informação para a coordenação de processos de desenvolvimento distribuídos. Neste trabalho, apresenta-se um resumo dos conceitos, requisitos e experiências que foram coletadas na literatura desta área (Capítulo 3).

A colaboração transparente é uma técnica para a construção de ambientes colaborativos a partir de aplicações pré-existentes. Esta técnica foi selecionada para servir como base da abordagem para o desenvolvimento de ambientes colaborativos voltados para o desenvolvimento de software. Este trabalho inclui uma breve revisão dos conceitos principais da proposta de colaboração transparente e experiências coletadas na literatura (Capítulo 4).

A arquitetura transparente para ambientes de desenvolvimentos colaborativos é apresentada, apontando as vantagens da aplicação da colaboração transparente para a construção de ambientes colaborativos. Os pontos principais desta arquitetura são apresentados e algumas experiências preliminares realizadas com alguns protótipos são comentadas (Capítulo 5). Este texto inclui o relato de experiências com a construção dos protótipos (Capítulo 6) e de avaliações (Capítulo 7) que foram realizados para determinar a viabilidade, o impacto no trabalho e a generalização da solução apresentada (Capítulo 8). Por fim, são apresentadas as contribuições obtidas e a

conclusão deste trabalho. O Anexo I apresenta parte dos instrumentos usados na avaliação que serve como exemplo para novos estudos.

# 2 Desenvolvimento Distribuído de Software

Este capítulo descreve o estado da arte relacionado ao desenvolvimento distribuído de software, que forma o contexto de pesquisa desta tese.

O desenvolvimento de software é uma atividade colaborativa (ARAÚJO *et al.*, 1997) (ALTMANN e POMBERGER, 1999) (BARTHELMESS e ANDERSON, 2002). Um projeto de software é o arcabouço de recursos dentro do qual o desenvolvimento de software ocorre. Araújo *et al.* (1997) classificam projetos de software com base em quatro aspectos da colaboração: comunicação, coordenação, memória de grupo e percepção. Cada um desses aspectos é independente e o desenvolvimento colaborativo é uma das classificações possíveis que resulta da combinação de cada aspecto.

Altmann e Pomberger (1999) apresentam cinco aspectos importantes do apoio ao desenvolvimento colaborativo de software: (a) colaboração espontânea, (b) construção de uma consciência de grupo, (c) comunicação direta e indireta, (d) conhecimento sobre o estado atual do projeto e (e) a reutilização de conhecimento com base em experiências concretas. A comunicação é vista como uma base para a coordenação entre os participantes e a cooperação é vista como uma das formas de coordenação entre os participantes.

Em um artigo mais recente, Barthelmess e Anderson (2002) apresentam argumentos a favor da natureza colaborativa do desenvolvimento de software por meio de uma análise com base na Teoria de Atividades. A classificação resultante considera quatro aspectos dos ambientes de desenvolvimento: (a) apoio à coordenação, (b) apoio à cooperação, (c) apoio à co-construção e (d) apoio à transição entre níveis. Da análise resulta, novamente, um destaque ao apoio à comunicação e ao compartilhamento de artefatos. Barthelmess e Anderson utilizam, também, como argumentos a noção de desenvolvimento de software como aprendizagem colaborativa (KEIL-SLAWIK, 1992) e solução colaborativa de problemas (FISHER, 1994).

Portanto, parece haver um consenso sobre a natureza colaborativa do desenvolvimento de software moderno. Além disso, existe um consenso de que os

ambientes de desenvolvimento continuam a oferecer apoio insuficiente para a comunicação e a percepção do trabalho. Ainda segundo Barthelmess e Anderson (2002), existem dois obstáculos para esse tipo de apoio. Primeiro, existe uma incompatibilidade entre (a) a concepção atual que considera o desenvolvimento de software como produção de artefatos, como se fosse uma tarefa de construção em uma fábrica, e (b) as idéias de desenvolvimento de software como um jogo, um aprendizado ou solução de problemas, onde a perspectiva de colaboração se torna mais evidente. O segundo obstáculo é de natureza técnica: o apoio existente, atualmente, para o trabalho em grupo, pode não ser suficiente para oferecer o apoio necessário para o desenvolvimento de software.

Por outro lado, parece não haver um consenso de como formalizar os aspectos da colaboração envolvidos no desenvolvimento de software, nem das maneiras como esses aspectos se relacionam. Além das três apresentadas, outras classificações são possíveis e cada uma delas tem a sua aplicação, vantagens e desvantagens. O consenso geral é de que o processo de desenvolvimento é colaborativo por natureza, devido principalmente a complexidade desta atividade, que normalmente excede a capacidade cognitiva e a preparação técnica de um único indivíduo.

A pesquisa no desenvolvimento colaborativo de software é impulsionada por duas áreas de trabalho:

- *Computer Supported Cooperative Work* (CSCW): esta área tem estudado as relações entre o apoio de programas de computador criados para auxiliar o trabalho de grupos (sistemas colaborativos ou *groupware*) e a organização do trabalho em um grupo ou entre grupos de usuários. CSCW é considerado um domínio de aplicação pelos especialistas da área, devido à complexidade das relações que foram encontradas e pela diversidade de famílias de *groupware* existentes. Um *groupware* típico oferece diversos serviços de colaboração adaptados para uma tarefa específica ou perfil de usuários (ELLIS *et al.*, 1991).
- *Computer Aided Software Engineering* (CASE): esta área tem estudado o apoio ao trabalho de profissionais do desenvolvimento de software através de aplicações de computador. Existe uma ampla variedade de ferramentas CASE e também de técnicas que acompanham o seu uso (PRESSMANN, 2002).

A intersecção entre essas duas áreas oferece um extenso campo para pesquisa,

no qual se situa esta tese. Existem diversas oportunidades para a aplicação de conceitos e técnicas de CSCW para aprimorar aspectos de colaboração apresentados por ferramentas CASE. Em particular, as ferramentas CASE, até o momento, têm se concentrado em apoiar uma alta percepção das tarefas e o compartilhamento de artefatos. Por outro lado, as oportunidades para a comunicação direta e a **percepção do trabalho** são muito baixas (FARSHCHIAN, 2001). Em ambientes onde a colaboração face-a-face é freqüente, essa deficiência das ferramentas CASE quase não é percebida. Diversos canais de comunicação são oferecidos naturalmente pelo ambiente físico de trabalho compartilhado. As pessoas são capazes de (a) perceber o ritmo de trabalho dos colegas, (b) encontrar interesses em comum e (c) realizar tarefas em pequenos grupos de forma natural e espontânea. No ambiente distribuído, os canais de comunicação são reduzidos e a capacidade dos desenvolvedores de trabalhar em conjunto é reduzida (HERSLEB *et al.*, 2001).

Em contrapartida, existe a necessidade da aplicação de conceitos e técnicas de Engenharia de Software na melhoria dos ambientes analisados nas propostas recentes da literatura. Principalmente, as críticas se concentram na estratégia *ad hoc* utilizada no desenvolvimento de protótipos e na falta de uma infra-estrutura comum que apóie as necessidades comuns desses ambientes, ao mesmo tempo em que permite diferenças significativas entre eles.

O restante deste capítulo está organizado da seguinte forma. A seção seguinte apresenta um breve histórico de algumas definições para o desenvolvimento colaborativo de software (Seção 2.1), seguida de alguns conceitos relevantes ao contexto de estudo (Seção 2.2). O desenvolvimento distribuído pode ocorrer de forma orgânica, ou seja, como um resultado de situações não previstas, ou ser resultado de um planejamento prévio. Alguns cenários são apresentados para ilustrar algumas das situações onde a distribuição ocorre (Seção 2.3). Em seguida, advoga-se que o apoio ao desenvolvimento de software com base em componentes deve incluir apoio à colaboração devido a características dessa técnica de reutilização (Seção 2.4). São apresentados dois aspectos importantes do apoio ao desenvolvimento: a coordenação de tarefas (Seção 2.5) e a cooperação entre participantes (Seção 2.6). Apesar do apoio existente, a literatura aponta diversos desafios que são também apresentados (Seção 2.7). Por fim, a conclusão resume os principais pontos deste capítulo

(Seção 2.8).

## 2.1 Breve Histórico

Em uma das primeiras demonstrações de um sistema de apoio à colaboração, um programa em FORTRAN foi utilizado para demonstrar a capacidade de visualização síncrona de um documento por dois participantes colocados em cidades diferentes (ENGELBART e ENGLISH, 1968). Quase trinta anos depois, as equipes distribuídas de desenvolvimento de software tornam-se uma característica da produção de software em larga escala e os primeiros estudos de campo são realizados (GORTON e MOTWANI, 1996).

Portanto, a idéia de distribuição física no desenvolvimento de software é antiga. Ao longo do tempo, a capacidade de mediação oferecida pelos computadores e a motivação para a distribuição dos participantes é que sofreram evoluções. Várias denominações foram propostas ao longo dos anos.

A idéia de uma equipe de desenvolvimento coordenada, com trabalho em paralelo e um local compartilhado para armazenamento de informações já estava bem estabelecida na **Engenharia de Software Concorrente** (*Concurrent Software Engineering*) e no apoio proposto para esse tipo de desenvolvimento (DEWAN e RLEDL, 1993). A **Engenharia de Software Colaborativa** (*Collaborative Software Engineering*) reconhece a necessidade de cooperação entre os envolvidos. Tanto de uma colaboração explícita, através de planejamento prévio dos processos de colaboração, quanto de uma colaboração mais informal, com base em negociação entre os participantes.

Com relação à existência de uma equipe geograficamente distribuída, existem duas áreas de pesquisa mais citadas na literatura. A **Engenharia de Software Global** (*Global Software Engineering*) é uma área que trata dos problemas estratégicos, técnicos e culturais do desenvolvimento de software globalizado (HERBSLEB e MOITRA, 2001). O termo Engenharia de Software é utilizado para se referir ao desenvolvimento de software em larga escala. O **Desenvolvimento de Software em Código Aberto** (*Open Source Software Development*) se refere a todo software cujo código fonte é definido como código aberto sob uma das licenças de código aberto para

estudo, alteração e melhoria de seu projeto e implementação (VÄLIMÄKI, 2005). A Engenharia de Software Global é uma prática de grandes empresas, enquanto o Desenvolvimento de Software Livre é uma ação conjunta de grupos de desenvolvedores. Essa diferença na motivação dos participantes se reflete nas relações entre os participantes da equipe e na própria concepção do que é uma equipe de desenvolvedores.

A denominação **Engenharia de Software Distribuído/Distribuída** (*Distributed Software Engineering*) apresenta pelo menos duas interpretações, que ficam mais evidentes na tradução para o português. Na primeira interpretação, os participantes estão trabalhando para a construção de um programa de computador (software) que deverá operar em um ambiente distribuído. Neste caso, o desenvolvimento de *groupware* pode ser entendido como um tipo de engenharia de software distribuído. Esta interpretação torna-se evidente quando se usam denominações mais precisas como Engenharia de Software Distribuído e Paralelo, onde software paralelo é um tipo específico de software distribuído, e a Engenharia de *Groupware* (FUKS *et al.*, 2002), onde *groupware* é um tipo de software distribuído que apóia o trabalho de grupos. Na segunda interpretação, os participantes estão trabalhando em um ambiente de desenvolvimento que oferece apoio à distribuição física dos participantes.

Esta tese está relacionada com as duas interpretações apresentadas. A primeira interpretação é a perspectiva do desenvolvedor do ambiente colaborativo em engenharia de software e a segunda interpretação é a perspectiva do desenvolvedor que vai trabalhar nesse novo tipo de ambiente de desenvolvimento. A abordagem proposta visa guiar o desenvolvedor a gerar um ambiente distribuído que oferece um conjunto de funcionalidades de interesse de um desenvolvedor que trabalha em condições de distribuição.

## 2.2 Conceitos Básicos

Três conceitos importantes ajudam a definir o desenvolvimento de software globalizado. O primeiro conceito é o da equipe virtual, pois o desenvolvimento globalizado, muitas vezes, é definido pela necessidade de composição equipes de trabalho cujos membros não se encontram localizados em um mesmo espaço físico.

Outra forma de caracterizar o desenvolvimento globalizado é através das dimensões tempo e distância. Em geral, a realização de uma tarefa de forma distribuída demanda mais tempo e envolve recursos também distribuídos, ou seja, separadas por uma distância física considerável. A avaliação do que é uma distância considerável é subjetiva. Desenvolvedores trabalhando em um mesmo prédio, mas em andares diferentes, podem considerar que essa distância impede ou reduz sua colaboração. A avaliação, embora subjetiva, apresenta alguns critérios que podem ser identificados. O tempo total da interação, do deslocamento, da preparação para o deslocamento, a disponibilidade do outro desenvolvedor e a freqüência dos encontros podem decidir se haverá uma interação face-a-face ou à distância.

### 2.2.1 Equipes Virtuais

Nem todo grupo é uma equipe e nem toda equipe obtém sucesso. Essa afirmação sugere que existem características específicas em uma equipe que a diferencia de um grupo de pessoas. DeMarco e Lister (1998) definem que uma equipe é um grupo com a consciência de um objetivo comum.

Constantine (1993) advoga que uma equipe de desenvolvimento pode ser organizada de acordo com "quatro paradigmas" diferentes. Para tarefas que são bem definidas e fazem parte da experiência da equipe é sugerido o **paradigma fechado**. Essas equipes apresentam relações de autoridade e hierarquia bem definidas, as tarefas são bem realizadas e o cronograma e orçamento facilmente controlados. No **paradigma aleatório**, a estrutura da equipe não é fortemente definida, as relações de autoridade se modificam de acordo com a tarefa realizada. Essas equipes podem atingir altos índices de inovação, mas resistem à introdução de controle e gerência de atividades. No **paradigma aberto**, existe um maior equilíbrio na hierarquia e, ao mesmo tempo, existe espaço para discussão e as decisões são baseadas em consenso. A quarta organização para equipes é o **paradigma síncrono**. O problema é particionado de forma tão eficiente que os integrantes da equipe podem trabalhar sem a necessidade de comunicação.

Equipe virtual é um termo genérico para designar grupos formados por pessoas, com um objetivo comum, que não se encontram em proximidade física (STEINFIELD *et al.*, 1999). Equipes virtuais possibilitam a reunião de habilidades de profissionais

dispersos em diferentes áreas geográficas.

Nesta tese, considera-se o apoio prioritário às interações entre elementos de uma equipe virtual. A colaboração entre equipes virtuais é um objetivo secundário. Considera-se, também, que as equipes virtuais estejam operando no paradigma aleatório ou no paradigma aberto.

Equipes virtuais globalizadas são caracterizadas pela sua duração temporária, pela diversidade cultural e pela ausência de um passado comum (JARVENPAA e LEIDNER, 1998). Ainda, existe uma grande dependência dessas equipes no apoio de *groupware* e de telecomunicações.

As vantagens em manter equipes virtuais são várias. Entre elas, podem ser destacadas: a facilidade em alocar membros em um maior conjunto de profissionais disponíveis, a possibilidade de manter um volume contínuo de trabalho que evita que a equipe permaneça ociosa e a possibilidade de mover parte da equipe para um local mais adequado a um determinado projeto. Por exemplo, existe uma grande demanda por programadores para a linguagem Java nas empresas dos EUA, enquanto existe uma abundância desses profissionais em universidade brasileiras. Em outro cenário, uma empresa pequena pode necessitar de um gerente de projeto em tempo parcial. O gerente pode participar da gerência de mais de uma equipe virtual, oferecendo suas habilidades para mais de uma empresa e dividindo o ônus de sua contratação. A terceira vantagem potencial é a possibilidade de movimentar parte da equipe para atender um grande cliente em um projeto. A equipe remota pode contar com a equipe local para proporcionar uma maior velocidade ao projeto e a empresa pode compartilhar a equipe local com diversos projetos remotos.

### 2.2.2 Tempo e Distância

O tempo e a distância são duas características afetadas pela distribuição do desenvolvimento de software (HERBSLEB *et al*., 2001). O tempo se refere ao atraso que existe na comunicação e nos horários diferentes de trabalho de acordo com a região onde as equipes estão localizadas. A distância contribui com o atraso e reduz as possibilidades de contatos diretos. As requisições realizadas para uma equipe virtual apresentam um maior tempo para atendimento, se comparadas com o tempo necessário

para atender a mesma requisição em uma equipe que trabalha em um mesmo espaço físico (HERBSLEB *et al.*, 2001). O atraso na comunicação é o principal fator que causa o atraso no atendimento da requisição, não apenas pela mediação da comunicação que ocorre por correio eletrônico, telefone e bate-papo, mas, principalmente, pela necessidade de comunicar o contexto da requisição para um membro que está trabalhando em um contexto de trabalho diferente ou desatualizado.

A distância é outra característica do desenvolvimento globalizado. Os participantes, normalmente, se encontram a uma distância que torna difícil a comunicação face-a-face. A razão é que os centros de desenvolvimento exploram recursos locais, portanto, dois centros colocados muito próximos estariam competindo por um mesmo recurso. Normalmente, os centros são dispostos em cidades, países e até continentes diferentes.

## 2.3 Cenários para o Desenvolvimento Distribuído de Software

Reich (1992) classifica trabalhadores de diversas profissões em categorias bem definidas. Os desenvolvedores de software podem ser classificados como "processadores simbólicos". Ou seja, trabalhadores que utilizam intensamente de raciocínio e que recebem uma formação especializada para a função. Uma característica dessa categoria é que os processadores simbólicos não se encontram fixados em um local específico por causa das condições de trabalho. Um "prestador de serviços" precisa se deslocar até o local onde o cliente se encontra, por outro lado, o "processador simbólico" pode realizar a sua tarefa em qualquer lugar. Por essa razão, as associações entre os desenvolvedores são mais flexíveis.

Conforme mencionado na Seção 1.1, podem ser organizados diversos cenários para o desenvolvimento distribuído de software. Do ponto de vista de importância econômica, os cenários de maior importância são o da empresa globalizada e o do grupo de desenvolvimento voluntário. O cenário da empresa desenvolvedora globalizada é uma área de atuação restrita às grandes empresas distribuídas em vários centros de trabalho. Por exemplo, a empresa estado-unidense Dell possui quatro centros de desenvolvimento de software interligados localizados nos EUA, Brasil, Espanha e Índia. Com a existência de recursos em centros diferentes, a maior dificuldade está em

coordenar esforços. Neste cenário, uma forma comum de organizar o trabalho é dividir atividades entre centros diferentes que se especializam em algumas atividades específicas. Por exemplo, um centro se especializa em elicitação de requisitos, outro em desenvolvimento e outro em testes. Dessa forma, a cadeia produtiva é segmentada em etapas e há uma redução na necessidade de comunicação e cooperação, por causa do esforço de coordenação prévio. Em geral, as soluções encontradas neste cenário são derivadas na visão de desenvolvimento de software como um sistema de produção industrial. A cooperação, normalmente, ocorre entre equipes com base em um processo de desenvolvimento.

O segundo cenário ocorre no desenvolvimento de software formado por grupos de desenvolvedores voluntários. Nesse tipo de desenvolvimento, um software simples é continuamente aprimorado por um grupo de voluntários. Em geral, esses voluntários não se encontram em um mesmo local e dedicam-se ao projeto em tempo parcial. A cooperação, normalmente, ocorre entre indivíduos com base em acordos sobre as funcionalidades que devem ser realizadas.

Um cenário misto, que obriga a colaboração de um indivíduo com uma equipe pode ocorrer em equipes que dependem de consultores e equipes que deslocam membros para outros locais.

A reutilização, em especial, o desenvolvimento com base em componentes, determina um conjunto de cenários adicionais em que as dependências entre componentes de software determinam a necessidade de cooperação entre desenvolvedores, principalmente, nos cenários onde ocorre a relação entre produtor e consumidor do componente em organizações distintas. Esses cenários são descritos na seção a seguir.

## *2.4 Desenvolvimento com Base em Componentes*

O desenvolvimento de software com base em componentes (DBC) é uma abordagem da Engenharia de Software que promove a reutilização de software. No DBC, todos os artefatos de software – desde aplicações, especificações de interfaces, arquiteturas, até modelos de negócio – podem ser construídos através da composição, adaptação e integração de componentes pré-existentes utilizando uma grande variedade

de configurações possíveis (D'SOUZA, WILLS, 1998).

Algumas abordagens de DBC (D'SOUZA e WILLS, 1998, BROWN, 2000, ATKINSON *et al.*, 2001, CHESSMAN e DANIELS, 2001,) apresentam excelentes resultados no particionamento de software e de tarefas, mas ainda assim não são suficientes para atingir o paradigma síncrono proposto por Constantine. A representação da informação que descreve o componente e as suas condições de uso é tema de pesquisa (TAYLOR *et al.*, 2000).

Estudos demonstram que a prática de reutilização ainda é muito dependente de comunicação entre as equipes produtoras e consumidoras de componentes. A documentação é suficiente para descrever o conhecimento de experiências, mas a comunicação direta é necessária para resolver diversos problemas e estabelecer relações de confiança entre os produtores e consumidores de software (GRINTER, 2001). Existem quatro cenários para o desenvolvimento de software, considerando a localização dos produtores e consumidores do componente de software (TAYLOR *et al.*, 2000). O cenário tratado nesta tese envolve a distribuição dos produtores e consumidores em locais diferentes. Além disso, um produtor fornece componentes de software para diversos consumidores e um consumidor adquire componentes de diversos produtores. Essa condição é referida como um *mercado de software* (WALLNAU *et al.*, 2001). Temos, portanto, um relacionamento *n* x *m* entre produtores e consumidores (Figura 2.1).

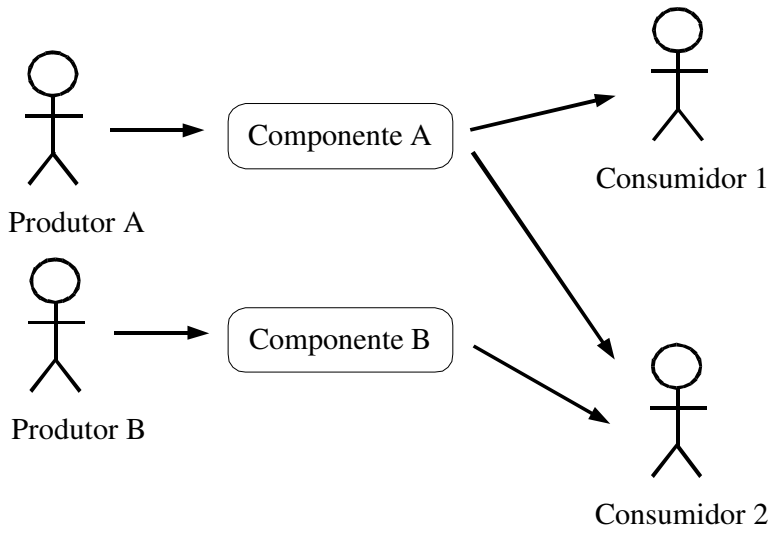



**Figura 2.1 O mercado de componentes de software.**

Nesta tese, considera-se que esses relacionamentos determinam uma *rede de conhecimento* (AUGIER e VENDELO, 1999), na qual as informações necessárias para a manutenção, evolução e aplicação do componente de software estão distribuídas de forma irregular entre os participantes. As redes de conhecimento são baseadas em relacionamentos colaborativos entre indivíduos e organizações que permitem o acesso rápido a conhecimento e recursos. Por outro lado, as técnicas de desenvolvimento de componentes em larga escala baseiam-se em documentação escrita dos componentes (TAYLOR *et al.*, 2000). Nossa proposta é complementar as práticas de documentação com a adição do apoio para essas redes de conhecimento.

A especificação e a documentação de um componente são muitas vezes insuficientes para comunicar as características essenciais do componente. Brown (2000) comenta que a documentação atual oferece muitos detalhes de implementação, mas tem dificuldades em registrar as condições de uso e funcionalidades oferecidas, que são essenciais para a localização e aplicação do componente. Ainda, podemos argumentar que parte do conhecimento necessário para a aplicação da reutilização não se encontra formalizado (WALENSTEIN, 2002). Portanto, existe a necessidade de comunicação direta entre os participantes. Existem outros tipos de conhecimento que fazem parte do desenvolvimento de software, além do conhecimento estruturado que é capturado na documentação escrita. Por exemplo, a gestão de conhecimento reconhece o conhecimento **tácito** e **situacional** (SVEIBY, 1999), aquele conhecimento que está de posse do indivíduo, mas é de difícil representação; e o conhecimento de "condições específicas de tempo e lugar" (HAYEK, 1945), ou seja, aquele que deriva de um ponto de observação privilegiado dentro de um sistema.

Além da classificação de Sveiby, no desenvolvimento de software, podemos organizar tipos de conhecimento como conhecimento de uso comum, tácito e ocupacional (*folk, tacit* e *craft knowledge*) (WALENSTEIN, 2000). O conhecimento de **uso comum** é aquele que é compartilhado pela equipe de desenvolvimento de forma implícita. Os participantes da equipe dispõem desse conhecimento, mas não se preocupam em representá-lo ou discuti-lo. Por exemplo, a hora aproximada de entrada e saída de cada participante da equipe ao trabalho é um conhecimento comum que não está prontamente disponível para quem é externo à equipe. O **conhecimento tácito** é aquele que é mantido por alguns indivíduos e que é difícil de explicitar. Por exemplo, o

processo de decisão de especialistas envolve uma série de análises que muitas vezes nem o próprio especialista consegue descrever de forma ordenada e determinística. O terceiro conhecimento é o **ocupacional**, e decorre da exposição contínua às tarefas, técnicas e ferramentas de desenvolvimento de software. Por exemplo, as instruções para compilar e executar um programa dificilmente são escritas, pois são consideradas óbvias para quem opera um compilador diariamente.

Walenstein (2000) argumenta que o esforço de uma ciência está em transformar todo o conhecimento em um conjunto organizado de fatos e regras. Entretanto, o conhecimento de todos os tipos está sempre se ampliando, reorganizando e, às vezes, deixando de ter validade. Com isso, é impossível atingir o objetivo de formalizar todo o conhecimento. Por outro lado, o conhecimento pode ser comunicado de um indivíduo para outro, desde que ocorra uma situação propícia para isso. Boa parte da importância da comunicação no desenvolvimento de software é decorrente dessa necessidade de explorar um contexto determinado para a extração de conhecimento.

Portanto, devido às limitações atuais na documentação de componentes e da natureza diversificada de conhecimentos que envolvem a prática do desenvolvimento de software, advogamos que o apoio à colaboração no DBC pode incentivar a reutilização de software. Em outras palavras, se o ambiente de DBC apresentar facilidades para a colaboração, parte do conhecimento que está ausente nos documentos poderia circular durante conversas mediadas pela própria infra-estrutura de comunicação.

A psicologia ambiental (GUNTHER, 2003) utiliza conceitos propostos por Gibson (1976) para explicar essas relações entre o indivíduo e o ambiente. Kreijn e Kirschner (2001) argumentam de forma similar, sobre o apoio à cooperação na área de aprendizado colaborativo, aplicando o conceito de *affordance*, proposto na área de psicologia ambiental por Gibson (1976). Segundo Gibson, um organismo percebe as possibilidades (*affordances*) do ambiente e as explora com base em um valor atribuído, um convite ou uma demanda oferecida por essa possibilidade. Ainda segundo Gibson, as possibilidades mais ricas são oferecidas por outros organismos presentes no ambiente. Portanto, de forma similar, as informações de percepção coletadas poderiam ser utilizadas para detectar quais os desenvolvedores que estariam mais aptos para participar dessas conversas, de acordo com suas atividades atuais e seu histórico de atividades. Embora essa seja uma das principais motivações para a construção desses

ambientes colaborativos, no contexto desta tese, não é um objetivo avaliar o impacto do apoio à colaboração na documentação ou no desempenho dos desenvolvedores.

É importante destacar que o DBC não é realizado de forma descontrolada. Existe uma organização do trabalho que envolve os contextos de atividade dos grupos de desenvolvedores envolvidos. Por exemplo, o apoio ao desenvolvimento de software baseado em componentes proposto por Werner *et al*. (2002) assume que os processos de desenvolvimento da modelagem de domínio e da aplicação propriamente dita são planejados, utilizando-se uma máquina de processos. Esta máquina possibilita a tomada de decisões gerenciais e o controle do fluxo de trabalho das equipes de engenheiros de software envolvidos nas atividades cooperativas de Engenharia de Domínio (análise de domínio, projeto de arquitetura para o domínio, projeto de componentes para reutilização) e de Aplicação (análise e projeto da aplicação, escolha de arquitetura e seleção de componentes) suportadas pelo ambiente. Ao estabelecer estes processos definem-se, entre outros aspectos, a seqüência de atividades, as ferramentas a serem utilizadas, os papéis dos desenvolvedores e os artefatos consumidos e produzidos. Estes artefatos são recuperados em repositórios de componentes na Internet, através de mediadores e ontologias capazes de integrar repositórios distribuídos.

Quando os desenvolvedores estão de posse dos componentes, é que as dependências múltiplas que se estabelecem entre os participantes são percebidas. O DBC impõe uma separação entre produtores e consumidores de componentes que precisa ser transposta em situações que ameacem a efetiva prática de reutilização. Por exemplo, se um consumidor de componentes encontra dificuldades no momento de reutilizar, é do interesse do produtor desse componente obter mais informações sobre as condições particulares do projeto, com vistas à melhoria do componente. A falha em reutilizar ameaça tanto o produtor, que perde uma oportunidade de comercializar seu produto, quanto o consumidor, que pode ter seu projeto prejudicado pela ausência do componente previsto.

## 2.5 Apoio para Coordenação de Tarefas

Em um ambiente de desenvolvimento orientado a processos, o desenvolvimento é resultado da encenação de uma descrição de passos lógicos.

Entretanto, embora o processo seja fundamental, existem outros fatores relevantes para a coordenação de tarefas. Apesar de muito do que deve ser feito estar previsto e representado no processo, ainda há margem para imprevistos e mudanças durante o desenvolvimento. Na prática, o processo concorre com uma série de fatores importantes, tais como o incentivo à participação, a influência de colaboradores no trabalho dos desenvolvedores, a auto-relação de comunidades de desenvolvedores e a facilidade de acesso às informações.

Esta seção apresenta alguns desses fatores que regulam a coordenação das tarefas do desenvolvimento de software.

### 2.5.1 Processos de Desenvolvimento de Software

Um processo de software oferece um arcabouço para o estabelecimento de um plano para o desenvolvimento de software. Um conjunto reduzido de atividades desse arcabouço pode ser aplicado a qualquer projeto de software, independente de seu tamanho e complexidade. Um outro conjunto de atividades deve ser adaptado para os requisitos de um projeto específico e de sua equipe de desenvolvimento.

Na literatura, existem diversas propostas de métodos que oferecem uma ampla lista de atividades que complementam a descrição do processo de software. Entre eles destacam-se o *Rational Unified Process* (KRUCHTEN, 2000), pela popularidade, e o *Catalysis* (D'SOUZA, WILLS, 1998), pela abrangência. Outros métodos estão disponíveis, por exemplo, o desenvolvimento baseado em componentes considera o *UML Components* (CHESSMAN e DANIELS, 2001) e o KOBRA (ATKINSON *et al*., 2001). Existe ainda uma tendência para a proposta de métodos de desenvolvimento ágeis, por exemplo, o *Extreme Programming* (BECK, 1999) e o SCRUM (RISING e JANOFF, 2000). O desenvolvimento de software aberto também apresenta seus métodos particulares (FELLER e FITZGERALD, 2000).

Independente do método considerado, a aplicação desse arcabouço para um projeto específico gera um plano de desenvolvimento contendo as responsabilidades de cada participante, seus respectivos direitos de acesso e um cronograma das tarefas que devem ser desenvolvidas. Desta forma, o plano determina o contexto de atividade de cada participante. Esses contextos restritos de trabalho promovem uma diminuição da

complexidade do problema através da decomposição do processo em etapas menores em uma sucessão lógica de ações coordenadas (BARTHELMESS, 2003).

Entretanto, é reconhecido que esses planos não podem ser definidos de forma completa previamente (MADHAVJI, 1992). O plano precisa ser revisto e alterado durante a sua encenação, ou seja, a realização concreta das atividades previstas para os diversos participantes, por diversos motivos. Por exemplo, desenvolvedores podem deixar o projeto, responsabilidades podem ser redefinidas, atrasos e mudanças de requisitos podem ocorrer. Além disso, o plano é uma descrição geral do que deve ser feito. Como se costuma dizer na prática, o plano é "uma trilha, não um trilho". Ou seja, cada participante deve completar os detalhes que não foram previstos no plano para garantir a execução da atividade dentro do que foi planejado (MANGAN, 1998).

A necessidade de adaptar ou completar o plano leva um participante a buscar outros recursos e o auxílio de outros participantes que não eram previstos. Este tipo de colaboração imprevista é um dos cenários que será tratado nesta tese. O apoio à colaboração neste cenário deve permitir a localização de outros participantes com contextos de trabalho semelhantes, ou que possam contribuir com algum conhecimento necessário, mas não previsto. Neste caso, a colaboração é a exceção, um participante deve conseguir realizar a tarefa alocada na maior parte das vezes. Em um outro cenário, consideramos que mais de um participante encontra-se associado a uma tarefa. A colaboração é a regra, pois o plano prevê que serão necessárias várias pessoas para cumprir a tarefa. Neste trabalho, este segundo cenário se torna importante quando o plano aloca pessoas dispersas geograficamente.

Outros trabalhos tratam das peculiaridades de processos de software globalizados (MAIDANTCHICK e ROCHA, 2002). Consideramos que o apoio por parte de processos já está resolvido no ambiente de trabalho destas equipes distribuídas quando se inicia a colaboração entre os participantes. Consideramos, também, que cada participante possui uma visão clara de suas tarefas e responsabilidades, que é definida pela gerência local do projeto ao qual o participante se encontra alocado.

### 2.5.2 Incentivos para a Participação

O esforço necessário para a inclusão e atualização da informação em um ambiente colaborativo pode ser alto (GRUDIN, 1994). Em diversos casos, determinados

participantes podem se sentir desmotivados ou explorados pelo grupo ou por outros participantes.

Na prática do desenvolvimento de software, podemos citar o mecanismo de incentivo e recompensa à participação do fórum de desenvolvedores Java. Ao ingressar no sistema, em uma analogia com um mercado auto-regulado, cada participante recebe uma quantidade determinada de uma unidade monetária fictícia (*duke dollar*). Esse mecanismo é descrito como um sistema de preços (*price system*) (HAYEK, 1945). Cada participante determina localmente o valor e a necessidade de um recurso. No sistema de preços, não existe uma entidade que centralize o planejamento nem um plano pré-definido. Essa abstração é utilizada para justificar o mercado de capitais e a livre iniciativa.

No caso dos *dukes*, o ambiente colaborativo permite "quantificar" o valor que cada participante atribui a um recurso. Essa moeda é utilizada para regular e incentivar a participação no ambiente. O participante que necessita de auxílio formula seu problema e oferece certa quantia como pagamento pela solução. O participante que contribuir com a solução pode utilizar a quantia recebida para oferecer um prêmio maior para a solução de seu próprio problema. Assim, quanto mais participativo e efetivo na colaboração, maior o seu potencial em atrair outro participante para resolver seu problema. Logicamente, são estabelecidos limites para a quantia oferecida para a solução de um problema e para o acúmulo individual, evitando uma disparidade muito grande entre os participantes.

### 2.5.3 Comunidades Auto-reguladas

Grupos que não dispõem de uma área de trabalho física comum precisam repensar as suas relações de hierarquia e controle. A distância entre os participantes dificulta a existência de um controle centralizado, o que muitas vezes leva a alguma forma de auto-organização espontânea.

Na proposta de translucência social (*social translucence*) (ERICKSON e KELLOG, 2000), a auto-organização é resultante do equilíbrio entre as intenções dos participantes em um meio de colaboração com baixa coordenação (*coordination*), alta percepção (*awareness*) e registro de ações (*accountability*). Nessas condições, Erickson e Kellog advogam que participantes bem intencionados estão sujeitos a "pressão social"

dos demais. A alta percepção oferece a informação para o planejamento de ações individuais com base na informação das ações dos demais participantes e o registro das ações executadas, disponível para todos os participantes, evita que um participante atue com a intenção de prejudicar outro participante. Esse tipo de organização é fundamentado em princípios otimistas. Os participantes se importam com a sua imagem perante aos demais e não tem a intenção de causar danos.

No desenvolvimento de software, encontramos uma situação similar em um tipo de sistema *Web*. O *WikiWikiWeb* (CUNNINGHAM e LEUF, 2001) é uma extensão para o servidor de hipertextos que possibilita a escrita de conteúdo através da própria página de visualização. Essa extensão favorece a autoria colaborativa assíncrona.

Do ponto de vista técnico, o sucesso do *WikiWikiWeb* é decorrente da simplicidade da sua implementação e da compatibilidade com diversos servidores e qualquer cliente que opere linguagem de marcação de hipertextos (*Hypertext Markup Language* – HTML).

Diversas comunidades que trabalham com desenvolvimento de software adotam o *WikiWikiWeb*. Possivelmente, a mais conhecida seja a *Portland Patterns Repository Wiki* (CUNNINGHAM, 2003). Outro exemplo, para o público em geral é a enciclopédia *on-line Wikipedia*.

Três aspectos são relevantes na extensão *WikiWikiWeb*. Primeiro, a auto-organização do conteúdo que emerge do conjunto de contribuições individuais. Segundo, a ausência de controles de acesso e censura centralizada. Os próprios participantes excluem de forma autônoma e isolada o material que julgam que não deve pertencer ao texto. Em terceiro lugar, está a ausência de uma administração ou representação da organização que mantém a instalação *WikiWikiWeb*.

O funcionamento dos *Wiki* é uma demonstração das capacidades de auto-organização que são necessárias para a criação de uma comunidade virtual. Os fundamentos da auto-organização são divulgados através de páginas que contém orientações dos criadores do sistema. Em mais de dez anos de existência, poucos atos de vandalismo ou incidentes foram registrados.

### 2.5.4 Ambientes com Base na *Web*

Este tipo de ambiente está sendo adotado principalmente no desenvolvimento de projetos de código aberto. Empresas de grande porte, como *Oracle* e *Nokia*, também começam a adotar este tipo de infra-estrutura colaborativa com base na *Web*.

Os dois ambientes de uso mais difundidos são o *SourceForge* (VA SOFTWARE, 2003) e o *SourceCast* (COLLABNET, 2003). Os serviços e arquitetura de ambos são similares. O serviço básico é formado por uma ferramenta de controle de versões de código fonte, adaptada para a obtenção de diversos tipos de artefatos através da *Web*. Essa estrutura facilita em maior grau a difusão de artefatos e em menor grau o seu compartilhamento.

## *2.6 Apoio à Cooperação entre Participantes*

O produto de software é uma construção coletiva. Diversos desenvolvedores contribuem mesmo para o menor dos programas, visto que as bibliotecas de programação e diversas ferramentas e sistemas necessários para a execução dos programas são criados por diferentes autores. O software de maior porte é resultado do esforço de uma equipe. Portanto, deve existir apoio suficiente para a cooperação entre os desenvolvedores, especialmente, para o compartilhamento de artefatos.

Esta seção apresenta o banco de dados do projeto de software, o repositório, e alguns fatores relacionados com o seu uso.

### 2.6.1 Repositórios

Um repositório é um banco de dados voltado para apoiar as necessidades de armazenamento de um projeto de engenharia. O repositório atende às necessidades de armazenamento dos diferentes componentes de um produto de software e de suas respectivas versões. Repositórios estão, freqüentemente, associados com a provisão de uma área de trabalho (*workspace*) individual que oferece ao desenvolvedor os componentes necessários a sua tarefa (ESTUBLIER, 2000).

Existem diversos produtos que implementam esses serviços. Ferramentas comerciais incluem o *Microsoft Source Safe*, o *Rational ClearCase* e o *Merant PCVS*.

Na prática corrente de desenvolvimento de software, o Sistema de Versões Concorrentes (CVS - *Concurrent Versioning System*), é a base para o chamado desenvolvimento em código aberto (*open source development*) (FOGEL e BAR, 2001). As características dessas ferramentas são muito similares e incluem algum apoio à colaboração entre os seus usuários. Existem três apoios que se destacam: (a) a notificação de alterações sobre componentes, (b) a resolução de conflitos de atualização e (c) a coordenação de ações com e sem o uso de bloqueios.

Entretanto, apesar da disponibilidade dos mecanismos para colaboração, os desenvolvedores enfrentam outros problemas de natureza não-técnica. De uma maneira geral, os mecanismos de apoio à colaboração existem, mas são preteridos pela sua complexidade ou substituídos pela interação face-a-face que é decorrente do trabalho em situação de proximidade física (HERBSLEB *et al.*, 2001).

Na prática, os componentes configurados correspondem a arquivos. Existem alternativas em desenvolvimento que possibilitam o trabalho sobre unidades menores ou maiores, levando em conta a estrutura lógica do componente (ver Seção 2.6.5).

### 2.6.2 Notificação de alterações

Os repositórios possibilitam que os participantes sejam notificados de alterações sobre um determinado artefato. A ativação desse mecanismo de notificação (*watcher*) deve ser solicitada pelo participante. Em comparação com as demais funcionalidades do repositório, a notificação é pouco utilizada. Fogel e Bar (2001) justificam que usuários típicos utilizam apenas as funcionalidades essenciais do repositório.

O principal motivo para o pouco uso das notificações está na descontextualização da notificação em relação à atividade do desenvolvedor. Ao solicitar o monitoramento do artefato, o desenvolvedor será continuamente informado de alterações. Essas alterações chegam a ele através de correio eletrônico ou através de meio similar, sem levar em conta a tarefa que está sendo realizada pelo desenvolvedor no momento da recepção do aviso. Em outras palavras, o mecanismo de notificação não considera que o desenvolvedor pode não ter interesse nessa informação quando está realizando uma tarefa que não se refere ao artefato monitorado. De forma similar, o desenvolvedor evita colocar avisos, pois não consegue prever com exatidão quanto

tempo irá durar o seu interesse na notificação.

Além da barreira para o uso do mecanismo por parte dos desenvolvedores, temos o problema do isolamento imposto pelo repositório. Entre a retirada e a devolução do artefato, o repositório não é capaz de informar sobre as operações que são realizadas no artefato. Apenas na devolução do artefato, os demais desenvolvedores poderão ter acesso à modificação. Também, não é possível informar quando as alterações serão devolvidas ao repositório, nem se estas serão simplesmente descartadas pelo desenvolvedor.

### 2.6.3 Resolução de Conflitos

Na prática, a comunicação entre os participantes é utilizada basicamente para resolver conflitos. Em geral, os desenvolvedores adotam uma postura ativa no desenvolvimento de soluções inovadoras que são apresentadas e discutidas apenas quando já existe um protótipo operacional.

Quando as alterações realizadas são incompatíveis com o trabalho de outros desenvolvedores, o repositório notifica o participante cuja ação gerou o conflito, enviando a ele as versões relevantes do artefato. O sistema aguarda que o participante corrija os problemas e submeta a requisição novamente.

O participante deve entrar em contato com os demais envolvidos por iniciativa própria. Na prática, entretanto, a comunicação não ocorre. O participante acaba por resolver sozinho o conflito existente. Estabelecer um canal de comunicação com outro participante não é um problema, devido às facilidades que existem no ambiente de proximidade física e os meios de comunicação atuais. Uma grande dificuldade é a contextualização (recomposição) do problema para um novo participante (GRINTER, 2001). Um grande obstáculo para a comunicação é a localização de um participante disponível que possua conhecimento que possa acrescentar algo à solução do conflito.

Ao minimizar a comunicação, o participante reduz a interrupção do trabalho dos demais participantes. O compromisso entre interromper o trabalho de um colega ou interromper o seu próprio trabalho é uma decisão difícil de ser realizada (HUDSON *et al*., 2002). É coerente afirmar que um grupo é mais produtivo quando todos os participantes possuem as condições necessárias para realizar suas tarefas. A interrupção temporária de um ou mais participantes pode ser a condição necessária para restabelecer

o grupo a sua capacidade máxima de trabalho. Por exemplo, se um participante mais experiente é interrompido para auxiliar um participante menos experiente, ao final o grupo gerará mais resultados, pois ambos podem prosseguir seu trabalho de forma mais produtiva após a interrupção. Este resultado é melhor do que o que seria obtido se apenas o participante experiente realizasse seu trabalho.

### 2.6.4 Coordenação de Ações

A coordenação das ações de cada participante pode ser realizada de forma pessimista ou otimista. Na forma pessimista, assume-se que as ações serão incoerentes e atingirão um estado inválido e indesejado. Neste caso, a ação de um participante deve bloquear as ações dos demais sobre o mesmo artefato. Na forma otimista, assume-se que as ações serão coerentes, mesmo ocorrendo de forma isolada, e o resultado final será um estado válido. Neste caso, mais de um participante realiza ações sobre um mesmo artefato.

O participante deve solicitar um bloqueio (trava, tranca, do inglês *lock*), se deseja trabalhar em isolamento sobre um determinado artefato. O uso de bloqueios é a estratégia preferida pelos participantes para a realização de um conjunto de alterações relacionadas. O uso de bloqueios impossibilita o paralelismo de atividades que concorrem pelo artefato bloqueado. Em geral, os participantes adotam a estratégia de geração de cópias do artefato (versões), que são trabalhadas em paralelo e posteriormente consolidadas em uma única versão em um momento futuro.

Em resumo, o bloqueio evita a ocorrência de conflitos, mas reduz o paralelismo no trabalho; o uso de cópias apenas posterga a detecção dos conflitos, mas possibilita um maior paralelismo de atividades.

### 2.6.5 Unidades de Trabalho

O tamanho e as características das unidades de trabalho do repositório determinam o menor item que pode ser compartilhado entre os desenvolvedores. Na maior parte dos repositórios, a unidade de trabalho é o arquivo. Existem propostas de repositórios que trabalham com unidades menores ou maiores, mas ainda não são amplamente adotadas na prática. O arquivo permanece a unidade de trabalho mais popular, pois a maioria das ferramentas de transformação e edição opera com essa

mesma unidade.

Quando o item do repositório é diferente de um arquivo, temos a ocorrência de multidimensionalidade, ou seja, é possível trabalhar sobre unidades lógicas que diferem da unidade utilizada no armazenamento. Chu-Carrol *et al.* (2002) argumentam que, em situações de distribuição, é necessário adotar a multidimensionalidade para permitir que os participantes tenham uma visão lógica dos artefatos que seja mais rica em semântica.

Existe certa discordância sobre a granularidade dos artefatos na literatura. O Coven (CHU-CARROL *et al.*, 2002) é um sistema de gerência de configuração que possibilita operar sobre unidades menores do que um arquivo completo. Por outro lado, a prática de DBC aponta na direção oposta, por exemplo, repositórios de componentes costumam indexar componentes que são compostos por diversos arquivos em uma única unidade de trabalho. Neste contexto, alguns trabalhos advogam que no desenvolvimento distribuído, o produto é integrado através de unidades maiores que um arquivo, ou seja, um componente em um de seus diversos formatos (ESTUBLIER *et al.*, 2001).

Argumentamos que a unidade de trabalho depende da representação utilizada na tarefa. Portanto, para decidir qual a unidade de trabalho utilizar, é necessário entender o contexto de trabalho do participante. Por exemplo, um desenvolvedor que tenta montar uma aplicação com base em componentes está interessado em unidades de trabalho maiores, o desenvolvedor do componente pode estar interessado na estrutura interna do componente de software, ou seja, em unidades menores.

Na prática, as aplicações é que são capazes de combinar ou dividir arquivos para obter a representação lógica dos artefatos adequada para as tarefas (ESTUBLIER, 2000).

## *2.7 Desafios do Desenvolvimento Globalizado de Software*

A literatura em geral destaca os problemas na prática do desenvolvimento globalizado de software. Minimizar esses problemas que são causados pelas dificuldades de comunicação e da percepção do grupo e do trabalho são os principais desafios do apoio ao desenvolvimento globalizado. Estudos empíricos (HERBSLEB *et al*.,2001) indicam que cinco problemas ocorrem com freqüência:

**Perda de contexto**: um desenvolvedor envia mensagens em um contexto de

trabalho para que seja recebida por outro desenvolvedor, provavelmente em um contexto diferente. Por exemplo, se o remetente faz referências a objetos em sua área de trabalho, o destinatário pode ter dificuldades em compreender a mensagem.

**Dificuldade de coordenação de ações**: as ações são normalmente acordadas previamente, entretanto, mudanças e incertezas fazem com que os desenvolvedores alterem os acordos firmados para se adequar a uma situação imprevista. Por exemplo, pode ser necessário adiar uma ação por causa de uma sobrecarga de trabalho em um outro projeto.

**Sobreposição e perda de trabalho**: As alterações realizadas no produto de software podem passar despercebidas devido à falta de sincronização entre as cópias. Por exemplo, um desenvolvedor pode ser obrigado a corrigir um componente defeituoso para que seu componente de software possa ser terminado. O responsável pelo módulo defeituoso pode já ter realizado o conserto ou pode ignorar o conserto realizado pelo colega.

**Dificuldades em localizar colaboradores**: em uma equipe virtual, torna-se mais difícil encontrar e entrar em contato com os desenvolvedores responsáveis por um componente de software.

**Dificuldades em evoluir artefatos de forma consistente**: por causa da baixa comunicação, os desenvolvedores que trabalham em um componente de software podem divergir em relação ao que deve ser realizado. Por exemplo, um desenvolvedor pode desejar criar poucos métodos com poucos argumentos enquanto outro acredita que seja melhor criar um número maior de métodos mais coesos e com um menor número de argumentos. Sem um acordo entre os dois, sucessivas versões do componente vão divergir nessa métrica, dificultando a evolução do artefato.

## *2.8 Conclusão*

Este capítulo apresentou o contexto desta tese. Foram apresentados um breve histórico e algumas definições importantes relacionadas com o desenvolvimento distribuído de software. Houve um esforço em diferenciar cenários do desenvolvimento distribuído e em apresentar o papel da reutilização de software na composição desses

cenários. Em seguida, foram apresentados dois aspectos importantes do apoio oferecido pelos ambientes de desenvolvimento que são o apoio à coordenação de tarefas e o apoio à cooperação entre participantes. O primeiro aspecto representado, principalmente, pelo processo de software e o segundo pelo repositório de software. Foram apresentados, também, alguns dos problemas existentes no desenvolvimento de software distribuído, apesar do apoio existente nos ambientes de desenvolvimento.

Foi apresentado, principalmente, o estado da arte e também da prática em desenvolvimento de software, com base na perspectiva tradicional de desenvolvimento de software como uma produção industrial. Essa perspectiva industrial é fundamental para organizar o desenvolvimento em grande escala, entretanto, os benefícios decorrentes dessa perspectiva já estão sendo explorados pela maioria das empresas que desenvolve software. Portanto, é necessário manter o apoio existente nos ambientes de desenvolvimento de software, mas ao mesmo tempo buscar novas perspectivas que explorem outras características inerentes ao desenvolvimento de software. Por exemplo, a característica colaborativa inerente ao processo de desenvolvimento de software favorece a perspectiva de um incentivo maior à colaboração entre os desenvolvedores, de forma a complementar a perspectiva tradicional de desenvolvimento. Entre duas organizações que desenvolvem software de forma distribuída, terá mais chance de sucesso aquela que conseguir incentivar seus desenvolvedores a uma maior colaboração.

No capitulo a seguir, algumas das propostas para apoio à colaboração no desenvolvimento globalizado de software são discutidas e utilizadas como exemplos de aplicação de apoio à percepção ou à colaboração no desenvolvimento de software.

# 3 Apoio à Colaboração no Desenvolvimento Distribuído de Software

Este capítulo apresenta uma análise crítica de práticas e ferramentas utilizadas no desenvolvimento de software que visam a coordenação de esforços das equipes envolvidas. A colaboração é necessária devido ao modo de produção atualmente empregado no desenvolvimento de software: cada participante está encarregado de uma especialidade e o trabalho concorrente de diversos participantes é necessário para completar o produto de software.

Este capítulo encontra-se organizado da seguinte forma. São apresentados três conceitos fundamentais para o entendimento do apoio oferecido pelos ambientes de desenvolvimento colaborativos estudados: a necessidade de apoio à tarefa e à colaboração (Seção 3.1), os modelos de percepção (Seção 3.2) e os modos de colaboração (Seção 3.3). Uma descrição breve dos ambientes revisados é realizada, seguida de um resumo dos ambientes (Seção 3.4), um modelo de características para o domínio de aplicação (Seção 3.5) e uma crítica das deficiências no apoio à colaboração e a sua abordagem de desenvolvimento nos ambientes analisados (Seção 3.6). Por fim, apresenta-se a conclusão deste capítulo (Seção 3.7).

## *3.1 Apoio à Tarefa e à Colaboração*

O apoio à colaboração é importante para satisfazer parte das necessidades de uma equipe de desenvolvimento. O objetivo da equipe é desenvolver software, portanto, o apoio à tarefa de desenvolvimento é fundamental. O apoio à colaboração é, normalmente, uma necessidade secundária, mas que se torna essencial quando se trata de equipes distribuídas.

As propostas de ferramentas e ambientes podem ser classificadas em relação a essas duas dimensões: apoio à colaboração e apoio à tarefa. Considerando as ferramentas revisadas neste capítulo, propomos uma classificação em quatro quadrantes

(Tabela 3-1). No primeiro quadrante, temos a situação na qual o potencial de apoio às duas dimensões é baixo. As ferramentas características desse quadrante são transformadores automáticos e outras ferramentas de baixa interatividade, como os compiladores de linguagens de alto nível. O exemplo usado é o *Java Development Kit* (JDK). Embora seja possível construir um ambiente de desenvolvimento simples com base em um editor de textos genéricos e um compilador, observa-se que as ferramentas utilizadas por profissionais encontram-se no quadrante ao lado.

No segundo quadrante, encontram-se algumas ferramentas de desenvolvimento características na modelagem e programação de software. Neste quadrante, aparecem as primeiras ferramentas que trabalham com gráficos ou modelos, assinaladas com um quadrado vazado.

**Tabela 3-1 Classificação de algumas ferramentas distribuídas nas dimensões de apoio à colaboração e apoio à tarefa.**

| Apoio à colaboração | Apoio à tarefa | |
|---|---|---|
| | Baixo | Alto |
| Alto | CO2DE, REDUCE, BSCW | Serendipity, DARGUML, Tukan, Palantir, Odyssey Share |
| Baixo | COAST | ARGOUML, Poseidon, Eclipse, Visual Smalltalk, Odyssey |

Três ferramentas são destacadas no quadrante de alto potencial para colaboração e baixo apoio à tarefa: BSCW (APPELT, 1999), REDUCE (CHEN e SUN, 1999) e CO2DE (MEIRE *et al.*, 2003). O BSCW oferece um conjunto de ferramentas de apoio à colaboração de propósito genérico. Diversas tarefas podem ser realizadas com o uso deste ambiente, mas sem que o usuário perceba o apoio específico ao seu domínio de aplicação. O REDUCE é um editor de textos colaborativo clássico que permite a edição síncrona de textos. O apoio à colaboração oferecido por este editor está classificado como mais elevado do que o apresentado pelo BSCW, pois apresenta mecanismos mais sofisticados de visualização, controle de concorrência e distribuição.

O CO2DE é um editor gráfico colaborativo que apresenta um maior apoio à tarefa do que as das ferramentas anteriores. A principal restrição do CO2DE em relação à tarefa se refere à baixa complexidade do editor de diagrama de classes implementado no protótipo. Neste quadrante, nota-se que existe um maior apoio potencial para a tarefa, à medida que a ferramenta se torna mais específica e restrita a uma tarefa no domínio de aplicação.

O quarto quadrante apresenta ferramentas com alto apoio tanto para colaboração quanto para a realização da tarefa. Essas ferramentas serão apresentadas em maiores detalhes na Seção 3.4. Nota-se que o apoio necessário ao desenvolvimento de software na situação de distribuição encontra-se neste quadrante superior direito.

## *3.2 Modelos de Percepção*

A percepção é um dos elementos fundamentais da colaboração (DOURISH e BELOTTI, 1992). A percepção se refere à capacidade de um elemento do grupo em entender o seu próprio contexto e a sua conexão com o contexto dos demais durante a realização de uma tarefa. Existem diversas propostas de percepção, por exemplo: percepção de área de trabalho (GUTWIN e GREENBERG, 1998), percepção de grupo (BEGOLE, 1999) e percepção de produto (FARSHCHIAN, 2001). A informação de percepção é utilizada para comunicar parte do estado do ambiente que interessa ao indivíduo. O estado do ambiente é um conjunto de fatos que podem ou não ser do conhecimento do indivíduo envolvido na colaboração. Para organizar o conjunto de fatos sobre o ambiente, é necessário criar um modelo de percepção desse ambiente. O modelo de percepção descreve o vocabulário que pode ser utilizado para comunicar fatos e detectar alterações sobre esses fatos. Por exemplo, para incentivar a percepção de área de trabalho utilizam-se informações de percepção que comunicam a posição das janelas, do indicador do mouse e de outros elementos relacionados com a representação do ambiente de janelas. O modelo de percepção, neste caso, deve ser suficiente para representar esse tipo de informação.

Rodden (1996) propôs o modelo de percepção espacial como uma forma de organizar a relação entre objetos em visualizações tridimensionais. Schummer (2001) argumenta que os artefatos são programas de computador podem ser organizados por meio de um modelo de percepção espacial. Schummer argumenta que as relações

expressas em um programa podem ser organizadas de forma a representar um espaço de software. A limitação dessa proposta é que o espaço de software apresenta mais de três dimensões, pois as relações existentes em um programa são várias. Portanto, o espaço de software não apresentaria a mesma familiaridade com o modelo tridimensional de um prédio. As distâncias entre os elementos do espaço de software não apresentam as propriedades de um modelo cartesiano. Por exemplo, uma classe apresenta certa dependência com sua superclasse. Entretanto, a relação no sentido inverso não se mostra igual: a superclasse não depende de suas subclasses.

Apesar desse problema, a idéia de espaço de software oferece um modelo de percepção que pode ser explorado para oferecer parte da informação necessária para os modelos de percepção de produto e de grupo.

Entretanto, nota-se que, dependendo da tarefa, existe a necessidade de complementar o espaço de software, principalmente com informações de visualização. Por exemplo, a percepção de área de trabalho tem por base informações sobre os elementos que constam na interface de usuário da aplicação. Nesse caso, é importante obter informações sobre a posição de janelas, ponteiros, barras de rolagem e outros elementos que não tem lugar na abstração do espaço de software. Com isso, argumentamos que a natureza da tarefa influencia no modelo de percepção que deve ser explorado.

## 3.3 Modos de Colaboração

Aparte ao apoio esperado para a realização de atividades colaborativas, Schummer e Haake (2001) propõem seis modos de colaboração (MoC – *Modes of Collaboration*): isolamento, nível de processo, nível de mudanças, nível de presença, comunicação e colaboração de alto acoplamento (Tabela 3-2). No modo de **isolamento** (*offline)*, o participante ou um grupo de participantes sinaliza que não deseja ser interrompido por eventos externos. Os autores argumentam que este modo de colaboração, ou melhor, de não-colaboração, é necessário para proporcionar o isolamento e a concentração necessários para a solução de problemas complexos. A comunicação assíncrona encontra-se disponível, caso o participante isolado necessite entrar em contato com outros participantes.

**Tabela 3-2 Modos de cooperação (MoC) propostos por Schümmer e Haake (2001).**

| *Modo (tradução)* | *Descrição* |
|---|---|
| Isolamento (*Offline*) | Nenhuma ação é monitorada. Contato através do correio eletrônico. Tempo de resposta indeterminado. |
| Nível de processo (*Process level*) | O sistema informa a tarefa de cada participante. |
| Nível de mudanças (*Change level/aware*) | Ações são registradas e comunicadas aos usuários interessados. |
| Nível de presença (*Presence level/aware*) | Informação sobre a visão do usuário é compartilhada. |
| Nível de comunicação (*Communication*) | Um usuário pode ser contactado de forma síncrona para responder questões. |
| Colaboração em alto acoplamento (*Tightly-coupled collaboration*) | Usuários podem participar de sessões colaborativas. |

No modo **nível de processo** (*process level*), o participante interage com uma interface que permite realizar o planejamento de tarefas. Essas informações sobre o que o participante pretende fazer são compartilhadas com os demais. Os participantes compartilham apenas a informação sobre o planejamento, sem detalhes sobre os artefatos concretos que são manipulados nas tarefas.

A informação sobre os artefatos aparece no terceiro modo de colaboração proposto, **nível de mudanças** (*change level*). Neste nível, todas as alterações do usuário são registradas para auxiliar o entendimento da evolução da edição e para realizar a detecção de conflitos de edição com outros participantes.

No nível de **presença** (*presence level)*, um conjunto maior de operações é monitorado, por exemplo, a visualização de artefatos também é registrada. Essa informação é passada para os demais participantes, que percebem a presença de um outro participante operando sobre o mesmo documento. Ao detectar outro participante, existe a opção de avançar para um dos demais modos de colaboração mais integrados.

No modo de **comunicação** (*communication),* os participantes podem trocar mensagens para discutir suas intenções sobre o artefato em comum e decidir por trabalharem juntos. No modo de **colaboração de alto acoplamento** (*tightly-coupled collaboration),* os participantes podem operar o ambiente de forma compartilhada, como se estivessem compartilhando os dispositivos de entrada de uma mesma estação.

Os modos de colaboração do *Tukan* (SCHÜMMER e HAAKE, 2001) são

usados para descrever diversas situações de trabalho e o apoio à colaboração necessário a cada uma delas. A sobreposição dos modos de colaboração, que pode ocorrer em um ambiente com mais de dois participantes, deixa claro os limites dessa classificação. Cada participante pode estar em uma sobreposição de modos de colaboração com o mesmo ou com diversos participantes.

Cada modo de trabalho possui um apoio à percepção característico. A abordagem apresentada nesta tese tenta facilitar a transição entre os modos de colaboração, com a proposta de uma arquitetura que permite desenvolver diferentes tipos de apoio à percepção que possam utilizar uma mesma massa de dados.

## 3.4 Exemplos de Ambientes Colaborativos Existentes

Esses ambientes são motivados pelas necessidades de equipes de desenvolvimento distribuídas, ou virtuais, característicos da globalização do desenvolvimento de software. Entre os requisitos desses ambientes, encontram-se: apoio à encenação de processos em equipes distribuídas, o incentivo à comunicação e à socialização, apoio à atividade individual e em pequenas equipes (2 a 6 pessoas). Tais ambientes objetivam uma maior eficácia e eficiência no desenvolvimento de software e, sobretudo, uma maior satisfação dos participantes (WERNER *et al*., 2002).

Alguns desses ambientes foram propostos em contextos diferentes e adaptados para o trabalho em situações de distribuição temporal ou geográfica. A revisão buscou grupos em atividade que apresentem propostas recentes. Na maior parte das vezes, o trabalho identificado faz parte de uma linha de pesquisa com vários anos de existência em um grupo interessado em CSCW e Engenharia de Software.

**Tabela 3-3 Literatura utilizada na revisão dos ambientes colaborativos.**

| *Ambiente* | *Resumo* |
|---|---|
| gSEE | Ambiente de uso geral, com base em tecnologia da *Web*. |
| MILOS | Ambiente integrado de uso geral que oferece apoio genérico, porém limitado, a diversos modos de colaboração. |
| *Tukan* | Ambiente com apoio para tarefas em alto grau de acoplamento, com apoio à programação *Smalltalk*. |
| *Serendipity* | Ambiente com apoio para tarefas em alto grau de acoplamento, com apoio à diagramação de modelos entidade-relacionamento. |
| *Palantír* | Ambiente com apoio para percepção de mudanças, integrado a um ambiente de desenvolvimento convencional. |

Nas próximas seções, são apresentados cinco exemplos de ambientes colaborativos (Tabela 3-3). Essa revisão oferece uma idéia do tipo de apoio à colaboração que vem sendo proposto para o desenvolvimento de software.

### 3.4.1 MILOS/MILOS XP/PRIME

O MILOS (*Minimally Invasive Long-Term Organizational Support*) é uma plataforma que oferece apoio à coordenação e colaboração de equipes de engenheiros de software (MAURER *et al.*, 2000). Nesta plataforma, estão sendo desenvolvidos diversos trabalhos pela equipe da Universidade de *Calgary*, Canadá. Um dos trabalhos mais recentes do grupo é o MILOS XP (MAURER e MARTEL, 2002). Maurer e Martel apresentam o MILOS XP como um ambiente orientado para metodologias ágeis de desenvolvimento de software (COCKBURN, 2001), em particular, o uso das técnicas de *Extreme programming* (XP) (BECK, 1999).

O sistema MILOS XP pode ser dividido em dois componentes complementares: apoio ao processo de software e compartilhamento de aplicações. No primeiro, o participante encontra apoio para a descrição de documentos que fazem parte de um processo de desenvolvimento de software ágil. O MILOS XP mantém a orientação voltada a processos (*workflow*) do ambiente MILOS original. Entretanto, as atividades são tratadas como "tarefas", que representam unidades de trabalho de menor duração e complexidade, coerente com a orientação do planejamento em XP.

Este primeiro componente é construído para apoiar a metodologia de programação, de acordo com a tarefa a ser realizada. Os documentos são formados por campos que são preenchidos com dados estruturados e semi-estruturados. Através de um ambiente centrado na *Web*, que apresenta uma arquitetura cliente-servidor, esses documentos encontram-se disponíveis através da rede. O objetivo dos participantes é preencher esses documentos da melhor forma possível em duplas. O trabalho em pares, em condição de distribuição, ocorre com o apoio do segundo componente do MILOS XP.

O segundo componente do MILOS XP possibilita que dois participantes compartilhem aplicações em uma arquitetura de distribuição ponto-a-ponto (*peer-to-peer*). O ambiente de desenvolvimento colaborativo se torna completo para a tarefa com a ativação das ferramentas convencionais disponíveis nas estações locais de trabalho de cada participante.

Em trabalhos iniciais, o segundo componente foi realizado com o uso do aplicativo *Microsoft NetMeeting*, que habilita serviços de compartilhamento de aplicações, videoconferência e troca de mensagens de texto. Trabalhos mais recentes do mesmo grupo investigam a proposta de uma implementação própria desse segundo componente (BOWEN e MAURER, 2002).

No *NetMeeting*, apenas um dos participantes tem o direito de escrita. Os demais participantes possuem apenas a visualização. O direito de escrita pode ser transferido de um participante para outro. Dessa forma, um participante realiza o preenchimento enquanto os demais podem comentar ou propor alternativas. Este arranjo é similar ao que ocorre na técnica de *extreme programming* em situações não distribuídas.

Os trabalhos no MILOS apontam alguns requisitos para o apoio de equipes virtuais, considerando tarefas relacionadas com processos ágeis (MAURER e MARTEL, 2002): coordenação, comunicação síncrona, notificação ativa e roteamento de informação, integração de gestão de conhecimento e encenação de processos.

Os dois últimos requisitos são mais bem desenvolvidos no ambiente MILOS/PRIME (*Process-Oriented Resource Information Management Environment*) (HOLTZ e MAURER, 2002).

A memória de grupo no MILOS/PRIME é formada pelo registro combinado de todos os documentos disponíveis no ambiente. Em especial, o conteúdo de discussões no ambiente é realizado via *Web*.

Dentro da classificação MoC, proposta por Schummer, o MILOS apresenta apoio para todos os modos de colaboração, embora não apresente uma característica marcante em nenhum deles, portanto, o MILOS é um sistema de uso geral.

### 3.4.2 Serendipity

O *Serendipity* é um ambiente para desenvolvimento de software que considera, principalmente, a colaboração em tarefas que envolvem a visualização coletiva e distribuída de artefatos (GRUNDY *et al.*, 2000). O editor colaborativo de diagramas de entidade e relacionamentos é um dos ambientes descritos nos trabalhos sobre o *Serendipity*.

Os componentes de software responsáveis pela visualização colaborativa de artefatos foram recentemente reescritos da linguagem C para a linguagem Java. Existem alguns protótipos em desenvolvimento que utilizam esses componentes batizados de *JView* e *JComposer* (GRUNDY e HOSKING, 2002). O primeiro componente, *JView*, atua capturando eventos da interface gráfica de usuário. O segundo componente é utilizado para determinar se os eventos coletados localmente devem ser propagados para as outras estações de trabalho.

A principal contribuição desta abordagem é a capacidade do ambiente em evoluir através do acréscimo de novos componentes. Alguns desses componentes estão relacionados com a colaboração em equipes virtuais. No *Serendipity*, a colaboração é apoiada com a adição de funções às ferramentas já instaladas no ambiente, ao invés de substituir uma ferramenta por uma versão colaborativa. Dessa forma, as melhorias no ambiente ocorrem de maneira integrada, onde todas as ferramentas são beneficiadas pelo novo componente introduzido.

A configuração da propagação de eventos é configurada pelo próprio usuário com o uso de uma interface gráfica (**Erro! A origem da referência não foi encontrada.**). Por outro lado, a principal limitação dos componentes está na sua aplicação em ferramentas disponíveis apenas no próprio *Serendipity*. Apesar das

características interessantes da configuração da colaboração no ambiente pelo próprio usuário e pela definição de componentes de colaboração, o apoio ao desenvolvimento de software permanece apoiando a modelagem de diagramas Entidade-Relacionamento, que está sendo atualmente substituída pela modelagem com a UML (*Unified Modeling Language*, OMG, 2003).

O *Serendipity* não apresenta apoio ao nível de processo, mas apresenta apoio aos demais modos de colaboração. O destaque do ambiente é a colaboração em alto acoplamento por meio de um editor colaborativo.

### 3.4.3 Tukan

Schummer e Schummer (*1999*) propõem a integração de apoio a informações de percepção em um ambiente de programação *Smalltalk*, chamado *Tukan*. A programação do *Tukan* é realizada por meio da biblioteca para programação de aplicações colaborativas COAST (*Cooperative Application Systems Technology*) (SCHUCKMANN *et al*., 1996). Uma versão em linguagem Java do COAST foi iniciada em 2003, mas não houve relato de sua conclusão. Os serviços do COAST auxiliam a programação do compartilhamento consistente de artefatos que são acessados em uma sessão colaborativa. A infra-estrutura do COAST também oferece apoio à percepção e à comunicação.

A maior parte das características do COAST é percebida em suas aplicações e o Tukan é uma dessas aplicações. Comentamos a seguir o *Tukan* como forma de apresentar os serviços do COAST e do *Tukan* ao mesmo tempo.

O ambiente visa o apoio de pequenas equipes de programadores a colaborar à medida que realizam alterações em um mesmo sistema. O ambiente oferece funcionalidades para a edição colaborativa de código fonte. A implementação de um ambiente de programação *Smalltalk* foi alterada para permitir este tipo de colaboração. Os elementos apresentados na interface de usuário são familiares ao programador *Smalltalk*.

A percepção e a coordenação na realização da tarefa são incrementadas através de um protocolo social apoiado nas indicações do teleapontador (*telecursor*) de cada participante. O teleapontador é um recurso muito utilizado em sistemas de

videoconferência, que possibilita perceber a posição do indicador do mouse de participantes remotos. Os participantes podem se comunicar através de mensagens instantâneas durante a edição do documento.

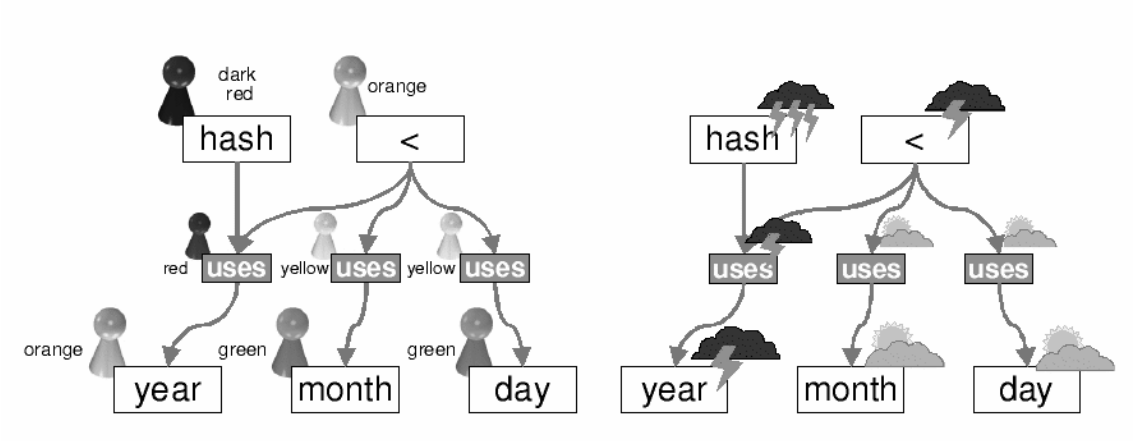



**Figura 3.1 Ícones para apresentação de informações de percepção (SCHÜMMER e HAAKE, 2001).**

A segunda contribuição do *Tukan* é a descrição do espaço de software, ou seja, das relações que existem entre os elementos que compõem a descrição do programa que está sob edição. A alteração em um artefato pode ocasionar a propagação de alterações sobre o código em todos os pontos onde a alteração tiver uma dependência com o restante do programa. No exemplo (Figura 3.1), uma alteração nos parâmetros dos métodos *hash* e *menor-que* (<) propaga alterações para outros métodos. No *Tukan*, as alterações são detectadas pelo compilador *Smalltalk* e apresentadas na própria interface do ambiente de edição. Cada método é considerado um artefato independente, portanto, as distâncias são calculadas e apresentadas em relação aos métodos. O programador, ao perceber os ícones no canto superior direito, apreende que existem alterações realizadas por outro programador que estão em conflito com as suas alterações correntes.

O *Tukan* não oferece notificação fora do modo de colaboração de alto acoplamento, ou seja, dois participantes podem trabalhar em uma mesma área sem notar a sobreposição do trabalho, a menos que tenham previamente concordado em receber notificações. Em outras palavras, o apoio à colaboração do *Tukan* não tenta interpretar ações e notificar participantes que ainda não estão cientes de sua intenção comum em trabalhar sobre o mesmo artefato (SARMA *et al.*, 2003).

Do ponto de vista do desenvolvimento do ambiente, as principais limitações do *Tukan* são: (a) a dependência com o ambiente de programação *Smalltalk* e (b) a ausência

de notificações fora do modo de colaboração de alto acoplamento. As alterações que permitem a colaboração no *Tukan* estão espalhadas no código original do ambiente *Smalltalk*. Uma evolução no ambiente *Smalltalk* faz com que toda a codificação tenha que ser revista.

### 3.4.4 Palantír

O *Palantír* (SARMA e HOECK, 2002, SARMA *et al.*, 2003) é um sistema que oferece informações de percepção sobre alterações no espaço de trabalho de um programador. As alterações sobre os artefatos são organizadas de acordo com uma estrutura hierárquica de unidades de encapsulamento. Os módulos do programa afetados por uma alteração são identificados com base em estimativas que levam em consideração a estrutura do programa. No *Palantír*, o espaço de trabalho de um programador é uma coleção de artefatos representando código fonte e recursos necessários ao desenvolvimento de software.

A organização da informação de percepção considera mais de um critério de análise: alterações em linhas de código, em palavras ou em interfaces de unidades de encapsulamento. Os diferentes critérios representam tentativas dos autores do *Palantír* em discernir alterações de baixo e de alto impacto na estrutura do programa. Por exemplo, a alteração de um nome de uma variável privada em uma unidade de encapsulamento resultaria em poucas propagações de alterações sobre o programa. Uma alteração em uma variável pública representaria uma mudança que pode obrigar a realização de alterações em outros trechos do código para que o programa volte a ser coerente.

São consideradas dependências com outras unidades de encapsulamento, além das que estão sendo editadas pelo desenvolvedor. O *Palantír* analisa a estrutura do item de configuração, através da detecção de dependências com outros artefatos, gerando uma estrutura hierárquica de dependências entre unidades de encapsulamento. As notificações são enviadas apenas para os desenvolvedores que estão trabalhando sobre um dos artefatos afetados pela alteração. Os desenvolvedores devem ativar a notificação. O desenvolvedor interpreta essas informações de percepção para planejar suas ações de forma a resolver conflitos de edição detectados com outros desenvolvedores.

Nas publicações, a monitoração de eventos no *Palantír* é realizada apenas no servidor do repositório, na prática, o *Palantír* monitora, também, o editor de código fonte durante a digitação dos desenvolvedores. As alterações dos artefatos são computadas e os resultados apresentados aos usuários que solicitaram o monitoramento.

A janela de interface do mecanismo utiliza cores para identificar as alterações realizadas por usuário diferentes. As versões de um mesmo artefato são representadas como retângulos empilhados, como em um bloco de cartas de baralho. Na lateral de cada retângulo, aparecem barras verticais coloridas para indicar a existência de conflitos entre as versões monitoradas.

Dentro da classificação dos modos de acoplamento, o *Palantír* trata somente do apoio à percepção de mudanças no produto.

### 3.4.5 *Global Software Engineering Environment* (gSEE)

A *World Wide Web* é uma das infra-estruturas de colaboração mais populares entre os usuários de computador. A idéia de adaptar as ferramentas e protocolos da *Web* para atender as necessidades de uma equipe de desenvolvimento de software distribuída é a principal motivação do *Global Software Engineering Environment* (gSEE) (FIELDING *et al.*, 1998). Fielding é um dos fundadores do Apache, organização que desenvolve o servidor de hipertextos mais utilizado na Internet, e contribuiu na definição de protocolos básicos da Internet como o protocolo de transferência de hipertextos (*Hypertext Transport Protocol* - HTTP).

Na *Web* convencional, a infra-estrutura oferece facilidades para a disseminação de conteúdo. A extensão necessária deveria permitir que os desenvolvedores obtivessem documentos, realizassem edições e os colocassem de volta no servidor de origem. Essa extensão da *Web* exige alteração do protocolo de transferência de HTTP com o acréscimo de apoio à autoria de documentos distribuída e o controle de versões de artefatos. A extensão proposta recebe o nome de WebDAV (*Web Distributed Authoring and Versioning*) (WEBDAV, 2003).

O WebDAV é uma extensão do protocolo de transferência de hipertextos HTTP que facilita a construção de aplicações de autoria distribuída de hipertextos (IETF, 2003). Esta extensão simplifica o desenvolvimento do ambiente colaborativo,

porém exige que todas as aplicações compartilhadas sejam reescritas no protocolo de transferência de hipertextos.

A extensão oferece serviços similares aos utilizados em outros ambientes colaborativos para a *Web*, como o BSCW e Dress (DRIDI e NEUMANN, 1999). A principal vantagem do WebDAV é sua padronização pela IETF, em parte, devido a influência de Fielding e Whitehead nessa entidade.

O WebDAV é utilizado para a criação de ambientes voltados para a *Web*. Por exemplo, o *Pervasive Collaborative Computing Environment* (PCCE) (PERRY *et al.*, 2002) permite encontrar e contactar outros participantes, enviar mensagens instantâneas de forma síncrona e assíncrona, compartilhar artefatos e ativar dinamicamente outras aplicações *Web* e sessões de videoconferência. Kantor e Redmiles (2001), utilizaram o WebDAV como base para implementação de sistemas de percepção.

Até o momento, nenhuma ferramenta CASE popular foi adaptada com sucesso para o ambiente Web. O atraso na resposta ao usuário, característico das aplicações *Web*, e a baixa complexidade da interface com o usuário são os principais fatores que reduzem o uso da *Web*. A latência de rede também dificulta o desenvolvimento iterativo de programas e modelos. O desenvolvedor deve aguardar a transferência da informação até o servidor e o retorno do resultado do processamento. Na edição de formulários, esse atraso não é tão grave, porém, na edição de diagramas e código fonte o atraso é inaceitável. Existe uma tendência de desenvolvimento de interfaces *Web* com maior interatividade, principalmente com o uso de programação do lado cliente, como *scripts* e *applets*. Entretanto, uma ferramenta CASE é um software de porte médio, o que dificulta a sua instalação sob demanda.

Na classificação de Schümmer, o gSEE apresenta apoio ao nível de processo e à comunicação assíncrona.

### 3.4.6 Resumo dos Ambientes Colaborativos

Os ambientes colaborativos apresentados abordam diferentes aspectos do apoio ao desenvolvimento colaborativo de software. Cada ambiente tem suas próprias contribuições, mas pontos em comum podem ser encontrados. O ambiente colaborativo ideal deveria combinar aspectos positivos de mais de um ambiente. Em particular seria necessário combinar as facilidades de edição colaborativa e diferentes tipos de

percepção (espaço de trabalho, produto).

MILOS e *Tukan* tratam da edição colaborativa e do compartilhamento de áreas de trabalho que permitam o trabalho síncrono de pequenas equipes (2 a 6 pessoas). A interatividade desse tipo de apoio visa reintroduzir algumas das facilidades encontradas quando os colaboradores podem dispor de uma área física contígua. Neste tipo de colaboração síncrona, é necessário algum mecanismo de controle de concorrência para tentar evitar a sobreposição de ações que poderiam levar a área de trabalho a um estado indesejado. Mecanismos de percepção são acrescentados ao ambiente colaborativo para oferecer apoio adicional para a coordenação de atividades de alto acoplamento.

Por sua vez, o *Palantír* enfatiza a necessidade de percepção de produto, ou seja, a percepção sobre a mudança de artefatos, mesmo sobre os que estão além da área de trabalho do participante. O apoio do *Palantír* oferece algum tipo de "alerta" sobre situações que precisam ser notificadas aos participantes. Esse tipo de comportamento não encontra similar no ambiente físico. A existência desses mecanismos de alerta adicionais pode compensar, em parte, as outras facilidades que são suprimidas no ambiente colaborativo apoiado por computador.

O *Serendipity* e o gSEE enfocam a questão do desenvolvimento de uma infra-estrutura básica de colaboração. O *Serendipity* permite aumentar a percepção sobre determinadas ações que ocorrem na interface das aplicações compartilhadas. O gSEE é uma proposta mais ampla que envolve a edição colaborativa, mas que também enfoca a percepção. Enquanto o *Serendipity* trata do compartilhamento de aplicações *desktop* (*Windows, icons, menus, and pointing device – WIMP*), o gSEE trata de aplicações Web. Nesta pesquisa, seguimos a abordagem do *Serendipity*, tratando de aplicações *desktop*. A principal razão para a escolha é que as aplicações *desktop* são ainda o tipo de aplicação mais comum no desenvolvimento de software. A interatividade e a facilidade de uso dessas aplicações ainda não foram igualadas pelas aplicações *Web*.

O *Palantír* e o *Serendipity* advogam pela capacidade do participante em configurar diferentes aspectos da colaboração. No primeiro, o participante pode escolher os **objetos** que tem interesse em monitorar; no segundo, o participante decide quais **eventos** deseja compartilhar. Essa flexibilidade pode permitir o apoio a um número amplo de cenários concretos de trabalho.

Diversos trabalhos foram omitidos da lista de exemplos. Por exemplo, as

propostas das ferramentas *SeeSoft* (EICK *et al.*, 1992) e *Augur* (DE SOUZA *et al.*, 2005) foram omitidas pois tratam de aspectos complementares ao tema de estudo. Ambas apresentam ênfase na apresentação de informações ao usuário e tratam pouco dos aspectos de distribuição. Também foi mantida fora da lista a ferramenta *Jazz* (CHENG *et al.*, 2003), desenvolvida, em código fechado, por um dos centros de pesquisa da IBM (*Watson Research Center*). A proposta se opõe à idéia de 'virtualização' do apoio à colaboração. Na proposta do *Jazz*, todo o apoio seria desenvolvido exclusivamente pela IBM e utilizado pelos demais desenvolvedores, dentro do ambiente de programação Eclipse.

## 3.5 Um Modelo de Características para o Domínio de Aplicação

O desenvolvimento de software clássico, seguindo o ciclo de vida cascata, começa pela análise de requisitos do sistema e se estende para projeto, implementação e testes (PRESSMAN, 2002). Entretanto, com o passar do tempo, os engenheiros de software constataram que muitos sistemas compartilhavam semelhanças, e que o retrabalho do desenvolvimento de software poderia ser diminuído através da reutilização de especificações, modelos e código-fonte. A prática de reutilização de software implica um aumento da produtividade da equipe de desenvolvimento e da qualidade e confiabilidade do produto, além de contribuir para a diminuição dos custos.

Contudo, para que a reutilização seja possível, o desenvolvimento de software precisa passar por uma etapa anterior, onde são analisadas, projetadas e implementadas as características comuns em relação ao domínio em questão. Essa etapa chama-se engenharia de domínio.

Dessa forma, podemos definir dois pontos de vista para o desenvolvimento de software: o desenvolvimento para reuso e o desenvolvimento com reuso (MOORE e BAILIN, 1991). O desenvolvimento para reuso, ou Engenharia de Domínio, se preocupa com a geração de componentes reutilizáveis em um determinado domínio. O desenvolvimento com reuso, ou Engenharia da Aplicação, se preocupa em construir aplicações reutilizando os componentes criados na Engenharia de Domínio. A Engenharia da Aplicação serve também como agente motivador para a construção de

novos componentes inexistentes na base de componentes do domínio.

Para representar o domínio de apoio à percepção utilizando modelos de domínio, em um nível de abstração que facilite a aquisição de conhecimento, construímos inicialmente um modelo de características do domínio (BRAGA, 2001). O modelo de características tem como objetivo representar graficamente os conceitos e funcionalidades do domínio. O seu nível de abstração é alto, se preocupando em representar o conhecimento descrito nos casos de uso existentes no domínio.

Utilizamos a abordagem proposta pelo projeto *Odyssey* (WERNER *et al.*, 2000), que usa extensões da UML para representar o conhecimento de um domínio, como, por exemplo, diagramas de contexto e de características (MILLER, 2001, OLIVEIRA *et al.*, 2005).

O modelo de características apresentado é resultado de uma análise do domínio de ambientes de desenvolvimento colaborativos, limitado ao contexto de apoio à percepção, dentro desse domínio. A aquisição de conhecimento foi realizada por meio da literatura revisada neste capítulo. Os pontos de variação e as variantes são as características de maior interesse (GRISS *et al.*, 1998). Cada ponto de variação indica um grupo de alternativas (variantes) que podem ser mutuamente exclusivas ou não.

A Figura 3.2 apresenta um modelo de características que modela os elementos em comum e a diversidade apresentada pelas ferramentas estudadas, usando a notação proposta por Oliveira *et al.* (2005).

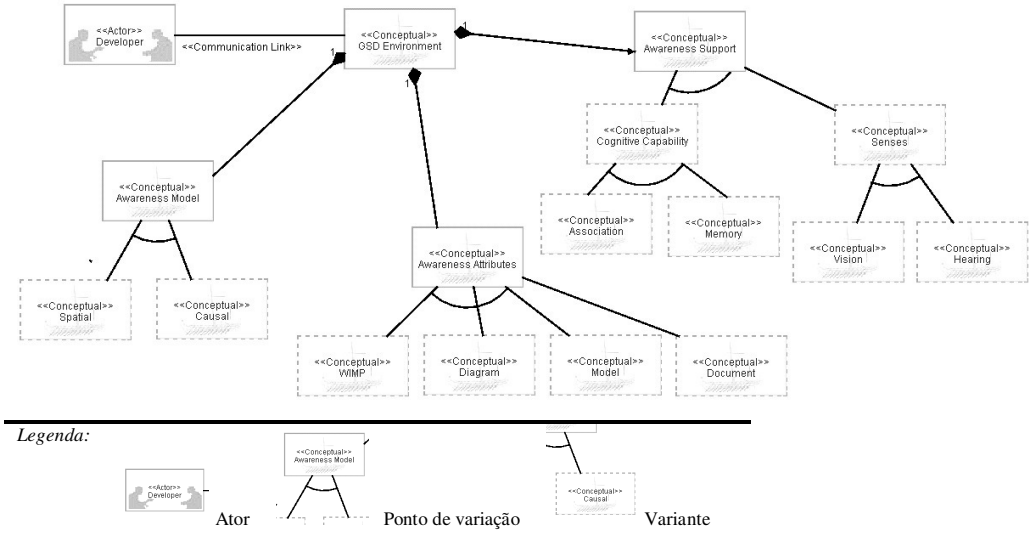



**Figura 3.2 Modelo de características para o domínio de ambientes globalizados de desenvolvimento**

**de software (Diagrama de características, notação Oliveira *et al.* (2005)).**

O primeiro conceito fundamental se refere aos aspectos da colaboração (*Collaboration Aspects*) e o papel da percepção na colaboração. A colaboração pode ser decomposta em quatro aspectos: comunicação, coordenação, memória e percepção (respectivamente: *Communication*, *Coordination*, *Memory* e *Awareness*). Todos os autores concordam que a percepção é um aspecto importante, mas que seu efeito é de alguma forma influenciado por outros aspectos.

O segundo conceito fundamental trata dos atributos do objeto compartilhado considerados na percepção (*Object Awareness Attributes*). O objeto compartilhado varia de acordo com a tarefa a ser realizada. Existem quatro conjuntos de atributos para o objeto compartilhado: modelo, diagrama, WIMP e documento (respectivamente: *Model*, *Diagram*, WIMP e *Document*). Os atributos do modelo são referências ao modelo da UML: classificadores, características, modificadores etc. Os atributos do diagrama são referências à sintaxe concreta da UML: caixas, linhas, texto e a posição dos elementos no diagrama. Os atributos WIMP são referências para elementos da interface de usuário: janelas, menus, ponteiros, barras de rolagem etc. Por fim, os atributos do documento são referências para uma estrutura de documentos com pastas, arquivos, linhas, colunas, palavras e letras.

É possível que um mesmo objeto na área de trabalho esteja representado em todos os quatro conjuntos de atributos. Por exemplo, um modelo de classes da UML é descrito como: (a) um modelo, que é uma entidade abstrata, sem representação visual; (b) um diagrama de classes, que usa uma notação para apresentar o modelo; (c) um documento, que pode ser um programa em uma linguagem de programação ou alguma outra notação textual para o modelo e, finalmente, (d) o diagrama e o documento são apresentados em uma interface de usuário que permite criar, alterar e inspecionar cada elemento do modelo (visualizador).

Os atributos do modelo existem nas três outras representações, entretanto, os atributos das representações não são representáveis no modelo. Por exemplo, a definição de uma classe A pode ser observada tanto no diagrama, na interface de usuário e no documento, pois é um atributo do modelo. Por outro lado, um documento apresenta linhas e colunas que não são relevantes ao nível do modelo. Apesar disso, com o uso do modelo é possível identificar um mesmo elemento nas demais representações. Por

exemplo, a linha que define uma classe no documento pode ser relacionada com a mesma definição no diagrama e a sua apresentação em algum elemento da interface de usuário. Esta última propriedade permite que o usuário perceba o elemento do modelo em todas as representações.

Os atributos de percepção são explorados em dois modelos de percepção (*Awareness Model*). Primeiro, a relação causal entre as mudanças é usada para ordenar eventos e para agrupar mudanças relacionadas. Por exemplo, é necessário criar a classe antes de atribuir um método a ela. A criação do método e da classe, executadas em uma seqüência, pode indicar que os dois elementos fazem parte de uma mesma ação atômica do ponto de vista do usuário final. Segundo, a relação espacial entre elementos pode indicar um relacionamento semântico. Por exemplo, uma alteração em um método causa também uma alteração em sua classe, devido a relação entre os dois atributos.

A Tabela 3-3 resume a ocorrência das características citadas. As duas características restantes são: (a) a capacidade cognitiva (*Cognitive Capabilities*) e (b) os sentidos (*Senses*) propriedades do indivíduo. As ferramentas analisadas atuam como amplificadores da capacidade cognitiva ou dos sentidos do usuário. As capacidades cognitivas consideradas são: (a) a memória (*Memory*), responsável pelo armazenamento e recuperação de informações, e a (b) associatividade (*Associativity*), responsável pela conexão entre informações. As características cognitivas aparecem relacionadas com o trabalho assíncrono. Os sentidos considerados são: (a) a visão (*Vision*), responsável pela percepção da luz, ou seja, de imagens e movimento, e (b) a audição (*Hearing*), responsável pela percepção do som. As características dos sentidos aparecem relacionadas com o trabalho síncrono.

As características propostas podem ser utilizadas para comparar sistemas existentes de forma mais objetiva. Esta classificação ainda não é definitiva e exige trabalhos futuros. A medida que novos sistemas forem revisados, novas características podem ser propostas, ampliando a classificação.

**Tabela 3-4 Características do modelo de domínio e a sua associação com as ferramentas.**

| *Características* | *Ferramentas* | | | |
|---|---|---|---|---|
| | Palantír | MILOS | Serendipity | Tukan |
| Atributos do Objeto Compartilhado | | | | |
| Modelo | ○ | ○ | ● | ○ |
| Diagrama | ○ | ○ | ● | ○ |
| WIMP | ○ | ● | ● | ● |
| Documento | ● | ● | ○ | ● |
| Proximidade no Modelo de Percepção | | | | |
| Causal | ○ | ● | ○ | ● |
| Espacial | ● | ○ | ● | ○ |
| Cognição | | | | |
| Memória | ● | ○ | ○ | ● |
| Associação | ● | ○ | ○ | ● |
| Sentidos | | | | |
| Visão | ● | ● | ● | ● |
| Audição | ○ | ● | ○ | ○ |

Legenda: ● presente ○ ausente

## *3.6 Deficiências no Apoio à Colaboração e à Abordagem de Desenvolvimento dos Ambientes*

Nesta seção, apresentamos o resultado de duas análises críticas dos ambientes estudados. O primeiro grupo apresenta requisitos ao apoio à colaboração oferecido (Seção 3.6.1). O segundo grupo apresenta requisitos à abordagem de desenvolvimento dos ambientes (Seção 3.6.2).

### 3.6.1 Requisitos quanto ao Apoio à Colaboração

Dois requisitos aparecem com maior destaque:

- **Estímulo à Socialização**

Os ambientes estudados procuram monitorar as alterações dos artefatos compartilhados. Os participantes recebem a informação de percepção que deve guiar as suas ações imediatas. Entretanto, pouca importância é dada ao estímulo de comunicação e integração entre os participantes. É essencial que o ambiente contribua para que o participante movimente-se entre os modos de colaboração, principalmente saindo da condição de prevalência do modo de isolamento.

Os participantes podem ter interesses em comum, que não são capturados na estrutura dos programas. Por exemplo, participantes podem utilizar as mesmas ferramentas, apresentar grandes intersecções em seus horários de trabalho, realizar tarefas semelhantes em projetos paralelos e outras características que justifiquem uma maior interação para a troca de experiências. O ambiente de trabalho favorece esse tipo de socialização e acreditamos que um ambiente colaborativo deva manter essa característica.

- **Organização de Informações de Percepção**

As informações de percepção são tratadas em um ciclo de estímulos e respostas. Embora seja eficiente para a solução de problemas pontuais, essa abordagem pode esconder problemas crônicos, ou seja, que se repetem em um ciclo mais amplo.

Por exemplo, imaginemos que dois participantes apresentam um histórico de sobreposição e conflitos de ações. Listar os artefatos afetados apenas estabelece soluções temporárias para uma situação que poderá se repetir em outros artefatos. Possivelmente, os dois participantes em conflito estejam perseguindo objetivos contrários como, por exemplo, um participante tenta minimizar o número de variáveis das chamadas de métodos de um programa enquanto o outro entende que deveria maximizar o número de variáveis. Nessas condições, o trabalho de um irá sempre desfazer o trabalho do outro.

Nesta tese, considera-se que o ambiente deve oferecer mais do que a mediação da informação. Para poder interferir na organização da informação, o ambiente precisa atribuir um significado para a informação e para o contexto do participante.

### 3.6.2 Requisitos quanto a Abordagem de Desenvolvimento

Alguns requisitos podem ser definidos quanto a abordagem de desenvolvimento dos ambientes estudados.

- **Extensão de Ferramentas de Trabalho**

O apoio para a realização das tarefas observado nas ferramentas analisadas é ineficiente, se comparado com ferramentas similares utilizadas na prática. Grande parte do esforço na construção de um ambiente colaborativo está na programação do apoio à tarefa. Além disso, o apoio não evolui para acompanhar avanços da técnica de desenvolvimento de software. Como resultado, o apoio à tarefa oferecido pelo ambiente colaborativo não é atraente para desenvolvedores profissionais, o que resulta em um ambiente que não é adequado para ser utilizado em situações reais de desenvolvimento. Nesse caso, não surge a oportunidade de coletar dados significativos sobre a sua utilização em tarefas concretas e em uso em longos períodos de tempo.

Portanto, é fundamental que as ferramentas existentes sejam mantidas para que os desenvolvedores se sintam motivados a utilizar o ambiente, tal como é proposto nesta tese.

- **Independência em relação a um Método de Trabalho Particular**

Para oferecer apoio mais específico, os ambientes pressupõem a existência de diversas relações entre os participantes, a existência de papéis e sobre a organização da equipe. Com isso, a equipe deve se adaptar ao ambiente colaborativo.

Advogamos que pode existir um apoio à colaboração que não utiliza tantas premissas e ainda assim seja útil, principalmente como meio virtual para a colaboração. Um método de trabalho particular poderia ser programado como resultado de uma configuração específica do ambiente, realizada pelos próprios participantes.

- **Apoio à Colaboração Específico**

Por outro lado, uma parte significativa de ambientes apresenta uma sobreposição com o apoio atualmente oferecido pela *Internet*, a *Web* e de *groupware* em geral. Neste trabalho, evitaremos tratar dos elementos nessa sobreposição, mantendo o uso de sistemas de comunicação síncronas e assíncronas e de hipertextos na *Web*. A proposta se concentra em alterar as aplicações que não oferecem recursos de apoio à

colaboração.

- **Implementação Descentralizada**

A abordagem de uma implementação centralizada, ou seja, dependente de um único servidor, facilita a implementação do ambiente colaborativo. Entretanto, um servidor estabelece dependências com uma determinada organização que mantém o servidor e a infra-estrutura necessária para seu funcionamento.

Essa dependência é inapropriada para modelos mais flexíveis de trabalho. Por exemplo, imagine que dois participantes de empresas diferentes decidem criar um projeto privado ou contribuir para alguma organização de software livre. Qual servidor deve ser utilizado?

Atualmente, existem facilidades para a programação de redes simétricas (ou *peer-to-peer*), ou seja, onde cada computador é, ao mesmo, tempo cliente e servidor. Com isso, cada computador pode ser o servidor de uma sessão ou equipe, com vantagens para distribuição de carga e para a independência dos participantes. Ainda, existe muita capacidade de processamento e armazenamento não devidamente explorada no lado cliente das redes de computadores. Neste trabalho, tratamos de redes simétricas.

- **Reutilização de aplicações e mecanismos de colaboração**

Uma análise preliminar dos ambientes indica que os componentes de software com maior potencial para reutilização são (a) as aplicações que suportam o trabalho individual e (b) os mecanismos que oferecem a informação de percepção e a coordenação entre as áreas de trabalho.

Uma abordagem para produção desses ambientes deve considerar o reaproveitamento de ambos. As aplicações devem ser recuperadas e integradas em uma estrutura que permita a sua interligação com os mecanismos de colaboração, sem a necessidade de se reescrever cada aplicação. As aplicações existentes, em geral, não são projetadas para oferecer esse tipo de apoio à colaboração.

Não é econômico reescrever todas as aplicações existentes para oferecer apoio à colaboração que, embora seja necessário, será possivelmente menos utilizado, em comparação com as demais funções dessas aplicações (GRUDIN, 1994).

## 3.7 Conclusão

A prática de Engenharia de Software está fundamentada em processos de software, métodos de desenvolvimento e diversas técnicas associadas. Essa característica tende a se manter no desenvolvimento globalizado. O investimento realizado na formação e treinamento de profissionais e na infra-estrutura já existente são dois dos principais motivos para a manutenção dessa característica. Estublier (2000) cita que diversas tentativas de alterar ferramentas e o modo de trabalho do indivíduo encontram resistências. Por exemplo, repositórios com serviços mais avançados do que os utilizados atualmente foram propostos, mas nunca adotados em larga escala, pois realizavam mudanças radicais no modo individual de trabalho. Além disso, aproveitar aplicações e práticas de sucesso pode facilitar a adoção gradual de *groupware* (GRUDIN, 1994).

O desenvolvimento de software é um processo colaborativo. A divisão do trabalho com base na divisão dos artefatos diminui a necessidade de comunicação direta entre os indivíduos. Porém, mesmo em técnicas como o desenvolvimento com base em componentes, podemos argumentar que a divisão dos artefatos e a sua reutilização aumenta a necessidade de comunicação ao invés de diminuí-la. Cada vez mais, o conhecimento que está de posse de um desenvolvedor ou parte interessada torna-se necessário. No DBC, advogamos que o mercado de componentes pode ser interpretado também como uma rede de conhecimento. A presença de um componente de software na área de trabalho de um desenvolvedor sinaliza a sua dependência com outros desenvolvedores. O consumidor de um componente passa a se interessar pelas ações do produtor do componente e também nas ações de outros consumidores.

A utilização de um novo meio para o apoio ao desenvolvimento de software oferece a oportunidade de observar características que antes permaneciam escondidas. A transição para ambientes com base na *Web* permite o registro em mensagens e hipertextos de uma estrutura social difícil de ser observada no processo de trabalho colocado. Registros de fóruns de discussão e hipertextos são uma das principais formas de observação do trabalho de equipes distribuídas (MOCKUS *et al*., 2000). Entretanto, a *Web* é um meio que exige a participação ativa do desenvolvedor tanto para o registro quanto para a seleção da informação necessária, o que restringe o seu uso por causa da

sobrecarga associada a essas atividades adicionais.

As ferramentas colaborativas apresentadas neste capítulo foram desenvolvidas através da alteração *ad hoc* de ferramentas CASE existentes. A alteração realizada desta forma exige um grande esforço de desenvolvimento e desvantagens na manutenção do sistema alterado. No Capítulo a seguir, é apresentado um conjunto de técnicas que permitem realizar as mesmas alterações de forma controlada, com maior potencial de reutilização das ferramentas e das alterações realizadas.

# 4 Colaboração Transparente

Neste capítulo, fazemos uma breve revisão de alguns conceitos sobre colaboração e características da abordagem de desenvolvimento de *groupware* com colaboração transparente (Seção 4.1). Apresentamos as características das propostas de colaboração transparente flexível (Seção 4.2) e colaboração transparente inteligente (Seção 4.3), que são a base da arquitetura proposta nesta tese e apresentada no Capítulo 5.

## *4.1 Definições*

A compreensão da colaboração é um requisito para a construção de aplicações colaborativas. A colaboração envolve diversos aspectos da interação entre indivíduos. Alguns dos aspectos mais citados incluem:

- **Comunicação**: a transmissão de informação explícita ou implícita (DIAS e BORGES, 1999, FUKS *et al*., 2002, LAURILAU e NIGAY, 2002);
- **Percepção**: a consciência das ações dos demais participantes (DIAS e BORGES, 1999, FUKS *et al.*, 2002);
- **Coordenação**: o controle e planejamento necessários para minimizar a sobreposição de ações e a proteção do trabalho já realizado (DIAS e BORGES, 1999, FUKS *et al.*, 2002, LAURILAU e NIGAY, 2002);
- **Memória compartilhada**: o registro de ações passadas (DIAS e BORGES, 1999);
- **Espaço compartilhado, cooperação, espaço de produção**: um local onde os resultados do trabalho são armazenados, percebidos e modificados pelos participantes (DIAS e BORGES, 1999, FUKS *et al.*, 2002, LAURILAU e NIGAY, 2002).

Uma aplicação colaborativa deve considerar a importância desses vários aspectos na tarefa que está sendo apoiada. Em geral, uma aplicação considera alguns

desses aspectos. Por exemplo, existem sistemas que consideram principalmente a comunicação, como o *Babble* (ERICKSON e KELLOG, 2000), e não enfatizam os demais aspectos.

Nos últimos anos, diversas aplicações têm sido propostas com ênfase na percepção. A informação de percepção é útil para permitir a coordenação e o controle sobre a colaboração. A demanda por apoio a este aspecto da colaboração resultou na proposta de sistemas dedicados à gerência do apoio à percepção (KIRSH-PINHEIRO *et al.*, 2002).

A percepção pode ser considerada como uma variação ou resultante da comunicação entre os participantes do ambiente colaborativo (ARAÚJO, 2000) (GEROSA *et al.*, 2002). Nesta tese, a definição para o conceito de percepção é: "uma compreensão das atividades de outros, a qual provê um contexto para a sua própria atividade" (DOURISH e BELLOTTI, 1992). Existe mais de uma classificação para tipos de percepção (STEINFIELD *et al.*, 1999, DOURISH e BELLOTTI, 1992).

Neste trabalho, enfatizamos a necessidade da chamada **percepção de espaço de trabalho** ou "percepção sobre a área de trabalho" (*workspace awareness*) (DOURISH e BELLOTTI, 1992, GUTWIN e GREENBERG, 1996, SCHLICHTER *et al.*, 1997). Segundo Gutwin e Greenberg, a percepção de espaço de trabalho é a informação I necessária para que o participante A conheça as interações do participante B com a área de trabalho [compartilhada], sendo que I torna possível a colaboração eficiente entre A e B. A informação I permite responder a questões como:

- Quem está participando da atividade?
- Onde eles estão trabalhando?
- Qual o nível de atividade deles sobre a área de trabalho?
- O quê eles estão fazendo?
- Quais as suas atividades e tarefas?
- O quê eles farão? Para onde irão?
- Quais alterações estão sendo realizadas e onde?
- Quais objetos estão sendo utilizados?

- O quê eles podem ver?
- Que ações eles são capazes de realizar?
- Em que locais eles estão autorizados a alterar?
- O quê eles esperam que eu faça?

Schlichter *et al.* (1997) revisam vários aspectos da percepção e destacam a necessidade de se considerar o acoplamento e o tempo de ocorrência entre a tarefa dos usuários. Com isso, as perguntas listadas assumem duas dimensões:

- **Tempo**: Quem está participando? Quem participou?
- **Acoplamento**: Quem está participando nesta área compartilhada? Quem está participando de outras áreas compartilhadas?

A informação de percepção é transmitida para o usuário da aplicação colaborativa através de um **mecanismo de percepção**. Um mecanismo de percepção deve coletar, distribuir, e apresentar informação. O mecanismo é importante nesta tese, pois desejamos facilitar a programação e reutilização dos mecanismos de percepção e mecanismos colaborativos em geral.

Alguns dos mecanismos considerados neste trabalho são componentes visuais que são integrados a janela da aplicação colaborativa: teleapontadores e janelas de radar (GUTWIN e GREENBERG, 1999) apresentam a posição do indicador do mouse e a área visível de um outro usuário para o usuário local; *gauges* de percepção (KANTOR e REDMILES, 2001) apresentam informações variadas sobre a composição e alterações na área de trabalho; indicadores de atividade (BORGES e PINO, 1999) organizam informações sobre a atividade na área de trabalho e ritmos de trabalho (BEGOLE *et al*., 2002) apresentam informações sobre a presença e disponibilidade dos participantes através de uma grade de horários.

O mecanismo depende da coleta de informações sobre as interações na área compartilhada, ou seja, na aplicação colaborativa. Em geral, os mecanismos já são reutilizados através de **bibliotecas de programação para *groupware*** como, por exemplo, o COAST *(SCHUCKMANN et al., 1996)* e o COPSE (DIAS, 1998, *DIAS e BORGES, 1999)*. Begole (1998) faz uma ampla revisão de bibliotecas disponíveis, muitas das quais permanecem disponíveis atualmente. A reutilização da infra-estrutura

de colaboração continua evoluindo com a proposta de *frameworks* orientados a objeto (LÓPEZ e SKARMETA, 2003) e servidores dedicados ao apoio à colaboração (KIRSH-PINHEIRO *et al.*, 2002). Essas facilidades para a programação de *groupware* ampliam a capacidade dos programadores em desenvolver aplicações colaborativas, reutilizando mecanismos pré-existentes.

Entretanto, as abordagens de desenvolvimento de *groupware* não consideram a reutilização das aplicações existentes que não foram projetadas para oferecer apoio ao trabalho de grupos. Nesta proposta, pretendemos ampliar a possibilidade de reutilização dessas aplicações para a construção de ambientes colaborativos. Para tanto, precisamos considerar uma abordagem de desenvolvimento que é diferente da que vem sendo explorada atualmente pelas bibliotecas e *frameworks* de desenvolvimento de *groupware*. Inicialmente, no desenvolvimento desta tese, foi abordado o problema do desenvolvimento de aplicações síncronas (MANGAN *et al.,* 2002). No Capítulo 5, é apresentada uma solução mais abrangente que oferece, também, apoio para aplicações assíncronas com base nas técnicas de *interpretação de eventos*.

As **abordagens de desenvolvimento** aplicadas para aplicações colaborativas síncronas fazem parte de uma entre duas categorias (LAUWERS e LANTZ, 1990). Na primeira categoria, as aplicações são especificamente projetadas para o apoio a grupos (*collaboration awareness),* ou seja, a aplicação é capaz de reconhecer cada um dos participantes e os seus papéis na colaboração (GREENBERG, 1990). Na segunda categoria, aplicações monousuário são compartilhadas e o apoio ao grupo é realizado por mecanismos de colaboração que são desconhecidos (ou *transparentes*) para as aplicações e seus desenvolvedores (*collaboration transparency*).

Na primeira categoria, enquadra-se o desenvolvimento da maior parte dos ambientes de desenvolvimento de software colaborativos apresentados no Capítulo 3. O uso de bibliotecas e *frameworks* para programação da colaboração podem ser enquadrados nesta primeira categoria sempre que o desenvolvedor da aplicação colaborativa tiver a necessidade de entremear instruções para controlar a colaboração no código da aplicação.

Para realizar uma aplicação colaborativa, duas preocupações (*concerns*) dos desenvolvedores devem ser unidas. A primeira preocupação é de como apoiar o trabalho

individual, ou seja, como tratar os requisitos do domínio de aplicação. A segunda preocupação é de como apoiar o trabalho do grupo, ou seja, como tratar dos requisitos da colaboração.

Dentre os ambientes de desenvolvimento colaborativos descritos, a proposta do *Palantír* é a que apresenta maior transparência para o desenvolvedor, pois o *Palantír* não interfere na definição da interface do repositório monitorado. Entretanto, o *Palantír* monitora o servidor da aplicação, enquanto que a colaboração transparente monitora as aplicações ou os clientes da aplicação.

Um **sistema de colaboração transparente** (*collaboration transparency system*) permite o compartilhamento **síncrono** de aplicações monousuário legadas por meio de alterações em seu ambiente de execução (BEGOLE, 1998).

Aplicações monousuário são aquelas que precisam tratar dos dados e execuções de um único usuário. Uma aplicação multi-usuário é aquela que reconhece preferências e dados privados de usuários diferentes. O desenvolvimento de aplicações monousuário é mais simples e, portanto, de menor custo, do que o de aplicações multi-usuário, o que vem a ser uma das justificativas para a sua abundância.

Se por um lado, aplicações monousuário são mais freqüentes no ambiente de trabalho do que aplicações multi-usuário, por outro, aplicações multi-usuário são mais freqüentes do que *groupware*. Aplicações colaborativas, em geral, não apresentam um conjunto de funcionalidades tão completo quanto o de suas equivalentes não colaborativas (LI e LI, 2002).

Inúmeras são as aplicações monousuário, instaladas e executadas em um único computador, que são usadas em diferentes ramos de atividade. Por esse motivo, a colaboração transparente é uma alternativa interessante para a implementação de apoio à colaboração em ramos de atividade onde podem ser encontradas diversas aplicações monousuário já desenvolvidas. Por exemplo, editores de texto e de diagramas são aplicações de alta complexidade que, em geral, são aplicações monousuário.

A idéia de compartilhar o uso de aplicações entre estações de uma rede não é recente. Engelbart e English (1968) já haviam explorado essa possibilidade com um dos primeiros sistemas de compartilhamento de telas registrado (NLS). Esse mesmo trabalho pioneiro propôs uma das primeiras interfaces gráficas de usuário e deu origem

ao dispositivo apontador (*mouse*).

Li e Li (2002) identificam três gerações de sistemas de colaboração transparente. A **primeira geração** permite o compartilhamento de telas. O NLS enquadra-se nessa geração. A **segunda geração** compartilha telas e aplicações, o que permite que o usuário mantenha alguma privacidade no seu ambiente, selecionando o que deseja compartilhar. Um sistema como o NetMeeting pertence a esta geração. A **terceira geração** explora algum tipo de conhecimento da semântica de operações da aplicação para oferecer um apoio mais adequado à colaboração. Os trabalhos de Begole (1998) e Li e Li (2002) enquadram-se nesta última categoria.

## 4.2 Colaboração Flexível

Begole (1998) apresenta uma arquitetura de colaboração transparente (*flexible collaboration transparency*) que permite incluir, em uma aplicação convencional, determinadas características de percepção (*awareness*) normalmente encontradas apenas em um *groupware*. Com essa abordagem, Begole advoga que é possível diminuir as diferenças em funcionalidade que separam as aplicações das duas diferentes categorias de desenvolvimento.

A colaboração transparente é implementada em diversas plataformas. A implementação mais popular em computadores pessoais é, possivelmente, o *NetMeeting* da Microsoft. Uma breve descrição do *NetMeeting* foi apresentada durante a discussão sobre o MILOS XP (Seção 3.4.1).

As principais críticas aos sistemas convencionais de colaboração transparente apontadas por Begole são: (a) ausência de trabalho concorrente, (b) uso de *WYSIWIS* estrito (*What you see is what I see*, em português, "O quê você vê é o que eu vejo"), (c) percepção de grupo limitada e (d) utilização ineficiente de recursos de rede. A proposta da *colaboração flexível transparente* trata de minimizar alguns desses problemas. No WYSIWIS estrito, apenas uma estação compartilha a sua tela com as demais estações. As demais estações se limitam a visualizar essa tela, mas não é possível realizar tarefas fora da estação compartilhada.

As características da colaboração flexível são comentadas nas seções seguintes. Cada seção resume o problema e a solução apresentada na colaboração flexível. Para propor as soluções apresentadas, Begole apóia amplamente a colaboração flexível na literatura de CSCW.

### 4.2.1 Trabalho concorrente

A **ausência de trabalho concorrente** na colaboração transparente convencional é decorrente da necessidade do controle de concorrência (*floor control*) entre os eventos originados nas aplicações compartilhadas.

Para preservar a intenção da ação de cada participante, o compartilhamento convencional bloqueia os eventos provenientes de todas as aplicações exceto uma. As ações do usuário que opera essa aplicação não bloqueada são percebidas nas demais estações. O uso do bloqueio no nível da aplicação evita que os demais usuários possam desenvolver qualquer ação sobre a aplicação. Tipicamente, os usuários revezam o controle da aplicação.

Na colaboração flexível, o bloqueio é realizado em uma granularidade menor. Dessa forma, é possível oferecer um paralelismo maior para os usuários da aplicação. A consistência da operação é obtida através de uma ordenação total de eventos e pelo uso de bloqueios em determinadas seqüências de operações que necessitam de atomicidade. A ordenação dos eventos na colaboração flexível adota a técnica de transformações operacionais descrita mais adiante (Seção 4.5).

### 4.2.2 WYSIWIS Relaxado

No modelo WYSIWIS relaxado, os participantes ganham a liberdade de visualizar porções diferentes da área de trabalho. A implementação de WYSIWIS relaxado é realizada na colaboração flexível através do controle dos eventos que devem ser compartilhados entre as aplicações. Cada aplicação pode visualizar áreas de trabalho de maneira independente. Eventos podem ser tratados de três maneiras diferentes: compartilhamento imediato, compartilhamento postergado e ignorado. Eventos que ocorrem na área de intersecção das janelas de cada aplicação devem ser compartilhados no modo imediato, eventos fora da área local de visualização são processados de forma

postergada ou ignorados.

Por exemplo, uma ação que ocorre fora da área de janela de uma estação não afeta o usuário local até o momento em que este visualiza o objeto afetado, ou um objeto dependente, ao movimentar sua área de visualização. Alguns eventos, como o movimento do mouse em uma área fora do campo de visualização local, podem ser ignorados sem prejuízo para a colaboração, isto é, não serão visualizados pelo participante.

### 4.2.3 Percepção de espaço de trabalho

A colaboração convencional conta apenas com teleapontadores. O trabalho concorrente associado com a adoção de WYSIWIS relaxado diminui a percepção que cada participante tem das ações dos demais. Na colaboração transparente, a informação de percepção é reintroduzida através de mecanismos de percepção síncrona, tais como teleapontadores e janelas em miniatura (*telepointers e miniviews*).

Teleapontadores permitem perceber a posição do indicador do *mouse* de um outro usuário que está operando sobre a mesma janela. Janelas em miniatura permitem uma visão panorâmica da área de trabalho compartilhada e oferece indicações sobre o posicionamento de cada participante.

### 4.2.4 Utilização otimizada dos recursos da rede

Na colaboração flexível, a arquitetura descentralizada e os diferentes tratamentos dados aos eventos permitem a utilização mais eficiente de recursos da rede do que na colaboração transparente convencional.

A arquitetura descentralizada permite um melhor balanceamento da carga de trabalho entre as diversas estações. A postergação e a omissão de eventos contribuem para diminuir o tráfego da rede.

### 4.2.5 Limitações da Colaboração Flexível

Begole (1998) aponta algumas limitações da colaboração transparente e da colaboração transparente flexível. A colaboração transparente é aplicável apenas quando

são apoiadas tarefas colaborativas que estendam modelos individuais de trabalho. Por exemplo, a edição de documentos colaborativa é uma extensão da edição de documentos realizada em aplicações monousuário.

Tarefas inerentemente colaborativas devem ser apoiadas por aplicações especificamente projetadas para uma finalidade particular, por exemplo, os processos de votação e tomada de decisão em grupo.

A colaboração flexível transparente depende da capacidade de interferir na fila de eventos da aplicação compartilhada. Duas técnicas são possíveis: a interferência na fila de eventos do ambiente de execução ou a da fila de eventos da aplicação. Begole adota a primeira opção, compartilhando eventos de mouse e teclado entre as aplicações. Além disso, é necessário tratar de eventos provenientes de outras fontes como eventos de rede, arquivos e outros dispositivos de entrada e saída.

Menus, janelas de diálogo e outros elementos da interface gráfica ou do modelo da aplicação podem não estar preparados para receber eventos de múltiplos usuários. Por exemplo, tipicamente, uma aplicação pode apresentar apenas uma opção de menu selecionada. Para contornar essas limitações, Begole (1998) substitui alguns elementos do sistema de execução da aplicação por outros elementos com capacidades colaborativas, ou bloqueia as operações dos demais até que uma determinada seqüência atômica de operações seja realizada por um dos usuários.

Entretanto, nem todos os eventos que alteram o estado de uma aplicação são provenientes do teclado e do *mouse*. A rede de computadores, o sistema de arquivos e outros sistemas podem gerar eventos que requerem a atenção da aplicação. Recentemente, foram propostas soluções satisfatórias para os problemas de acesso a conexões de rede, arquivos e outros sistemas externos a aplicação (BEGOLE *et al.*, 2001). A solução desses problemas envolve alterações adicionais no ambiente de execução, além do que seria necessário para monitorar e inserir eventos na aplicação.

Como limitação geral, o compartilhamento não pode ser utilizado em aplicações que executem tarefas dependentes de tempo, por causa da latência variável da rede de comunicação.

## *4.3 Colaboração Inteligente*

Li e Li (2002) abordam o compartilhamento de aplicações heterogêneas. A crítica aos sistemas de colaboração dos autores é em relação a simetria da rede de aplicações compartilhadas. Todas as aplicações devem ser idênticas (produto e versão) para que os eventos gerados em uma estação possam ser interpretados corretamente nas demais estações.

A colaboração, segundo Li e Li, pode ser necessária entre colaboradores que utilizam aplicações diferentes, mas que realizam as mesmas tarefas. Nesse caso, as seqüências de comandos em cada aplicação são diferentes, mas correspondem a uma mesma ação. Li e Li propõem que é possível criar uma descrição de comandos (um *script*) que permite mapear comandos de uma aplicação para os comandos equivalentes em outra aplicação. Esse processo é chamado de **tradução de eventos**. A proposta é chamada de colaboração transparente inteligente (*Intelligent Cooperation Transparency* - ICT).

A ICT não altera o ambiente das aplicações, apenas captura os eventos dos dispositivos de entrada. As seqüências de comandos são traduzidas para uma representação independente da aplicação e enviadas para as demais estações. Cada estação traduz a seqüência independente para os comandos equivalentes da aplicação local.

### 4.3.1 Limitações da Colaboração Inteligente

A colaboração inteligente altera a fila de eventos dos dispositivos de entrada de uma aplicação. Essa abordagem gera problemas na injeção de eventos. Para receber um evento remoto, a interface de usuário local tem de ser interrompida para que o evento remoto seja inserido. Todos os parâmetros do evento precisam ser inseridos na fila de eventos, como se estivessem sendo digitados. Essa operação é sujeita a falhas, embora Li e Li advoguem que é tecnicamente viável. Em ambientes de desenvolvimento, a execução de algumas funções (por exemplo, geração de código, compilação e engenharia reversa) pode levar vários minutos, Durante esse intervalo de tempo, a aplicação permaneceria interrompida.

A colaboração flexível utiliza polimorfismo para substituir determinados objetos que controlam o ambiente de execução da aplicação, por exemplo, as classes que controlam a geração de números aleatórios e entrada e saída de arquivos e elementos da interface, tais como barras de rolagem (*scrollbars*). A colaboração inteligente não realiza essas substituições, por dois motivos (a) impossibilidade de apresentação de mecanismos de colaboração dentro da janela da aplicação e (b) desconhecimento do estado interno da aplicação.

A colaboração inteligente não considera que gestos (*gesture*) diferentes na interface podem corresponder a uma mesma ação. É natural para o usuário ativar a mesma ação através de itens de menu, botões, teclas de atalho ou menus sensíveis ao contexto. Esses quatro gestos devem ser descritos separadamente para que possam ser interpretados corretamente.

A descrição dos eventos pode variar em versões diferentes de uma mesma aplicação. Ainda, um usuário pode personalizar a interface removendo e acrescentando elementos de interface ou módulos opcionais. Uma aplicação típica reconhece centenas de gestos diferentes e cada um deve ser descrito através de um *script* para que o compartilhamento pleno das aplicações aconteça. Sobre esta última limitação, Li e Li acrescentam que os usuários podem reduzir o número de ações que desejam compartilhar a um conjunto de comandos necessários e suficientes para a realização da tarefa.

## 4.4 Transformações Operacionais

A transformação operacional é uma técnica que permite que ações locais possam ser imediatamente executadas e em seguida enviadas para os demais editores. As operações remotas são transformadas antes de serem aplicadas localmente, de forma a manter a consistência entre as operações (CHEN e SUN, 1999).

Um documento é representado em um editor colaborativo como uma seqüência de símbolos. Sobre essa seqüência são realizadas apenas duas operações (a) inserção de um símbolo e (b) remoção de um símbolo na seqüência (SUN e ELLIS, 1998). Essas duas operações são as bases para outras operações mais complexas. Por exemplo, uma alteração é a combinação de uma remoção e de uma inserção no mesmo ponto da

seqüência.

Como ilustração do funcionamento geral da transformação operacional, vejamos como ocorreria a edição colaborativa da seqüência “fgh” por dois usuários A e B (Figura 4.1). Imaginemos que o usuário A realiza a operação inserir('x',1) que insere o caractere 'x' na posição 1 da seqüência, enquanto que o usuário B realiza a operação inserir('y', 2). Imediatamente, a seqüência do usuário A torna-se “xfgh” e a seqüência do usuário B fica “fygh”. A seguir as operações locais são propagadas de um editor para outro.

A operação inserir('x',1) pode ser realizada diretamente na seqüência do usuário B, gerando então a seqüência “xfygh”. A operação não necessita transformação, pois não opera sobre o trecho da fita alterada pela operação local - inserir('y', 2). Por outro lado, se a operação inserir('y', 2) for aplicada na seqüência do usuário A, teremos como resultado “xyfgh”. Essa operação removeria a sincronização das seqüências editadas. Para obter o efeito correto, a operação deve ser realizada como inserir('y', **3**), ou seja, deve levar em conta a alteração na fita causada pela operação local – inserir('x',1). A transformação sobre a operação, com base nas alterações locais realizadas, gera a seqüência esperada: “xfygh”.

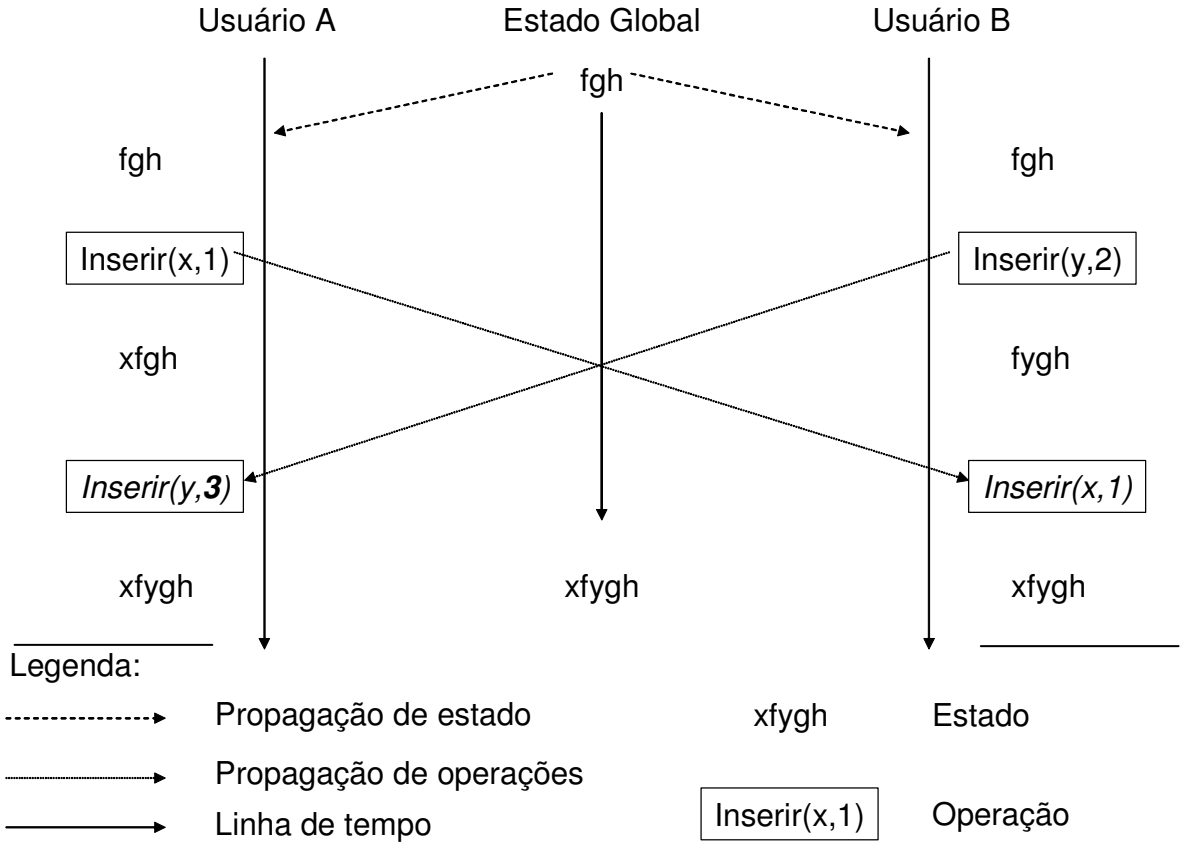



**Figura 4.1 Ilustração do funcionamento geral da transformação operacional (Adaptado de Sun e Ellis, 1998).**

Essa técnica permite que os editores locais mantenham estados consistentes, sem a necessidade de sincronizar todas as operações em um servidor. Cada estação calcula as transformações necessárias, considerando o histórico de eventos locais. Dessa forma, o tempo de resposta local de cada editor não apresenta uma diferença significativa em tarefas colaborativas e individuais. Tanto a colaboração flexível quanto a colaboração inteligente utilizam esta técnica que permite a edição colaborativa em redes simétricas. Do ponto de vista de infra-estrutura, a transformação operacional é uma técnica que permite que a computação ocorra sem a necessidade de ordenação total de todas as operações.

A transformação operacional oferece apoio à edição colaborativa síncrona. Molli *et al* (2002) demonstraram uma variação dessa técnica para apoiar a colaboração assíncrona e multi-síncrona. Na colaboração assíncrona, a fila de eventos remotos permanece armazenada e vai acumulando operações que foram executadas pelo grupo,

mas não foram ainda integradas na seqüência local. O usuário decide o momento quando as operações serão integradas na sua cópia, o que permite alternar entre períodos de isolamento e de edição colaborativa em um mesmo editor.

Na colaboração multi-síncrona, um grupo de usuários é particionado gerando seqüências de edição alternativas. Dois ou mais usuários compartilham entre si operações que não são conhecidas pelos demais usuários. Essa seqüência alternativa pode até mesmo evoluir para um estado incompatível com a seqüência original. Esse tipo de colaboração é importante para permitir a exploração de alternativas de solução e espaços de trabalho para projetos. A transformação operacional é utilizada neste trabalho para apoiar diferentes modos de colaboração. A ordenação total de eventos é a base de diversos editores síncronos. A transformação operacional permite que eventos com diferentes ordenações locais resultem em um mesmo estado global consistente.

## 4.5 Modelo Semântico da Aplicação

A colaboração transparente depende da utilização de um modelo semântico da aplicação. Em linhas gerais, esse modelo é utilizado para decidir como tratar e disseminar os eventos locais coletados com os eventos que chegam através da rede, provenientes de outras aplicações.

O modelo que oferece os elementos necessários para representar software deve ter por base uma estrutura genérica e extensível, como é o caso da UML (Figura 4.2). Na Figura 4.2, nomes em itálico indicam classes abstratas. Todas as classes são subclasses de um mesmo elemento (*ModelElement*), exceto a própria classe *ModelElement*. Por simplicidade, esses relacionamentos foram omitidos da figura.

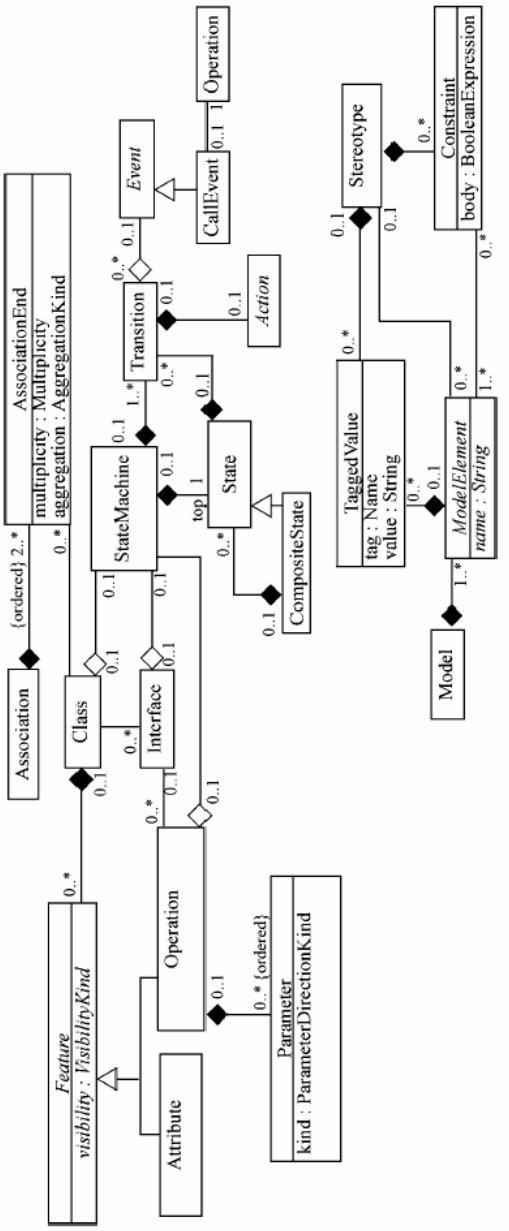



**Figura 4.2 Simplificação do Meta-modelo da UML 1.5 usada por Medvidovic *et al.* (2002).**

Por polimorfismo, um grande número de elementos pode ser tratado em uma infra-estrutura de percepção voltada para software como se fosse uma instância de *ModelElement* ou de suas classes derivadas. Com isso, o modelo da UML além de oferecer um conjunto abrangente de elementos, ainda contribui para reduzir a complexidade da programação do apoio à percepção.

Begole utiliza um modelo de aplicação que permite distinguir entre três tratamentos distintos para os eventos locais. Li e Li ampliam esse modelo para reduzir

seqüências de eventos em ações da interface de usuário que são propagadas para as demais aplicações.

Nesta tese, ampliamos ainda mais esse modelo para reduzir as ações de interface em ações que são significativas para os participantes. Desse modo, as ações podem ser processadas e apresentadas para outros participantes, mesmo que não estejam operando a mesma aplicação. Em outras palavras, estaremos gerando informação de percepção com base em eventos de operação de aplicações.

## 4.6 Computação Sensível ao Contexto

Uma tendência na computação distribuída é a Computação Sensível ao Contexto (*Context Aware Computing*) (COUTAZ *et al.*, 2005). A noção de contexto é usada nessas aplicações para adaptar uma aplicação às condições e limitações do usuário. A noção mais simples de contexto está relacionada com a localização física do usuário. As aplicações utilizam a localização para selecionar informações que podem ser úteis para a orientação do usuário. Outro uso comum do contexto é a seleção de configurações de uma aplicação que se adaptem às limitações de um dispositivo. Por exemplo, uma mesma aplicação pode usar som para informar o usuário de um telefone móvel e usar um gráfico para informar um outro usuário em um computador pessoal.

Atualmente, existe uma preocupação com a relação do contexto com a percepção. Do ponto de vista de implementação de sistemas colaborativos, o interesse nesse tipo de aplicação é a maneira como o contexto é organizado. Coutaz *et al.* (2005) propõem sete elementos de uma infra-estrutura para a computação sensível ao contexto (Figura 4.3).

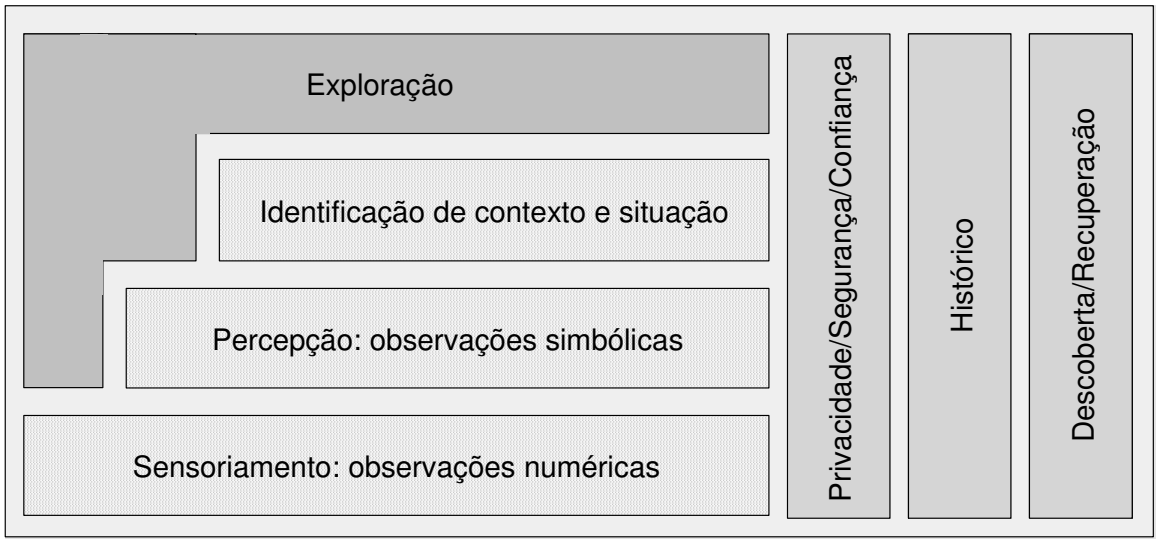



**Figura 4.3 Níveis de abstração para uma infra-estrutura de propósito geral para computação sensível ao contexto (COUTAZ *et al.*, 2005).**

Em um paralelo com a abordagem desta tese, sistemas de apoio à percepção e aplicações sensíveis ao contexto apresentam diversas similaridades. O **sensoriamento** é o elemento responsável pela obtenção de informações sobre o ambiente. As informações são coletadas de forma contínua sobre um ambiente que se altera continuamente pela ação de agentes. As informações coletadas são amostras discretas de um ambiente contínuo. A informação coletada não representa o ambiente como um todo, nem representa um aspecto completo do ambiente. Por exemplo, imaginemos um **sensor** que coleta a temperatura de uma sala em um dado instante. A temperatura coletada não é suficiente para caracterizar a sala como um todo, o sensor coleta a temperatura no ponto onde está instalado. Nada pode ser informado sobre a temperatura no extremo oposto da sala. A coleta de temperatura é realizada em determinados instantes, não é possível afirmar qual a temperatura exata da sala entre duas coletas consecutivas. Entretanto, um sistema de controle de temperatura é capaz de funcionar, pois tem por base um conjunto de premissas que permitem que as lacunas de informação sejam completadas. Por exemplo, as premissas do sistema de controle de temperatura são: espera-se que o sensor seja colocado em uma posição adequada dentro da sala e que a temperatura do ambiente não sofra alterações bruscas.

A **percepção** é um elemento das aplicações sensíveis ao contexto que tem como função realizar uma interpretação simbólica das medidas numéricas do sensoriamento. No exemplo do sistema de controle de temperatura, as medidas

numéricas de temperatura podem ser interpretadas como valores simbólicos como: frio, quente, agradável. O termo percepção tem um significado diferente em CSCW. Na computação sensível ao contexto, a percepção é uma função do sistema que resume ou reduz a complexidade das informações do sensoriamento. Em CSCW, a percepção é um processo cognitivo do participante, que pode ser incentivado por informações oferecidas pelo sistema.

O elemento de **identificação de contexto e situação** determina condições que influenciam no comportamento esperado pelo sistema. Por exemplo, uma sala de eventos pode ser usada para festas, palestras, reuniões e cada um desses eventos pode ter uma temperatura ideal pré-definida.

O elemento de **exploração** trabalha sobre os dados dos outros três elementos descritos. No exemplo, quando a percepção temperatura estiver agradável, temos uma situação em que nenhuma ação é necessária. Nas situações de quente ou frio, o sistema de refrigeração ou aquecimento devem ser acionados de acordo.

Os três elementos restantes são ortogonais, ou seja, fazem parte de uma infra-estrutura explorada pelos demais elementos. Na proposta, o histórico e a descoberta e recuperação são realizados pelo espaço de tuplas. A privacidade, segurança e confiança são elementos que não foram explorados nesta tese.

## *4.7 Modelos Computacionais para Computação Distribuída*

Todo sistema colaborativo para apoio a equipes distribuídas é também um sistema distribuído. O paradigma mais utilizado atualmente para a programação de sistemas distribuídos é o cliente/servidor. A principal limitação desse paradigma é o aproveitamento inadequado dos recursos da rede de computadores. A capacidade de armazenamento e processamento dos clientes é pouco utilizada pelo paradigma. O paradigma simétrico (ou *peer-to-peer*) proporciona um melhor aproveitamento dos recursos da rede, porém apresenta uma complexidade maior ao desenvolvedor. O paradigma simétrico é utilizado atualmente na programação de aplicações *peer-to-peer* e também em computação em grade (*grid computing*).

A programação de sistemas distribuídos pode adotar diferentes modelos, sendo os mais comuns o modelo de troca de mensagens e o modelo de memória

compartilhada. A memória compartilhada apresenta baixo desempenho e a troca de mensagens apresenta alta complexidade no endereçamento e composição das mensagens. Entretanto, existem modelos computacionais alternativos com base em troca de mensagens que oferecem facilidades para programação.

No modelo computacional de Carriero e Gelernter (2001), a computação pode ser dividida em dois componentes ortogonais: um síncrono e outro assíncrono. O componente síncrono é qualquer *locus* de atividade (*activity locus*) que realiza uma tarefa de cada vez. Por exemplo, um processador simples é um componente do tipo síncrono. O segundo componente é uma coleção de componentes síncronos que se comunicam de forma assíncrona (*asynchronous ensemble*). Um computador do tipo *desktop* é uma coleção de dispositivos eletrônicos. Neste modelo, um usuário seria mais um exemplo de um *locus* de atividade e um usuário e um computador formariam um exemplo de uma coleção assíncrona.

Carriero e Gelernter chegam a conclusão que esses dois componentes - a computação e a coordenação - são ortogonais, no sentido de que podem ser desenvolvidos modelos independentes para esses componentes. A união de dois desses modelos gera um modelo de computação completo. A partir dessa idéia, Carriero e Gelernter procuraram desenvolver novos modelos de coordenação (*Linda* e *Lifestreams*) e um novo modelo de computação (Linguagens simétricas). A pesquisa foi iniciada na universidade de Yale, por volta de 1980.

O modelo computacional adotado nesta tese tem por base a orientação objeto. Por outro lado, o modelo de comunicação adotado é uma implementação do modelo de comunicação gerativa (*generative communication*) implementado na linguagem Linda (GELERNTER, 1985). Um conceito fundamental da implementação da linguagem é o espaço de tuplas (*tuple space*).

Um espaço de tuplas é uma implementação do paradigma de memória associativa para programação distribuída que oferece um repositório de tuplas, que pode ser utilizado de forma concorrente. Um exemplo de aplicação seria uma computação na qual um grupo de processos produz dados enquanto outro grupo consome esses dados. Os produtores colocam dados em tuplas no espaço e os consumidores removem do espaço determinadas tuplas que satisfaçam um critério de busca. O espaço de tuplas

acumula persistência, comunicação e controle de concorrência em um único sistema.

Existem implementações de espaços de tuplas em diversas linguagens de programação. Na plataforma Java, a implementação é especificada por meio do serviço *JavaSpaces*, da biblioteca de programação distribuída *Jini*. Nessa implementação, uma tupla é um objeto cujos atributos são tipos simples. Com isso, as tuplas podem ser usadas pelo programador de aplicação exatamente como um objeto. A plataforma *Neem* é a única proposta, dentro do contexto de ambientes colaborativos, que adota o espaço de tuplas como modelo de comunicação (BARTHELMESS e ELLIS, 2002).

O *Lifestreams* também influenciou a abordagem descrita nesta tese. Uma *lifestream* é um histórico de todos os documentos e mensagens que foram recebidas por um determinado usuário. O conceito do *Lifestreams* tem origem na idéia similar de *chronicle streams* (Gelernter, 1993). No *Lifestreams*, todo o histórico de comunicação permanece armazenado indefinidamente. Um usuário pode realizar buscas como "Onde está aquela mensagem que recebi na semana passada?".

Entretanto, do ponto de vista de modelagem, a tupla tem como limitação o uso exclusivo de atributos simples. Por outro lado, as informações coletadas do ambiente estão relacionadas entre si e é necessário que seja criada uma representação uniforme para os fenômenos que podem ser observados no ambiente. A solução encontrada é o uso de conceitos da *Web* Semântica, descritos na seção a seguir.

## 4.8 Web Semântica e Anotações

A Web é um vasto depósito de conhecimento. Existe informação em grande quantidade, gerada por fontes diferentes e sobre os mais variados assuntos. A *Web* excede a capacidade de qualquer indivíduo em extensão e complexidade. A informação encontra-se conectada por referências que ligam documentos de acordo com a intenção do produtor do texto. Entretanto, nem todos textos sobre um mesmo assunto estão interligados. Por exemplo, para encontrar documentos sobre "vinhos finos", um usuário deve recorrer a uma busca com base na ocorrência de palavras do texto. Na *Web* convencional, não existe uma classificação universal de assuntos ou de sinônimos que permita a classificação dos documentos e sua recuperação automática.

A *Web* Semântica é uma evolução da *Web* convencional que permite

justamente que os documentos da *Web* sejam descritos e organizados com o uso de um vocabulário controlado (ex. *Dublin Core*). Em particular, a proposta mantém a estrutura atual dos documentos e acrescenta um conjunto de informações adicionais (anotações) que relacionam ou descrevem documentos. Uma anotação é uma declaração na forma: sujeito, predicado e objeto. As anotações são parte do *Resource Description Framework* (RDF) (WORLD WIDE WEB CONSORTIUM, 2005a, WORLD WIDE WEB CONSORTIUM, 2005b, SANTANCHÉ, 2003).

A estrutura da Web e do espaço de software apresentam diversas similaridades. Ambos são espaços discretos formados por elementos que apresentam ligações entre si. A estrutura de informação se presta à diferentes interpretações. Por exemplo, pode ser criado um índice de documentos que têm fundo azul e outro índice de documentos que tratam sobre "jardinagem". Um índice é completamente independente do outro. Também é possível que pessoas diferentes criem índices sobre "jardinagem", contendo documentos diferentes. Logo, é necessário permitir a coexistência de múltiplas tentativas de organização da informação.

A solução encontrada, nesta tese, é o uso de anotações que registram a informação coletada por diferentes sensores. Cada tupla permanece independente, mas composta de três atributos: sujeito, predicado e objeto. Em um mesmo atributo, os valores possíveis são determinados pelos elementos existentes no espaço de software. O espaço de software serve como vocabulário básico, o que permite identificar quando um mesmo sujeito aparece em tuplas diferentes.

## 4.9 Conclusão

As técnicas de colaboração transparente sinalizam com uma possibilidade de integração de aplicações existentes com mecanismos de colaboração diversos. Essas técnicas são aplicadas no compartilhamento síncrono de aplicações e, posteriormente, também no compartilhamento assíncrono. A colaboração transparente possibilita a criação de áreas de trabalho compartilhadas, mas não oferece, atualmente, apoio para informações de percepção.

A segunda geração do compartilhamento de aplicações desperta interesse comercial com a proposta de aplicação em ambientes de escritório. *Groove* (GROOVE

NETWORKS, 2003) e *Microsoft Messenger* (MICROSOFT, 2003) são exemplos de aplicações direcionadas para a colaboração de computadores pessoais (*desktop collaboration*). Através dessas aplicações, os participantes podem compartilhar aplicações, efetuar comunicação com voz, vídeo-conferência ou texto. Groove Networks é o nome de uma companhia criada por Ray Ozzie, um dos fundadores da *Lotus Notes*. O *Microsoft Messenger* é uma evolução do *Microsoft NetMeeting*. Ambos possibilitam a criação de uma rede ponto-a-ponto (*peer-to-peer*) para o compartilhamento de arquivos, troca de mensagens síncronas e assíncronas e compartilhamento de aplicações até o momento restritas. O Microsoft Word e o Microsoft PowerPoint são, no momento, as principais aplicações disponíveis. Existe pelo menos uma proposta do uso do *Groove* no contexto de desenvolvimento de software colaborativo (DEFRANCO-TOMMARELLO e DEEK, 2002).

No próximo capítulo, será apresentada a proposta de uma variação da técnica de colaboração transparente adaptada e adequada ao tipo de apoio à colaboração que foi observado nos ambientes para desenvolvimento distribuído analisados. A variação proposta visa atender aos requisitos listados. Em particular, a possibilidade de uma exploração da semântica do espaço de software, do apoio de interações assíncronas e da organização da informação de percepção.

# 5 Uma Proposta de Arquitetura de Colaboração Transparente

Conforme apresentado nos capítulos anteriores, a oferta de apoio à percepção nos ambientes de desenvolvimento de software encontra-se limitada, principalmente, pela maneira como esse apoio é implementado. O apoio à percepção, geralmente, encontra-se associado a uma única ferramenta, que costuma não oferecer apoio adequado às tarefas relacionadas ao desenvolvimento de software. Além disso, os ambientes estudados apresentam um conjunto de funcionalidades que poderia ser realizado fora da aplicação por uma infra-estrutura de apoio em comum. Apesar desses problemas apresentados nas implementações existentes, existe uma demanda pelo apoio às equipes de desenvolvimento distribuídas, que necessitam de ambientes com melhor apoio à colaboração, incluindo o apoio à percepção de grupo, de espaço de trabalho e de produto.

Este capítulo aborda o problema de como apoiar a construção de aplicações que facilitem a colaboração em equipes virtuais. Esta proposta está centrada em uma "Arquitetura de Colaboração Transparente" (ACT) que descreve elementos que possibilitam a criação de espaços de trabalho com maior apoio à percepção a partir de ferramentas CASE e de componentes de apoio à percepção pré-existentes.

Uma arquitetura é uma descrição dos elementos discretos que fazem parte de um sistema e das interações entre esses elementos. Cada elemento se torna um item de um vocabulário que pode ser usado para comunicar a descrição do sistema e analisar suas propriedades (SHAW e GARLAN, 1996, BOSCH, 2000). Neste trabalho, a arquitetura de colaboração transparente busca identificar os elementos necessários para a criação de um sistema colaborativo adequado ao desenvolvimento de software. O adjetivo transparente advém da característica da arquitetura em definir um ambiente colaborativo que (a) exige pouco esforço adicional por parte do usuário durante sua operação e (b) opera integrado com outro sistema pré-existente, sem que seja necessária uma alteração deste sistema pré-existente. Com essa abordagem, esperamos reduzir (a) a

resistência à adoção desse ambiente colaborativo e (b) o esforço necessário para criar e operar tal ambiente. O primeiro objetivo é atingido pela integração de ferramentas CASE, cujos usuários tem a opção de participar da colaboração sem abandonar seu ambiente de trabalho convencional, que também reduz o esforço de operação. O esforço para a criação o ambiente é reduzido, principalmente, quando ocorre a reutilização da implementação de elementos da arquitetura, em particular, nos sensores.

O capítulo encontra-se organizado da seguinte forma. Inicialmente, são apresentados alguns dos requisitos técnicos e de usabilidade para a arquitetura de colaboração transparente (Seção 5.1). A seguir, é apresentada uma visão geral das decisões de projeto da arquitetura que satisfazem esses requisitos (Seção 5.2). A percepção é o aspecto da colaboração que é apoiado pela abordagem, portanto, são apresentados um modelo de percepção (Seção 5.3) e um ciclo de informação de percepção (Seção 5.4), que são as principais abstrações que orientam a proposta da arquitetura. A descrição da arquitetura é apresentada em seguida (Seção 5.5). Para orientar a aplicação da abordagem, um processo de desenvolvimento que organiza as atividades relacionadas com o desenvolvimento de sistemas com o uso da arquitetura proposta também é descrito (Seção 5.6). É apresentada ainda uma comparação com trabalhos relacionados (Seção 5.7). Ao final, é feito um resumo deste capítulo (Seção 5.8) destacando as principais contribuições da abordagem proposta e suas limitações.

## *5.1 Requisitos*

Esta seção enumera algumas características desejáveis em sistemas que sejam gerados a partir da abordagem proposta. O papel da arquitetura aqui é garantir que esses requisitos sejam atendidos por meio das características dessa infra-estrutura de apoio comum e que se refletem nas propriedades dos sistemas construídos a partir dela.

### 5.1.1 Requisitos Técnicos

Do ponto de vista técnico, a arquitetura de colaboração transparente deve atender aos seguintes requisitos:

- RT1. A interface de programação utilizada na aplicação original deve

permanecer inalterada. Na prática, este requisito determina que o código adicionado para realizar o apoio à percepção não seja visível ao programador da aplicação original.

- RT2. O apoio à percepção não deve interferir no desempenho da aplicação original. Como medida prática, o tempo de resposta da aplicação original, nas funcionalidades relacionadas com o apoio à tarefa, não deve ser alterado de forma significativa.
- RT3. Os recursos de rede e de processamento disponíveis devem ser utilizados de forma eficiente. Em geral, a computação tende a se concentrar em um servidor que acaba por se tornar um ponto crítico de falha e a limitar a escalabilidade da solução. A capacidade de processamento e memória dos clientes são quase sempre sub-utilizadas.
- RT4. Deve ser permitida a combinação de serviços de apoio à percepção em uma mesma instância da aplicação original. A existência de um único serviço de apoio à colaboração em cada aplicação é uma limitação dos ambientes colaborativos estudados.
- RT5. Os requisitos da aplicação original não precisam ser alterados, se for do interesse do desenvolvedor. Em outras palavras, as preocupações relacionadas com o apoio à colaboração não devem obrigatoriamente ser incluídas entre as preocupações do desenvolvedor da aplicação original.

Conforme apresentado no capítulo anterior (Seção 4.2), a lista de requisitos de Begole (1998) apresenta requisitos similares aos requisitos RT1, RT2 e RT3 para sistemas de colaboração transparentes, os quais foram adaptados para o escopo desta tese. Os requisitos RT4 e RT5 são propostos nesta tese e não são tratados por Begole. Os requisitos RT1 e RT5 são necessários para garantir a idéia de transparência nas perspectivas de projeto e de implementação da aplicação original. Begole defende a possibilidade da criação de uma aplicação transparente, com apoio síncrono para edição colaborativa. O requisito RT4 decorre da necessidade de desenvolver uma família de aplicações, onde os serviços de colaboração são um dos pontos de variação (GRISS *et al.*, 1998).

### 5.1.2 Requisitos de Usabilidade

Além dos cinco requisitos técnicos, a colaboração transparente deve atender a três requisitos de usabilidade:

- RU1. Os diferentes modos de trabalho dos desenvolvedores devem ser respeitados e a alteração de um modo de trabalho para outro deve ser permitida. Como forma de proteger a privacidade e a produtividade individuais, o uso dos mecanismos pode ser ativado e desativado a critério do usuário.
- RU2: O conteúdo da área de trabalho individual não deve ser alterado como resultado de um serviço de colaboração. O serviço não deve introduzir alterações na área de trabalho local sem uma ação consciente do usuário.
- RU3. A oferta de informação de percepção deve ser ampla, permitindo a consulta a diferentes tipos de informação, incluindo: área de trabalho, grupo e produto.

O requisito RU1 decorre da natureza complementar do apoio à percepção. Nem todo usuário necessita ou deseja participar. O requisito RU2 delimita a responsabilidade do apoio à percepção como um mecanismo de informação. A maioria das ferramentas CASE exige uma área de trabalho estável e consistente. O usuário deve ser informado do estado das áreas remotas, mas deve ter controle total de sua área local. A integração das áreas de trabalho deve ser realizada, preferencialmente, por meio de serviços das próprias aplicações individuais. O requisito RU3 decorre de estudos que apontam o aumento da quantidade e da qualidade das informações sobre as atividades de outros participantes como uma das principais demandas dos desenvolvedores distribuídos geograficamente (HERBSLEB *et al*., 2001, HERBSLEB e MOCKUS, 2003).

## *5.2 Visão Geral*

Muitas das deficiências encontradas no apoio à percepção nos ambientes colaborativos de desenvolvimento de software são decorrentes da abordagem usada na sua implementação: o apoio à percepção é construído por meio de uma alteração de um ambiente colaborativo ou de um ambiente de desenvolvimento existentes. O problema

dessas alterações é que o ambiente resultante apresenta baixo apoio à tarefa, quando é construído como um alteração de um ambiente colaborativo, ou é restrito a uma única ferramenta, quando é construído como uma alteração de um ambiente de desenvolvimento (Seção 3.6).

Esta seção apresenta uma abordagem alternativa de construção que adota a estratégia de alteração de ambientes de desenvolvimento existentes, mas sem restringir o serviço a uma única ferramenta. Com essa abordagem busca-se guiar o desenvolvedor do ambiente colaborativo a oferecer um conjunto de funcionalidades para atender necessidades de desenvolvedores que trabalham na condição de distribuição. Assume-se que a equipe trabalha no paradigma aleatório ou aberto. Um **modelo de computação distribuída** é aplicado para armazenar e distribuir as informações de percepção coletadas nas aplicações. A **integração das aplicações** ocorre de forma transparente para o programador do ambiente de desenvolvimento e para o usuário. Os serviços de colaboração são realizados por **componentes de software** reutilizáveis. Um **processo de desenvolvimento** organiza as atividades necessárias para realizar a abordagem proposta.

Com essas quatro características é possível construir ambientes colaborativos para o desenvolvimento de software que apresentam um grau mais elevado de apoio à tarefa e à percepção do que é observado nos sistemas existentes atualmente. A reutilização de componentes de software permite que (a) uma mesma alteração seja reutilizada em diferentes ambientes e que (b) um mesmo ambiente reutilize mais de uma alteração.

### 5.2.1 Ambientes Distribuídos

A literatura sobre implementação de ambientes colaborativos discute as vantagens e desvantagens da utilização de arquiteturas distribuídas ou centralizadas (Seção 4.6). A arquitetura centralizada apresenta menor complexidade de programação e requer uma infra-estrutura mais simples. Os aspectos negativos da infra-estrutura centralizada são a baixa escalabilidade, a baixa tolerância a falhas e o uso ineficiente de recursos de comunicação e de processamento da rede. As arquiteturas distribuídas apresentam a relação inversa, porém as dificuldades de implementação e programação das arquiteturas distribuídas reduzem a sua aplicação prática. Além disso, diversas redes

locais restringem o uso de arquiteturas distribuídas para reduzir a possibilidade da operação de código mal-intencionado. Entretanto, recentemente, o interesse por arquiteturas distribuídas foi retomado, principalmente pela característica de uso eficiente dos recursos disponíveis na rede. Um dos esforços atuais na pesquisa em sistemas distribuídos está na construção de infra-estruturas de grade. Através da computação em grade é possível alcançar um maior aproveitamento de recursos geograficamente dispersos (FOSTER, 2002).

Existem dois modelos tradicionais para a programação de sistemas distribuídos: a troca de mensagens e o compartilhamento de memória. Para o desenvolvedor de ambientes colaborativos, o nível de abstração oferecido por esses modelos é muito baixo. De fato, uma das funções básicas das bibliotecas para desenvolvimento de ambientes colaborativos é, justamente, a de prover um meio de comunicação (GREENBERG, 2001). Contudo, a possibilidade de comunicação entre ambientes de colaboração fica restrita aos ambientes que foram desenvolvidos com a mesma biblioteca. Por exemplo, um ambiente desenvolvido com o *GroupKit* não pode se comunicar com um ambiente desenvolvido com o COAST, a não ser que seja criado um adaptador específico (OCHOA *et al.*, 2004).

Além disso, a utilização de *frameworks* para colaboração (como o COAST e o *GroupKit*) apresentam vantagens, mas também dificuldades pela falta de uma orientação em como construir uma aplicação específica (BECKER e BACELO, 2001).

Para contornar a complexidade da programação distribuída e a incompatibilidade entre bibliotecas, foi adotado um modelo alternativo de programação: o espaço de tuplas (CARRIERO e GELERNTER, 1988). Uma tupla é um registro de dados que é gravado em um processo que se encontra em uma estação e pode ser lido ou removido por um processo em qualquer outra estação da rede através do espaço de tuplas. Um espaço de tuplas é uma estrutura de dados distribuída que oferece uma camada de abstração que permite a comunicação e a coordenação de processos distribuídos.

O espaço de tuplas atende aos requisitos RT3 e RT4. Os recursos de rede são gerenciados pela implementação do servidor de espaço de tuplas. O espaço de tuplas é gerado por um grupo de servidores. Replicação e particionamento de dados e outras técnicas de sistemas distribuídos oferecem um desempenho alto e uma utilização

eficiente dos recursos de rede. Cada tupla deve pertencer a um tipo de tupla, que é uma descrição dos campos do registro da tupla. Do ponto de vista de programação, os conceitos de tupla e tipo de tupla é similar ao conceito de objeto e classe. Qualquer processo pode realizar a leitura de uma tupla, que permanece disponível no espaço até que seja removida por algum dos processos. As tuplas formam uma descrição da informação de percepção disponível que permite a interoperabilidade entre os serviços de colaboração.

A ACT apresenta uma arquitetura de distribuição simétrica (ou replicada) (BEGOLE, 1998). Cada estação de trabalho na rede é, ao mesmo tempo, um cliente e um servidor. Dessa forma, cada estação adicionada nessa rede contribui com a capacidade de armazenamento e processamento que é utilizada para realizar o processamento dos eventos capturados.

Redes simétricas são apontadas como uma alternativa que apresenta vantagens sobre sistemas centralizados (ANDERSON *et al.*, 2000). Entre as vantagens estão: melhores relações de escalabilidade de serviços, reconfiguração de topologia, tolerância a falhas e balanceamento de carga (BEGOLE, 1998). Begole advoga ainda que a descentralização da estrutura favoreça a adoção por parte dos usuários que passam a não depender da infra-estrutura de terceiros.

Diversas tecnologias facilitam a construção desse tipo de sistema, de forma que um sistema simétrico possa ser programado com facilidades semelhantes àquelas encontradas na programação cliente/servidor (ADAMS, 2002, SUN, 2003). Nesta tese, foram testadas, em diferentes protótipos, as implementações de um servidor de mensagens, da chamada remota de mensagens e do serviço de espaço de tuplas.

### 5.2.2 Componentes de Software para Colaboração

O conceito de componentes é aplicado de duas maneiras diferentes nesta abordagem. Um componente de software pode representar uma funcionalidade aparente para o usuário final: um *widget*. Além de ser um componente visual, integrado com a visualização da aplicação, esse componente de software apresenta uma interface de integração com o restante da infra-estrutura colaborativa. Do ponto de vista do usuário final, o apoio à colaboração disponível em qualquer ambiente pode ser descrito como um grupo de componentes visuais. Um usuário se refere e se lembra de um sistema

colaborativo por seus serviços visíveis na interface. Por exemplo, um teleapontador e uma janela de radar são componentes visuais visíveis na interface de usuário de um sistema de compartilhamento de aplicações.

Um ambiente colaborativo é um sistema complexo e distribuído, composto de diversas partes que interagem entre si. Do ponto de vista do desenvolvedor, a infra-estrutura é composta por um conjunto de componentes de software que não são visíveis na interface de usuário da aplicação, embora ofereçam os serviços necessários para os componentes visuais. Por exemplo, o serviço de comunicação e o serviço de controle de concorrência em um sistema de compartilhamento de aplicações são componentes de software invisíveis para o usuário final.

A divisão do ambiente colaborativo apresenta dois aspectos importantes. O primeiro é a capacidade de reutilização de partes de um sistema existente em um novo sistema. Bibliotecas como o *GroupKit* e o COAST foram criadas visando esse potencial para reutilização. O segundo aspecto é a possibilidade de substituição de componentes. Um componente do sistema pode ser substituído por outro componente que satisfaça as mesmas interfaces. Existem duas implicações sobre esse segundo aspecto. Primeiro, é possível que um sistema operacional seja aperfeiçoado pela substituição de um componente por outro de maior qualidade ou mais adequado para uma dada tarefa. Por exemplo, um teleapontador pode ser substituído por um teleapontador semântico para reduzir a necessidade de comunicação entre as aplicações compartilhadas. A segunda implicação da substituição é a possibilidade de dispor de mais de um fornecedor para um componente. As bibliotecas existentes para desenvolvimento de ambientes colaborativos não levam em conta a substituição. O desenvolvedor está limitado aos componentes disponíveis na biblioteca que é desenvolvida por um único fornecedor. Por exemplo, o serviço de comunicação apresenta a mesma interface para diferentes implementações de um teleapontador. Entretanto, nas bibliotecas atuais, um teleapontador não pode ser substituído e nem interoperar com outros componentes de outra biblioteca.

Cada componente visual exige um conjunto de componentes não-visuais. Por sua vez, um componente não-visual exige outros componentes não-visuais. Um componente pode substituir ou excluir outros componentes em um ambiente colaborativo. Na abordagem descrita, as relações entre os componentes são explícitas,

de forma que a sua interoperabilidade seja prevista.

### 5.2.3 Integração de Aplicações de Uso Individual

O apoio à tarefa, no desenvolvimento de software, é realizado por aplicações de uso individual. Em particular, as tarefas relacionadas com a programação e a modelagem de software são realizadas com aplicações WIMP (*Windows, Icons, Menus, and Pointer*).

Existem três requisitos para a integração das aplicações existentes:

- RI1: Coleta de informações sobre objetos e sobre a fila de execução. As aplicações integradas não são projetadas com os requisitos de colaboração em mente. Entretanto, é possível obter informações sobre os objetos e sobre a fila de comandos, utilizando-se de diferentes técnicas de programação.
- RI2: Integração de informação de percepção às aplicações. A informação de percepção deve ser apresentada ao usuário de forma integrada ao ambiente, reduzindo o esforço para obter e interpretar essa informação. A adição de novas visões é obtida pela sincronização de janelas independentes, pela sobreposição de telas e pela adição de painéis. A integração atende, principalmente, ao requisito RU2.
- RI3: As aplicações devem apresentar um modelo semântico discreto. A atualização por meio de eventos é adequada para transferir instâncias de modelos de objetos discretos. Cada elemento apresenta tamanho reduzido e pode ser transferido separadamente, desde que se encontre uma maneira de mover as referências entre elementos. Essa técnica não é adequada para a movimentação de modelos contínuos ou dependentes de tempos, como, por exemplo: imagens, vídeo e som. A atualização por meio de eventos é vantajosa quando a informação sobre as alterações em um modelo é menor que a informação para descrever o novo estado do modelo. A transferência de eventos reduz o espaço de armazenamento e do canal de comunicação necessários. Em modelagem e programação, as transformações são geralmente reduzidas. Com exceção do uso de ferramentas de engenharia reversa e de geração de código, uma sessão de trabalho, geralmente, envolve

a edição de um grupo restrito de elementos. As primeiras edições são compostas de grandes transformações e as seguintes tendem a ser de volume menor.

Foram utilizadas quatro técnicas na construção dos protótipos: a extensão de aplicações, a notificação de eventos, a substituição de objetos e a programação orientada a aspectos. Com o uso dessas técnicas, é possível atender aos requisitos RT1 e RT2 (Capítulo 6).

### 5.2.4 Requisitos e Características da Abordagem

Nesta seção, são relacionados os requisitos e as características da abordagem proposta. As características da abordagem atuam sobre os requisitos identificados.

A computação distribuída atua sobre os requisitos de uso eficiente de recursos (RT3) e da combinação de serviços de colaboração (RT4). A eficiência é obtida pela movimentação da gerência de recursos do código da aplicação para o código do servidor de tuplas. A gerência se torna mais eficiente, pois é realizada de forma global pelo servidor, que por vezes deve dividir recursos entre diversas aplicações. Além disso, os algoritmos e técnicas usadas na programação do servidor são mais sofisticados do que aqueles ao alcance do programador de aplicação médio. A combinação de serviços é favorecida pelo paradigma de programação adotado, pois a informação encontra-se disponível para todos os processos envolvidos na colaboração, com a possibilidade de integração entre processos que executam linguagens de programação diferentes. As tuplas formam um modelo ontológico que é público para todos os desenvolvedores. Um componente pode usar tuplas definidas para outro componente. A tecnologia pode ser usada para aproveitar recursos das estações de trabalho.

A integração de aplicações é um ponto importante da abordagem, pois atua sobre três requisitos técnicos (RT1-3). A adaptação de interfaces é uma técnica da programação com componentes de software. A necessidade de evolução de sistemas implicou o desenvolvimento de algumas técnicas para a integração de sistemas legados e interoperabilidade entre sistemas que são aplicadas nesta abordagem. É natural que essas técnicas possuam suas limitações. Em alguns casos, o sistema legado e seu ambiente de execução simplesmente não podem ser expandidos ou alterados. Nos casos onde as técnicas apresentadas se aplicam, espera-se um sobrecarga durante a execução.

Para fins de avaliação de desempenho, será considerado significativo o atraso maior do que 5% do tempo de resposta original nas funcionalidades relacionadas à tarefa.

O desenvolvimento com base em componentes de software atua sobre os requisitos RT1, RT4 e RT5. A decomposição do sistema em componentes atua para preservar os requisitos e a interface de programação da aplicação original.

A relação entre requisitos e características é resumida na Tabela 5-1.

**Tabela 5-1 Atuação das características da abordagem proposta sobre os requisitos técnicos identificados.**

| *Requisitos/Características* | *Computação em grade* | *Integração de aplicações* | *Componentes de software* |
|---|---|---|---|
| RT1 Preservação da API da aplicação | ○ | ● | ● |
| RT2 Preservação do desempenho da aplicação | ○ | ● | ○ |
| RT3 Uso eficiente de recursos | ● | ● | ○ |
| RT4 Combinação de serviços de colaboração | ● | ○ | ● |
| RT5 Preservação dos requisitos da aplicação | ○ | ○ | ● |

Legenda: ● contribui ○ não contribui

Os requisitos de usabilidade são responsabilidades dos componentes desenvolvidos como pontos de variação na arquitetura (GRISS *et al.*, 1998).

## 5.3 Modelo de Percepção com base em Anotações

Conforme apresentado anteriormente (Seção 4.8), o uso de anotações favorece a representação de diferentes perspectivas sobre um modelo. O uso de anotações em um modelo de percepção é útil para representar múltiplas interpretações sobre a informação de percepção coletada. Existe então, uma única representação do espaço de software, que é acrescida de anotações relevantes dentro da perspectiva de cada participante.

Um modelo objetivo de percepção necessita comunicar três elementos fundamentais: o objeto, a ação e o agente da ação (Figura 5.1). Em um ambiente de desenvolvimento de software, os objetos são representados por alguma das especializações de *ModelElement* do meta-modelo da UML. São quatro ações possíveis: criar, remover, alterar e consultar um objeto. Uma ação causa uma mudança sobre o

modelo, levando um objeto de um estado (*before*) para um novo estado (*after*). Um estado é caracterizado por um conjunto de propriedades do objeto e seus valores correspondentes. O agente da ação é um desenvolvedor, normalmente, o próprio operador da aplicação de modelagem. Uma ação pode ser resultado da discussão entre vários agentes. Por exemplo, na programação em duplas, a ação é resultado do consenso entre os dois desenvolvedores envolvidos, embora apenas a autoria do operador da ferramenta seja registrada. As ações apresentam uma relação de ordem parcial entre si.

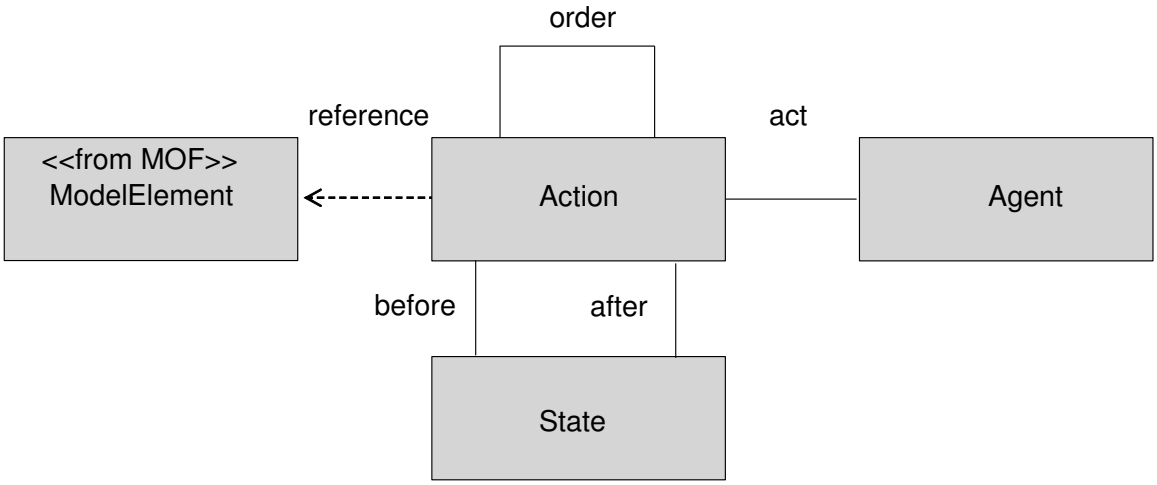



**Figura 5.1 Elementos fundamentais de um modelo objetivo de percepção para modelos de software (Diagrama de classes da UML).**

A complexidade do modelo de objetos é irrelevante, uma vez que todo objeto é uma sub-classe de *ModelElement*. Dessa forma, cada ação do modelo de percepção aponta para um elemento do modelo de software. Essa relação entre modelos tem por base a proposta de anotações sobre modelos da *Web* Semântica. O modelo de software é um modelo de fatos que é utilizado para a construção de modelos específicos.

A descrição dos estados é viável, pois um *ModelElement* é construído com base em um modelo de dados genérico (*Generic Semantic Data Model* - GSM) composto de três estruturas discretas: objeto, propriedade e valor (*Object*, *Property*, *Value*). Um valor é sempre um elemento discreto que pode ser representado em um tipo de dados básico (um número, uma cadeia de caracteres, uma referência a outro elemento).

O modelo de software é manipulado por uma aplicação que mantém a consistência do modelo e das operações. Por exemplo, a aplicação impede que uma classe seja composta por pacotes. Em outras palavras, a aplicação garante que o modelo de software seja consistente com as regras de formação definidas no modelo da UML. O

modelo de percepção não realiza testes de consistência. Uma vez que as ações são capturadas, a partir do modelo de software mantido pela aplicação, não é necessário que a infra-estrutura de colaboração verifique essas regras novamente. O processamento é mais eficiente, mas depende da confiança atribuída aos sensores (Seção 4.6) instalados nas aplicações.

Foram também desenvolvidos dois modelos objetivos de percepção com base em anotações. Um modelo anota alterações em um modelo de documentos e outro em interfaces de usuário do tipo WIMP. Os modelos foram usados para representar informações de percepção sobre repositórios e de compartilhamento de telas.

O modelo de percepção deve ser entendido como um modelo aproximado. Existe a possibilidade de que algumas das ações não sejam capturadas. A implementação dos sensores pode ser incompleta ou os sensores podem estar desligados na aplicação que está sendo usada para alterar o modelo de software. Com isso, formam-se lacunas na ordem parcial de ações.

## *5.4 Ciclo de Informação de Percepção*

Do ponto de vista técnico, a informação utilizada no modelo de percepção passa por um ciclo desde que é coletada a partir das ações de um usuário, armazenada, organizada, distribuída, apresentada, analisada e, finalmente, destruída.

### 5.4.1 Coleta

A etapa de coleta tem como objetivo determinar os elementos do modelo de percepção associados com cada ação existente na fila de eventos da aplicação. O principal problema, nesta etapa, é realizar a coleta sem alterar as aplicações existentes (RT1).

Não é adequado que o usuário seja envolvido no processo de coleta, pois isso implicaria tarefas adicionais para o usuário final que, além da sobrecarga cognitiva associada, iriam distrair o usuário da atividade principal que é o desenvolvimento de software. A coleta implícita de informações deve ser realizada com uma das técnicas de integração de aplicação, conforme as facilidades oferecidas pela aplicação. Do ponto de

vista da arquitetura, é importante separar a coleta das demais preocupações. O componente de coleta deve ser simples e coeso. Deve ser considerada a existência de dados incompletos devido a sensores insuficientes ou usuários não participantes. Exemplos de sensores são apresentados juntamente com os protótipos no Capítulo 6.

### 5.4.2 Armazenamento

A colaboração assíncrona exige a existência de um mecanismo de persistência que armazene a informação de percepção. Na arquitetura proposta, escolhemos o servidor de espaço de tuplas. Devido à granularidade escolhida pelo modelo de percepção, espera-se a necessidade de um grande espaço de armazenamento. O espaço de tuplas favorece o armazenamento, pois é uma estrutura escalável, distribuída e persistente.

### 5.4.3 Organização

A organização da informação de percepção inclui os processos de filtragem e transformação de eventos. A filtragem é baseada na recuperação de tuplas que satisfaçam determinados critérios de consulta e é posterior ao armazenamento. Desta forma, reduz a sobrecarga de apresentação, mas não favorece a redução do espaço de armazenamento, nem a privacidade do usuário.

O principal processo de transformação é a interpretação de eventos (Seção 5.5.5), que é encarregada de gerar novos eventos, de mais alta ordem semântica, a partir de um grupo de eventos existentes. A quantidade de eventos após a interpretação é sempre reduzida, embora os eventos originais sejam preservados no repositório.

### 5.4.4 Distribuição

A distribuição de informação é realizada por notificação e por consulta, com base em um critério de consulta. Na notificação, uma tupla é enviada para um processo, sempre que satisfizer um critério específico. Na consulta, são recuperadas tuplas existentes que satisfazem o critério estabelecido. Em geral, um componente vai usar a consulta para recuperar tuplas que foram geradas antes do início da execução do

processo e a notificação para acompanhar a criação de novas tuplas.

### 5.4.5 Consumo

O consumo da informação ocorre quando da sua apresentação ao usuário final ou pela sua retenção por um filtro de informação. Em mecanismos que trabalham em tempo real, o consumo da informação é instantâneo, ou seja, uma vez distribuída e apresentada, a informação não é mais necessária. Por outro lado, nos demais mecanismos, a informação pode ser consumida (consultada) em diferentes sessões de execução de um mecanismo.

Em alguns casos, a informação é coletada na estação de trabalho antes mesmo de ser consolidada no repositório central. Ocorre, então, o problema de notificar o cancelamento de uma informação armazenada e possivelmente distribuída e consumida. Por exemplo, imaginemos que um desenvolvedor crie algumas classes e depois decida abandonar a alteração sem realizar o *commit*, no repositório de software do projeto. A informação coletada na estação, durante a sessão de trabalho, vai indicar a criação de uma classe que não existe no repositório central, gerando uma inconsistência entre a informação de percepção e o modelo de software. A solução adotada é a notificação da destruição da classe que não se encontra no modelo de software, logo após a notificação da criação da classe. Entretanto, a percepção de um desenvolvedor que está ciente da criação é diferente da percepção do desenvolvedor que ainda não estava ciente da criação da classe. Para o primeiro, a criação tem maior relevância, enquanto que para o segundo, a criação e a destruição são eventos que se anulam e não tem maior relevância, a não ser que o segundo desenvolvedor esteja interessado no histórico dos artefatos envolvidos.

O consumo da informação de percepção é uma etapa que é realizada, em sua maior parte, pelas extensões colaborativas. Existem trabalhos que tratam especificamente desta etapa e das possibilidades de estabelecer filtros de informação e de seleção de extensões apropriadas para uma sessão colaborativa (DAVID e BORGES, 2001).

### 5.4.6 Destruição

Na prática, o armazenamento da informação de percepção é limitado pelo espaço de armazenamento disponível e pela perda de desempenho causada pelo alto volume de informação a ser processado para responder às consultas. Em um sistema concreto, uma política deve ser adotada para a destruição das anotações ou para a sua movimentação para um novo espaço de armazenamento.

Por exemplo, as anotações sobre um elemento do modelo de software poderiam ser destruídas quando o elemento deixasse de existir em todas as cópias do modelo. Por exemplo, quando uma classe deixasse de existir no repositório central e em todas as áreas de trabalho, seria possível remover todas as anotações sobre essa classe. Todas a ações sobre a classe seriam invalidadas por sua remoção do modelo. Entretanto, essa regra não é determinante da destruição de uma anotação, pois existem outras características a serem analisadas.

A remoção de um elemento do modelo de software gera uma modificação em todos os elementos associados ao elemento removido. Por exemplo, ao remover uma classe de um modelo, ocorre uma modificação no pacote associado a esta classe. Portanto, a anotação sobre um elemento de software inexistente é necessária para comunicar a situação de elementos que ainda persistem no modelo de software. Se a anotação sobre a remoção da classe fosse destruída, a anotação sobre a alteração do pacote seria perdida.

A destruição da informação de percepção não é uma operação trivial. A informação pode ser necessária no futuro para um estudo da evolução do projeto ou para facilitar sua auditoria. A opção mais simples é mover a informação de percepção para mídia terciária. A movimentação da informação seria controlada pelo ciclo de desenvolvimento dos projetos. A decisão pela destruição da informação é administrativa e é provável que seja determinada pela data da coleta ou outro parâmetro de interesse da gerência do projeto.

## *5.5 Arquitetura de Colaboração Transparente (ACT)*

A ACT, proposta nesta tese, resulta da adaptação das arquiteturas transparentes

para colaboração síncrona propostas por Begole (1998) e Li e Li (2002) com o acréscimo do conceito de *eventos semânticos*. Eventos semânticos descrevem o significado das operações realizadas pelo usuário da aplicação e são gerados a partir de interpretações dos *eventos de entrada* e *eventos de aplicação* das arquiteturas transparentes. Com essa infra-estrutura baseada em eventos, a ACT facilita a programação de um grupo de serviços de colaboração mais amplo do que o encontrado em propostas similares. A concepção da ACT tenta unificar serviços de percepção em espaço de trabalho (MAURER e MARTEL, 2002) (SCHÜMMER e HAAKE, 2001), serviços de colaboração assíncrona e percepção de produto (SARMA *et al.*, 2003) e memória individual e de grupo com base em ações.

Na ACT, os eventos estão isolados da aplicação que os geram, o que promove o desacoplamento entre o produtor e o consumidor dos eventos. Neste aspecto, a ACT foi inspirada nos sistemas de percepção com base em eventos e notificações (KANTOR e REDMILES, 2001).

A primeira contribuição da ACT é a proposta de eventos semânticos, que têm significado definido por um modelo independente da aplicação. A segunda contribuição é a integração de diferentes serviços de percepção em uma mesma infra-estrutura de apoio comum.

Esta abordagem permite que pequenas alterações em ambientes de trabalho concretos levem a um incremento da informação disponível para apoiar a colaboração entre os participantes. Em particular, além do potencial para a realização de atividades síncronas e assíncronas, a coleta de informações pode ser utilizada para identificar oportunidades de colaboração que não seriam evidentes para os participantes, mesmo que tivessem oportunidades para contato face-a-face.

A definição de uma arquitetura é uma etapa do processo básico de desenvolvimento com base em componentes (BROWN, 2000) (TEIXEIRA, 2003) que consiste de três passos básicos (Figura 5.2):

- **Entender o contexto:** estudo das práticas e sistemas atuais para fornecer uma melhor compreensão do domínio do problema. Neste caso, o domínio do problema é o apoio ao desenvolvimento de software colaborativo. Os sistemas atuais analisados foram citados no Capítulo 3.
- **Definir a arquitetura:** encapsular e distribuir funcionalidades entre componentes da

arquitetura, quais serão desenvolvidos ou adquiridos de terceiros. Os componentes manipulados no nível da arquitetura não apresentam implementação, simbolizando, apenas, os *serviços* fornecidos e consumidos entre eles. A descrição da ACT é resultado desta etapa.

- **Fornecer a solução:** implementar e montar o software, com a escolha de tecnologias e infra-estruturas adequadas, e também envolve testar e implantar o software no ambiente do usuário. Esta etapa foi realizada através da construção, por terceiros, de protótipos (Capítulo 6).

Esse processo pode ser realizado em ciclos iterativos, onde o resultado do primeiro ciclo alimenta a entrada do ciclo seguinte. Na proposta da arquitetura, realizamos diversos ciclos, um para cada protótipo desenvolvido.

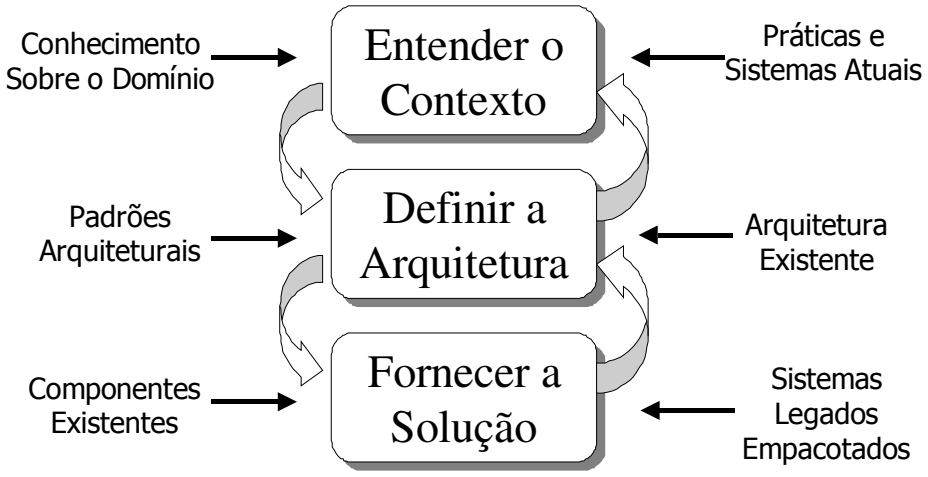



**Figura 5.2 Passos básicos do desenvolvimento baseado em componentes.**

## 5.5.1 Elementos da Arquitetura

Os elementos da arquitetura de implementação são apresentados na Figura 5.3.

Um **sistema de eventos e notificação** oferece serviços de roteamento e transferência de informações entre as estações da rede simétrica. Serviços de **controle de sessão**, **autenticação e autorização** e **catálogo** oferecem uma infra-estrutura que pode ser compartilhada entre as estações. Essa infra-estrutura é dependente da implementação utilizada. O **controle de sessão** é utilizado para coordenar a transferência contínua de eventos entre estações que estão apoiando tarefas de alto

acoplamento. Por exemplo, se duas estações estão realizando a edição colaborativa de um mesmo artefato, é estabelecido um canal de comunicação de alta prioridade para esses eventos, criando um grupo de comunicação dentro da rede.

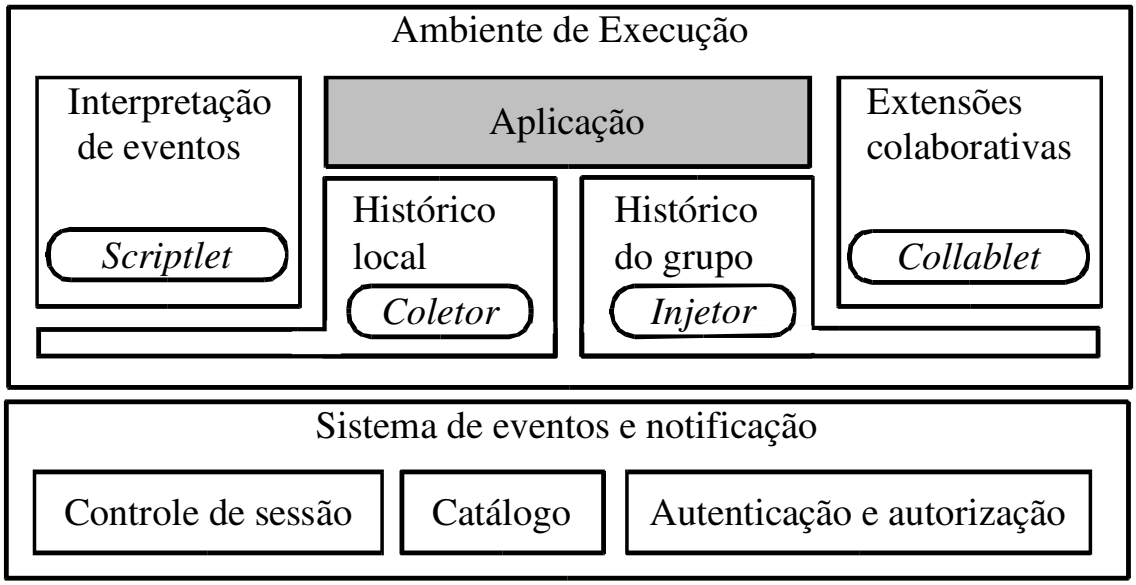



**Figura 5.3 Principais elementos da Arquitetura de Colaboração Transparente (ACT)**

A transferência de eventos nos demais casos ocorre através de notificação e consultas, com uma prioridade menor para poupar recursos da rede. Por exemplo, os eventos que estão sendo utilizados para a apresentação de informação de percepção assíncrona podem tolerar um maior atraso na recuperação de informações (na ordem de segundos) do que o atraso da edição e percepção síncrona (na ordem de milésimos de segundo).

A **autenticação e autorização** são implementadas em servidores que pertencem a infra-estrutura da rede. Uma estação local deve identificar o usuário antes de permitir o acesso ao sistema. Com base nessa identificação local, o usuário é então autorizado ou não a participar da colaboração.

O **catálogo** armazena informações sobre as aplicações e seus eventos que são apresentadas aos usuários para que determinem quais eventos devem ser monitorados. Essas mesmas informações são utilizadas pelas extensões colaborativas e pelos interpretadores de eventos.

A aplicação compartilhada, ou monitorada, encontra-se em um **ambiente de**

**execução** próprio em uma estação da rede. O ambiente de execução é um sistema em execução que oferece serviços básicos para a aplicação, provavelmente gerados pelo sistema operacional, um ambiente de janelas e por algum outro sistema dependente da plataforma de implementação da aplicação.

A aplicação está associada aos elementos descritos na Seção 5.5.2 (Figura 5.4). A fila de eventos da aplicação sofre dois tipos de interferência. Primeiro, os eventos da aplicação são coletados através de um sensor, um programa de computador, que coleta eventos (**coleta**). Os eventos coletados são colocados em um espaço de armazenamento local da estação (**histórico local**). O histórico tem um registro completo dos eventos de um ou vários usuários. Cada evento está associado com um único usuário, mas eventos de diferentes usuários podem estar localizados em um mesmo histórico local.

Segundo, a fila de eventos pode receber eventos provenientes de outras estações. Os eventos remotos chegam a estação local e são armazenados em um **histórico de eventos do grupo**, ordenados por ordem de chegada. Os eventos são enviados de uma estação para outra através do controle de sessão ou da solicitação de ***collablets***. *Collablets* são agentes genéricos que consultam e alteram os eventos disponíveis, a fim de apoiar a colaboração. Um *collablet* é uma abstração de programação para facilitar o desenvolvimento de mecanismos de colaboração na ACT, conforme apresentado mais adiante.

A **interpretação de eventos** é a técnica responsável por "reduzir" seqüências de eventos de nível semântico mais elementar em eventos de nível mais alto. O uso de um sistema de eventos e notificação é característico da **invocação implícita**, um estilo de arquitetura de software que evoluiu a partir da orientação a objetos (SHAW e GARLAN, 1996). O estilo divide os elementos da arquitetura entre geradores e consumidores de eventos. Os eventos, geralmente, transferem dados ou controle. O consumidor do evento não é conhecido pelo gerador de eventos. Um mecanismo de manipulação de eventos realiza a coleta e distribuição de eventos para grupos de consumidores. Os consumidores informam os eventos que desejam receber ao sistema de manipulação de eventos. A ACT permite também a consulta aos eventos passados, que é característica do estilo arquitetural *blackboard*, que utiliza um depósito de dados que pode ser consultado e atualizado pelos elementos da arquitetura (BOSCH, 2000). A combinação desses dois estilos arquiteturais facilita contornar os problemas de (a) baixo

desempenho no consumo de eventos da *blackboard* e de (b) ausência de memória de eventos passados da invocação explícita.

A manutenção de sistemas que combinam essas características é simplificada pelo baixo acoplamento entre os elementos da arquitetura, o qual facilita a substituição e inclusão dinâmica de elementos (BOSCH, 2000).

O ponto negativo da combinação desses estilos é que não é possível determinar uma especificação do comportamento total do sistema (BOSCH, 2000). Por exemplo, a reação aos eventos depende da presença de determinados consumidores de eventos, que podem estar ou não presentes. Ainda, um produtor de eventos que gera eventos incorretos pode ocasionar reações corretas localmente, porém incorretas globalmente, por parte dos consumidores de eventos. Para minimizar esses problemas, é necessário controlar a qualidade dos geradores de eventos e garantir a presença de elementos que sejam essenciais ao processamento.

### 5.5.2 Aplicação

Os principais elementos da proposta da ACT já se encontram instalados na estação de trabalho do participante. São eles: a *aplicação*, elemento que oferece *visualizadores* que permitem ao usuário perceber os objetos que estão sendo manipulados (*visão*) e operar sobre eles (*operação*). As operações são registradas através de *eventos*. Os objetos são armazenados em um *meio persistente*, em geral um repositório de projeto.

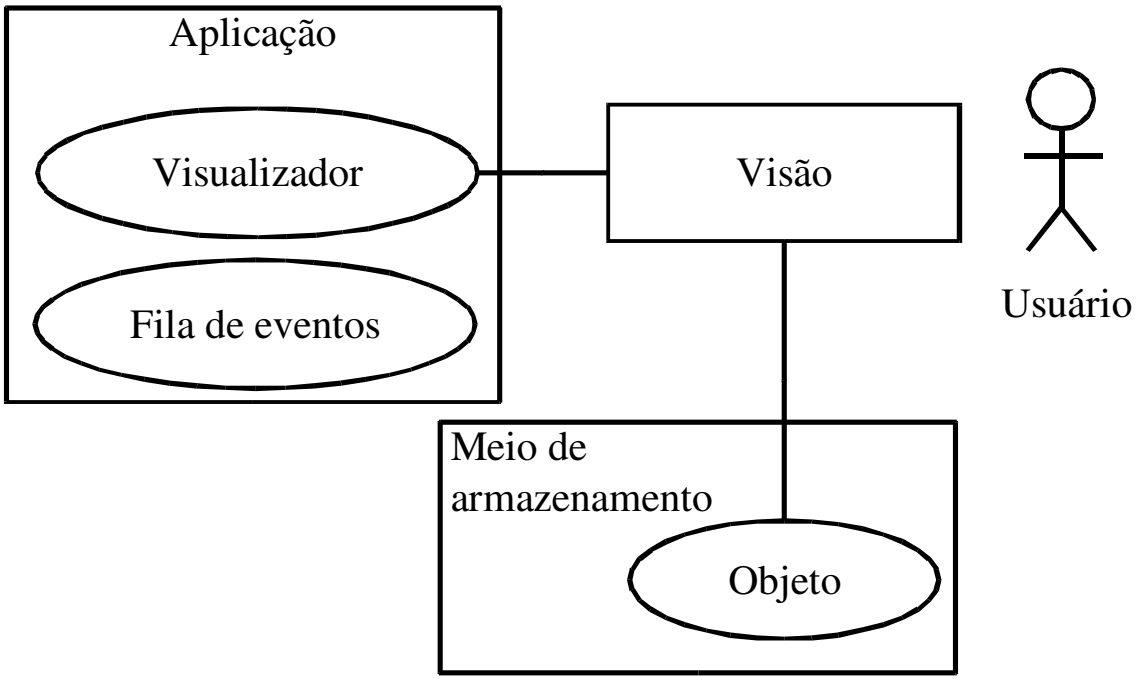



**Figura 5.4 Conceitos gerais: a relação da aplicação cliente, visualizador e a visão do usuário final - adaptado de (ANDERSON *et al.*, 2000).**

Utilizamos definições para esses termos adaptadas da proposta do *Chimera*, um sistema hipermídia para ambientes de desenvolvimento de software heterogêneos (ANDERSON *et al.*, 2000). A função do *Chimera* é possibilitar a marcação e navegação entre artefatos por um usuário. O objetivo da ACT é diferente, mas as características das aplicações são similares, portanto, adaptamos a formalização já realizada pelo *Chimera*.

As definições formam um modelo sobre o qual podemos realizar uma descrição do espaço de trabalho, dos objetos manipulados e das ações do usuário.

As definições dos principais conceitos utilizados para descrever uma aplicação na ACT são as seguintes (Figura 5.4):

- **Objetos**: são entidades persistentes cuja estrutura interna é irrelevante para a ACT;
- **Aplicação**: um programa de computador que apresenta visualizadores. Uma aplicação apresenta ao menos um visualizador;
- **Visualizador**: uma entidade que permite perceber representações visuais de objetos. As operações oferecidas por um visualizador variam e incluem, geralmente, navegação, criação, edição e remoção de objetos;

- **Visão**: denota um par (*visualizador*, *objeto*). Um objeto pode ser apresentado em mais de uma visualizador e, portanto, participar de mais de uma visão;
- **Usuários**: usuários interagem com as visões. O conceito de usuários permite identificar as interações de usuários diferentes;
- **Propriedades**: um par (*nome*, *valor*), onde *nome* é um identificador persistente, como uma cadeia de caracteres e *valor* é uma informação em qualquer formato arbitrário suportado na implementação. Propriedades são utilizadas para registrar meta-informação que não é mantida pelos visualizadores. Por exemplo, a identificação do autor de um objeto pode ser expressa como um par ("autor", *usuário*);
- **Relação**: uma associação entre dois ou mais objetos. Cada relação pode pertencer a um *tipo de relação*, que é um identificador persistente dependente do visualizador.
- **Evento**: um registro de uma operação realizada no visualizador pelo usuário. O evento deve registrar as informações necessárias e suficientes para a sua interpretação incluindo visualizador, objetos afetados e outros parâmetros. Existem dois tipos de eventos: **eventos operacionais**, que são coletados e processados pelas aplicações e **eventos semânticos**, que são eventos que interpretam seqüências de eventos operacionais, atribuindo a esses um significado no modelo de percepção.
- **Fila de eventos**: seqüência de eventos gerados em ordem de ocorrência das ações do usuário;
- **Meio de armazenamento**: um mecanismo responsável pelo armazenamento persistente de objetos. O meio de armazenamento é capaz de recuperar objetos através de algum tipo de identificador único e persistente.

Uma aplicação típica de edição utilizada no desenvolvimento de software apresenta todos os conceitos citados. Nota-se o predomínio de ferramentas que apresentam diversos visualizadores simultâneos, pois os artefatos de software apresentam várias representações diferentes, cada uma adequada a um tipo de trabalho (Figura 5.5).

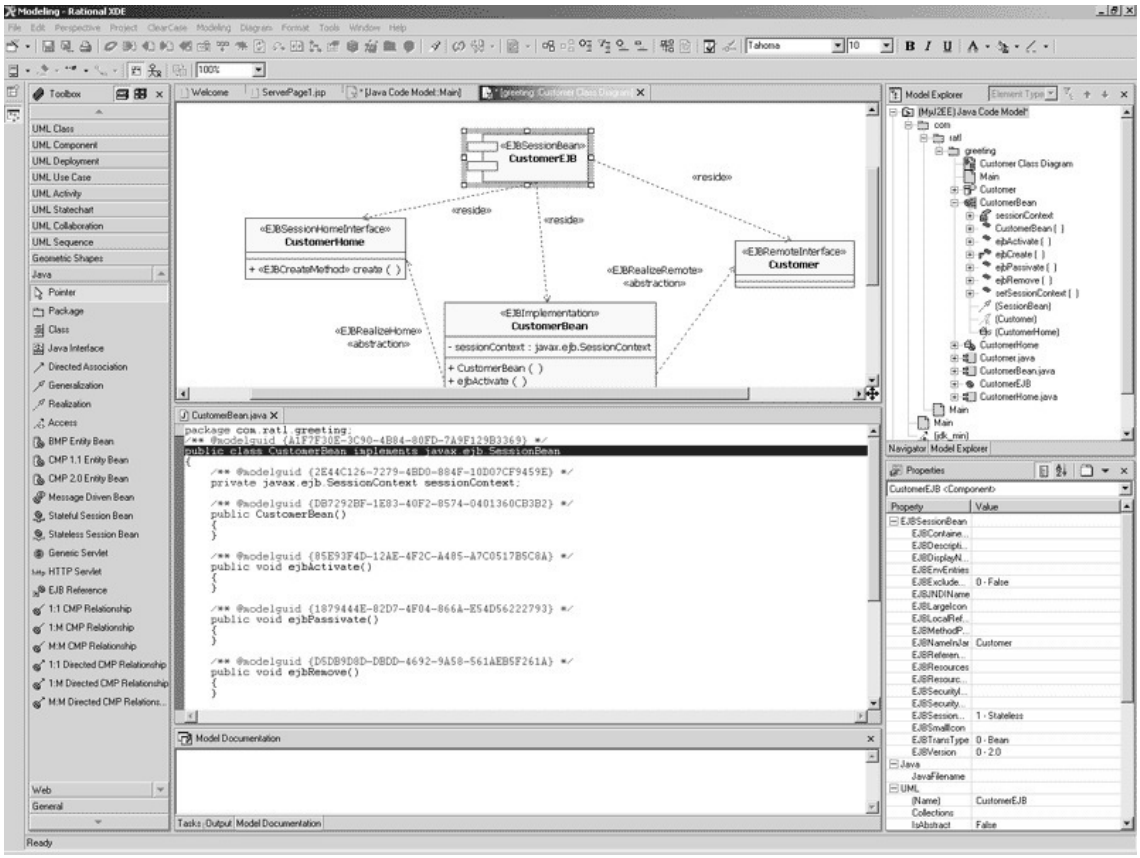


**Figura 5.5 Exemplo de interface de usuário de alta complexidade (RATIONAL, 2003).**

No exemplo citado, podemos notar que a ferramenta apresenta pelo menos cinco visualizadores ativos e outros tantos disponíveis, mas não presentes no campo de visão do usuário. Podemos perceber que dois visualizadores no centro da tela estão mostrando duas representações de diferentes visões para um mesmo objeto. A interação do usuário com essas visões gera eventos que comandam alterações nos visualizadores e nos objetos subjacentes. Todos esses elementos estão presentes na aplicação com a finalidade de apoiar uma tarefa realizada por um indivíduo. A noção do grupo se forma pela influência que essas ações individuais geram nos objetos compartilhados. Alguns dos eventos são consumidos localmente e não geram efeitos persistentes nos objetos compartilhados, como a rolagem de janelas e o redimensionamento de janelas. Outros eventos geram um efeito permanente, que será percebido pelos demais integrantes do grupo, em velocidades diferentes.

### 5.5.3 Histórico de Grupo e Históricos Locais

O histórico local é relevante na determinação do perfil de trabalho do usuário. O histórico de grupo é relevante para determinar as ações dos demais que não estão ainda refletidas na área de trabalho de todos os participantes. A informação do histórico permite inferir indícios sobre horários de trabalho, tempo e freqüência de uso de aplicações e operações sobre artefatos, além da própria enumeração de artefatos manipulados.

Embora distantes fisicamente, os desenvolvedores compartilham entre si fragmentos de modelos que são reutilizados e transformados para gerar novas aplicações. O enfoque deste trabalho é tratar o problema de isolamento entre os desenvolvedores, através da coleta de informações sobre os hábitos de trabalho e interesses individuais, ou seja, o perfil de trabalho do desenvolvedor. Partimos da premissa que as ações do usuário estão de alguma forma correlacionadas com seu perfil.

Promover a colaboração em uma equipe distribuída é uma tarefa difícil, pois os participantes necessitam (a) saber da existência e disponibilidade de seus colaboradores, (b) avaliar os custos e benefícios da atividade de colaboração e, então, (c) tomar ações para encenar a colaboração.

A existência e disponibilidade de colaboradores é uma informação obtida com facilidade. Os modernos sistemas operacionais e algumas aplicações encontradas no ambiente de trabalho dispõem desse tipo de informação. Existem certas dificuldades na identificação consistente de usuários, como a existência de múltiplos mecanismos de autenticação em um mesmo ambiente. A disponibilidade do colaborador é um item mais difícil de determinar do que a existência de um colaborador. A disponibilidade envolve a disposição e a contribuição potencial do colaborador.

A análise dos eventos do histórico local e do grupo são os pontos principais do apoio à descoberta de colaboradores. A informação armazenada no histórico é coletada de forma transparente, portanto, não envolve esforço consciente do usuário. Devido ao baixo esforço para a obtenção das informações, a proporção entre sua utilidade e esforço para sua obtenção é alta.

### 5.5.4 Interfaces para Coleta de Eventos

Em geral, uma ferramenta CASE apresenta cinco pontos de extensão. Os pontos de extensão são: (a) o sistema de execução da plataforma, (b) o sistema de janelas do ambiente de execução, (c) o mecanismo de extensibilidade oferecido pela ferramenta, (d) o mecanismo de persistência e (e) o repositório da ferramenta.

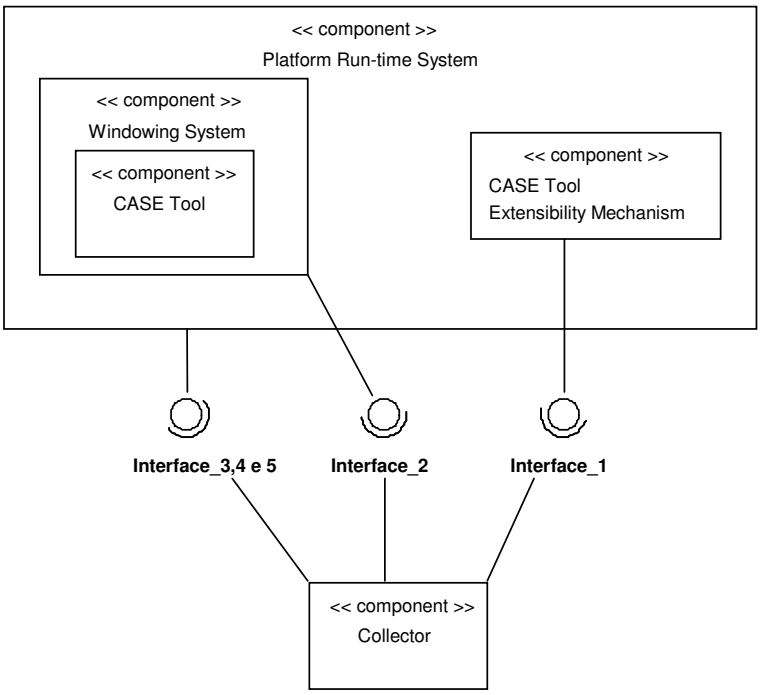



**Figura 5.6 Interfaces de integração com a ferramenta CASE (Diagrama de componentes da UML).**

Cada ponto de extensão está associado com um sistema de execução que apóia a execução da aplicação. O sistema de execução da plataforma é o sistema que executa as instruções do código da aplicação. Por exemplo, na plataforma Java, esse sistema é a máquina virtual Java (*Java Virtual Machine*). O monitoramento das instruções de execução é uma técnica utilizada por depuradores e ferramentas similares. O sistema de janelas do ambiente de execução é responsável pela coordenação das janelas e demais elementos da área de trabalho. As instruções específicas para a manipulação da interface de usuário são executadas por esse sistema. O mecanismo de extensibilidade oferecido pela ferramenta é um módulo específico que permite a integração de novas ferramentas que adicionam funcionalidades à ferramenta original. O mecanismo de persistência é um

módulo utilizado pela ferramenta para realizar o armazenamento e recuperação de dados. Por fim, o repositório da ferramenta é o sistema que centraliza e integra os dados manipulados pelas ferramentas.

As interfaces dos pontos de extensão oferecem informações diferentes. A integração de elementos visíveis área de trabalho como, por exemplo, teleapontadores ou outra extensão colaborativa para compartilhamento de janelas ou telas, pode usar informações importantes por meio da interface do sistema de janelas do ambiente de execução (Interface_2). A interface do repositório, geralmente, oferece informações relacionadas à representação da informação persistente no repositório (Interface 5). As informações dessas duas interfaces exigem um maior processamento para que seu significado possa ser interpretado dentro da semântica esperada pelo usuário final.

Informações mais adaptadas à semântica da aplicação são obtidas por meio das demais interfaces: as interfaces Interface_1, Interface_3 e Interface_4 oferecem informações sobre o modelo interno utilizado pela aplicação. As interfaces Interface_3 e Interface_4 apresentam o modelo do ponto de vista do programador da aplicação, enquanto a Interface_1 oferece uma interface de programação simplificada, direcionada para o programador de extensões da aplicação.

A Interface_5 apresenta diferenças em relação às outras três interfaces. Primeiro, as informações oferecidas estão consolidadas. As demais interfaces oferecem informações enquanto ainda encontram-se na estação de trabalho e não foram comunicadas a nenhuma outra estação. Com isso, torna-se claro que a Interface_5 não permite romper o isolamento entre os desenvolvedores. Segundo, uma desvantagem da Interface_5 é que muitos repositórios armazenam informações de forma diferente daquela que é manipulada pela ferramenta. Ao recuperar dados diretamente do repositório é necessário interpretá-los, da mesma forma que a ferramenta os interpreta.

As interfaces Interface_2 e Interface_1 favorecem a coleta de informações com uma semântica mais próxima da usada no diálogo com o usuário. Por exemplo, um repositório de arquivos trabalha com abstrações diferentes das usadas por uma ferramenta de desenvolvimento orientado a objetos. A proximidade com a semântica usada no diálogo é um fator importante para tornar a informação mais significativa do ponto de vista do usuário.

A Interface_1 é provida com a finalidade de permitir a criação de ferramentas adicionais. A complexidade e diversidade de técnicas de desenvolvimento que podem ser adotadas pelos usuários impedem que uma ferramenta seja construída com todo o apoio adequado para as diferentes necessidades de cada processo de trabalho específico. Sendo assim, o desenvolvedor da ferramenta CASE gera uma ferramenta com funções básicas e oferece a Interface_1 como meio de complementar sua ferramenta com ferramentas adicionais desenvolvidas por terceiros. A Interface_1 oferece informações úteis para o desenvolvimento de mecanismos de percepção, embora não seja construída com esse fim específico.

As interfaces devem prover dois tipos de serviços: enlace e coleta. O serviço de enlace estabelece uma conexão entre o Coletor e o componente monitorado. As funções básicas são o início, a negociação e a conclusão de uma sessão de coleta. O serviço de coleta oferece funções de consulta de dados e notificação. As interfaces Interface_1, Interface_2 e Interface_3 oferecem o serviço adicional de negociação de área visual. A área visual é um espaço da área de trabalho do usuário onde as informações de percepção são apresentadas.

### 5.5.5 Interpretação de Eventos

O modelo do Chimera trata da relação entre as ações do usuário e a geração de operações sobre objetos por meio da aplicação e seus visualizadores. O item que necessita de maior esclarecimento é justamente o objeto. Consideramos que o objeto seja uma instância de um modelo de informação estruturada e discreta (Seção 5.3). No caso das aplicações que foram alteradas com sucesso, esse modelo corresponde ao proposto na *Unified Modeling Language* (UML), com base na *Meta Object Facility* (MOF).

A complexidade das informações manipuladas nesse tipo de visualizador não é capturada em sua totalidade nos eventos semânticos da ACT. Os objetos vão sendo descobertos a medida que eventos sobre eles vão sendo gerados. Caso as aplicações gerem eventos sobre objetos inexistentes, a colaboração não deve tentar detectá-los. As aplicações geram eventos que não são verificados pela infra-estrutura. Os detalhes são coletados nos eventos operacionais que são repassados entre as aplicações. Apenas as

aplicações podem interpretar corretamente os eventos operacionais.

A interpretação de eventos é o processo que reconhece uma seqüência característica da operação de uma aplicação e gera um novo evento que representa essa mesma seqüência. A seqüência original é utilizada para promover sessões síncronas de edição colaborativa, ou para registrar e reproduzir ações locais. Os eventos gerados a partir dela são utilizados para atualizar um modelo dos objetos e ações que pode ser explorado para gerar informações de percepção assíncronas e perfis de usuário.

Existem três filas de eventos disponíveis:

- fila de eventos de entrada e execução da aplicação: $e_1, e_2, ..., e_m$;
- fila de eventos da aplicação: $a_1, a_2, ..., a_n$;
- fila de eventos semânticos: $s_1, s_2, ..., s_r$;

onde $m > n > r$.

A primeira fila é composta por eventos de baixo nível, que correspondem ao registro realizado pelo sistema de janelas e do ambiente de execução da aplicação. A segunda fila é composta por eventos de mais alto nível que correspondem às operações da aplicação ativadas pelos eventos da primeira fila. A terceira fila é composta de eventos que correspondem às operações que são descritas através de propriedades associadas aos eventos.

**Tabela 5-2 Relação entre os eventos de operação da interface de usuário e a semântica pretendida na operação.**

| *Eventos da Interface* | *Eventos semânticos* |
|---|---|
| Mover mouse para a posição $x_1$,$y_2$ | |
| ... | |
| Mover mouse para a posição $x_n$,$y_n$ | |
| Clicar com o botão esquerdo | Clicar no botão “Criar classe” |
| Mover mouse para a posição $x_k$,$y_k$ | |
| Clicar com o botão esquerdo | Criar classe |
| Mover mouse para a posição $x_m$,$y_m$ | |
| Clicar com o botão esquerdo | Selecionar classe criada |
| Digitar a letra ‘F’ | |
| Digitar a letra ‘o’ | |
| ... | |
| Digitar a letra ‘r’ | |
| Digitar <ENTER> | Renomear classe para “Fornecedor” |

A correspondência entre eventos é descrita através de um programa que é notificado quando um conjunto de eventos aguardados ocorreu (*scriptlet*). Um scPara cada seqüência de eventos, o *scriptlet* associa outros eventos de mais alta ordem. Os *scriptlets* são ativados pelo reconhecimento das seqüências de eventos no histórico local. Os eventos associados a seqüência reconhecida são inseridos na fila local apropriada e tornam-se disponíveis, juntamente com os eventos operacionais.

Por exemplo, uma seqüência de eventos dos dispositivos de entrada pode ser associada com eventos de aplicação correspondentes (Tabela 5-2). Quanto mais seqüências forem descritas, mais ampla será a capacidade de apoiar a colaboração.

Os eventos das diferentes filas apresentam quantidade e granularidade diferentes (Figura 5.7).

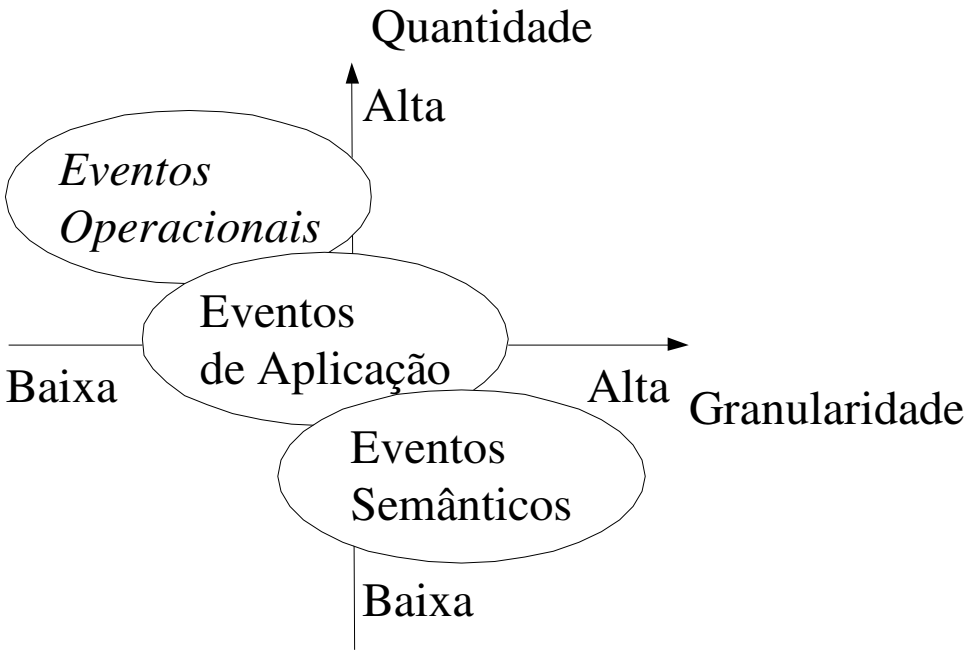



**Figura 5.7 Relação entre quantidade e granularidade de eventos.**

Eventos operacionais ocorrem em maior quantidade e menor granularidade. Por exemplo, um evento operacional como “mover mouse para ponto x, y” ocorre com alta freqüência e oferece uma quantidade reduzida de informações. No outro extremo, eventos semânticos ocorrem em menor quantidade e maior granularidade. Por exemplo, o evento semântico “Marcos criou uma classe com nome 'Fornecedor'” é resultado da redução e interpretação de diversos eventos de menor ordem e, também, da obtenção de informações adicionais, como nome do usuário corrente e nome da classe, que são provenientes da inspeção do estado do ambiente de execução e da aplicação. Os eventos de aplicação representam em quantidade e granularidade intermediários que podem ser utilizados para facilitar a descrição de outros eventos. Por exemplo, os eventos de aplicação “Marcos é o usuário corrente do sistema”, “usuário ativou a opção 'criar classe'” e “usuário renomeou classe para 'Fornecedor'” são eventos que podem ser utilizados para descrever o evento semântico “Marcos criou uma classe com nome 'Fornecedor'”.

### 5.5.6 Extensões Colaborativas

Por causa da alta disponibilidade de eventos, propomos um mecanismo para a

descoberta de colaboradores com base no histórico individual gerado a partir de amostras dos eventos coletados das aplicações integradas. A descoberta de colaboradores pode ser utilizada para identificar oportunidades de colaboração que não seriam evidentes para os participantes, mesmo que tivessem oportunidades para contato face-a-face. Esse conjunto de serviços atende a demandas de diversos modos de colaboração necessários para apoiar diferentes cenários concretos do desenvolvimento de software globalizado.

Uma extensão colaborativa é um mecanismo que consulta, cria ou remove eventos dos históricos. As extensões são programadas através de *collablets*. Um *collablet* é uma abstração de programação que visa facilitar a programação de extensões da interface de usuário de aplicações legadas com o uso de transparências, janelas independentes, painéis adicionais e outras decorações de interface de usuário.

*Collablet* é um neologismo formado por analogia aos termos *applet* e *servlet* usados na plataforma Java, e também por analogia ao termo *viewlet* da aplicação *Viewlet Builder* da empresa *Carbon*. O *collablet* é aquele que permite ou facilita a colaboração (*collaboration let*).

Um *collablet* controla uma porção da área do *desktop* da estação de trabalho e se comunica com a ACT consumindo e gerando eventos. O *collablet* utiliza a área de interface para interagir com o usuário para apresentar e coletar informações. O usuário pode ativar e desativar *collablets* para controlar a interrupção do trabalho e a carga de informação recebida da maneira que achar conveniente.

O conceito de *collablet* visa facilitar a programação de mecanismos de colaboração que façam uso de eventos da ACT. A idéia é oferecer um conjunto de classes e interfaces (*Application Programming Interface* - API) ao programador de *collablets* que permita receber notificações quando da ocorrência de eventos que satisfaçam uma determinada expressão de busca. A API, também, permite que um *collablet* gere um evento novo e o introduza na memória local da estação, tornando-o disponível para outras aplicações. A API está implementada na plataforma Java, com base na API do espaço de tuplas (*JavaSpaces*).

Um exemplo com duas estações de trabalho pode ilustrar o uso dessa API (Figura 5.8) que apresenta quatro primitivas: escrever, notificar, ler e apagar.

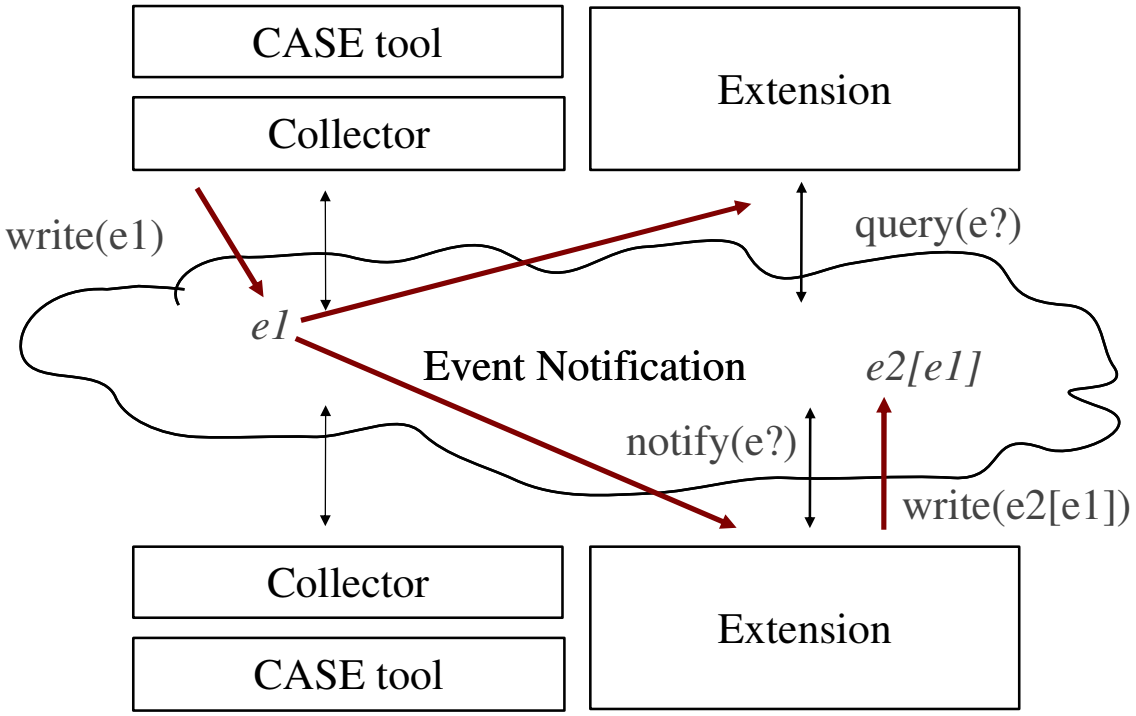



**Figura 5.8 Duas estações em comunicação via espaço de tuplas.**

O coletor (*collector*) normalmente registra eventos que foram observados pelos sensores colocados na ferramenta CASE (*CASE Tool*). Na Figura 5.8, uma estação está gravando um evento por meio de uma tupla escrita no espaço de tuplas (*write(e1)*).

A recuperação dos eventos se dá através de dois comandos: notificar e ler. O comando notificar (*notify(e?)*) solicita que toda a escrita de tupla do tipo seja comunicada. Apenas as tuplas escritas após a execução do comando serão comunicadas. O outro comando, ler (*query(e?)*), recupera uma tupla existente no espaço que satisfaça o modelo de consulta. Cada execução do comando ler recupera apenas uma tupla. Para realizar a associação entre duas tuplas, é necessário gravar a identificação de uma tupla como um campo de outra tupla (*write(e2[e1])*). Dessa forma, é possível representar informações de maior complexidade, com a referência entre tuplas servindo como uma forma de estabelecer relacionamentos entre elementos do modelo.

A última primitiva, apagar (*take(e?)*), que não está representada na Figura 5.8, remove uma tupla de forma permanente do espaço de tuplas. A primitiva é raramente utilizada, pois a remoção de tuplas do espaço é uma tarefa realizada pelos administradores do espaço de tuplas.

## *5.6 Processo de Desenvolvimento*

A descrição do processo de desenvolvimento descreve papéis e atividades necessárias para a aplicação da abordagem.

### 5.6.1 Papéis

A ACT pressupõe a existência de usuários que desempenham ações em contextos diferentes. São definidos cinco papéis:

- **P1: Desenvolvedor da Ferramenta CASE (***CASE Tool Developer***)**: responsável pela construção da ferramenta que oferece o apoio individual às tarefas de desenvolvimento de software. Este papel pode ser responsável pela implementação de um mecanismo de extensão da própria ferramenta.
- **P2: Desenvolvedor da Extensão Colaborativa (***Collaboration Enhancement Developer***)**: responsável pela construção de um mecanismo de colaboração, com ênfase em percepção, utilizando os eventos disponíveis. O desenvolvimento da extensão deve ser realizado como uma sobreposição de uma visualização existente ou como uma janela independente.
- **P3: Administrador da Base de Eventos (***Event Base Administrator***)**: define os eventos disponíveis, administra a evolução de eventos, grupos de servidores e o ciclo de vida dos eventos.
- **P4: Desenvolvedor de Integração (***Integration Developer***)**: responsável por construir um programa que transforma eventos. Eventos locais da ferramenta CASE são coletados e transformados em eventos do modelo de percepção; eventos do modelo de percepção são transformados em novos eventos, com maior significado para o desenvolvedor.
- **P5: Administrador de Ambiente (***Environment Administrator***)**: define ambientes colaborativos compostos por aplicações e extensões colaborativas com base nas necessidades de um grupo específico em um cenário específico. O Administrador de Ambiente pode solicitar o desenvolvimento de novas extensões ou integrações.

Um único indivíduo pode realizar todos os papéis listados. Sugere-se que, em uma organização que utilize a ACT, eventos e aplicações sejam administrados em um

catálogo. Os catálogos poderiam ser compartilhados entre instalações, incentivando a reutilização.

### 5.6.2 Atividades

São previstas quatro atividades no processo que leva à composição de ambientes colaborativos com base na abordagem proposta (Figura 5.9).

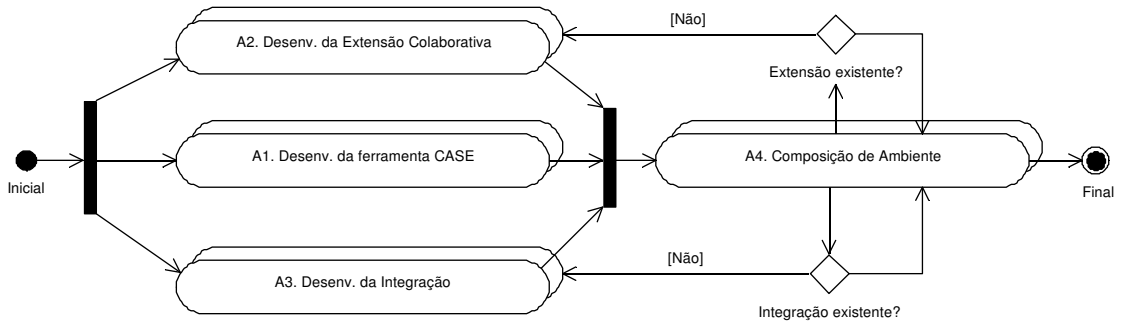



**Figura 5.9 Visão geral do processo de aplicação da abordagem proposta para a composição de ambientes colaborativos (Diagrama de atividades da UML).**

A descrição das atividades é a seguinte:

- **A1 Desenvolvimento da ferramenta CASE**

O desenvolvimento de uma ferramenta CASE é uma atividade complexa que demanda vários anos para a produção de uma ferramenta estável e confiável. As ferramentas são normalmente desenvolvidas por meio de ciclos de desenvolvimento que duram de dois a seis meses. As versões sucessivas de uma ferramenta tentam manter compatibilidade com suas versões anteriores, o que confere certa estabilidade às interfaces descritas logo a seguir.

Por causa da característica de transparência da abordagem, nenhuma restrição é feita sobre a atividade de desenvolvimento da ferramenta CASE. O papel de desenvolvedor da ferramenta CASE é, naturalmente, exercido por diversos papéis diferentes, do ponto de vista desta atividade.

O produto final desta atividade é a determinação das cinco interfaces para uma

ferramenta CASE (Seção 5.5.4), que serve de entrada para a próxima atividade.

- **A2 Desenvolvimento da extensão colaborativa**

O desenvolvimento da extensão é uma atividade voltada para a reutilização do mecanismo de percepção. As principais tarefas desta atividade são (a) identificar o serviço do mecanismo, possivelmente, por meio da observação de uma implementação ou descrição de um mecanismo já implementado; (b) identificar características visuais estáticas e dinâmicas da interface de usuário do mecanismo e (c) identificar os elementos do modelo de percepção que é utilizado pelo mecanismo.

- **A3 Desenvolvimento da Integração**

O uso de padrões de projeto facilita a realização da integração, em particular, os padrões *Command* e *Observer* (GAMMA *et al.*, 1997). O padrão *Command* materializa a fila de eventos da aplicação em um objeto que ordena as operações solicitadas pelo usuário. Cada operação é descrita em um comando, o que facilita a detecção das operações locais. O padrão *Observer* permite a um objeto receber a notificação sobre mudanças em um outro objeto. Esses dois padrões de projeto estão presentes na biblioteca de programação Java, onde a maioria dos protótipos foi implementada. O padrão *Command* é utilizado nas classes *Action* do pacote *Swing* e o padrão *Observer* no modelo de componentes *JavaBeans* e na própria classe *Object*.

- **A4 Composição de um Ambiente**

Esta atividade é uma atividade com reutilização das aplicações e extensões na composição de novos ambientes. Os elementos que irão compor o novo ambiente são determinados pelas características dos usuários, das suas ferramentas e do contexto de trabalho.

Essas características auxiliam a decidir quais as extensões que podem ser úteis para uma determinada equipe. A escolha das extensões determina a existência de coletores compatíveis com as aplicações envolvidas.

## *5.7 Trabalhos Relacionados*

A Arquitetura de Colaboração Transparente (ACT) estende a arquitetura de

transparência de Begole (1998), pela adoção de interpretação de eventos proposta por Li e Li (2002). A interpretação de eventos permite obter uma descrição do significado dos eventos compartilhados. Na proposta de Li e Li, o significado dos eventos é utilizado para promover o compartilhamento de aplicações heterogêneas. Na ACT, a interpretação é utilizada para promover serviços de coordenação e colaboração assíncronas. Com isso, pretende-se oferecer um espectro mais amplo de apoio adequado para as necessidades do desenvolvimento de software globalizado, onde os participantes precisam trabalhar em diferentes modos de colaboração.

A proposta de *collablets* permite que a ACT seja ampliada através da programação de novos mecanismos. Li e Li propõem o treinamento na interpretação de eventos. A arquitetura de implementação da ACT adota o uso de sensores nas estações clientes. Nesse sentido, a implementação é similar a usada no *Palantír* (SARMA *et al*., 2003). Porém, a interpretação dos eventos e dos objetos não é reproduzida dentro de um servidor. Na ACT, toda a interpretação é realizada nas estações de trabalho, o que facilita a interpretação de novos eventos e objetos sem a necessidade de interferir na implementação da ACT.

Neste trabalho, o apoio se destina aos usuários cuja principal tarefa colaborativa é o compartilhamento de editores gráficos. Os visualizadores considerados são aplicações capazes de apresentar diagramas bi-dimensionais, onde o conhecimento necessário para o entendimento do software está semi-formalizado. Diagramas são o principal instrumento de comunicação entre desenvolvedores e a maior parte das discussões de alto-nível, que envolvem criatividade e incerteza, ocorrem sobre esse tipo de representação.

Além disso, o conhecimento proveniente de diversas fontes é explicitado nos diagramas, o que pode levar a uma maior necessidade de colaboração nesse tipo de documento. Outros tipos de documento podem ser compartilhados, bastando para tanto que se estude o mapeamento das informações de seu visualizador para o modelo da ACT. Cada classificador da UML (classe, pacote, estado, caso de uso) é mapeado como um "objeto" nos modelos da ACT, as propriedades da UML são mapeadas para "propriedades" da ACT e os relacionamentos são mapeados para "relações". A UML oferece um modelo mais detalhado que o da ACT. Mantendo o modelo da ACT genérico podemos definir diversos outros mapeamentos. Por exemplo, o modelo da

ACT pode ser usado para representar informações como “Marcos criou a classe 'Fornecedor'”, com base em um mapeamento dos elementos da UML, mas também pode afirmar que “Antônio apagou o arquivo Y” com base em outro modelo que considere um modelo de sistema de arquivos.

O modelo de percepção de Rodden descreve um grafo onde os nodos representam objetos e as arestas representam relações entre os objetos. Existe uma função *r(o1, o2)* que determina a correlação *c* entre dois objetos, onde *c* varia entre 0, para objetos não relacionados, até 1, para objetos fortemente relacionados (RODDEN, 1996). O modelo de Rodden é muito utilizado como base para outros modelos de percepção. A função *r* no modelo de Rodden pode ser comparada as análises de impacto de mudanças exploradas no Palantír (SARMA *et al.*, 2003). No caso da ACT, a função *r* pode ser definida de diversas maneiras. Ou seja, uma mesma coleção de objetos e eventos pode gerar interpretações diferentes. O modelo da ACT oferece a oportunidade de definir uma relação r com base em diversos critérios. Por exemplo, podemos utilizar o conceito de relação para determinar uma função *r1* que considera a estrutura estática de classes definidas através de relações ou propriedade da ACT e uma função *r2* que calcule o (inverso) do número de operações realizadas no ambiente entre a última operação sobre o objeto *o1* e a subseqüente operação sobre o objeto *o2*. A definição de *r1* seria adequada para avaliar o impacto sobre alterações em um diagrama. A definição de *r2* poderia ser utilizada para avaliar a correlação entre as definições de dois objetos. A ocorrência de operações conjuntas freqüentes sobre um grupo de objetos pode revelar uma dependência semântica que não é evidente pela estrutura estática das classes.

A plataforma de agentes CUMBIA (MORENO *et al.*, 2003) explora a possibilidade da utilização de informações coletadas na área de trabalho de computadores pessoais como forma de promover colaborações oportunísticas. A coleta das informações ainda não está totalmente definida na CUMBIA, mas a abordagem de gestão de conhecimento a partir da coleta de informações das aplicações guarda alguma semelhança com a proposta do histórico da ACT.

Algumas características descritas na ACT podem ser encontradas em outros sistemas. Sistemas que avaliam a usabilidade de interfaces também utilizam coletas de eventos de interface (YVORY e HEARST, 2001). A interpretação de eventos de entrada em eventos de aplicação é uma técnica utilizada para reduzir o espaço necessário para

registrar a operação da aplicação por parte de um usuário. Sistemas de colaboração com base em comunicação, como o *Babble* (ERICKSON e KELLOG, 2000), baseiam a coordenação nas áreas compartilhadas em protocolos sociais. Sistemas de recomendação podem utilizar informações sobre o histórico do usuário para encontrar preferências e indicar outros objetos similares (CLAYPOOL *et al*., 2001). Ambientes colaborativos voltados para o ensino, como o AgentSheets (REPENNING *et al.*, 2001) adotam a possibilidade de configuração por parte do usuário sem a necessidade de um programador. Os ambientes evoluem com a adição de novos componentes de aprendizado, mediante programação. Percepção com base em memória de eventos é explorada em sistemas como o POLITeam (MARK e BORDETSKY, 1998). As informações são coletadas durante o uso das aplicações com o monitoramento combinado de eventos ocorridos nos dispositivos de entrada (teclado e mouse) e no ambiente de execução das aplicações. O monitoramento das ações de um usuário para a captura de seu processo de trabalho é uma prática proveniente de repositórios ativos (YE e FISCHER, 2000).

A proposta da arquitetura Clover (LAURILAU e NIGAY, 2002) é fundamentada em três conceitos: comunicação, coordenação e produção (*communication*, *coordination* e *production*). O conceito de produção se refere à área compartilhada, onde estão os objetos criados (produzidos) pelos usuários. Laurillau e Nigay advogam que uma arquitetura de *groupware* deve buscar o equilíbrio entre esses três conceitos. A Clover é uma arquitetura conceitual, a ACT pode ser caracterizada como uma arquitetura de implementação coerente com os conceitos da Clover.

As facilidades de extensão da ACT são similares as do *framework* ANTS (LÓPEZ e SKARMETA, 2003). Serviços de percepção, modelo de componentes e a integração com *middleware* existente são três das características do ANTS que se assemelham a esta proposta. Entretanto, as extensões no ANTS se referem ao apoio à colaboração. O ANTS oferece uma biblioteca para a programação de aplicações colaborativas. Por outro lado, a similaridade da ACT com o ANTS poderia indicar que a implementação da ACT poderia ser realizada, em parte, com o uso deste *framework*. Com isso, seria possível concentrar o desenvolvimento da interpretação de eventos. Este ponto ainda está para ser investigado. Outra arquitetura similar ao ACT é a WWG (MARQUÈS e NAVARRO, 2001). A WWG é aplicada na construção de ambientes de

ensino e oferece mecanismos de percepção assíncronos. A difusão de eventos da WWG é muito similar a da ACT. A infra-estrutura da WWG baseia-se na extensão WebDAV (WEBDAV, 2003).

## 5.8 Conclusão

Os elementos da ACT apresentam responsabilidades que favorecem a implementação de edição colaborativa, mecanismos de informação de percepção e obtenção de perfil de usuário. A ACT descreve uma família de ambientes colaborativos similares e uma mesma implementação com base na ACT poderia ser adaptada para uma mesma implementação pode atender a diversos cenários concretos. Como princípio, a ACT é não intrusiva, tanto na aplicação legada, quanto no modo de trabalho do usuário da aplicação. Entretanto, a aplicação da ACT é restrita a colaboração que evolui a partir do modelo individual de trabalho e que depende de baixa coordenação implementada pelo ambiente, ou seja, na mediação. Mecanismos de coordenação mais eficientes podem ser implementados através de protocolos sociais ou de extensões colaborativas que atuem sobre os históricos antes que os eventos sejam difundidos.

A ACT ainda descreve papéis e distribui responsabilidades para diversos perfis de desenvolvedores. Através dos papéis a programação de mecanismos de colaboração pode ocorrer em paralelo com a evolução das aplicações. A integração das aplicações e a configuração dos ambientes colaborativos são distribuídas entre os próprios usuários do ambiente.

Na ACT, o custo da obtenção de eventos é único, ou seja, a coleta por um mecanismo de um evento de entrada requer o mesmo esforço que o da coleta de uma ação semântica por parte do usuário. Com isso, espera-se incentivar a criação de novos mecanismos de colaboração. A programação de novos mecanismos requer o aprendizado de uma API reduzida e da lista de eventos catalogada no ambiente.

Por fim, o histórico da ACT facilita a avaliação das possibilidades de apoio à colaboração, antes da construção de um *groupware* através da obtenção de métricas sobre a geração dos eventos (*baseline*). Após a introdução do apoio à colaboração, é possível continuar monitorando o trabalho e verificar o impacto das alterações. Esse potencial para a observação do trabalho individual e em grupo pode levar a um

entendimento mais adequado da colaboração entre os participantes.

# 6 Protótipos Desenvolvidos

Este capítulo descreve os protótipos desenvolvidos com o objetivo de testar as principais idéias envolvidas na Arquitetura de Colaboração Transparente (ACT) e a refinar a arquitetura como um todo. A ACT define os principais elementos que compõem cada protótipo. Dentre os elementos, o que apresenta maior complexidade é o coletor de informação. A descrição dos protótipos se concentra na explicação de como o coletor foi realizado, visto que a transparência depende do sucesso na construção desses coletores. Os coletores variam de acordo com o modelo de percepção que utilizam e com a técnica de programação usada na construção.

A construção dos protótipos tem como pré-requisitos (a) a capacidade de extensão de aplicações e (b) a capacidade de instrumentação das aplicações por meio de alterações no ambiente de execução ou nos códigos fonte e objeto dessas aplicações.

Foram desenvolvidos seis protótipos

- Compartilhamento de aplicações
- CineOdyssey
- Ariane
- Infra-estrutura MAIS
- *GAW*/*CVS*-*Watch*/Ritmo de trabalho
- Coletores para o *Rational Rose*

Os protótipos são parte integrante da infra-estrutura colaborativa proposta no projeto *OdysseyShare* (WERNER *et al.*, 2002), que visa investigar a aplicação de apoio a colaboração no processo de desenvolvimento de componentes de software.

Os protótipos foram integrados em três aplicações:

- *Odyssey*
- Eclipse
- *Rational Rose*

Quatro sistemas já existentes foram utilizados como fonte de informação para realizar a coleta passiva de dados:

- sistema de janelas e entrada de teclado: *Java Swing e Abstract Window Toolkit* (AWT)*;*
- sistema de persistência: *Java Data Objects* (JDO);
- sistemas de extensão: carga dinâmica de módulos dos ambientes *Odyssey* e *Eclipse*, *Rational Rose Extensibility Interface* (REI);
- sistema de execução: *Java Virtual Machine* (JVM).

Para interferir e mediar a colaboração, a ACT exige a configuração prévia dos pontos de colaboração em cada aplicação. Um ponto de colaboração pode ser identificado por um gesto (*gesture,operação de um dispositivo de entrada, por exemplo, movimentar o* mouse *é um gesto*) na interface de usuário ou por um evento específico na execução da aplicação. A descrição dos eventos da aplicação pode ser reutilizada pelo grupo de participantes que operam a mesma aplicação. A descrição de eventos pode ser realizada sem a necessidade de programação, desde que seja criada uma técnica para a determinação dos pontos de colaboração através do monitoramento e análise da execução da aplicação.

Fica a cargo de cada usuário determinar, previamente, quais aplicações e respectivos eventos devem ser compartilhados. A descrição dos eventos é variável para cada aplicação, embora muitos eventos se mantenham os mesmos. O desenvolvedor seleciona os eventos de acordo com critérios próprios. Fatores que podem influir na decisão de compartilhar são privacidade, consumo de recursos e objetivos em relação a colaboração (GRUNDY *et al.*, 2000).

Uma vez configurada a aplicação, a arquitetura de colaboração transparente é capaz de oferecer serviços de colaboração similares aos encontrados em *groupware*. Dependendo do grau de modificação nessas aplicações legadas e no seu ambiente de execução, diversos serviços de colaboração são possíveis (BEGOLE, 1998).

Por exemplo, uma aplicação que permita o monitoramento dos seus eventos de ativação e desativação, permite determinar informações de percepção importantes tais como os horários e a duração da ativação das aplicações, ou seja, o ritmo de trabalho do operador. Apesar de reduzida e fragmentada, essa informação é relevante no

desenvolvimento globalizado, pois várias equipes apresentam poucas oportunidades para interações síncronas, devido a diferenças de turno de trabalho ou fusos-horários (BEGOLE *et al.*, 2002).

Acrescentando-se a essa informação, a atividade do mouse e o teclado do participante, podemos determinar o volume de atividade do participante no espaço de trabalho (BORGES e PINO, 1999, KANTOR e REDMILES, 2001). Se os principais comandos de edição de uma aplicação encontram-se compartilhados, é possível a visualização (MANGAN *et al.*, 2002b) e, possivelmente, a edição colaborativa tanto síncrona (BEGOLE, 1998 e LI e LI, 2002) quanto assíncrona (MOLLI *et al.*, 2002).

Além disso, foram desenvolvidos e testados mecanismos para o processamento e apresentação de diferentes tipos de informação de percepção associados, principalmente, com a percepção assíncrona. Foram explorados mecanismos assíncronos de coordenação (BORGES e PINO, 1999, KANTOR e REDMILES, 2001), ritmo de trabalho (BEGOLE *et al.*, 2002) e percepção de grupo (KREIJN E KIRSCHNER, 2001). Os protótipos não esgotam as possibilidades de implementação de *collablets*, nem apresentam mecanismos de percepção inéditos. Novos *collablets* ainda podem ser criados sob demanda, com base em novos tipos de eventos e requisitos de colaboração. Contudo, os protótipos demonstram as necessidades de uma família representativa de mecanismos de percepção que podem ser atendidas pelos coletores propostos, dentro da condição de transparência para o programador da aplicação original.

Por fim, foram realizadas implementações de sensores na ferramenta *Rational Rose*, por meio de sua interface de extensão (*Rose Extensibility Interface*). Os protótipos foram integrados também com o ambiente de programação *Eclipse*, por meio da sua plataforma que é baseada em um núcleo central ampliado por módulos complementares (*Eclipse plug-ins*).

As duas últimas seções apresentam comentários sobre o uso da abordagem por terceiros e um resumo dos protótipos desenvolvidos. Nas seções a seguir, são apresentados detalhes de cada um dos seis protótipos desenvolvidos.

## 6.1 *Shared Workspaces*: Compartilhamento de Área de Trabalho

Os *collablets*, desenvolvidos para compartilhamento de área de trabalho, implementam mecanismos síncronos de percepção voltados para a coordenação de visualização colaborativa em espaços de trabalho compartilhado (GUTWIN e GREENBERG, 1999). Três *collablets* (teleapontador, janela em miniatura e bate-papo) foram acoplados ao ambiente de desenvolvimento Odyssey (WERNER *et al.*, 2000), formando a base para a proposta do ambiente colaborativo síncrono do OdysseyShare (WERNER *et al.*, 2002). O ambiente *Odyssey* visa prover uma infra-estrutura de apoio à reutilização baseada em modelos de domínio. A reutilização de software no ambiente *Odyssey* ocorre através dos processos de *Engenharia de Domínio*, cujo objetivo é construir artefatos reutilizáveis voltados para problemas de conhecimento específico, e de *Engenharia de Aplicação*, que tem o objetivo de desenvolver aplicações reutilizando esses artefatos. O *Odyssey* oferece o conjunto de aplicações sobre as quais se deseja aumentar o potencial para a colaboração. Essas aplicações incluem diversas ferramentas específicas para o desenvolvimento de software, tais como editores gráficos para diversos tipos de diagramas com apoio para rastreabilidade, aplicação de padrões e críticas. No ambiente, estão disponíveis uma máquina de processos (MURTA, 2002) e um repositório de componentes (SOUZA *et al.*, 2001) que apóiam outros aspectos de colaboração.

Dois *collablets* oferecem informações de percepção do espaço de trabalho e um terceiro apóia a comunicação entre os participantes. O ambiente resultante oferece apoio para a visualização colaborativa de artefatos e para a comunicação direta entre os participantes (Figura 6.1).

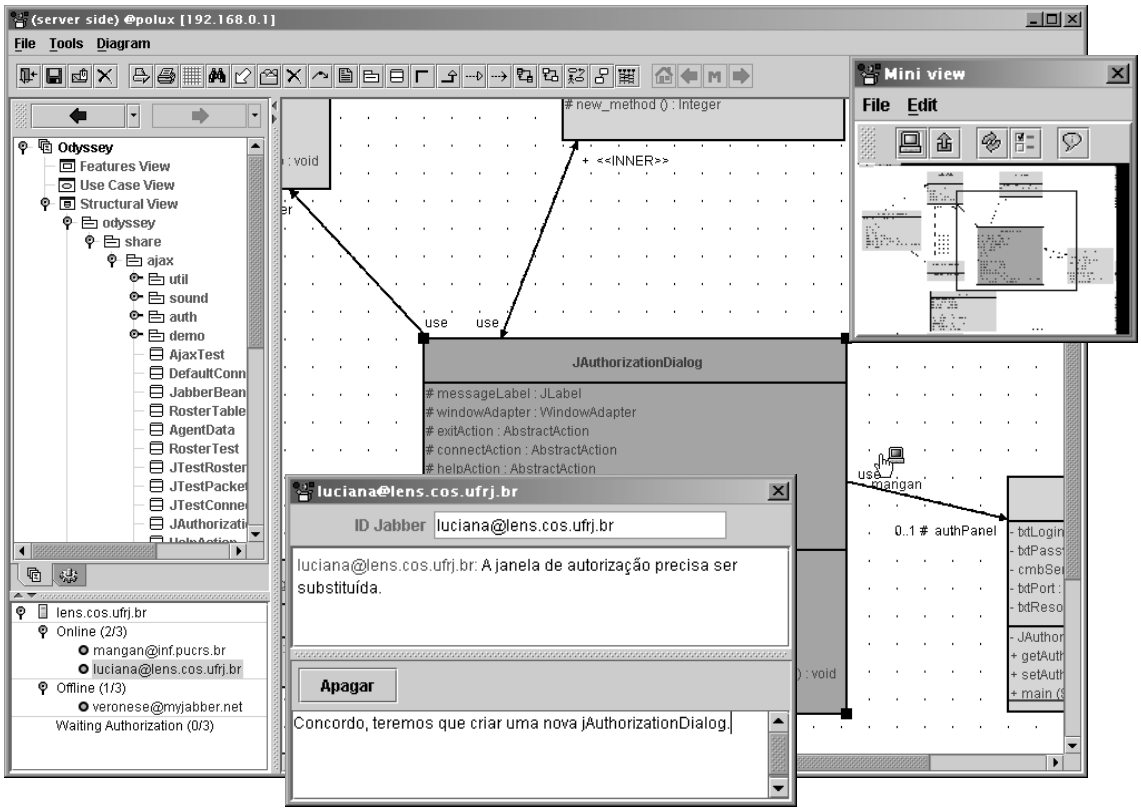



**Figura 6.1 Collablets no OdysseyShare SDE: bate-papo, teleapontador e janela em miniatura (MANGAN *et al.*, 2002b).**

O usuário poderá visualizar sobre a área das janelas de aplicação o indicador do dispositivo apontador remoto e o cursor de edição remoto de um ou mais usuários. Teleapontadores, cursores de edição e outras decorações e sinais sobrepostos são *collablets*, ou seja, componentes de software independentes.

O usuário tem a liberdade de selecionar nenhum, um, dois ou três dos *collablets* disponíveis. Durante a visualização colaborativa, o usuário percebe um atraso característico de aplicações distribuídas devido às condições de latência e banda da rede de computadores. Ainda, um ligeiro atraso no processamento é observado, visto que o processador local estará encarregado de parte do trabalho necessário para manter o registro, a interpretação de eventos e a transmissão de mensagens.

A versão inicial do protótipo exigia alterações mínimas no ambiente *Odyssey* original. Em termos de alterações no código, apenas duas classes (dentre as mais de 600 classes que compõem a aplicação) foram alteradas, com um total de apenas cinco linhas de código incluídas (Figura 6.2). A alteração foi realizada por um programador que não tinha conhecimento prévio do código da aplicação, com base em um processo de

depuração manual da execução da aplicação. Basicamente, as alterações são necessárias para controlar a ativação dos componentes e a integração com os menus da aplicação.

```
1. Workspace w = new Workspace();
2. w.setJFrame(this);
3. w.setJScrollPane(sp);
4. w.setup();
5. w.setActive(true);
```

**Figura 6.2 Alterações no OdysseyShare (Classe *WinModelagem*, linguagem Java).**

Os *collablets* são resultado de um esforço em tornar mecanismos de apoio à colaboração síncrona em componentes de software (MANGAN *et al.*, 2002).

O editor de textos *Stylepad* é uma outra aplicação que foi alterada com o uso da abordagem apresentada. A alteração de uma segunda aplicação tem como objetivo demonstrar a independência dos *collablets* em relação ao *Odyssey*. O *Stylepad* foi utilizado, também, como protótipo da tese de Begole (1998), com a proposta da Colaboração Transparente Flexível (CTF). Os resultados obtidos na visualização colaborativa são semelhantes na CTF e na ACT. A CTF permite realizar a edição colaborativa síncrona, que foi prevista, mas não implementada na ACT.

Entretanto, a CTF de Begole não pode ser combinada com outra proposta. A CTF substitui objetos da biblioteca Swing por versões colaborativas desses objetos. A ACT utiliza a notificação para acrescentar comportamento aos objetos, sem substituí-los. A vantagem da ACT é que vários mecanismos podem coexistir sobrepostos, sem que a inclusão de um novo *collablet* remova um *collablet* existente.

Nas duas aplicações, a coleta é realizada por meio do sistema de janelas. Entretanto, além do desenvolvimento do coletor, é necessário escolher um meio para transferir a informação coletada de uma estação para outra. O sistema de eventos e notificação foi implementado com o uso da tecnologia *Jabber* (ADAMS, 2002). A função principal desse mecanismo é a realização de comunicação através de mensagens de texto. A possibilidade de extensão desse mecanismo permite a criação de mensagens contendo semântica bem definida, representada por marcações na mensagem.

O formato de representação é adequado para a identificação dos elementos semânticos da mensagem, mas pode não ser adequado ao processamento dos eventos da edição. Principalmente, porque a representação é realizada na linguagem de marcação extensível (Ex*tensible Markup Language* - XML) que ocupa um espaço e um tempo adicional para a criação e processamento das mensagens, se comparada com

alternativas, tais como, objetos distribuídos e comunicação via soquetes (*sockets*). A principal limitação atual desse mecanismo de troca de mensagens é em relação ao processamento eficiente de um alto volume de mensagens. Essa limitação está sendo resolvida com os avanços no processamento de XML. Entretanto, o uso de XML não é adequado para o processamento de informações em formato binário como imagens, som e vídeo. Nesses casos, a comunicação foi implementada através de um canal independente.

A principal vantagem no uso de XML é a facilidade de conversão dessas informações para outros formatos que permitam acesso a sistemas de inferência, mineração de dados, processamento estatístico e visualização de grandes volumes de dados.

No ambiente *OdysseyShare*, a comunicação é apoiada basicamente por mensagens instantâneas semi-estruturadas. Considera-se interessante o uso associado de áudio e vídeo-conferência. Entretanto, as mensagens de texto oferecem maiores facilidades à busca de informação.

No contexto de troca de mensagens, cada mensagem contém a indicação do remetente, destinatários, conteúdo, ordenação na seqüência de mensagens. O servidor *Jabber* realiza o roteamento de mensagens com base nas informações contidas na mensagem. Um cliente *Jabber* apresenta as mensagens recebidas e interage com o usuário por meio de uma interface de bate-papo convencional.

A representação em XML permite que a mensagem transporte marcações semânticas adicionais. Por exemplo, uma mensagem pode ser marcada como uma "pergunta" que aguarda uma mensagem específica marcada como "resposta". Essas marcações facilitam a construção de mecanismos sofisticados de troca de mensagens que operam sobre uma mesma rede de comunicação. Para interpretar as mensagens corretamente, os clientes em cada ponta devem extrair as marcações do corpo da mensagem. As marcações que não são processadas pelos clientes são ignoradas, sem prejuízo para a comunicação. Através dessas marcações, é possível criar grupos de discussão e aninhamento de assuntos.

A tecnologia *Jabber* é compatível com diversos sistemas de mensagens populares como o *MSN Messenger*, *Yahoo!Messenger* e *ICQ*. A compatibilidade permite a troca de mensagens e a operação por um participante que tenha um cadastro

válido em qualquer desses sistemas. Essa característica contorna um dos principais problemas de um sistema de mensagens que é o estabelecimento de uma massa crítica de usuários. Além disso, os usuários ficam livres para utilizar o sistema de mensagens de sua preferência.

A áudio e video-conferência são realizadas com o uso de componentes de software compatíveis com o protocolo de vídeo-conferência H.323 (ITU, *International Telecommunication Union*, 2003). Os clientes utilizados em testes foram o *Windows NetMeeting*, *CuSeeMe* e *Gnome Linux GnomeMeeting* acionados através de uma marcação específica enviada através do protocolo *Jabber* e interpretada pelo cliente *Jabber*. Os demais clientes (*ICQ*, *CUSeeMe*) devem iniciar a aplicação de vídeo conferência manualmente. As desvantagens da vídeo-conferência são: (a) o consumo de recursos (capacidade de processamento, banda de rede e espaço para armazenamento) e (b) a distração introduzida pela riqueza das mídias de áudio e vídeo.

Durante a operação do protótipo, percebeu-se a utilidade do uso de marcações especiais para sinalizar o estado do participante durante a sessão, similar aos ícones que demonstram emoções (*smiles*, *emoticons*) em programas de bate-papo. As marcações indicam dúvidas, permissão para falar, controle do ambiente, ação em andamento e confirmação de atenção concentrada e outras sinalizações que auxiliam a conduzir a participação durante a sessão. Notou-se, também, a possibilidade de uso de marcação para indicar perguntas, respostas e outros modificadores que facilitem a organização das mensagens.

As mensagens são ordenadas e ficam registradas como parte da memória do grupo. As discussões ficam associadas aos artefatos e outros elementos do contexto colaborativo que envolve as ações de cada usuário. Pode-se utilizar um dos elementos desse contexto para recuperar essas mensagens em um momento posterior. De forma similar, mensagens podem ser colocadas no ambiente como se fossem anotações.

O canal de comunicação com base em mensagem pode ser utilizado como forma de interação entre o apoio à colaboração e ao participante. Espera-se que a inserção de mensagens no contexto da conversa possa ser útil para adicionar informação relevante ao contexto. Considera-se, ainda, possível que os participantes possam utilizar um vocabulário restrito para solicitar informações ao sistema de apoio a colaboração. Por exemplo, a frase “Quem criou o artefato Contrato?” poderia ser usada para

recuperar um dos atributos do artefato durante a colaboração. O uso dessa forma de interação pode ter aceitação variável entre os usuários. Esperamos que a prática do ambiente auxilie a determinar fatores que indiquem a preferência individual para o recebimento de notificação. Apesar da facilidade da implementação e uso de serviços de comunicação com base em mensagens, a sua aplicação deve considerar os problemas que podem ser ocasionados pelas dificuldades de interpretação dos participantes (PIMENTEL *et al.*, 2003).

Informações de percepção básicas como a presença e a disponibilidade de um participante são oferecidas como parte do mecanismo de comunicação. O monitoramento da interface de usuário das aplicações gera informações de percepção mais detalhadas sobre o volume de atividade e o foco do interesse dos demais participantes. Essas informações combinadas permitem determinar quem está presente no ambiente colaborativo e qual a visão que cada um possui sobre o ambiente. O *OdysseyShare* captura e apresenta informações sobre o espaço de trabalho dos participantes usando os eventos da Tabela 6-1.

**Tabela 6-1 Eventos básicos da arquitetura de colaboração transparente (modelo de percepção de espaço de trabalho).**

| ***EVENTO*** | ***Parâmetros*** |
|---|---|
| MOVE_VIEWPORT | Point p1, Point p2 |
| SELECT_VIEW | ID |
| MOVE_POINTER | Point p |
| RESIZE_VIEWPORT | Dimension d |
| RESIZE_VIEW | Dimension d |

Os eventos são coletados no sistema de janelas do ambiente de execução da plataforma Java, processados e introduzidos de volta ao ambiente através de mecanismos de percepção específicos como o teleapontador e a janela de radar.

Aplicações de compartilhamento de tela, como o *Microsoft NetMeeting*, são normalmente utilizadas neste tipo de estudo para comparar o desempenho do apoio proposto com um sistema genérico. No nosso caso, problemas técnicos impossibilitaram essa comparação, pois a aplicação implementada em Java não pode ter sua tela

compartilhada, ao menos na versão do *NetMeeting* disponível na época. A justificativa era de que a plataforma Java possui um sistema de janelas independente do usado pelo ambiente operacional Microsoft Windows. Além disso, o protótipo consegue operar em plataformas heterogêneas (ex. *Microsoft Windows* e *Red Hat Linux*). Com isso, a observação se restringe à caracterização do uso dos componentes de compartilhamento de área de trabalho.

As alterações realizadas no ambiente de execução consideram apenas a captura de eventos de interface (Figura 6.3). Os eventos são coletados a partir do sistema de janelas utilizado pela aplicação, o que garante a transparência por parte do programador da aplicação.

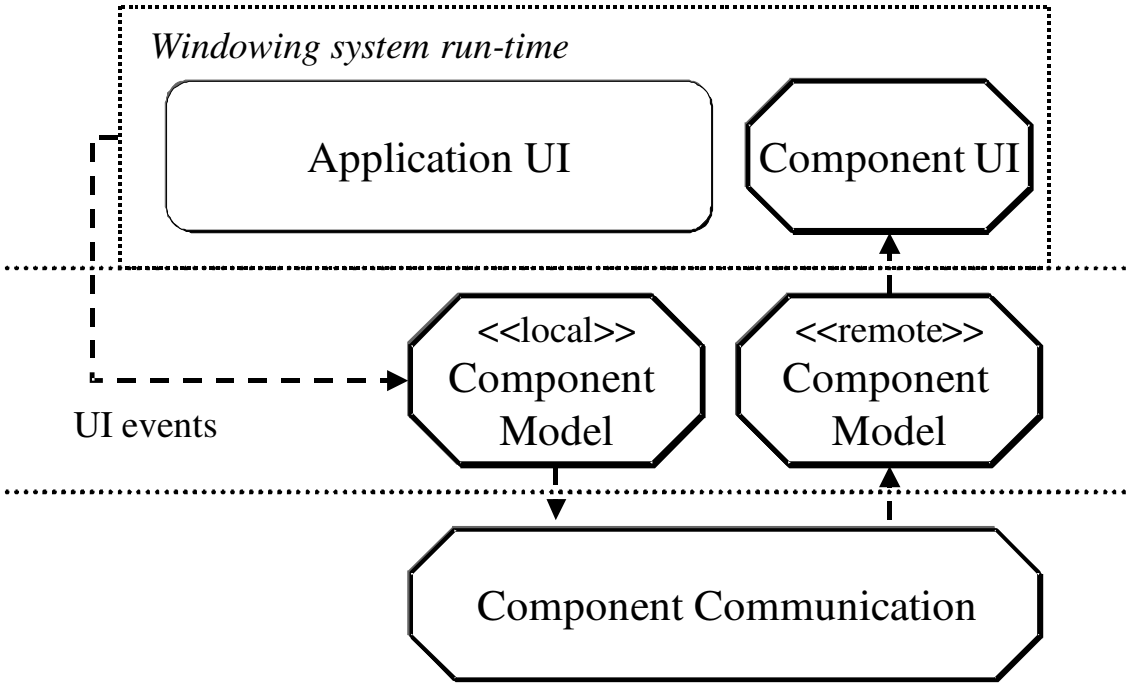



**Figura 6.3 Versão preliminar da ACT (MANGAN, WERNER, BORGES, 2002).**

Com base nessa versão preliminar, foram obtidos protótipos operacionais que oferecem os serviços de comunicação, visualização colaborativa de diagramas e percepção em espaços compartilhados.

## *6.2 Cine Odyssey: Captura e Reprodução de Históricos de Eventos*

O *Cine Odyssey* é o protótipo responsável pela captura e reprodução de

históricos de eventos. Este protótipo foi a primeira tentativa de implementação de edição colaborativa no *Odyssey*. Durante a construção deste protótipo, foi tomada a decisão de não oferecer apoio à edição colaborativa síncrona pelas seguintes razões:

(a) dificuldades técnicas na inserção de eventos remotos na fila de eventos da ferramenta local. Apesar de ter sido possível atualizar o modelo de representação interna de quase todas as aplicações de teste, foram observados problemas na atualização da interface de usuário.

(b) dificuldades na integração com os mecanismos de gerência de configuração. A inserção de uma alteração remota na aplicação local causa a perda da informação de autoria e faz com que a alteração remota seja identificada como um conflito de edição entre os desenvolvedores. Para solucionar esse problema, seria necessário alterar o mecanismo de gerência de configuração para que este levasse em consideração as alterações que foram movidas entre uma estação e outra. Uma alternativa seria determinar que apenas um dos desenvolvedores ficasse encarregado de inserir o resultado da seção de volta no repositório. Essa alternativa evita a existência de cópias do resultado da ação e, portanto, contorna a geração de conflitos. Contudo, a informação de autoria continuaria sendo perdida e os demais usuários teriam que abandonar os dados da seção colaborativa e atualizar suas cópias por meio do repositório.

(c) dificuldades na integração com o modo de trabalho do desenvolvedor. Duas características dos desenvolvedores limitam a utilidade da edição colaborativa síncrona. Primeiro, os desenvolvedores trabalham em diferentes modos de colaboração. Dois desenvolvedores podem estar trabalhando no mesmo intervalo de tempo, mas um deles pode não estar disposto a interromper sua atividade individual para iniciar uma sessão colaborativa. Segundo, no desenvolvimento globalizado, os desenvolvedores normalmente trabalham em horários diferentes, com pouca possibilidade de intersecção. Além disso, os desenvolvedores tendem a sofrer maior influência dos colegas que compartilham o mesmo local de trabalho, que monopolizam as interações síncronas pela facilidade de interação e pela necessidade de socialização.

A edição assíncrona pode ser adotada para contornar esses três problemas. Os desenvolvedores podem determinar o momento quando as ações remotas serão integradas em seu ambiente local. O mecanismo mais comum para controlar o fluxo de ações é a idéia de uma bandeja ou caixa de entrada. Os eventos remotos ficam

armazenados nessa caixa de entrada e o usuário decide quando e quais deles devem ser inseridos na área local.

Apesar da edição colaborativa síncrona ter sido abandonada, o *Cine Odyssey* serviu para definir novos eventos descritos na Tabela 6-2. Esses eventos foram utilizados na construção dos demais protótipos. O histórico e a interpretação de eventos foram propostos a partir da experiência com o *CineOdyssey*. Este protótipo destacou o potencial para a gravação e reprodução de sessões colaborativas ou de registros individuais da utilização das ferramentas e para a análise desses eventos.

**Tabela 6-2 Eventos básicos da arquitetura de colaboração transparente (modelo de percepção abstrato)**

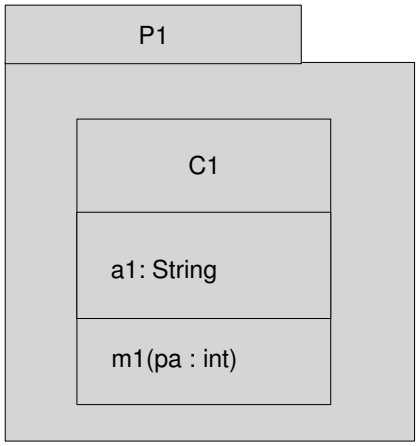
| ***EVENTO*** | ***Parâmetros*** |
|---|---|
| CREATE | ID |
| RETRIEVE | ID |
| UPDATE | ID |
| DELETE | ID |
| VIEW | ID |
| SELECT | ID |
| CLASSIFY | ID, IDCLASSE |
| COMPOSE | ID_CONTENTOR, ID_CONTEUDO |
| ADD_PROPERTY | ID, NAME, VALUE |

Com esses nove eventos é possível representar as atividades realizadas sobre um diagramador. Com esses nove eventos é possível representar boa parte da atividade realizada sobre um diagramador. Com eles é possível alimentar modelos de percepção que oferecem informações úteis para mecanismos de apoio à colaboração assíncrona, descoberta de perfil de usuário e histórico das experiências dos usuários.

Por exemplo, imaginemos que o usuário Cláudia faça a seguinte seqüência de operações em um editor: abra um diagrama UML contendo o desenho de duas classes 'a' e 'b', em seguida, remova a classe 'b' e crie uma nova classe 'c'. A seqüência de eventos gerada seria, aproximadamente: RETRIEVE B; RETRIEVE A; SELECT B; DELETE B; CREATE C; CLASSIFY C, “classe UML”; ADD_PROPERTY C, “name”, “c”. Essa seqüência representa fatos observados na aplicação. As letras A, B e C são

identificadores dos objetos. O nome do objeto, como é apresentado na aplicação, é representado por uma propriedade. A Tabela 6-3 apresenta um exemplo um pouco mais complexo, com os eventos necessários para representar a edição de um diagrama de classes simples.

**Tabela 6-3 Representação de um diagrama simples utilizando uma seqüência de eventos.**

| *Modelo* | *Seqüência* |
|---|---|
|  | CREATE 1<br>ADD_PROPERTY 1, “name”, “P1”<br>CLASSIFY 1, “class”<br><br>CREATE 2<br>ADD_PROPERTY 2, “name”, “C1”<br>CLASSIFY 2, “package”<br>COMPOSE 2,1<br><br>CREATE 3<br>ADD_PROPERTY 3, “name”, “a1”<br>ADD_PROPERTY 3, “type”, “String”<br>CLASSIFY 3, “attribute”<br>COMPOSE 3,2<br><br>CREATE 4<br>ADD_PROPERTY 4, “name”, “m1”<br>CLASSIFY 4, “method”<br>COMPOSE 4,2<br><br>CREATE 5<br>ADD_PROPERTY 5, “name”, “pa”<br>ADD_PROPERTY 5, “type”, “int”<br>CLASSIFY 5, “method_parameter”<br>COMPOSE 5,4 |

A principal dificuldade na programação do *CineOdyssey* foi a localização dos pontos do código da aplicação *Odyssey* onde os eventos devem ser coletados. O *Odyssey* conta com uma interface de extensão que comunica esses eventos como parte do mecanismo de carga dinâmica (MURTA *et al.*, 2004).

## 6.3 Ariane: Apoio à Percepção de Produto

O *Ariane* (VIEIRA *et al.*, 2003, VIEIRA, 2003) é o segundo protótipo que explora um modelo de percepção com base na estrutura da informação, em contraponto

ao protótipo *Shared Workspaces* que explora um modelo de apresentação. Diferente do *CineOdyssey* que utiliza uma técnica dependente da aplicação e da capacidade de modificação do código de uma aplicação, o *Ariane* explora as facilidades de notificação de uma técnica de persistência orientada a objetos.

A especificação *Java Data Objects* (RUSSEL, 2004) define um mecanismo de notificação que comunica eventos a cada alteração no modelo de uma aplicação persistente. A notificação é possível, pois a aplicação é modificada de forma automática por um pós-compilador para incluir o código necessário para interceptar todos os comandos que alteram ou consultam valores de atributos de objetos Java persistentes.

O propósito desse sistema de notificação é permitir que o mecanismo de persistência sincronize o modelo de dados em memória principal com o modelo de dados em memória secundária. Entretanto, percebeu-se que existe um potencial para a coleta de informações de percepção. A notificação permite o conhecimento do estado e das transições de estado do modelo da aplicação sem a necessidade de um programador que tenha conhecimento sobre a codificação da aplicação.

Além disso, existem duas vantagens adicionais em se colocar a coleta diretamente na aplicação e não diretamente no banco de dados. A primeira, é que com o monitoramento da aplicação é possível romper o isolamento entre as estações. Os dados que são coletados ainda não são de conhecimento dos demais usuários. Dessa forma, a informação de percepção pode ser utilizada para alertar sobre possíveis conflitos antes que estes ocorram no repositório. A segunda vantagem, é que a coleta na aplicação considera o modelo de negócios da aplicação e não o modelo de persistência que está armazenado no banco de dados. Por exemplo, é comum que o modelo de negócios orientado a objetos seja mapeado para um modelo relacional. As informações sobre o modelo de negócios são, geralmente, mais próximas da perspectiva do usuário da aplicação, o que torna muito mais simples comunicar as mudanças coletadas (Figura 6.4).

Widget EventMonitor

| When | Who | Where | What | How |
|---|---|---|---|---|
| Tue Oct 21 15:09:... | vaninha | modelo.componente.ModeloComponente@1272602452-54 | DATA_CREATED | SensorJDO@za.co.... |
| Tue Oct 21 15:09:... | vaninha | modelo.componente.logico.ModeloLogicoComponente@327457969-55 | DATA_CREATED | SensorJDO@za.co.... |
| Tue Oct 21 15:09:... | vaninha | modelo.componente.interfaces.ModeloInterfacesComponente@11556... | DATA_CREATED | SensorJDO@za.co.... |
| Tue Oct 21 15:09:... | vaninha | modelo.componente.logico.ModeloLogicoComponente@327457969-57 | DATA_CREATED | SensorJDO@za.co.... |
| Tue Oct 21 15:09:... | vaninha | modelo.componente.interfaces.ModeloInterfacesComponente@11556... | DATA_CREATED | SensorJDO@za.co.... |
| Tue Oct 21 15:09:... | vaninha | modelo.Dominio@1020539875-41 | DATA_RETRIEVED | SensorJDO@za.co.... |
| Tue Oct 21 15:09:... | vaninha | jdbc:microsoft:sqlserver://localhost | TRANSACTION_ROLLBAC... | SensorJDO@za.co.... |
| Tue Oct 21 15:09:... | vaninha | jdbc:microsoft:sqlserver://localhost | SESSION_ENDED | SensorJDO@za.co.... |
| Tue Oct 21 17:00:... | mangan | jdbc:microsoft:sqlserver://localhost | SESSION_STARTED | SensorJDO@za.co.... |
| Tue Oct 21 17:00:... | mangan | jdbc:microsoft:sqlserver://localhost | TRANSACTION_STARTED | SensorJDO@za.co.... |
| Tue Oct 21 17:00:... | mangan | modelo.ListaDominios@604681121-1 | DATA_RETRIEVED | SensorJDO@za.co.... |
| Tue Oct 21 17:00:... | mangan | modelo.Dominio@1020539875-1 | DATA_RETRIEVED | SensorJDO@za.co.... |
| Tue Oct 21 17:00:... | mangan | modelo.Dominio@1020539875-2 | DATA_RETRIEVED | SensorJDO@za.co.... |
| Tue Oct 21 17:00:... | mangan | modelo.ListaDominios@604681121-21 | DATA_RETRIEVED | SensorJDO@za.co.... |

**Figura 6.4 Exemplo de dados coletados com o protótipo Ariane.**

O resultado da coleta é uma grande massa de dados com cada transição realizada pelos desenvolvedores sobre suas cópias locais do modelo de software editado no *Odyssey*. Um elemento do modelo recebe o mesmo identificador em todas as cópias, por necessidade do mecanismo de persistência. Dessa forma, é possível, por exemplo, identificar quando um mesmo objeto é alterado por mais de um desenvolvedor.

A análise da massa de dados pode ser realizada com o uso de técnicas de mineração de dados. Por exemplo, o uso do esquema estrela (Figura 6.5) pode ser aplicado para decompor a informação coletada em diferentes dimensões.

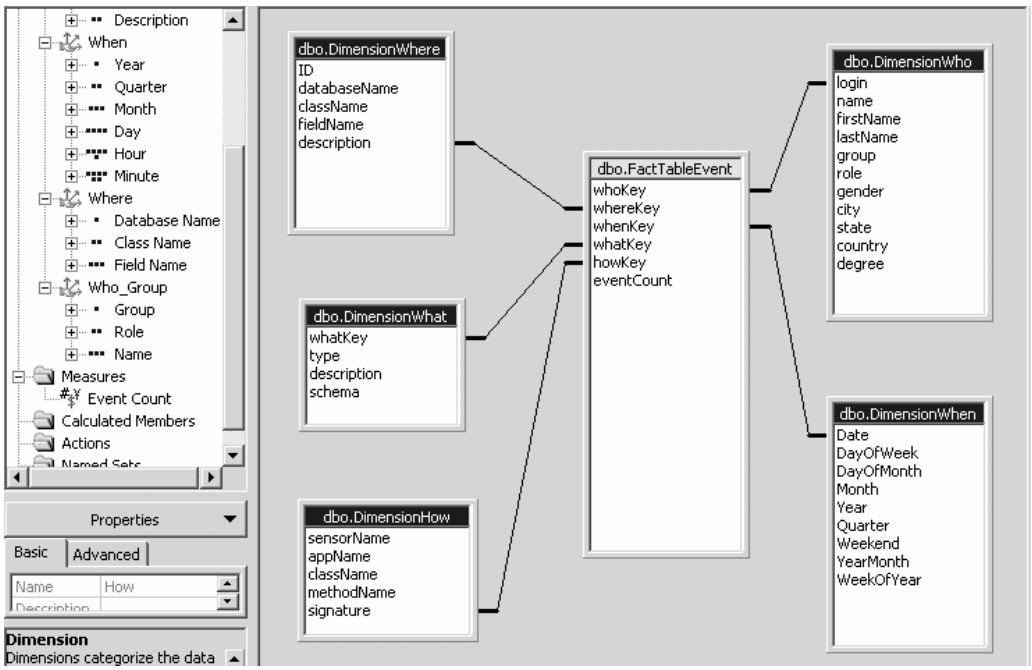



**Figura 6.5 Análise dos dados com o uso do esquema estrela.**

A partir do esquema estrela, o usuário pode montar consultas com o uso de uma ferramenta de mineração de dados convencional (Figura 6.6).

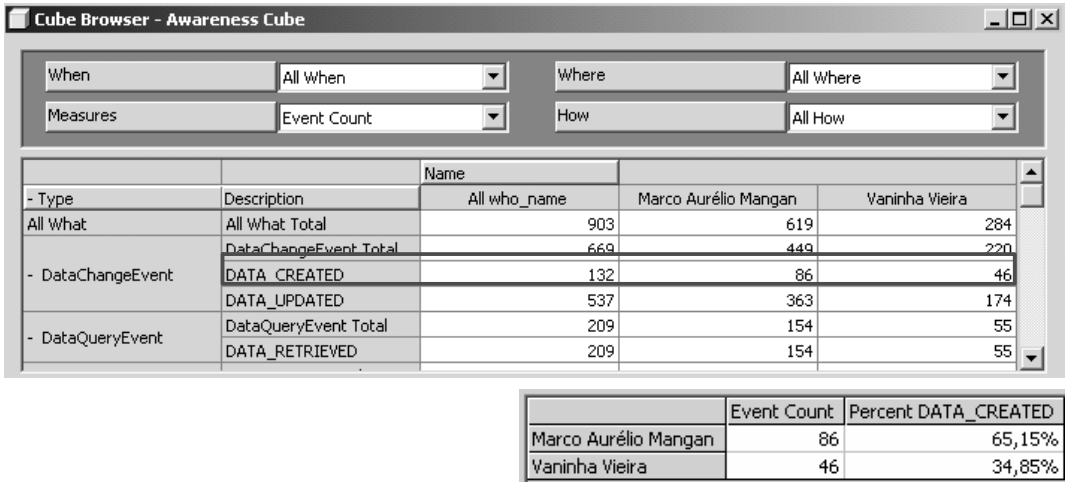



**Figura 6.6 Exemplo de consulta sintética realizada sobre os dados coletados no Ariane.**

O principal obstáculo técnico para o uso do mecanismo de persistência como fonte de informação de percepção é que a especificação informa as interfaces exigidas, mas não interfere na sua implementação. A especificação é implementada por diversos produtos (*JDOGenie*, *Castor*, *Versant*). Dessa forma, seria possível alterar o código de um dos produtos, mas a alteração não estaria presente em novas versões do produto nem em outros produtos.

A solução encontrada foi o uso de técnicas da programação orientada a aspectos (KICZALES *et al.*, 2001) para associar a assinatura dos métodos da especificação JDO com a ativação de coletores de dados, em uma implementação JDO específica (Figura 6.7). Com isso, toda a implementação compatível com a especificação JDO pode ser monitorada, sem a necessidade de alteração manual do código. A alteração é realizada por um pós-compilador que adiciona o código da notificação com base nas chamadas de métodos específicos.

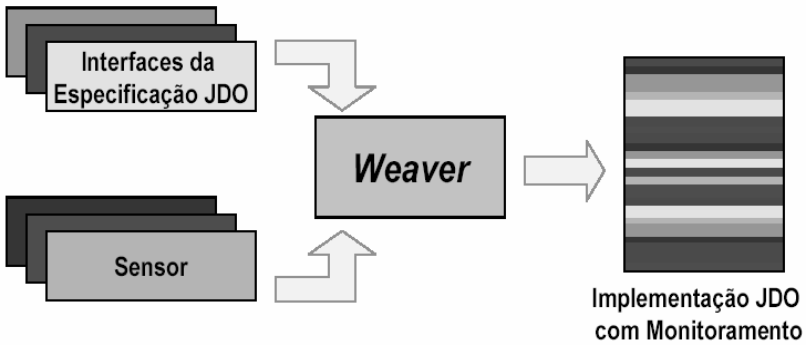



**Figura 6.7 Uso de aspectos para integrar a notificação de eventos JDO com o coletor.**

A abordagem exige conhecimento técnico para a colocação dos aspectos, mas não obriga o conhecimento de uma implementação JDO em particular. Dada uma implementação, a execução do pós-compilador providencia a instrumentação necessária para a operação do coletor.

## 6.4 MAIS: Apoio à Percepção de Produto

A infra-estrutura para percepção Multi-síncrona (MAIS – *Multi-synchronous Awareness Infrastructure*) (LOPES *et al.*, 2004a, LOPES *et al.*, 2004b, LOPES, 2005) destaca determinadas informações de percepção de produto com base na relevância relativa para cada usuário a partir do histórico individual.

O apoio à interpretação de eventos permite a exploração de serviços de perfil de usuário e de cálculo da distância entre participantes e artefatos. Com a memória implementada, torna-se possível a programação de extensões para: (a) D**escoberta de colaboradores**: dentre as diversas barreiras inseridas pela separação no tempo e no espaço está a de encontrar um parceiro de trabalho que possa contribuir com opiniões e soluções para problemas em um contexto específico (MOCKUS e HERBSLEB, 2002). Em geral, indivíduos que pertencem a comunidades tendem a se auxiliar mutuamente, obtendo melhores resultados do que os obtidos por esforços isolados. O problema está em como formar essas comunidades de desenvolvedores que desejam e são capazes de se auxiliar quando as oportunidades para colaboração não estão presentes. (b) **Revisão de Experiências:** permitiria a localização e compartilhamento de seqüências de operação das aplicações. Com isso podemos incentivar a revisão e também a reutilização e a disseminação de experiências. (c) **Cálculo de Distância Semântica entre Artefatos e Colaboradores**: os eventos semânticos representam fatos sobre as atividades dos participantes. O número de eventos tende a crescer rapidamente, a medida que as contribuições dos participantes são realizadas. A fim de organizar a extração de informação útil para a colaboração, podemos determinar uma distância semântica entre dois participantes. Essa informação pode ser utilizada para localizar colaboradores que estão trabalhando em um mesmo "local", ou seja, sobre um mesmo conjunto de objetos. A distância também pode considerar outros critérios, como o uso das mesmas aplicações ou intersecções entre horários de trabalho de diferentes

participantes.

Para obter uma avaliação da distância entre dois participantes, é necessário atribuir valores para as combinações de eventos semânticos propostos para o *OdysseyShare* (Tabela 6-2).

Por exemplo, digamos que o participante A tenha criado uma classe 'a' e o participante B tenha visualizado a classe 'a' através de alguma aplicação. Existe entre os participantes uma relação de proximidade no 'espaço de software'. Ou seja, é provável que ambos demonstrem interesse por abstrações semelhantes. Se o participante B realizar uma alteração na classe 'a', podemos argumentar que a proximidade dos dois é maior. Nessa interpretação, a distância d(A:create(a), B:view(a)) é maior do que d(A:create(a), B: update(a)). Com isso, advogamos que podemos calcular a distância entre dois participantes levando em conta o significado das operações.

Podemos explorar o registro de ações dos participantes para encontrar outras relações. Por exemplo, um participante pode solicitar uma notificação para cada alteração em artefatos de seu interesse. Relacionamentos entre as entidades podem ser usados para calcular a propagação de alterações sobre outros artefatos.

A distância pode ser utilizada para calcular a probabilidade de (a) sobreposição ou continuidade entre as ações dos participantes ou (b) contextos de trabalho semelhantes, mesma tarefa, mesmos problemas. O 'espaço de software' pode ser definido de diversas formas, sem se limitar a estrutura do objeto original. Uma única classe pode ser decomposta em seus elementos, formando um espaço de software onde as operações dos participantes encontram-se dispersas. Por exemplo, a Figura 6.8 demonstra a visualização da classe *BookManager* e seus métodos e atributos no visualizador da aplicação CAST. Métodos e atributos são conectados à classe através da relação *bind* (B), métodos se relacionam com atributos através das relações *atribute read* e *atribute write* (Ar e Aw). Um modelo de percepção sobre esse visualizador é mais detalhado do que sobre um diagrama UML.

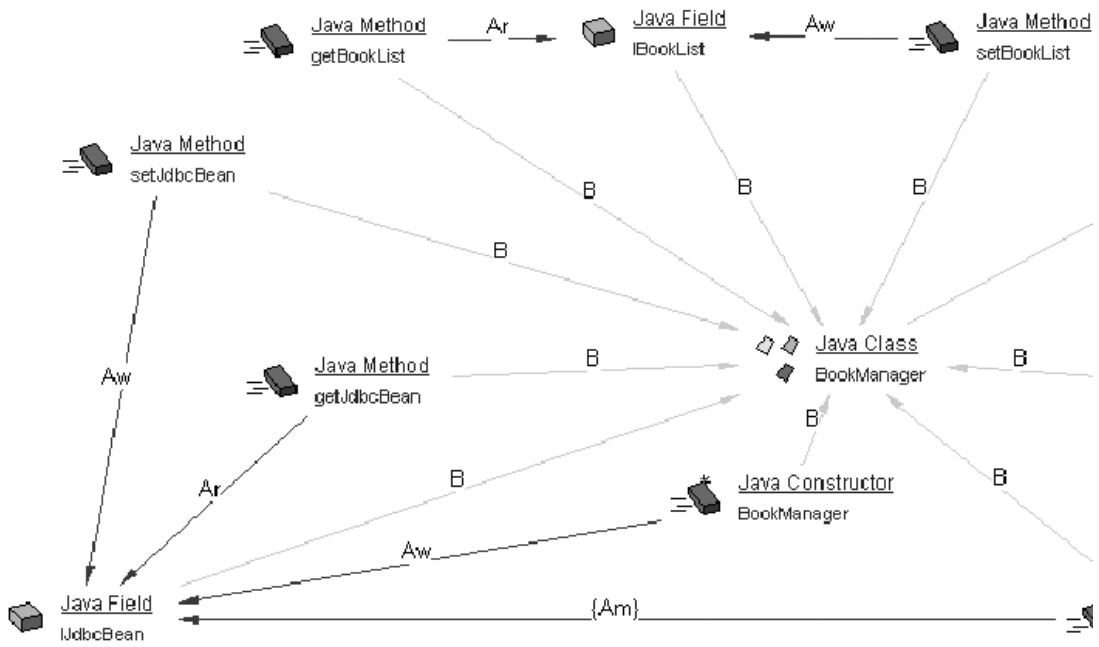



**Figura 6.8 Distância semântica e dependência multidimensional entre artefatos na ferramenta CAST**

Fica a critério do programador da extensão colaborativa a coleta de eventos e o estabelecimento de relações que julgue pertinente. No caso do MAIS, foram explorados elementos da UML (método, atributo), apesar do visualizador disponível não destacar esses elementos na interface com o usuário. Por exemplo, os elementos principais do visualizador de diagrama de classes da UML no *Odyssey* são classes e pacotes. No MAIS, o usuário pode optar por elementos contidos nas classes (método, atributo) para obter maiores detalhes sobre a mudança (Figura 6.9).

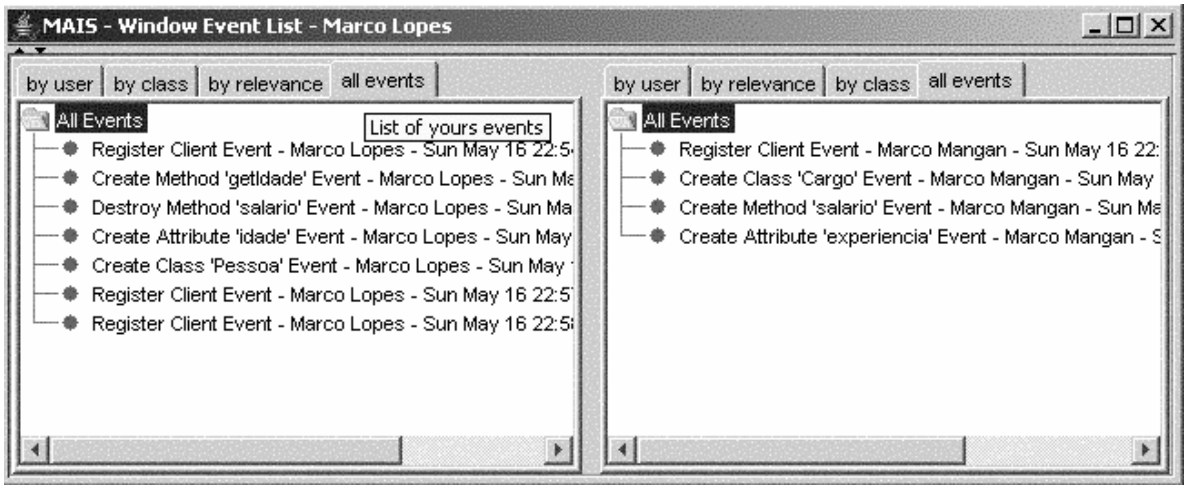



**Figura 6.9 Interface de usuário do MAIS.**

A interface do MAIS organiza os eventos em duas listas: operações locais e operações remotas. As operações locais correspondem ao histórico de mudanças realizadas pelo usuário corrente. As operações remotas combinam as operações locais realizadas pelos demais usuários.

Dois usuários apresentam um ponto de interesse comum quando existe uma coincidência de elementos em suas listas de operações. Por exemplo, se dois usuários criam objetos em um mesmo diagrama. O MAIS usa o conceito de distância semântica para calcular o índice de coincidência. Por exemplo, dois usuários que alteram atributos da mesma classe realizaram operações sobre elementos diferentes (atributo), mas esses elementos estão próximos, pois pertencem a uma mesma classe.

O *Odyssey* conta com uma interface de extensão que comunica os eventos de interesse ao MAIS, como parte do mecanismo de carga dinâmica (MURTA *et al.*, 2004). A interface de extensão é utilizada por todos os outros módulos opcionais do *Odyssey*.

## 6.5 GAW: Apoio à Percepção de Grupo

O componente visual para percepção de grupo (GAW – *Group Awareness Widget*) (MANGAN *et al.*, 2004, SILVA *et al.*, 2004, SILVA, 2005) organiza informações de percepção de grupo e de produto em uma linha de tempo (Figura 6.10).

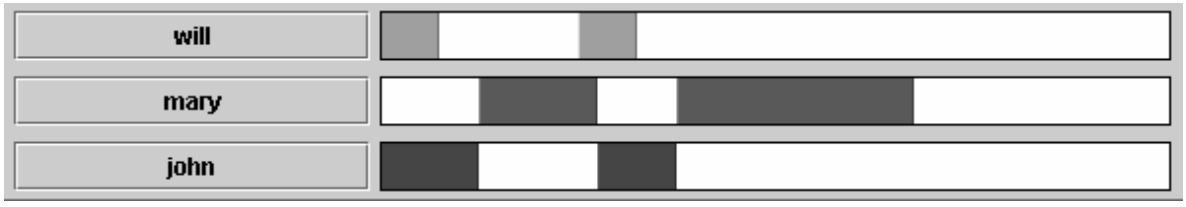



**Figura 6.10 Três desenvolvedores e seus ritmos de trabalho em um GAW.**

São duas as motivações para o uso do GAW: (a) realizar uma implementação de protótipo que operasse no *Odyssey* e também em outra aplicação de amplo uso na modelagem ou programação e (b) explorar eventos coletados pelo MAIS.

O objetivo do GAW foi atingido por meio das interfaces de extensão do Eclipse SDE (Figura 6.11) e do *Odyssey* (Figura 6.12). A integração com o Eclipse considera um modelo de percepção baseado nos conceitos de pasta, documento e linha. Esse modelo é coerente com a visualização percebida pelo usuário durante uso da

ferramenta. A informação é coletada por uma modificação no ambiente e pela criação de um *plug-in* para visualização das informações.

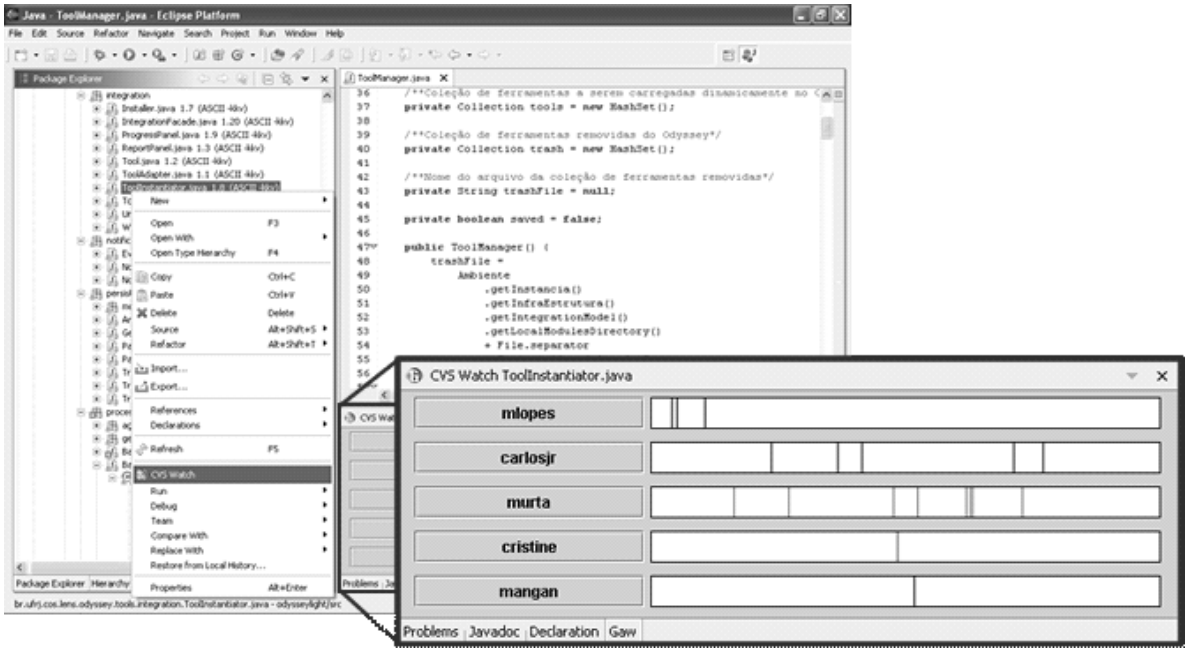



**Figura 6.11 O *CVS Watch* integrado com o ambiente Eclipse.**

A integração com o Eclipse exige um esforço de adaptação da interface de usuário do protótipo ao ambiente e também um novo sensor, adequado ao repositório CVS.

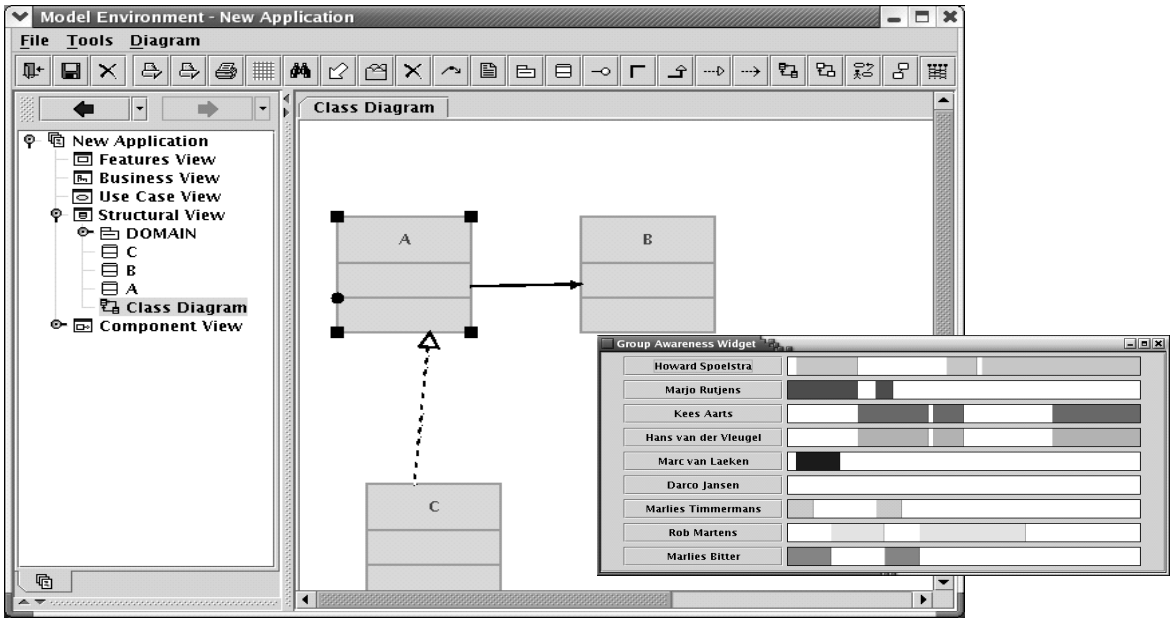



**Figura 6.12 O *Group Workrhytm* integrado com o ambiente *Odyssey*.**

No *Odyssey*, a integração considera um modelo de percepção com base em

conceitos de modelo, elemento e propriedades. Novamente, esses conceitos são significativos para o usuário da aplicação integrada. A informação é coletada por meio do mecanismo de extensão e carga dinâmica de módulos disponível no ambiente. O coletor utilizado pelo GAW é o mesmo coletor usado no MAIS. O que ilustra a possibilidade de reutilização de componentes de um mecanismo em outro mecanismo similar. Ainda no *Odyssey*, a informação apresentada permite determinar sessões de trabalho, pois cada elemento alterado nos modelos gera um evento.

O GAW é um protótipo que utiliza tanto o espaço de tuplas, quanto um repositório compatível com CVS. Neste caso, não apenas o ambiente integrado é diferente, mas também o sensor e o local de armazenamento das informações. Essa característica multiplica a probabilidade de reutilização, pois permite atender a um maior número de situações.

## 6.6 Coletor Semântico para o Rational Rose

Os cinco protótipos descritos até o momento foram desenvolvidos na plataforma Java, sendo quatro deles integrados com a aplicação *Odyssey*. A proposta do sexto protótipo é verificar a possibilidade de coleta de informação, usando técnicas similares, em outra plataforma de desenvolvimento.

O *Rational Rose Modeler* (atualmente, *IBM Rose Modeler*) foi escolhido por ser uma aplicação bem conhecida e amplamente utilizada. O *Modeler* é uma aplicação nativa da plataforma Microsoft Windows, desenvolvida na linguagem C. O *Rose Modeler* foi portado também para a plataforma *Linux*, dado o amplo apoio oferecido da linguagem nessa plataforma. Diferente da linguagem Java, a linguagem C não possui mecanismos de execução ou de persistência tão flexíveis. Os mecanismos mais prováveis para a integração com o *Rose Modeler* são o sistema de janelas da plataforma *Windows* e o mecanismo de extensão da própria ferramenta. O sistema de janelas é muito similar ao da plataforma Java, o que não causa surpresa, pois ambos os sistemas de janelas são muitos similares.

O coletor foi implementado na linguagem C# e opera com o mesmo conjunto de eventos utilizado pelo coletor dos protótipos MAIS, GAW e Ariane. Neste caso, temos duas realizações de um mesmo componente que apresentam a mesma interface de

programação, mas encontram-se instalados em plataformas diferentes.

No exemplo (Figura 6.13), o coletor está gerando os eventos que correspondem à criação de uma classe.

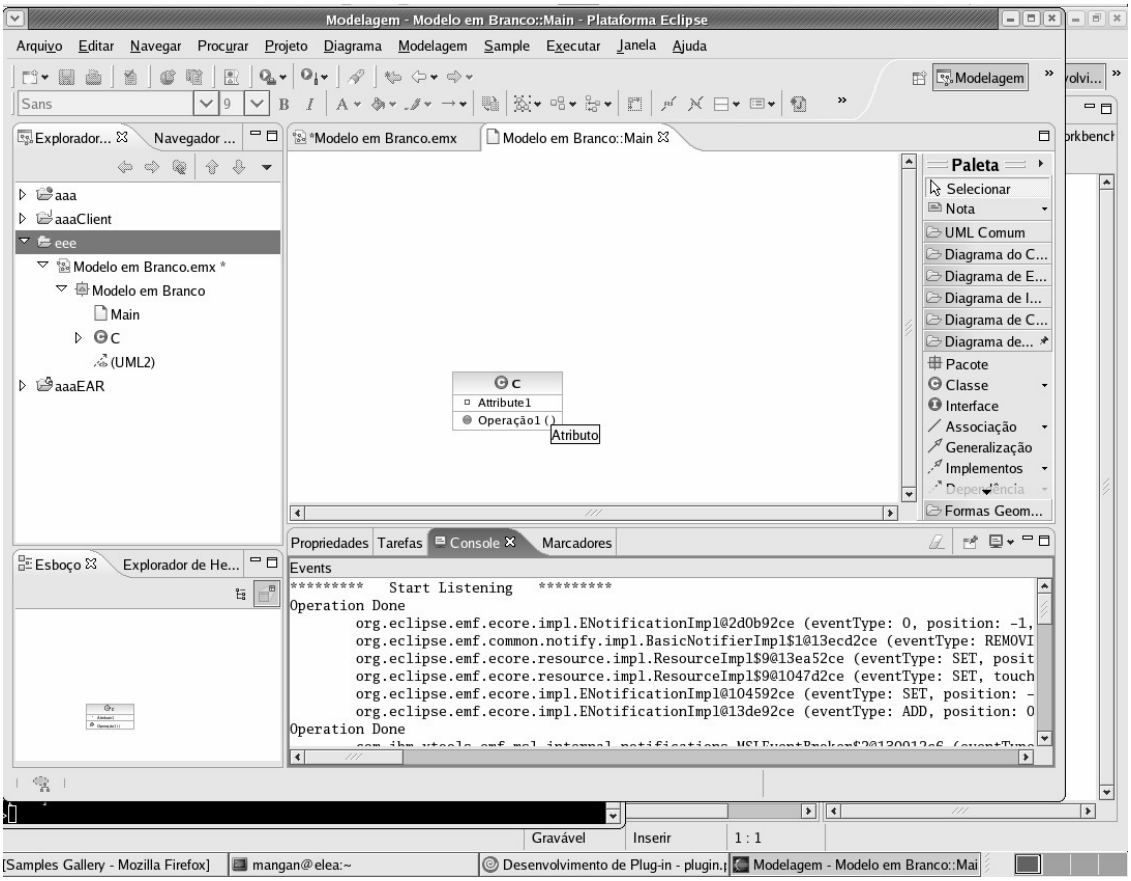



**Figura 6.13 Exemplo de coleta no *Rational Rose Modeler* integrado com o Eclipse.**

A construção deste coletor ajuda a demonstrar que a abordagem pode ser aplicada fora do ambiente *Odyssey*, pois as ferramentas CASE atuais apresentam um conjunto de características similares. O coletor consegue realizar a comunicação com o espaço de tuplas via *web services* ou diretamente pela biblioteca do espaço de tuplas que possui implementações em Java e C# (GIGASPACES, 2004).

## *6.7 Comentários sobre a Aplicação da Abordagem de Desenvolvimento*

Os protótipos foram desenvolvidos em conjunto com a equipe de reutilização de software da COPPE/UFRJ. A experiência de desenvolvimento de protótipos com a

participação de terceiros foi importante para descobrir as principais dificuldades apontadas pelo desenvolvedor de extensões colaborativas.

Nos protótipos *Shared Workspace* e *CineOdyssey* houve a dificuldade de determinar quais as tuplas que seriam necessárias para representar a informação coletada. Nos protótipos subseqüentes, a definição das tuplas foi aproveitada e, com isso, os desenvolvedores puderam se concentrar em outros aspectos da implementação. O espaço de tuplas persistente permite inclusive que se executem testes utilizando as tuplas gravadas, sem a necessidade de executar a aplicação original. Com isso, os testes se tornam mais rápidos, até porque a extensão é um programa de baixa complexidade.

O teleapontador e a visão em miniatura compartilham a mesma definição de tuplas. Os demais protótipos utilizam uma segunda definição de tuplas. Com isso, incentiva-se a reutilização de informação coletada mesmo entre extensões diferentes. Por exemplo, um desenvolvedor pode observar dados de um projeto com o uso do GAW e outro desenvolvedor pode ver os mesmos dados usando o MAIS.

A programação com o uso das tuplas exige um treinamento específico. Os desenvolvedores, normalmente, utilizaram analogias com sistemas de bancos de dados. A consulta de tuplas foi, em geral, associada com a consulta por exemplos (*query by example*) utilizada em alguns sistemas de informação. Uma vez recuperada, a tupla se comporta como um objeto convencional. A notificação de tuplas foi entendida por analogia com o sistema de notificação usado na programação de interfaces de usuário, no qual uma rotina é acionada sempre que uma determinada ação ocorre na interface. As analogias foram feitas por iniciativa dos desenvolvedores. Durante a explicação do espaço de tuplas, não foram feitas analogias.

A principal dificuldade para os desenvolvedores é a concorrência introduzida pela comunicação em rede. Nas versões iniciais dos protótipos, a interface de usuário, geralmente, apresentou problemas de sincronização com os dados da aplicação. De fato, todos os desenvolvedores tinham experiência prévia em programação, mas nenhum havia tido contato com a prática da programação concorrente. Em particular, na plataforma Java, uma aplicação com interface gráfica deve ser programada com certos cuidados, quando existir mais de uma linha de execução em atividade. As notificações que chegam pelo espaço de tuplas introduz, pelo menos, uma segunda linha de

execução. Existem mecanismos específicos para tratar disso, entre eles a classe *SwingWorker*, que permite sincronizar o estado da interface de usuário.

Durante o desenvolvimento, o espaço de tuplas se oferece apenas o serviço de persistência, pois o desenvolvedor utiliza apenas uma estação para executar a extensão colaborativa. Durante os testes com mais de uma estação de trabalho, em tempo real, é que se percebe o serviço de comunicação, quando se percebem dados que foram gravados por outras estações aparecendo. Nestes testes, uma estação acaba por ter de apresentar dados para os quais não recebeu um evento de interface correspondente e os problemas de sincronização são percebidos com maior facilidade.

Outra dificuldade dos desenvolvedores está em perceber a utilização do espaço de tuplas como memória auxiliar. A tendência do desenvolvedor é criar rotinas que obrigam a recuperação de todo o espaço de tuplas e a sua manutenção em memória principal. A maneira mais adequada de trabalhar com o espaço de tuplas é armazenar resultados intermediários no próprio espaço de tuplas. Por exemplo, ao se calcular o total de operações que dois usuários realizaram em um mesmo objeto, esse total deve ser gravado novamente no espaço de tuplas. Com isso, uma vez computado, não é necessário gerar esse total até que uma nova tupla seja gravada por um desses dois usuários, sobre o objeto em questão. A estratégia incorreta seria guardar a lista de tuplas em memória e realizar a contagem em memória toda vez que o total fosse necessário. Da mesma forma, o espaço de tupla pode ser usado para persistir informações sobre o estado da extensão colaborativa. Por exemplo, o MAIS utiliza uma tupla adicional para registrar o último evento notificado para um determinado usuário. Sendo assim, o estado da extensão colaborativa é recuperado da mesma forma em qualquer ponto da rede e não é necessário que a extensão use outro meio de armazenamento.

Na opinião dos desenvolvedores, o espaço de tuplas é mais simples de ser programado em comparação a outros sistemas de persistência ou comunicação (ex. sistema de banco de dados ou invocação de métodos remotos) que foram usados em versões iniciais dos protótipos, antes da introdução do espaço de tuplas. Entretanto, o primeiro contato é difícil, pois o espaço de tuplas não é equivalente a outro sistema que tenha sido usado por eles anteriormente.

## 6.8 Resumo dos Protótipos

Ao longo do desenvolvimento desta tese, as características dos protótipos foram sendo alteradas para oferecer um apoio mais específico para as situações de equipes distribuídas. O primeiro protótipo desenvolvido favorece a interação síncrona, em alto acoplamento. Os demais protótipos utilizam informações mais relacionadas com a semântica do espaço de software e com a interação assíncrona, com baixo acoplamento entre tarefas (Tabela 6-4).

**Tabela 6-4 Características do modelo de domínio e a sua associação com os protótipos.**

| *Características* | *Ferramentas* | | | | |
|---|---|---|---|---|---|
| | Shared Workspaces | Cine Odyssey | Ariane | MAIS | GAW |
| Atributos do Objeto Compartilhado | | | | | |
| Modelo | ○ | ● | ● | ● | ● |
| Diagrama | ○ | ○ | ○ | ○ | ○ |
| WIMP | ● | ○ | ○ | ○ | ○ |
| Documento | ○ | ○ | ○ | ○ | ● |
| Proximidade no Modelo de Percepção | | | | | |
| Causal | ○ | ● | ● | ● | ● |
| Espacial | ● | ○ | ● | ○ | ○ |
| Cognição | | | | | |
| Memória | ○ | ● | ● | ● | ● |
| Associação | ○ | ● | ● | ● | ● |
| Sentidos | | | | | |
| Visão | ● | ● | ● | ● | ● |
| Audição | ○ | ● | ○ | ○ | ○ |

Legenda: ● presente ○ ausente

A diversidade dos protótipos é similar à diversidade verificada nos sistemas estudados (Tabela 3-4). Com isso, podemos argumentar que a arquitetura proposta possui potencial para servir como infra-estrutura para o domínio de aplicação estudado.

## 6.9 Conclusão

Os protótipos desenvolvidos demonstram a viabilidade da arquitetura como infra-estrutura para o apoio à percepção. A participação de terceiros no desenvolvimento auxilia a verificar a viabilidade da proposta.

O sucesso na construção dos protótipos corrobora com as hipóteses sobre o potencial para reutilização de componentes de software na construção de ambientes com maior apoio à percepção. O espaço de tuplas é um elemento importante no apoio à reutilização, pois permite reduzir o acoplamento entre os componentes que produzem e os componentes que consomem as tuplas. Um consumidor não está ligado diretamente a um consumidor. Um componente pode ser substituído por outro componente que produza e consuma um mesmo conjunto de tipos de tuplas. Por fim, as tuplas servem como um "vocabulário comum" que descreve a informação que está disponível para exploração das extensões colaborativas.

Não existe prova definitiva de que a abordagem proposta permita o desenvolvimento com menor custo e em menor tempo dos componentes de software necessários para a construção do ambiente. Uma avaliação empírica dessa afirmação seria praticamente inviável. A variação na habilidade e aplicação entre os desenvolvedores seria muitas vezes superior à vantagem dada pela abordagem. Ou seja, é provável que um desenvolvedor com habilidade alta consiga resultados melhores independente da técnica ou abordagem utilizadas. O problema de determinar se dois desenvolvedores apresentam as mesmas habilidades é difícil de resolver.

O que a construção dos protótipos demonstra é que é possível manter um conjunto de componente de software interoperáveis, mesmo quando não foram realizados pelo mesmo desenvolvedor. Existe um potencial muito alto de reutilização dos elementos responsáveis pela coleta e armazenamento dos eventos. Esses elementos podem ser recombinados para a criação de novos ambientes e substituídos por outros componentes que apresentem as mesmas interfaces. Com isso, fica demonstrado que o domínio estudado apresenta um amplo potencial para sustentar um processo de reutilização de software que pode propiciar uma maior economia na construção e uma melhoria contínua da qualidade dos sistemas desse domínio.

No Capítulo 7, os protótipos são avaliados, de forma empírica, em relação a sua

usabilidade. Em particular, os requisitos de usabilidade (Seção 5.1.2) devem ser verificados. Essa perspectiva complementa a avaliação da abordagem ao demonstrar que o protótipo é capaz de apoiar tarefas de desenvolvimento de software, nas condições que foram previstas para sua utilização.

# 7 Avaliação dos Protótipos

Um dos aspectos mais interessantes da pesquisa em trabalho colaborativo é a realização de estudos de caso no contexto de trabalho de um grupo. Nesta tese, o contexto de trabalho são tarefas de desenvolvimento de software e os integrantes do grupo são desenvolvedores. Neste capítulo, são apresentadas três avaliações realizadas ao longo do desenvolvimento desta tese, as quais permitiram experimentar a reação de terceiros no uso dos protótipos desenvolvidos.

As avaliações foram criadas usando elementos em comum, segundo uma metodologia proposta nesta tese (Seção 7.1). A metodologia descreve passos do planejamento e operação das avaliações, que podem ser reutilizados ou adaptados para diferentes estudos. A metodologia é resultado da adaptação de material e planejamento de outros estudos e também da própria evolução dessas adaptações.

Foram realizadas três avaliações no contexto desta tese. Na primeira e na segunda avaliação, participaram alunos dos cursos de engenharia de software da Universidade Federal do Rio de Janeiro. Na terceira avaliação, participaram desenvolvedores de software profissionais. Cada avaliação descreve uma tarefa na qual um dos protótipos pode ser aplicado no contexto de trabalho de uma equipe de desenvolvimento. A primeira tarefa foi a revisão de documentos com a aplicação dos componentes de área de trabalho (Seção 7.2). Nessa primeira tarefa, caracteriza-se o trabalho de uma dupla de desenvolvedores que necessitam compartilhar informações para completar a tarefa. No cenário proposto, o conhecimento necessário para a realização da tarefa encontra-se dividido entre os participantes, ou seja, um dos participantes não possui toda a informação necessária para completá-la sozinho.

A segunda tarefa foi a compreensão da evolução de documentos com a aplicação dos componentes de percepção de produto (Seção 7.3). Nessa tarefa, o desenvolvedor enfrenta o problema de atualizar a sua percepção de um artefato que foi alterado pela contribuição de outros desenvolvedores.

Finalmente, a terceira tarefa foi a localização de especialistas com a aplicação

dos componentes de apoio à percepção (Seção 7.4). Nessa tarefa, o desenvolvedor enfrenta a dificuldade de compreender a participação de desenvolvedores que trabalham em um projeto desconhecido para ele. Ao final, é apresentado um resumo das avaliações (Seção 7.5).

## 7.1 Metodologia

Do ponto de vista de pesquisa em trabalho colaborativo, existem dois motivos de interesse para essas avaliações. O primeiro é observar o próprio processo de colaboração que emerge das ações do grupo. O segundo motivo é observar a interação entre o usuário e o sistema de apoio à colaboração. Essas avaliações foram planejadas para explorar esse segundo motivo, pois ele está diretamente relacionado com as funcionalidades apresentadas pelo sistema e com a expectativa do usuário em relação ao sistema. No primeiro motivo, a atenção se volta para o uso do sistema. Do ponto de vista do desenvolvimento de software profissional, a realização dessas avaliações está diretamente ligada ao interesse da organização em aprender com suas próprias experiências (LANDES *et al.*, 1999). Ou seja, ao aplicar a avaliação em um grupo de desenvolvedores específico, obtém-se informações que podem ser utilizadas para auxiliar esse grupo a desenvolver um processo de melhoria contínua em relação a sua postura quanto à colaboração e à adequação do sistema colaborativo às necessidades particulares desse grupo. Portanto, no contexto desta tese, o que está sendo apresentado nesta seção é um guia, ou metodologia, de como realizar as avaliações dos componentes, acompanhado de três estudos realizados.

Bridgman (1955) declara que existem tantas metodologias quanto existem diferentes pesquisadores. Ou seja, de fato, cada pesquisador desenvolve sua própria metodologia, combinando idéias de diversas fontes. Portanto, a descrição de um estudo não está completa se não houver uma descrição da metodologia utilizada.

Jail (1991) elabora a idéia da avaliação e da experimentação como uma arte. Uma mesma avaliação ou experimento pode ser planejado de diversas maneiras diferentes e o resultado pode ser manipulado de diversas formas para gerar este ou aquele resultado. Essas afirmações são verdadeiras para avaliações em diversas áreas e, portanto, também se aplicam para a avaliação em computação. Desta forma, o que pode se esperar de toda avaliação é um viés, gerado de forma intencional ou acidental.

Portanto, a apresentação da metodologia cumpre o papel de expor o processo da avaliação de forma explícita.

Esta seção apresenta a descrição dos passos que levaram ao planejamento e realização das avaliações propostas. A metodologia é apresentada como forma de externalizar o processo de formação das avaliações e, também, para incentivar a criação de novas avaliações. A metodologia proposta para o primeiro estudo foi publicada (MANGAN *et al.*, 2002) e, posteriormente, aplicada aos demais estudos.

Podem ser identificados cinco passos para a definição das avaliações.

**Passo 1.** O primeiro passo é a definição de um conjunto de situações que ocorrem durante o desenvolvimento de software, onde se percebe a necessidade de uma melhoria na percepção dos desenvolvedores. Esse conjunto de situação chama-se aplicação. A descrição da aplicação informa as situações onde o apoio à percepção pode ser aplicado. A aplicação é descrita como uma tarefa que faz parte do processo de desenvolvimento de software. A tarefa deve ter um objetivo bem definido e ser claramente identificável.

Nas avaliações a seguir, as tarefas escolhidas têm duração reduzida. Parte-se da premissa de que uma atividade colaborativa é composta de uma seqüência de tarefas. Antes de observar a atividade, é necessário entender as tarefas individualmente.

**Passo 2.** Considerando uma única aplicação, ainda são possíveis diversos estudos. Portanto, o segundo passo na criação de uma avaliação é a descrição do projeto e procedimento da avaliação. A primeira decisão é determinar qual o mecanismo que será avaliado. Há dois tipos de estudo possíveis: (a) um estudo de caracterização, no qual as propriedades do uso de um mecanismo são descritas e (b) um estudo comparativo, no qual um mecanismo é comparado com outro mecanismo similar ou com a situação de ausência de mecanismo. No primeiro tipo de estudo, temos um ensaio para encontrar variáveis mensuráveis. Diversas medidas são utilizadas de forma exploratória para tentar encontrar quais delas podem ser usadas para caracterizar os efeitos positivos e negativos do uso do mecanismo. No segundo tipo de estudo, as medidas já são conhecidas e a influência dos mecanismos sobre essas medidas é que é o principal elemento de estudo.

A escolha entre o estudo de caracterização e o estudo comparativo determina a

maneira como os objetivos e as hipóteses da avaliação são formulados. Estas, por sua vez, influenciam nos instrumentos utilizados e, também, são definidos nesse passo. Em um estudo de caracterização, o principal desafio é encontrar e avaliar as medidas relevantes. Ao final do estudo, novas medidas podem ser encontradas e algumas das medidas podem ser descartadas. No estudo de caracterização, a viabilidade da operação da avaliação tem um papel importante, pois ainda não se tem informação suficiente para afirmar se os fenômenos de interesse vão ocorrer e se a sua observação e sua avaliação serão possíveis.

Uma vez escolhido o tipo de estudo, deve-se passar à descrição dos objetivos e das hipóteses. O modelo de definição de objetivos (BASILI, 1986, WOHLIN *et al.*, 2000) é um formato padronizado de descrição de estudos experimentais composto de cinco declarações. A descrição é formatada para destacar as declarações que são delimitadas por palavras-chave. As palavras-chave são propostas em inglês no original. As traduções dessas cláusulas ainda não encontraram um consenso entre diferentes autores brasileiros, uma possível tradução para o modelo é apresentada a seguir:

**Objeto de estudo** <descrição do objeto de interesse do estudo>

**Propósito** <descrição da finalidade do estudo>

**Foco de qualidade** <descrição das principais medidas do estudo>

**Perspectiva** <indicação de um dos papéis envolvidos no estudo>

**Contexto** <descrição de condições referentes ao contexto do estudo>

A finalidade do modelo é auxiliar a definir e a comunicar, de forma breve e direta, os principais elementos do estudo e pode ser utilizado para identificar semelhanças entre estudos diferentes. O modelo é amplamente adotado pela comunidade de engenharia de software experimental e empírica (TRAVASSOS *et al*., 2002, SAMPAIO *et al*, 2004).

Os objetivos delimitam o contexto para a formação das hipóteses do estudo. São desenvolvidas duas versões da descrição de uma hipótese. A primeira versão da descrição é informal, com o objetivo de expressar uma das questões de pesquisa que envolva uma relação entre as variáveis. Cada uma dessas descrições é transformada em uma segunda descrição mais formal que expressa uma relação matemática, normalmente quantitativa. A segunda versão da hipótese oferece uma hipótese nula e uma ou mais

hipóteses alternativas. A definição desse conjunto de hipóteses exige a determinação das medidas que serão utilizadas no estudo. Entretanto, para poder determinar essas variáveis é necessário descrever em maior detalhe a tarefa que será utilizada na operação do estudo, ou seja, descrever o contexto, o objetivo e as ações que serão realizadas pelos participantes.

**Passo 3.** O terceiro passo é a descrição da tarefa e da operação da avaliação. A aplicação descrita no primeiro passo se refere a uma classe de tarefa; neste passo, é necessário escolher uma tarefa representativa dessa classe. A descrição da tarefa demanda a escolha de ferramentas e dados de execução concretos. Sobre a ferramenta escolhida, um ou mais componentes de percepção serão instalados. Em um estudo de comparação, ainda é necessário encontrar uma condição similar com o uso de um componente ou funcionalidade semelhante para que a comparação possa ser realizada. A descrição da tarefa descreve um contexto e um problema a ser resolvido. A tarefa estará concluída quando uma solução for encontrada ou quando o participante desistir de continuar.

A tarefa será executada por voluntários que precisam ser informados do que deve ser realizado. Naturalmente, as características e o desempenho dos participantes apresentam variações. O material apresentado aos participantes deve ser preparado previamente e registrado em documentos para reduzir a possibilidade de que os participantes sejam influenciados pelas variações geradas pelo pesquisador que conduz a avaliação. Por exemplo, a apresentação do material, se realizada oralmente, pode sofrer uma melhoria ao longo do tempo, a medida que o pesquisador se acostuma a repetir a apresentação. A maior parte da informação referente à avaliação é comunicada na forma de documentos impressos. Entretanto, os participantes podem expor dúvidas ao longo da avaliação. As dúvidas causam alguma variação no material, que é prejudicial para fins de comparação entre os participantes, mas é benéfica para a melhoria dos materiais e do planejamento do estudo.

O primeiro documento apresentado ao participante contém as instruções para a realização do estudo. A condução da avaliação é dividida em etapas para auxiliar a conter a complexidade da execução, orientar o participante e o envolver na manutenção do formato da execução. O participante pode usar essa informação para decidir se concorda em participar e para avaliar se dispõe do tempo necessário. A lista de etapas

varia de acordo com a tarefa proposta.

O segundo documento apresentado ao participante é o termo de compromisso, que declara ao participante os limites da sua participação no estudo, suas responsabilidades durante a avaliação, o tempo previsto para a conclusão e outras informações que sejam necessárias para que o participante decida se aceita continuar sua participação. O termo, também, informa que os dados da avaliação não estão sujeitos a serem utilizados para classificar o desempenho dos participantes nem influenciarão sua avaliação como estudante ou profissional, sempre que isso se aplique ao participante. Esse documento dificilmente necessita de adaptação para um estudo particular.

O documento de descrição da tarefa apresenta ao participante um contexto de trabalho, os dados de entrada disponíveis e o resultado esperado. O documento de descrição de tarefa pode ser acompanhado de documentos auxiliares, tais como, modelos de software e trechos de programas. Este documento é dependente da aplicação, descrita no Passo 1.

O documento de treinamento comunica as principais funcionalidades do componente de percepção utilizado. O documento é uma apresentação ou gravação de tela que demonstra o uso prático do protótipo. No caso de um estudo de comparação, deve ser criado um treinamento para o segundo mecanismo comparado que seja similar em duração e profundidade. O treinamento é dependente do protótipo e ferramentas utilizadas.

Os quatro documentos descritos apresentam informações ao participante. A coleta de informações é realizada com o uso de três instrumentos: questionários, gravações de tela e entrevistas.

Os questionários são utilizados para tornar a condução da avaliação mais objetiva e para auxiliar a concentração do participante em um aspecto particular da avaliação. Sempre que possível, as questões são adaptadas a partir dos questionários já existentes. Existem três tipos de questionários. O questionário de dados do participante, aplicado antes da execução da tarefa, pretende obter informação sobre a experiência anterior, formação, habilidades, aptidão e outros dados descritivos (variáveis independentes). O questionário da tarefa é o instrumento para a coleta dos resultados da tarefa. O questionário de avaliação, aplicado após a execução da tarefa, pretende obter

informação sobre a sessão do ponto de vista do participante. Esse terceiro questionário contém questões de usabilidade selecionadas a partir de questionários similares (PERLMAN, 2002, MANGAN *et al.*, 2002). São apresentadas, também, questões sobre sugestões para o protótipo e para o procedimento da avaliação.

A gravação é útil para rever a execução do procedimento e da tarefa, o que auxilia no aprimoramento do estudo. Além disso, esses dois elementos da avaliação podem ser sujeitos à opinião de terceiros ou de pesquisadores que tenham interesse no mesmo fenômeno sob outra perspectiva. O preenchimento dos instrumentos é gravado, pois é durante o preenchimento dos instrumentos que as dúvidas são manifestadas. A gravação de tela foi realizada em função de sua facilidade de execução, se comparado com a gravação em vídeo. Não é necessário transportar o equipamento de gravação nem converter o formato de vídeo para manipulação no computador. A gravação de tela captura a perspectiva do usuário durante a sua participação. Foi utilizado o programa *TechSmith Camtasia* (TECHSMITH, 2002). O programa alternativo *Lotus ScreenCam* (LOTUS, 2002) falhou na gravação de telas de uma aplicação Java.

A entrevista é uma conversa informal com os participantes, após o preenchimento do último questionário e a conclusão da gravação de tela, explora-se a opinião dos participantes em um formato menos restrito que o do questionário. Com base nas respostas dos questionários, essa conversa também serve para confrontar respostas discordantes entre os participantes, explorando as diferenças entre eles e obtendo as motivações que ponderam suas respostas. A entrevista permite descobrir perspectivas que não foram previstas pelo pesquisador.

**Passo 4.** Do ponto de vista do desenvolvedor de software, a avaliação de uma ferramenta tem por base o consumo de recursos e a qualidade do produto. Uma terceira medida relevante é a satisfação do usuário com o uso da ferramenta. Essas três medidas são ortogonais entre si.

As medidas de tempo e qualidade são geralmente objetivas. A medida de tempo é bem definida. Por outro lado, a medida de qualidade deve ser associada a uma medida relevante na tarefa como o número de erros encontrados ou o número de classes em um diagrama.

A medição da usabilidade utiliza uma escala *Likert* (LIKERT, 1932; ROBSON,

2002) padronizada com cinco valores, de 1 (concordo plenamente) até 5 (discordo totalmente). A escala *Likert* avalia o quanto uma pessoa concorda ou discorda de uma afirmativa. A escala é utilizada para estabelecer uma forma de comparação entre opiniões subjetivas de diferentes participantes, pois a variação de cada participante é colocada em uma escala finita e discreta.

**Passo 5.** O último passo do planejamento é refletir sobre o que pode ser aprendido com a avaliação, caso tenha-se sucesso em realizá-la. As medidas são coletadas ao longo da condução do estudo e pela análise posterior da gravação de tela. Em seguida, é feita a tabulação dos dados coletados. As medidas qualitativas são também organizadas em tabelas.

A análise estatística (FREUND e PERLES, 1999) vai tratar, inicialmente, do resumo dos dados de freqüência e da realização de cruzamentos entre as variáveis independentes (ex.: formação, aptidão) e dependentes (medidas definidas no passo 4). Para comunicar os resultados e para realizar comparações, utiliza-se como medidas de localização e de variação os valores da média e desvio padrão observados. Devido ao tamanho reduzido das amostras ($n < 30$), adota-se o teste de *Student* (t) para avaliar diferenças entre médias. Um exemplo dos questionários utilizados encontra-se no Anexo I. No relatório das seções seguintes, constam apenas os resultados da análise que sejam significativos ou inesperados e a descrição de opiniões e sugestões foi resumida.

Existem diversas interpretações possíveis para os dados coletados, que variam de acordo com a perspectiva do observador. Nas avaliações a seguir, adota-se a perspectiva do desenvolvedor que vai direcionar a observação para encontrar a possibilidade de melhorias dos componentes.

## *7.2 Avaliação dos componentes de percepção de área de trabalho*

Nesta seção é apresentada a primeira observação planejada dos protótipos desenvolvidos no ambiente *OdysseyShare*. Ao todo foram conduzidos apenas dois casos, com um total de dois pares de participantes. No primeiro caso, os participantes revisaram um modelo simplificado de um sistema de informação. No segundo caso, foi utilizado um modelo relacionado com o domínio de aprendizagem colaborativa.

### 7.2.1 Aplicação: Revisão por pares de modelos de software

O propósito de uma revisão por pares (WIEGERS, 2002) é procurar por defeitos ou por oportunidades de melhorias em um artefato de trabalho em construção. Acredita-se que a introdução de tarefas de revisão por pares no processo de desenvolvimento aumente as chances de encontrar erros mais cedo e, portanto, reduzir os custos da sua correção. A revisão por pares é adotada por diversas metodologias de desenvolvimento ágeis, incluindo a *extreme programming*. Sessões de revisão não têm como objetivo a edição de um artefato para remoção de seus defeitos. Durante a condução de revisões em interações síncronas, o desafio durante a edição está em manter as visões de cada participante sincronizadas.

A possibilidade de realizar uma revisão por pares com participantes em locais diferentes permite que a revisão possa contar com especialistas que não se encontram disponíveis localmente. Com isso, o número de pessoas que podem participar da revisão se torna maior e as possibilidades de contribuição se multiplicam. Por outro ângulo, podemos imaginar que um especialista que utiliza o apoio oferecido pode com isso participar de um número maior de revisões e, portanto, aumentar seu conhecimento de soluções e da diversidade de projetos em que opinou.

O apoio oferecido pelos componentes de percepção de área de trabalho se destina à realização de revisões informais e não se limita a um método de revisão particular. A colaboração é motivada por permitir aos usuários conduzirem revisões *ad hoc*, ou seja, no momento em que julgarem necessário, usando algum dos editores oferecidos pelo ambiente *Odyssey* como cenário para a condução das revisões. O apoio de uma técnica específica de revisão como, por exemplo, uma revisão estruturada, pode ser realizada com o esforço dos próprios participantes em manter uma estrutura na colaboração, ou com a especialização do protótipo utilizado.

A tarefa de revisão é um cenário específico que restringe o número de ações dos participantes sobre o ambiente. As ações do usuário se limitam a deslocar barras de rolagem e apontar para artefatos nos diagramas. Mesmo assim, a tarefa de revisão utiliza um meio de colaboração similar ao que oferecem papel e lápis sobre uma mesa, caso os participantes se encontrassem no mesmo local. A escolha da tarefa de revisão *ad hoc* decorre, em parte, das limitações do apoio à colaboração desenvolvido. Entretanto, a

opção pela tarefa de revisão reduz o número de variáveis na avaliação, tornando mais importante o papel do posicionamento dos participantes no espaço compartilhado.

### 7.2.2 Descrição do projeto e do procedimento da avaliação

A proposta desta avaliação é utilizar a informação do sistema de janelas local como fonte de informação de percepção da área de trabalho, que poderia guiar revisores que compartilham uma mesma visão sobre um diagrama. Um participante enfrenta dificuldades de comunicação quando tenta manter seus colegas informados sobre sua localização e posição de edição corrente na área compartilhada, em uma sincronização *WYSIWIS relaxada*. Os participantes podem sobrepor essas dificuldades pela verbalização explícita da sua posição para seu colega remoto por meio de áudio ou troca de mensagens. Essa verbalização explícita pode sobrecarregar a comunicação e, quando inadequada, pode causar impacto na produtividade da interação e na satisfação dos participantes.

Este é um estudo de caracterização que conta com a observação da interação gravada e a opinião dos participantes. O modelo de definição de objetivos (WOHLIN *et al.*, 2000) para este estudo fica preenchido da seguinte forma:

**Objeto de estudo** indicadores de posição de edição e localização da janela de edição.

**Propósito** caracterizar o uso dos indicadores.

**Foco de qualidade** esforço realizado e satisfação do usuário.

**Perspectiva** do revisor.

**Contexto** uma dupla de revisores, conferindo a descrição textual de um sistema e o correspondente diagrama de classes.

As hipóteses sendo testadas são:

**H1:** Os participantes utilizam os indicadores para reduzir a necessidade de orientação verbal.

**H2:** Os participantes consideram que estão menos propensos a cometer erros quando se referem aos artefatos do modelo enquanto são apoiados pelos indicadores.

**H3:** Os participantes consideram que a tarefa foi realizada de maneira satisfatória com o uso dos indicadores.

Os participantes selecionados foram quatro estudantes de computação. O *Odyssey* é um ambiente para pesquisa e uso acadêmico, portanto, julgou-se apropriado utilizar alunos como participantes. Cada participante satisfez três critérios de inclusão: (a) ser um usuário do *Odyssey*, (b) conhecer a notação da UML e (c) não ter experiência com o uso do protótipo. Os participantes formaram duas duplas.

Pelo menos três avaliações com condições semelhantes podem ser encontradas na literatura (BEGOLE *et al.*, 1999, MAURER e MARTEL, 2002, FAVELA *et al.*, 2004).

### 7.2.3 Tarefa

A avaliação conta com cinco instrumentos para registrar o compromisso, ações e contribuições dos participantes. Sendo esses instrumentos: um termo de adesão, dois questionários que são aplicados antes e depois da execução da tarefa, uma conversa informal com os participantes após o preenchimento do último questionário e a gravação de tela da execução da tarefa, conforme explicado na metodologia (Seção 7.1).

A conversa informal ocorre com a presença de quatro pessoas. Os dois participantes que formam um mesmo par são entrevistados ao mesmo tempo pela equipe de avaliação (composta do autor e de um aluno de mestrado).

Três documentos fazem parte da descrição da tarefa: uma descrição dos objetivos da tarefa, uma descrição de requisitos e um diagrama de classes. A descrição de requisitos foi dividida em duas partes, para induzir a necessidade de comunicação entre os participantes. Deste modo, ambos os participantes necessitam compartilhar e obter informações do seu par para poder revisar o diagrama completo.

Durante a avaliação, o diagrama completo está disponível na tela; os outros dois documentos estão impressos. A avaliação apresenta seis etapas, listadas a seguir:

Etapa 1: Preenchimento do termo de autorização

Etapa 2: Preenchimento do primeiro questionário

Etapa 3: Realização de uma apresentação informal dos participantes

Etapa 4: Leitura por parte de cada participante da sua metade do documento de requisitos

Etapa 5: Execução da tarefa de revisão

Etapa 6: Preenchimento do segundo questionário

A gravação de tela foi realizada apenas na etapa 5. A condução da avaliação ocorreu em uma sala onde os computadores dos participantes estavam separados por uma distância de aproximadamente três metros, o suficiente para que eles pudessem conversar, mas dispostos de uma maneira que impedia a visão da tela do outro participante. Essa configuração em uma mesma sala pode influenciar o comportamento dos participantes. No caso de comparar dados coletados em salas diferentes, essas alterações teriam de ser investigadas. Essa configuração foi adotada para simplificar a avaliação. Os participantes encontram o ambiente configurado para compartilhar os indicadores de edição e de localização entre os dois computadores que serão utilizados. Cada participante utiliza microfone e fone-de-ouvido para que a comunicação entre os participantes fique registrada, juntamente com a gravação de tela. Antes de iniciar a tarefa, é realizado um teste de gravação de um minuto para verificar que o ambiente encontra-se funcionando corretamente.

Os participantes realizam uma sessão individual de revisão do diagrama com o uso de sua metade do documento de requisitos. De forma independente, cada participante observa o funcionamento dos indicadores de posição e localização. Quando ambos estão satisfeitos com a sua revisão individual, inicia-se o uso da área de trabalho compartilhada.

### 7.2.4 Medidas

O total de orientações verbais (o) utilizadas pelos participantes (ex. “esta classe aqui”, “a classe do outro lado do relacionamento”) é uma das medidas previstas para determinar o entendimento dos indicadores. A produtividade é medida pelo tempo total para realizar a tarefa (t) e o número de defeitos ou melhorias encontrados (d). A quarta medida é a satisfação do usuário, avaliada com uma escala *Likert* por meio de questões de usabilidade.

Devem ser levadas em consideração variáveis independentes que afetam as medidas observadas. Entre elas, o conhecimento dos revisores e sua experiência nos seguintes tópicos: revisão por pares, operação do ambiente e entendimento dos

indicadores.

### 7.2.5 Resultados e discussão

Visto que o estudo se limita a caracterizar o uso dos componentes, as observações se concentram em registrar a relação dos participantes com o apoio presente durante a realização da tarefa.

O uso de orientações verbais (o) foi observado durante toda a avaliação com uma média de ocorrência de mais uma orientação a cada 40 segundos. A orientação inclui o uso do ponteiro para indicar uma das classes do diagrama e o uso do nome da classe. A orientação ocorre quando os participantes durante a transição passam a falar sobre uma nova classe. Devido ao uso combinado da especificação impressa, algumas orientações foram mais incisivas, quando um participante estava lendo o texto enquanto o outro participante tentava apontar para um elemento do diagrama. O indicador foi mais efetivo quando utilizado para apontar elementos do diagrama nessas situações onde um dos participantes tenta localizar uma referência em um contexto diferente daquele onde ele está trabalhando. O indicador apresenta referências sem qualquer ambigüidade. Não existe discussão ou réplica uma vez que os dois participantes podem confirmar que estão ambos apontando para o mesmo elemento.

A primeira sessão de revisão durou cerca de 20 minutos e a segunda cerca de 40 minutos. Conforme esperado, o tempo total (t) foi proporcional ao número de elementos do diagrama analisado. Em média, os participantes revisaram uma classe a cada dois minutos.

O total de defeitos e melhorias (d) foi dentro do esperado. Foram encontrados, sobretudo, problemas de conformidade entre o que está descrito na especificação textual e no diagrama. Os revisores não chegaram a determinar defeitos, pois os participantes não julgaram ter informação suficiente para decidir entre relatar um defeito e relatar uma sugestão de melhorias. Por exemplo, os revisores indicaram que o nível de abstração do diagrama pode ser decisivo. Os diagramas apresentados foram considerados como diagramas de alto nível, ou seja, relacionados com as etapas iniciais de análise. Apesar disso, defeitos injetados nas especificações foram corretamente detectados, como, por exemplo, a inexistência de elementos no diagrama que são

descritos apenas na especificação. Por outro lado, houve a indicação de melhorias não esperadas como a sugestão de registrar o comportamento dinâmico descrito na especificação com o uso de diagramas de seqüência.

Os participantes manifestaram dúvidas sobre as condições da revisão. Segundo o que foi exposto nas instruções, o diagrama não deve ser editado. Entretanto, o ambiente permite alguns tipos de alteração que não são refletidas na estação remota. Por exemplo, os participantes alteraram o formato de elementos e o tamanho de letras para facilitar a leitura, introduzindo uma certa imprecisão na apresentação dos indicadores entre as estações. Visto que as especificações textuais são diferentes para cada participante, surgiu a dúvida se os diagramas são iguais ou diferentes para cada participante. Os participantes sugeriram o uso da ação de arrastar-e-soltar sobre o *miniview*. Foram feitos também comentários sobre o esforço para programar uma nova funcionalidade. Os participantes também demonstraram preocupação com o espaço ocupado em disco pela gravação de tela.

Na primeira sessão, um dos participantes relatou como um defeito uma característica do diagramador do ambiente de desenvolvimento de software. O *Odyssey* não apresenta relacionamentos associativos e a diagramação foi realizada com o uso de uma classe associativa auxiliar.

Durante o uso, ocorreram três perdas de conexão cuja causa não foi detectada. Apenas uma das perdas ocorreu durante a sessão colaborativa. A segunda sessão foi interrompida aos 8 minutos e reiniciada. As outras duas perdas ocorreram durante o teste de conexão. A tecnologia empregada utiliza um objeto remoto para representar cada estação de trabalho. Com isso, ao perder uma conexão, apenas a movimentação de um dos participantes permanece e a do outro se perde. Apenas o participante na estação que perdeu a conexão consegue perceber o problema, enquanto o outro continua a trabalhar normalmente. Restabelecer a conexão obriga a criação de uma nova sessão de trabalho.

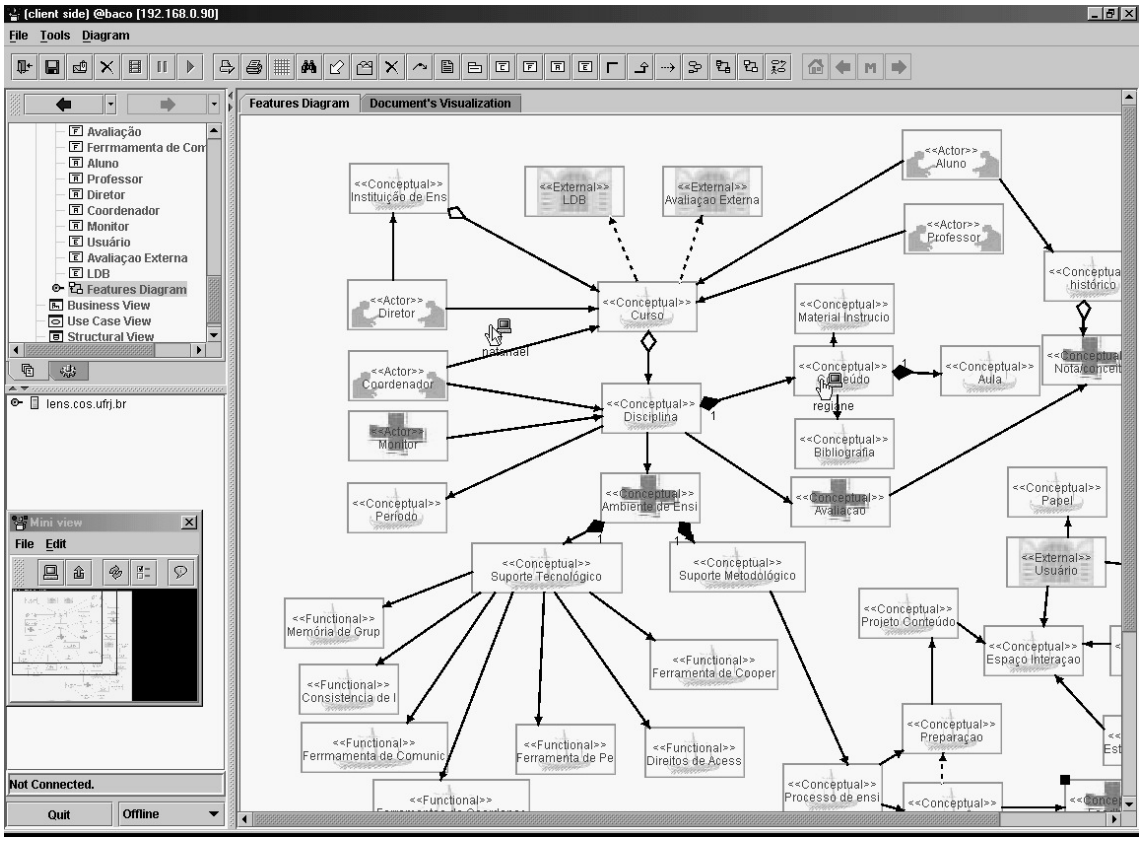



**Figura 7.1 Segunda sessão de avaliação, modelo de domínio para CSCL.**

Nas duas sessões, os participantes demonstraram cansaço após alguns minutos de interação. A principal causa de impaciência foi a falta de um critério para o encerramento da revisão. Os participantes preocuparam-se em corresponder a uma expectativa de total de defeitos encontrados ou de tempo para realizar a tarefa. Na prática, essas restrições não são conhecidas e, portanto, foram omitidas nas instruções.

A estratégia de revisão foi diferente entre cada sessão, por decisão dos participantes. Em realidade, não houve discussão sobre qual seria a abordagem, mas durante a sessão a estratégia emergiu da interação entre os participantes. Na primeira sessão, a revisão foi organizada pelos itens do diagrama, cada participante localiza todos os elementos da especificação textual que podem se referir ao item em discussão. Na segunda sessão, os participantes seguiram a ordem da especificação textual, movendo-se pelo diagrama para identificar onde a declaração da especificação se localiza. Na segunda estratégia, o uso dos componentes de percepção é mais intenso, pois um dos participantes move sua localização para acompanhar uma declaração do texto e obriga ao outro participante a se mover também.

Esta avaliação alertou para dois problemas relacionados ao tipo de apoio à percepção avaliado quando aplicado ao desenvolvimento de software. Primeiro, que o apoio oferecido limita-se a criar a idéia de compartilhamento visual da área de trabalho. Ou seja, os componentes de apoio à percepção não reconhecem nenhuma semântica associada aos objetos sendo manipulados. Notou-se que um apoio assim limitado apresenta poucas vantagens em relação às ferramentas comerciais similares como o *NetMeeting* (MICROSOFT, 2002). Segundo, o cenário proposto obriga o uso síncrono da ferramenta, ou seja, os participantes necessitam encontrar um horário em comum para realizar a tarefa. Esse horário em comum é uma condição difícil de ser estabelecida em equipes distribuídas. O terceiro problema é decorrente da arquitetura simétrica desse primeiro protótipo. Cada estação é ao mesmo tempo cliente e servidor, o que gera problemas de configuração, segurança, consistência e tolerância à falhas. Com base nessas constatações, buscou-se explorar a semântica dos objetos envolvidos, apoio ao uso assíncrono e uma nova arquitetura de distribuição.

Do ponto de vista da metodologia, após essa avaliação foram introduzidos o documento de treinamento e o uso de legendas para análise dos vídeos.

## *7.3 Avaliação dos componentes de percepção de produto*

O conceito de percepção está associado a um estado mental de determinado indivíduo (SOHLENKAMP, 1998). Portanto, é desejável observar a utilização do protótipo MAIS, para determinar a sua aplicabilidade em determinado cenário, levando a possíveis novos requisitos e melhorias. Esta seção apresenta um estudo, de foco qualitativo, realizado sobre a utilização do protótipo, na atividade de modelagem concorrente. O planejamento foi realizado em conjunto com um aluno de mestrado.

### 7.3.1 Aplicação: Atualização de documentos relacionados

Um projeto de software é normalmente composto por um conjunto de documentos relacionados. A medida que o projeto evolui, documentos são atualizados e novos documentos são criados. Os documentos são alterados por diversos participantes de forma concorrente e cada participante percebe as alterações em momentos diferentes. Se as alterações não são percebidas ao mesmo tempo, podemos encontrar entre os

participantes, diferentes percepções sobre o conteúdo de cada documento.

Em geral, os desenvolvedores não se interessam por todas as mudanças que ocorrem nos documentos do projeto. Apenas os documentos necessários à tarefa são considerados, como uma forma de conter a complexidade do trabalho. Sendo assim, periodicamente, um desenvolvedor encontra um documento que é de seu conhecimento e que já foi lido anteriormente. A primeira dúvida do desenvolvedor é se esse documento foi alterado ou não. Caso tenha sido alterado, deseja-se saber quais são os pontos alterados.

O uso do protótipo MAIS se concentra, principalmente, na etapa de convergência das cópias do modelo compartilhado em uma cópia consistente, armazenada em um repositório central. Esta convergência é realizada com base na fusão e adaptação das mudanças realizadas pelos desenvolvedores nessas cópias. Como as adaptações podem ser custosas com relação a tempo e esforço, já que esse grupo de desenvolvedores pode estar interagindo nas cópias do modelo compartilhado de forma isolada, é interessante antever possíveis adaptações. Para isso, é necessário que cada desenvolvedor conheça as mudanças realizadas pelos demais. Uma das formas de se conhecer essas mudanças é comparar o que há de diferente em duas cópias do modelo compartilhado. Considerando essa necessidade, a informação percepção de mudança do produto torna-se útil para detectar essas alterações e evitar que o desenvolvedor tenha que rever todos os documentos que já foram utilizados em uma tarefa recente.

### 7.3.2 Descrição do projeto e do procedimento da avaliação

A proposta desta avaliação é usar a informação apresentada pelo componente de percepção de produto para auxiliar um desenvolvedor a localizar as mudanças recentes sofridas por um documento.

O modelo de definição dos objetivos deste estudo é descrito pela estrutura a seguir:

**Objeto de estudo** a lista de informações de percepção de produto.

**Propósito** caracterizar a utilidade do mecanismo.

**Foco de qualidade** satisfação em decorrência do ganho de rapidez e eficiência na

realização da atividade de modelagem.

**Perspectiva** do desenvolvedor.

**Contexto** quatro estudantes de engenharia de software, utilizando o ambiente *Odyssey*, em uma atividade que envolve a revisão informal de um documento em relação a alterações realizadas em um modelo relacionado.

As hipóteses a serem testadas são:

**H1:** Os participantes identificam as alterações em menos tempo com o uso da lista de informações de percepção.

**H2:** Os participantes encontram uma maior usabilidade na lista de informações.

### 7.3.3 Tarefa

Para a realização deste estudo, foram convidados quatro voluntários. Dois são alunos do curso de Ciência da Computação desta universidade e outros dois são alunos do Programa de Engenharia de Sistemas e Computação deste Programa. Os voluntários possuem alguma experiência anterior em modelagem de software e no uso do ambiente *OdysseyShare*. Porém, nunca utilizaram o componente de percepção de produto.

Os instrumentos desta avaliação seguem a proposta de instrumentos explicada na metodologia (Seção 7.1) Os voluntários utilizaram a versão 1.1.0 do ambiente *OdysseyShare*. Dois voluntários foram selecionados para utilizar esta versão do ambiente, combinada com a versão 1.1.0 do protótipo de componente de percepção.

O procedimento adotado para execução no estudo é resumido pelos passos a seguir listados:

Etapa 1: Assinatura do termo de compromisso por cada participante.

Etapa 2: Preenchimento do questionário de caracterização de participante.

Etapa 3: Leitura, por parte de cada participante, da descrição da tarefa.

Etapa 4: Explicação oral sobre o uso do protótipo e do ambiente *OdysseyShare*.

Etapa 5: Teste de gravação de vídeo.

Etapa 6: Realização da tarefa e gravação desta.

Etapa 7: Preenchimento do questionário sobre a execução da tarefa.

A tarefa contemplada neste estudo consiste na identificação, pelos participantes, das modificações realizadas entre duas versões de um modelo de classes. O documento que descreve a tarefa apresenta uma breve descrição sobre os elementos constituintes deste modelo, derivado do modelo de classes que descreve a representação de eventos no protótipo MAIS.

Os participantes deste estudo são organizados em dois grupos, quanto a realização da tarefa proposta: (i) os que utilizaram o protótipo para identificar as mudanças (p1 e p3) e (ii) os que não o utilizaram (p2 e p4). Foram fornecidas, para cada participante, as cópias do modelo de classes referentes às versões anterior e posterior deste. As modificações sobre o modelo de classes em questão foram realizadas no próprio ambiente *OdysseyShare* e registradas pelo protótipo MAIS. Esse histórico de eventos foi persistido, pois seria utilizado na realização da tarefa com o protótipo.

As sessões de utilização do protótipo foram individuais. Cada participante leu as instruções apresentadas no documento de descrição da tarefa, além de explicações orais sobre a finalidade do protótipo (para aqueles que o utilizaram). Explicações sobre a utilização do ambiente *OdysseyShare* não foram necessárias, devido à experiência de todos os participantes na utilização deste. Os participantes que utilizaram o protótipo MAIS o fizeram como se fossem desenvolvedores, que estariam manipulando concorrentemente o modelo de classes em questão. Desta forma, foi possível observar as mudanças realizadas que derivaram a versão anterior do modelo na posterior deste, registradas na forma de eventos.

Conforme as mudanças eram identificadas, estas eram reportadas em uma seção do documento de descrição da tarefa. Estas modificações foram descritas na forma de texto livre.

### 7.3.4 Medidas

O total de mudanças observadas (m) pelos participantes (ex: “a classe X foi acrescentada”) é uma das medidas previstas para determinar o desempenho do participante. Foram realizadas dezenove mudanças entre as versões do modelo. O tempo (t) para concluir a tarefa é uma medida de produtividade avaliada. A satisfação do

usuário (u) é a terceira medida do estudo, dentro de uma perspectiva de usabilidade.

O número de mudanças observadas é indicado pelo participante no questionário sobre a tarefa. O tempo é obtido pela análise da gravação de tela. A satisfação é avaliada por um questionário de usabilidade, conforme foi relatado na metodologia (Seção 7.1).

### 7.3.5 Resultados e discussão

Os participantes apresentam experiência relacionada à modelagem concorrente de artefatos descritos pela notação UML e à utilização de aplicações colaborativas similares. Esta informação foi inferida pelo preenchimento do questionário de caracterização de participante.

As medidas de tempo (t) e do número de mudanças (m) não foram afetadas pelo uso do protótipo. Entretanto, o protótipo obteve uma melhor avaliação quanto à usabilidade.

O participante p2 relatou que, tomando como base o ferramental oferecido a este para realização da tarefa, fica muito difícil comparar as duas versões do modelo, mesmo quando se tem um número pequeno de classes envolvidas. Este participante também informou que nem todas as informações de mudanças eram visíveis pelos diagramas, o que fez com que este tivesse que editar cada elemento do modelo para visualizar suas propriedades, com a finalidade de observar se houveram ou não modificações.

Já o participante p4 relatou que, para modelos de classe muito pequenos, a procura visual das diferenças pode ser simples e eficiente. Contudo, este participante se sentiu inseguro ao realizar a tarefa de identificação de mudanças, devido à ausência de um ferramental próprio para tal. Este participante informou que a observação visual das modificações não é segura quanto à identificação destas.

Apesar do participante p1 ter levado mais tempo que o participante p2 para identificar as mudanças, o primeiro as descreveu em um nível de detalhe maior do que o segundo. Além disso, o participante p1 relatou que, além de observar os eventos no protótipo MAIS, verificava se as modificações que este apresentava estavam condizentes com as versões do modelo de classes utilizado na tarefa. Este participante informou que, com o uso do protótipo MAIS, foi fácil a identificação das mudanças

ocorridas em cada elemento do modelo, principalmente aquelas que não eram aparentemente visíveis (isto é, mudanças que não eram possíveis de se identificar apenas comparando visualmente os diagramas referentes às versões do modelo) .

Porém, o participante p1 relatou que, apesar de ser possível identificar, no protótipo MAIS, a ocorrência de determinada mudança, existem casos que essa informação não descreve os detalhes dessas alterações (por exemplo, em uma mudança de assinatura de um determinado método, a versão atual do protótipo MAIS apenas informa que este método mudou, não oferecendo informações sobre o que mudou). Além disso, este participante sugeriu que os detalhes referentes a um determinado evento fossem apresentados ao clicar com o botão direito do mouse.

Por fim, o participante p3 informou que não utilizou, de maneira efetiva, o protótipo MAIS para a realização da tarefa, realizando um trabalho de "garimpo" das modificações. Em suma, este participante realizou a tarefa como se não possuísse o apoio do protótipo. A justificativa para este fato, segundo este participante, foi a ausência de uma descrição mais detalhada de informações sobre o protótipo. Além disto, este participante relatou que o treinamento na utilização do protótipo deveria ser realizado antes da execução do estudo, já que, na visão deste, o tempo despendido para essa atividade foi insuficiente.

De posse das informações geradas pela realização do estudo, é possível identificar que o planejamento deste necessita de melhorias. A configuração do ferramental para execução do estudo foi problemática, ficando dependente de certos recursos computacionais do ambiente de execução destes. A realização do estudo com os quatro participantes demonstrou oportunidades de melhorias no planejamento deste e no protótipo MAIS. Apesar do número de participantes envolvidos neste estudo não ser estatisticamente relevante, foi possível obter um panorama inicial sobre os pontos positivos e negativos do protótipo, destacados no processo de utilização. Foram identificadas necessidades de melhorias no planejamento do estudo, já que a insuficiência do treinamento oferecido para a utilização do protótipo fez com que um participante não o utilizasse para a realização da tarefa. Além disso, observou-se a necessidade de uma melhor adaptação do questionário de execução de tarefa preenchido pelos participantes que não usam o protótipo.

Na execução deste estudo, foi observado que uma ferramenta como o protótipo MAIS é útil na identificação de mudanças entre duas versões de um modelo, dado o relato da dificuldade de realização da tarefa sem ele. Observou-se também que os participantes, que não utilizaram o protótipo MAIS, gostariam de ter um ferramental para apoio a realização da tarefa estipulada. Porém, foi identificado que o protótipo, em certas situações, deveria prover um maior nível de detalhe em relação a uma determinada mudança (evento).

A utilização do protótipo não acelerou, no contexto deste estudo, a identificação das modificações realizadas, como definido na hipótese deste. Acredita-se que isto ocorreu por motivo de desconfiança do participante quanto à autenticidade da informação provida pelo protótipo. Na medida em que os usuários do protótipo acreditem nas informações do protótipo, é razoável imaginar que este tempo pode ser reduzido.

Porém, a hipótese de que a utilização do protótipo MAIS incrementaria o estado de percepção de mudanças dos participantes pode ser observada. Na realização da tarefa proposta pelo estudo, as modificações descritas pelo participante que usou efetivamente o protótipo possuíam um maior nível de detalhe, se comparado aos que executaram a tarefa sem o apoio do protótipo.

## *7.4 Avaliação dos componentes de percepção de grupo*

Nesta seção é apresentada a segunda observação planejada de um protótipo, integrado, neste caso, com o ambiente *Eclipse*. Ao todo foram conduzidas seis observações, com um total de seis participantes, analisando um projeto de desenvolvimento real. Novamente, a observação adaptou instrumentos e a concepção das observações realizadas anteriormente.

### 7.4.1 Aplicação: Encontrar especialistas em artefatos

A busca por especialistas em artefatos é uma parte importante do problema de alocação de recursos na engenharia de software. Em um projeto, a execução de uma tarefa normalmente envolve a manipulação de um conjunto de artefatos, por exemplo, arquivos de código fonte. Um especialista em artefatos é aquele que apresenta

conhecimento relevante sobre os requisitos, concepção ou implementação de um dado artefato e que é importante para a evolução e manutenção deste artefato. O especialista pode possuir informações não documentadas sobre discussões, alternativas de projeto e decisões tomadas. Esse conhecimento é útil também na compreensão e adaptação de artefatos de software empacotados como, por exemplo, componentes de software.

Herbsleb *et al.* (2001, 2003) estabeleceram por meio de observações empíricas que a manutenção da colaboração é relativamente simples, do ponto de vista técnico, pois existem diversas alternativas disponíveis. No contexto de grandes empresas, não é raro encontrar sistemas de áudio e vídeo-conferência e outras facilidades que nem sempre estão disponíveis para o usuário comum. Entretanto, os mesmos estudos relatam que, na maioria dos casos, a colaboração não é iniciada porque os participantes não são capazes de identificar quem são seus colaboradores potenciais.

Em alguns contextos, a informação sobre quem é o especialista sobre um determinado artefato pode se encontrar incompleta ou desatualizada. A informação pode estar incompleta, quando se encontra fora da esfera de influência da gerência local, e desatualizada, quando foi coletada em um ambiente dinâmico. Em particular, no desenvolvimento de software globalizado (DSG), um gerente pode não ter acesso aos dados dos desenvolvedores remotos e a informação pode estar desatualizada se é mantida, manualmente, pelos desenvolvedores e seus gerentes. Em contextos locais, existem diversas oportunidades para que esta informação seja atualizada durante conversas casuais e o convívio diário. Na prática, os desenvolvedores em um mesmo local, que são responsáveis por manter essas informações atualizadas, percebem poucos benefícios, pois já estão cientes das mesmas. Portanto, argumenta-se que o uso de coleta implícita é a melhor alternativa para manter a informação sobre os especialistas de artefatos atualizada e completa.

Trabalhos relacionados ao apoio de programação distribuída em duplas (SCHUMMER e HAAKE, 2001; FAVELA *et al*., 2004; MAURER e MARTEL, 2002) não abordam o problema da formação de pares. Ao menos dois estudos (FAVELA *et al*., 2004; MAURER e MARTEL, 2002) apresentam resultados conflitantes sobre a aplicação de mecanismos de apoio à percepção na programação distribuída. Possivelmente, a experiência dos desenvolvedores possa ser um dos fatores a serem considerados para explicar tais conflitos. Um par com baixa experiência sobre os

artefatos manipulados pode obter um desempenho inferior, independente do mecanismo de apoio à colaboração existente durante a sessão de trabalho. Além disso, outros tipos de conhecimento podem ser determinantes como, o grau de habilidade necessário para a realização da tarefa e o conhecimento de conceitos do domínio da aplicação.

### 7.4.2 Descrição do projeto e do procedimento da avaliação

A proposta desta avaliação é usar a informação de um repositório como fonte de informação de percepção de grupo que poderia guiar desenvolvedores a encontrar outros desenvolvedores para colaborar. O repositório registra ações de *commit*, para cada artefato. Uma ação de *commit* ocorre quando as alterações realizadas na área de trabalho local do desenvolvedor são transferidas para o repositório. Foram avaliadas duas representações diferentes para essa informação. A primeira é uma representação em forma de tabela do registro de atividades do repositório. A segunda é uma representação gráfica dos mesmos dados.

O modelo de definição de objetivos para este estudo é o seguinte:

**Objeto de estudo** representações gráficas e textuais para informação de percepção.

**Propósito** avaliar o uso de ambas as representações.

**Foco de qualidade** esforço realizado e satisfação do usuário.

**Perspectiva** do programador distribuído.

**Contexto** seis desenvolvedores profissionais, divididos em dois grupos de três desenvolvedores, que procuram individualmente por especialistas sobre artefatos.

As hipóteses a serem testadas são:

**H1:** Os participantes selecionam desenvolvedores diferentes dependendo da representação.

**H2:** Os participantes selecionam desenvolvedores em menor tempo com a representação gráfica.

**H3:** Os participantes encontram uma maior usabilidade na representação gráfica.

Os participantes selecionados são seis desenvolvedores profissionais de um centro de desenvolvimento de software global localizado no sul do país. O centro está integrado com outros dois centros nos Estados Unidos e na Espanha. Um participante

deve satisfazer três critérios de inclusão: (a) ser um desenvolvedor profissional, (b) ser um programador Java, e (b) ser um usuário do ambiente Eclipse. Os participantes são distribuídos aleatoriamente em dois grupos. Um grupo recebe a representação textual e o outro recebe a representação gráfica. Ambos os grupos recebem a mesma tarefa e um treinamento em formato de apresentação da ferramenta de percepção que vão utilizar. Portanto, temos dois tratamentos e cada grupo irá receber apenas um tratamento, sem ter conhecimento de que existe um segundo tratamento diferente.

Os três questionários do estudo anterior foram adaptados para esta tarefa. O primeiro é preenchido antes da realização da tarefa e concentra-se em determinar a experiência prévia do participante. O segundo questionário é preenchido durante a tarefa e contém perguntas sobre os artefatos e desenvolvedores do projeto que devem ser respondidas com o uso da representação recebida. O terceiro questionário concentra-se em questões de usabilidade e é preenchido após a realização da tarefa.

O procedimento apresenta seis etapas, listadas a seguir, acompanhadas do tempo médio para conclusão:

Etapa 1: Preencher o termo de autorização (5 minutos).

Etapa 2: Preencher o questionário sobre experiência prévia (10 minutos)

Etapa 3: Assistir o treinamento (10 minutos)

Etapa 4: Executar a tarefa (30 minutos)

Etapa 5: Preencher o questionário sobre usabilidade (10 minutos)

A gravação de tela durante a execução da tarefa (Etapa 4) foi realizada da mesma maneira que nas avaliações anteriores.

### 7.4.3 Tarefa

O fenômeno de interesse é a seleção de um desenvolvedor para desempenhar uma tarefa distribuída. Existem duas abordagens para provocar a ocorrência desse fenômeno. A primeira é solicitar diretamente aos participantes para selecionar um desenvolvedor em um contexto hipotético. A segunda é solicitar ao participante para realizar uma tarefa mais ampla em que este tenha a necessidade de selecionar um colaborador. A desvantagem da primeira abordagem é que a questão principal da pesquisa pode se tornar aparente para o participante. A segunda abordagem apresenta a

desvantagem de aumentar o tempo total necessário para a avaliação e de conter elementos que não são imediatamente importantes para esta pesquisa, tais como, problemas inerentes da programação em pares distribuídos. Optou-se pela primeira abordagem, a mais direta, para reduzir o tempo que é solicitado ao participante. Além disso, devido a complexidade da tarefa proposta, o participante não é capaz de controlar seus resultados, mesmo se os objetivos da pesquisa estiverem evidentes.

A tarefa proposta apresenta a descrição de uma tarefa de um projeto que exige a inclusão de novas funcionalidades em um conjunto existente de arquivos fonte. A descrição também informa ao participante que algumas informações sobre as classes não se encontram documentadas. Nem todas as classes do projeto são relevantes para a tarefa. O participante deve usar o navegador de classes, ferramenta presente no ambiente Eclipse, para localizar algumas classes que satisfaçam um determinado critério descrito nas instruções. Os participantes devem navegar pelo repositório e selecionar três desenvolvedores para entrar em contato. A tarefa se encerra quando o participante concluiu a seleção. O repositório utilizado foi o repositório do projeto Ant, disponível pela Internet (ANT, 2005).

### 7.4.4 Medidas

A seleção dos três desenvolvedores mais ativos é esperada como resposta correta. Deste modo, o tempo total (t) utilizado para alcançar a resposta é uma das medidas principais. Também foi medido o tempo de resposta em cada questão no segundo questionário. Por exemplo, o tempo para encontrar as classes e o tempo para encontrar os desenvolvedores é contabilizado separadamente. Apenas o tempo para encontrar os desenvolvedores é influenciado pelo tratamento. Portanto, as medidas não influenciadas pelo tratamento são utilizadas para comparar o desempenho dos participantes. A seleção dos participantes (s) é outra medida. No mínimo, dois dos desenvolvedores mais ativos devem ser listados para uma resposta correta. O participante pode propor outros desenvolvedores se a escolha for explicada por outra regra, que não a freqüência de contribuições. Por exemplo, um participante pode selecionar os autores dos artefatos e com isso encontrar uma resposta correta, ainda que inesperada. A avaliação da satisfação do usuário (u) por meio de questões de usabilidade, utilizando uma escala *Likert*, com base em uma avaliação de usabilidade

também aplicada nos estudos anteriores.

### 7.4.5 Resultados e discussão

O trabalho assume que a distribuição da população que está sendo amostrada obedece a forma de uma curva normal com desvio padrão similar, que é condição para a aplicação do teste de *Student*.

**Seleção de desenvolvedores:** As hipóteses testadas são H0: $s_t = s_g$ e HA: $s_t <> s_g$, o teste rejeita a hipótese nula se as seleções tiveram uma ou nenhuma intersecções. Os participantes indicaram ao menos dois entre os três desenvolvedores ativos, em cada resposta. Portanto, a hipótese nula não pode ser rejeitada, ou seja, os participantes selecionam desenvolvedores diferentes de acordo com as representações utilizadas.

Nas respostas, o terceiro desenvolvedor escolhido é um dos cinco desenvolvedores mais ativos. Portanto, mesmo havendo essa discordância na resposta, notou-se uma intenção em escolher sempre um dos desenvolvedores mais ativos. A variação na escolha do terceiro desenvolvedor pode ser explicada, em parte, por uma característica da representação gráfica. Notou-se que quando um grupo de ações de *commit* ocorre em um intervalo de tempo curto (menos de uma hora), a representação gráfica comprime todo o grupo e desenha apenas uma única ação de *commit*. Onde o participante da representação textual percebe diversas ações de *commit* em um espaço de poucos minutos, o participante da gráfica conta apenas uma ação de *commit*.

**Tempo transcorrido:** As hipóteses são H0: $t_t = t_g$ e HA: $t_t > t_g$, $\alpha = 0,05$, o teste rejeita a hipótese nula se $t > 2.132$, onde 2.132 é o valor constante na tabela de valores de t para quatro graus de liberdade. Caso contrário, a hipótese nula é aceita ou reserva-se a opção de encontrar uma justificativa. Os dados considerados para o cálculo de t são os tamanhos da amostra: $n_t = n_g = 3$, os tempos médios (x) e os desvios padrão (s) transcorridos na gravação de tela: $x_t = 43$ minutos, $s_t = 7$ minutos, $x_g = 34$ minutos, $s_g = 3$ minutos, o que leva ao valor de $t = 1.854$. Esse valor de t não é suficiente para rejeitar a hipótese nula ($t < 2.132$). Deste modo, com um grau de significância de 0,05, pode-se concluir que não há uma evidência real de que as representações gráficas reduzem o tempo para completar a tarefa, usando-se para isso o teste de *Student* (t), para amostras independentes.

A medição dos tempos é objetiva, porém, pode ser alterada em função de outras variáveis. Por exemplo, o tempo que o participante utiliza para completar tarefas não relacionadas com as representações testadas pode ser usado para diferenciar participantes naturalmente mais rápidos dos que são mais lentos. Na avaliação do uso do ambiente, os participantes que foram alocados para a representação gráfica apresentaram tempos maiores para completar as tarefas iniciais. Em uma tarefa relacionada sem o uso das representações testadas, os participantes associados com a representação gráfica sofrem uma desvantagem significativa no tempo de uso do navegador de classes (questão 2, segundo questionário). Mesmo com essa desvantagem aparente, o grupo que utilizou a representação gráfica alcançou uma média similar à alcançada pelo outro grupo. Em um estudo posterior, pode-se avaliar a proporção entre o tempo usado nas tarefas que não são influenciadas pelo tratamento para ponderar os tempos das tarefas que são influenciadas pelo tratamento. Contudo, não foi possível encontrar valores adequados para determinar uma ponderação, com base nos dados obtidos nesta avaliação.

**Satisfação do usuário:** As hipóteses são H0: $u_r = u_g$ e HA: $u_r > u_g$. Com sete questões (Tabela 7-1), em uma escala *Likert* com cinco valores, de 1 (pior) até 5 (melhor), a usabilidade avaliada foi maior para a representação gráfica.

**Tabela 7-1 Questões sobre a experiência de utilização dos componentes.**

| | | |
|---|---|---|
| 22 | Em geral, considero a minha experiência satisfatória. | 1 2 3 4 5 NA |
| 23 | Os objetivos da tarefa foram atingidos. | 1 2 3 4 5 NA |
| 24 | Conseguiu efetivamente realizar minha tarefa. | 1 2 3 4 5 NA |
| 25 | Estou satisfeito com a lista dos participantes da equipe. | 1 2 3 4 5 NA |
| 26 | Estou satisfeito com as escolhas que realizei na seleção de colaboradores em potencial. | 1 2 3 4 5 NA |
| 27 | Estou totalmente satisfeito com a tarefa. | 1 2 3 4 5 NA |
| 28 | Eu usaria a ferramenta para realização de seleção de colaboradores na prática. | 1 2 3 4 5 NA |

Estas notas são subjetivas, do ponto de vista do usuário. A representação textual (RH) obteve nota 4 com mínimo de 3 pontos. A representação gráfica (GAW) obteve nota 4,5, com mínimo de 4 pontos (Figura 7.2).

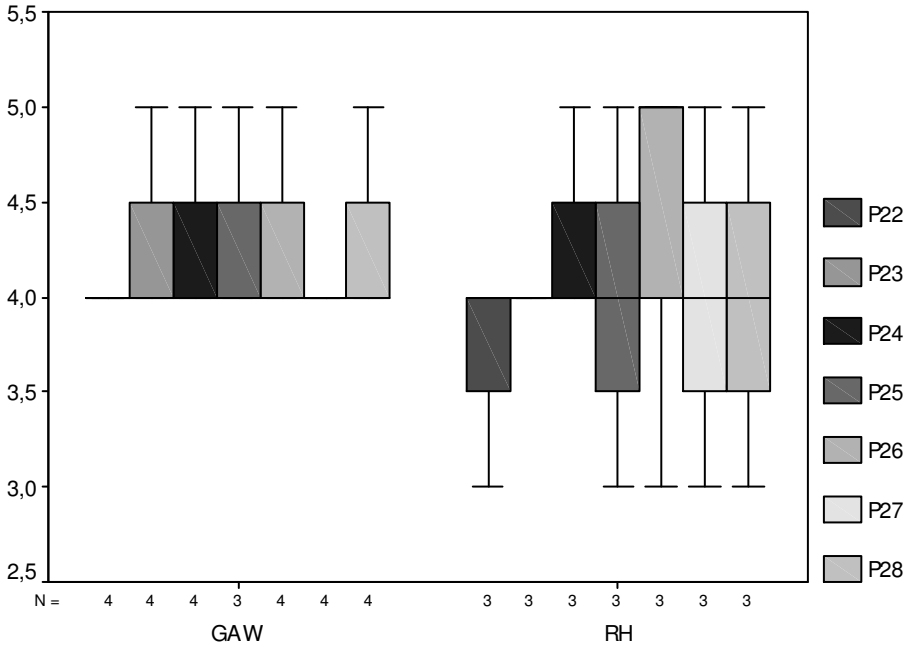



**Figura 7.2 Avaliação do GAW e *Resource History* nas questões de usabilidade.**

## 7.4.6 Observações adicionais

A utilização da gravação de tela durante o estudo permite observações adicionais. A maior parte do tempo usada pelos participantes está relacionada com a seleção e comparação de valores de diferentes arquivos. Dado que a representação mostra informações de um único arquivo, os usuários devem selecionar múltiplos arquivos em seqüência e lembrar das diferenças entre eles. Por exemplo, ao comparar a data de criação, alguns participantes realizaram anotações sobre a data mais antiga e o arquivo encontrado até o momento. Claramente, as representações estariam melhor adaptadas para a tarefa se permitissem exibir múltiplos arquivos simultaneamente.

Duas práticas foram encontradas, em ambos os tratamentos. Primeiro, os participantes usaram resultados encontrados em uma questão para reavaliar e corrigir questões anteriores. Segundo, os participantes registravam uma resposta preliminar e então retornavam para a representação para conferir a resposta. A partir dessas práticas, observa-se que os participantes trataram as tarefas seriamente, como se sua competência pessoal estivesse sendo avaliada. Em um uso prático, podemos esperar que as respostas sejam dadas em um tempo mais curto.

Alguns usuários gostariam de ter os dados extraídos para uma planilha ou

programa estatístico, de modo que eles pudessem analisar conjuntos maiores de dados ou outros gráficos. Os participantes mostraram interesse em representações adicionais das informações.

Ao menos um usuário gastou bastante tempo verificando os dados e detalhes de cada ação de *commit* para ter certeza que estava interpretando a representação gráfica corretamente. Após algum tempo, o usuário estava confiante sobre sua interpretação e abandonou esse comportamento. Uma melhor indicação da representação gráfica correspondente e os dados subjacentes poderiam tornar essa interpretação mais fácil e dar mais confiança ao usuário novato.

As análises do vídeo combinadas com os comentários dos participantes levaram à implementação de seis melhorias na representação gráfica:

a) seleção múltipla;

b) seleção de arquivos recursiva;

c) adoção de uma interface com ação de arrastar-e-soltar (*drag-and-drop*);

d) inclusão de uma indicação explícita do eixo de tempo;

e) apresentação de detalhes do evento (*tooltip*);

f) exportação de dados em formato de texto.

Teoricamente, essas melhorias significam uma redução do tempo necessário para realizar a tarefa, posto que algumas das dificuldades observadas nos usuários foram minimizadas com as alterações realizadas. Entretanto, um novo estudo seria necessário para testar essa nova hipótese.

## 7.5 Conclusão

Este capítulo apresentou avaliações sobre três dos protótipos desenvolvidos. A primeira tarefa é a mais interativa, realizada por duplas de participantes. As duas tarefas seguintes são realizadas por apenas um participante que interage com um artefato de software que é resultado do trabalho do grupo. As tarefas estão relacionadas com a compreensão da evolução de artefatos, que é uma habilidade necessária para a prática das técnicas de reutilização. A reutilização, ou seja, a tarefa de localizar, compreender a adaptar o que foi feito por outros, intensifica a necessidade de colaboração entre o produtor e o consumidor do componente ou artefato reutilizável. Nas três tarefas, o

papel do consumidor do artefato é auxiliado pelo apoio à percepção a identificar elementos do histórico do componente que podem ser úteis para sua melhor compreensão ou para a tomada de decisão em um contexto de tarefa.

A generalização dos resultados dessas avaliações é muito restrita devido a diversos fatores. Primeiro, existem as diferenças entre os participantes. Durante a operação do sistema, ocorreram casos em que uma funcionalidade foi bem avaliada por um participante e mal avaliada por outro. Claramente, existem inúmeras variáveis desconhecidas relacionadas com a caracterização dos participantes. Segundo, não existem dados sobre a população estudada. É quase impossível realizar estimativas estatísticas sem que se tenham parâmetros básicos para a classificação de indivíduos. Terceiro, o desempenho dos participantes durante a avaliação pode ser decorrente de limitações do sistema, da descrição da tarefa ou do instrumento de coleta. Entretanto, apenas o esforço contínuo em propor e reproduzir estudos pode oferecer respostas para essas lacunas no conhecimento.

Da perspectiva de trabalho colaborativo, os resultados são pouco significativos, em decorrência das amostras reduzidas e da ausência de um modelo de colaboração que explique os resultados encontrados. Os planejamentos dos estudos apresentados é a principal contribuição das avaliações planejadas. A avaliação acrescenta ao componente de percepção a capacidade de ser avaliado em condições específicas. Da perspectiva de desenvolvimento de *groupware*, as sugestões dos participantes, os problemas encontrados durante a execução levaram a proposta e implementação de melhorias nos protótipos.

# 8 Conclusão

Este capítulo conclui esta tese apresentando as contribuições deste trabalho (Seção 8.1), as limitações (Seção 8.2) e perspectivas futuras (Seção 8.3) que fornecem indicações de continuação deste trabalho, que compõem inclusive a lista de interesses do autor nos próximos anos de sua pesquisa.

Espera-se que a arquitetura e os protótipos apresentados sirvam para o desenvolvimento de novas ferramentas e novos estudos de caso e que estes contribuam também para um melhor entendimento do papel da colaboração no apoio ao trabalho, em particular, no desenvolvimento de software.

## *8.1 Contribuições Obtidas*

Um aspecto importante da pesquisa em ambientes colaborativos diz respeito a estudos de caso em contextos concretos de uso contínuo do ambiente. Entretanto, para que tais observações sejam possíveis é necessário que esses ambientes sejam desenvolvidos e, sobretudo, utilizados na prática. Diversos fatores concorrem para a falta de utilização desses ambientes colaborativos adequados ao trabalho colaborativo de equipes de desenvolvimento de software. Entre eles, está a falta de uma cultura que valorize o potencial da colaboração em uma área onde o trabalho que gera modelos e implementações é, aparentemente, realizado por indivíduos isolados que interagem com suas ferramentas CASE, transformando uma representação de software em uma nova representação, até que o produto planejado passa a existir. De fato, os ambientes de desenvolvimento de software tradicionais procuram oferecer para cada usuário a idéia de que os objetos sendo manipulados são alterados apenas pelas ações imediatas e locais que estão sendo realizadas.

No nosso entendimento, essa sensação de isolamento é falsa, exceto em projetos muito reduzidos e de curta duração. Em projetos da indústria, o que se observa é que os desenvolvedores estão continuamente estabelecendo comunicação entre si, em

pequenos grupos que se concentram em uma determinada tarefa. Para confirmar essa afirmação basta observar qualquer sala onde o desenvolvimento de software esteja ocorrendo. Existem diversas incertezas e mudanças que obrigam o desenvolvedor a consultar a opinião de colegas mais experientes, mais bem informados ou que, simplesmente, possam oferecer novos dados sobre o problema em questão. Em diversas situações, as dúvidas se referem aos sistemas legados, mantidos ou em desenvolvimento, e as rotinas, classes ou componentes de software que estão sendo continuamente atualizados e adaptados. Dessa forma, nota-se que o trabalho individual gera influência no trabalho de outros, mesmo que isso não seja percebido ou valorizado pelos participantes. O apoio à percepção, construído com a abordagem apresentada, torna essa influência evidente na forma de informação de percepção que pode ser utilizada pelos desenvolvedores para coordenar melhor suas atividades e localizar fontes de informação.

A alteração da cultura se dá pela conscientização e educação dos desenvolvedores para o potencial de agir como um grupo, em que cada membro está ciente das ações dos demais. Mas de uma maneira pragmática, existe a necessidade de demonstrar essas possibilidades e de verificar as condições em que essas vantagens podem vir a surgir. Disto resulta a necessidade de construir uma diversidade de propostas de ambientes colaborativos e de colocá-los em prática. Essa necessidade se intensifica a medida que novas práticas de desenvolvimento, como o desenvolvimento globalizado e os processos ágeis distribuídos, passam a exigir a interação entre indivíduos e equipes que tem como meio principal de interação e de trabalho o computador.

A proposta desta abordagem para a construção de ambientes de desenvolvimento de software serve de guia para a construção de ambientes colaborativo dentro da perspectiva de transparência. Diversas decisões de projeto foram incorporadas à ACT, cuja divisão em componentes de software interligados favorece a implementação e a reutilização desses componentes. Com isso, foi atingido o objetivo de facilitar a programação e reutilização de mecanismos de apoio à percepção e à colaboração. Com essa abordagem, foi possível incentivar alunos a desenvolver protótipos que foram integrados com sucesso nos ambientes *Odyssey* e *Eclipse*.

Existem diversos tipos de documentos e repositórios de informação que são

utilizados no desenvolvimento de software. A abordagem proposta se concentra naqueles documentos estruturados que tem relação com o "espaço de software", o ambiente artificial que é modificado pelos desenvolvedores de software. Em situações de distribuição, como é o caso do desenvolvimento de software globalizado, a percepção de modificações é importante para a coordenação de elementos dessas equipes distribuídas. Diversos ambientes colaborativos com apoio à percepção foram revisados nesta tese e foi identificado então um domínio de aplicação com alto potencial para reutilização de software. A abordagem proposta oferece uma alternativa para construção de ambientes nesse domínio estudado.

O ambiente *Odyssey*, desenvolvido no contexto do projeto *Odyssey*, foi estendido para comportar as modificações geradas pelos protótipos desenvolvidos, dando então origem ao apoio à percepção do *OdysseyShare*. Inicialmente, foi desenvolvido apoio para a integração visual de áreas de trabalho, implementando o apoio para mecanismos de percepção de área de trabalho. Em seguida, foi realizada a inclusão de funcionalidades de troca de mensagens (bate-papo) e de indicadores de estados dos participantes. Essas primeiras alterações constituíram o núcleo do *OdysseyShare*, que foi posteriormente complementado pela proposta de máquina de processos e do repositório de componentes. Em uma segunda etapa, procurou-se investigar informações relacionadas com as operações sobre os artefatos. Foram então desenvolvidos os protótipos *Ariane*, MAIS e GAW que implementam o apoio para mecanismos de percepção de produto e percepção de grupo.

Foram produzidos como contribuições desta pesquisa:

- A definição de uma abordagem para a integração de apoio para a colaboração em cenários reais de desenvolvimento de software, envolvendo pequenas equipes (2 a 6 pessoas) que trabalhem em tempo integral ou parcial dentro de condições de distribuição geográfica ou temporal. A estratégia visa orientar as equipes na construção e seleção dos mecanismos de colaboração e na operação do apoio;
- A proposta de uma metodologia para planejamento e observação de cenários e estudos de caso que ofereceram dados mais concretos sobre os efeitos do apoio à colaboração sobre o trabalho dos participantes;
- A implementação de uma arquitetura transparente que facilite a integração e a

observação dos efeitos do apoio à colaboração nesses ambientes, com base em um conjunto de técnicas adaptadas ou inéditas para a coleta e injeção dos eventos necessários para obter e apresentar informações pertinentes à colaboração e especificação de interfaces de componentes para cada mecanismo de apoio à colaboração adotado.

- Uma análise do domínio de conhecimento relativo ao desenvolvimento distribuído de software e da família de aplicações propostas para esse domínio, dentro do aspecto de apoio à percepção;
- A construção de protótipos com base na abordagem proposta;
- A realização de observações sobre a usabilidade dos protótipos.

São contribuições da arquitetura ACT para a colaboração transparente: (a) os eventos semânticos, (b) a combinação de mecanismos de apoio à percepção e (c) a possibilidade de reutilização desses mecanismos.

## 8.2 Limitações

A principal limitação da proposta é a sua aplicação a uma família de aplicações restrita. As generalizações foram realizadas com base em um conjunto restrito de sistemas de apoio à percepção. Não foi realizada uma revisão sistemática ou exaustiva dos sistemas de apoio à percepção que permita afirmar a predominância do tipo de apoio à percepção. Entretanto, dentro da literatura estudada, a abordagem demonstra-se viável na criação de sistemas que apresentam apoio à percepção similar ao que foi observado nas propostas da literatura.

Grande parte do sucesso na aplicação da abordagem se deve ao comprometimento da equipe de reutilização de software. Em todos os protótipos, ocorreram situações não previstas pela abordagem e componentes existentes tiveram que ser adaptados para atender a novas demandas. Tudo isso ocorreu por causa do esforço contínuo em desenvolver para a reutilização. Sem esse esforço adicional, possivelmente, cada protótipo poderia ter se tornado um sistema independente. Esse esforço adicional é característico do desenvolvimento de componentes.

Não ocorreu uma avaliação sistemática do processo de desenvolvimento de

software com o uso da abordagem proposta. O desenvolvimento dos protótipos se estendeu por longos períodos de tempo. Os desenvolvedores foram acompanhados semanalmente, mas não houve um registro preciso das atividades. Embora seja uma limitação da observação realizada, é necessário ressaltar que não foi localizado um outro processo similar que apresente um registro conforme se gostaria de ter realizado nesta tese.

## 8.3 Perspectivas Futuras

O esforço realizado no contexto desta tese no sentido de viabilizar a abordagem para o desenvolvimento de ambientes colaborativos de desenvolvimento de software resultou em contribuições que permitiram uma análise do domínio e dos desafios do desenvolvimento dessas aplicações, a realização de protótipos e estudos de caso.

As perspectivas futuras podem ser divididas em três áreas: maiores facilidades para o uso da abordagem, ampliação da família de aplicações atendida pela abordagem e incentivo e observação da colaboração nos ambientes modificados.

Na primeira área, destaca-se a necessidade do desenvolvimento ou aplicação de ferramentas que auxiliem na composição de ambientes com base em componentes. A composição de ambientes colaborativos com base em extensões, aplicações e sensores é uma tarefa realizada sem muito apoio automático. As interfaces dos componentes poderiam ser verificadas para testar a conectividade e as dependências. Neste sentido, serão realizadas tentativas de criação de componentes visuais que simplifiquem a ligação entre componentes e a configuração de ambientes a partir de uma interface de usuário intuitiva. Outra possibilidade seria investigar linguagens e sistemas de descrição de arquiteturas de software. Existe, portanto, um potencial para meta-programação que não foi explorado. Outra iniciativa nessa área seria organizar uma biblioteca de componentes disponíveis para a composição de ambientes. Para tanto, seria ideal recomendar a utilização de componentes com base em características do ambiente de colaboração. Em especial, seria importante investigar um modelo de domínio de aplicação que consolide os principais conceitos da área de percepção e que apresente rastros para os componentes existentes.

Na área de ampliação da família de aplicações, destaca-se a necessidade de

permitir que um número maior de tipos de aplicações possa ser desenvolvido dentro da abordagem proposta. Em especial, a injeção de eventos não foi realizada na implementação atual. Essa falta restringe o tipo de extensão que pode ser desenvolvido. Seria importante verificar os limites da injeção de eventos nas aplicações integradas, investigando interfaces de extensibilidade e outras técnicas que permitam a reprodução de eventos em um ambiente local, rumo a possibilidade de editores colaborativos e da realização de captura e reprodução de seqüências de comandos.

A terceira área é composta de ações para incentivar o uso prático do apoio à percepção no desenvolvimento de software. Nesta área, exigiria ações tanto para observar o uso prático quanto para argumentar sobre as vantagens do uso prático. Parece importante planejar e realizar observações sistemáticas, por parte de terceiros, de equipes de desenvolvimento de software usando a abordagem em condições de uso contínuo e em cenários de distribuição. Do ponto de vista prático, seria também necessário definir estratégias de distribuição de informações e para remoção de informações da área de memória compartilhada, uma vez que o seu uso contínuo pode tornar informações desatualizadas ou cujo volume excessivo reduza o desempenho do sistema. Outra forma de incentivar o uso seria explorar relações mais significativas entre os eventos, com o objetivo de alertar o usuário de situações de conflito ou de interesse. Por exemplo, seria importante realizar avaliações do uso dos componentes de percepção e sua associação com métricas e informações da gerência de projeto.

Além disso, as observações realizadas durante o uso dos protótipos indicam que existe um grande potencial para o refinamento de cada um dos mecanismos de apoio a percepção investigados. Por exemplo, depende da reação do uso prático a proposta de formas de representação gráfica para a informação de percepção que sejam integradas com os diagramas e visualizadores da ferramenta. Aqui as possibilidades se dividem em duas. Primeiro, o uso de técnicas para interferir na visualização de diagramas tais como, a sobreposição de imagens usada nos teleapontadores. Segundo, a notação expandida que irá conter símbolos para representar alterações no produto e outras informações de percepção.

As três áreas de perspectivas futuras apresentam potencial. A primeira área está relacionada com pesquisa em reutilização e técnicas de desenvolvimento com base em componentes, portanto, ainda é um tema relevante e as aplicações desenvolvidas na

ACT podem servir como um estudo de caso para essas técnicas. A ampliação da família de aplicações apresenta desafios técnicos, mas também está relacionada com limites da reutilização e da capacidade de delimitação de domínios e famílias de aplicação. A terceira área é relevante tanto para a Engenharia de Software quanto para o Trabalho Colaborativo. Para a Engenharia de Software, pela possibilidade de que o apoio à percepção influencie na qualidade do processo e do produto de software. Do ponto de vista do Trabalho Colaborativo está o interesse em testar e evoluir diferentes formas de apoio à colaboração e observar seu uso prático.

# Referências Bibliográficas


ADAMS, D. J., 2002, *Programming Jabber*, Boston, O'Reilly, 480p.

ALTMANN, J., POMBERGER, G., 1999, "Cooperative Software Development: Concepts, Models, and Tools", In: *Proceedings of the Technology of Object-Oriented Languages and Systems*, Santa Barbara, USA, Aug, pp 194-207.

ANDERSON, K. M., TAYLOR, R. N., WHITEHEAD JR, J. E., 2000, "Chimera: Hypertext for Heterogeneous Software Development Environments", *ACM Transactions on Information Systems*, v.18, n.3 (July), pp. 211-245.

ANT, 2005, Apache Ant CVS Repositories, http://ant.apache.org/cvs.html.

APPELT, W, 1999, "WWW Based Collaboration with the BSCW System", In: Proceedings of the 26th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM), *Springer Lecture Notes in Computer Science*, 1725, pp.66-78; Dec, Milovy.

ARAÚJO, R., DIAS, M., BORGES, M. R. S., 1997, "A Framework for the Classification of Computer Supported Collaborative Design Approaches", In: *Proceedings of the III CYTED-RITOS International Workshop on Groupware (CRIWG)*, Madrid, Spain, Oct., pp. 91-100.

ARAÚJO, R. M., 2000 (cap 3), "Groupware" e (cap 5), "Percepção". In: *Ampliando a Cultura de Processos de Software - Um Enfoque Baseado em Groupware e Workflow*, Tese de DSc., COPPE/UFRJ, Rio de Janeiro, Brasil, 240 p.

ATKINSON, C. *et al.*, 2001, *Component-based Product-line Engineering with UML*, Addison-Wesley, 528p.

AUGIER, M., VENDELO, M. T., 1999, "Networks, Cognition and Management of Tacit Knowledge", *Journal of Knowledge Management*, v.3, n.4 (June), pp. 252-261.

BARTHELMESS, P., ANDERSON, K. M., 2002, "A View of Software Development Environments Based on Activity Theory". In: *Computer Supported Cooperative Work, v.*11, n.1-2, pp. 13-37.

BARTHELMESS, P., ELLIS, C. A., 2002b, The Neem Platform: An Extensible Framework for the Development of Perceptual Collaborative Applications. In: *Proceedings of the EDCIS 2002*, Lecture Notes in Computer Science, v. 2480, Jan, pp. 547-562.

BARTHELMESS, P., 2003, "Collaboration and coordination in process-centered software development environments: a review of the literature", *Information & Software Technology*, v.45, n.13, pp. 911-928.

BASILI, V., *et al.*, 1986, "Experimentation in Software Engineering". *IEEE Transactions on Software Engineering*, v.12, n.7(Jul).

BATTIN, J. R., CROCKER, R., KREIDLER, J., SUBRAMANIAN, K., 2001, "Leveraging Resources in Global Software Development", *IEEE Software*, v.18, n.2 (March/April), pp. 70-77.

BECK, K., 1999, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 190 p.

BECKER, K.; BACELO, A. P., 2001, "Considerations on the application of object-oriented reuse technology to the CSCL". In: *Proceedings of the 7th International Workshop on Groupware (CRIWG'2001)*, Darmstadt, IEEE Press. v. 1. p. 164-169.

BEGOLE, J., 1998, *Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application-Sharing Systems*, Virginia Politechnic Institute, USA, Tese de DSc, 182 p., http://scholar.lib.vt.edu/theses/available/etd-011999-152714/.

BEGOLE, J., ROSSON, M. B., SHAFFER, C. A., 1999, "Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application-Sharing Systems", *ACM Transactions on Computer-Human Interaction*, v.6, n.2 (Jun), pp. 95-132.

BEGOLE, J., SMITH, R. B., STRUBLE, C. A., SHAFFER, C. A., 2001, "Resource Sharing for Replicated Synchronous Groupware", *Transactions on Networking*, v.9, n.6 (Dec), pp. 833 - 843.

BEGOLE, J., TANG, J. C., SMITH, R. B., YANKELOVICH, N, Y., 2002, "Work rhythms: analyzing visualizations of awareness histories of distributed groups", In: *Proceedings of the Conference on Computer Supported Cooperative Work*, New Orleans, USA, ACM Press, pp.334-343.

BOEHM, B., GRÜNBACHER, P., BRIGGS, B., 2001, "Developing Groupware for Requirements Negotiation: Lessons Learned", *IEEE Software*, v.18, n.3 (May), pp. 46-55.

BORGES, M. R. S., PINO, J. A., 1999, "Awareness Mechanisms for Coordination in Asynchronous CSCW", In: *Proceedings of the 9th Workshop on Information Technologies and Systems (WITS '99)*, Charlotte, Dec., pp. 69-74.

BORGHOFF, U. M., SCHLICHTER, J. H., 2000, *Computer Supported Cooperative Work: Introduction to Distributed Applications*, Springer Verlag, 529p.

BOSH, J., 2000, *Design and Use of Software Architectures*, Addison-Wesley, 368 p.

BOWEN, M., MAURER, F., 2002, "Designing a Distributed Software Development Support System Using a Peer-to-Peer Architecture", In: *Proceedings of the Workshop on Cooperative Supports for Distributed Software Engineering Processes*, 6 p.

BRAGA, R. M. M., 2001, *Busca e Recuperação de Componentes em um Ambiente de Reuso*, Tese de D.Sc., COPPE/UFRJ.

BRIDGMAN, P. W., 1955, "On Scientific Method". In: *Reflections of a physicist*. New York: Philosophical Library, 392 p.

BROWN, A., 2000, *Large-Scale Component-Based Development*, Prentice Hall, 512p.

BUXMANN, P., KÖNIG, W., 2000, *Inter-Organizational Cooperation with Sap Systems: Perspectives on Logistics and Service Management*, Springer Verlag, 183p.

CARRIERO, N., GELERNTER, D., 2001, "A computational model of everything", *Communications of the ACM*, v.44, n.11(Nov.), pp. 77-81.

CHESSMAN, J., DANIELS, J., 2001, *UML Components: a Simple Process for Specifying Component-Based Software*, Addison-Wesley, 208p.

CHEN, D., SUN C., 1999, “A distributed algorithm for graphic objects replication in real-time group editors”. In: *Proc. of the International ACM SIGGROUP Conference on Supporting Group Work*, pp. 121-130, Nov.

CHENG , L., HUPFER, S., ROSS, S., PATTERSON, J., CLARK , B., DE SOUZA, C. R. B., 2003, “Jazz: a collaborative application development environment”. In: *OOPSLA´03 Companion*, p. 102-103.

CHU-CARROL, M. C., WRIGHT, J., SHIELDS, S., 2002, “Supporting Aggregation in Fine Grained software Configuration Management”, In: *Software Engineering Notes*, San Diego, USA.

CLAYPOOL, M., LE, P., WASEDA, M., BROWN, D., 2001, “Implicit Interest Indicators”, In: *Proceedings of ACM Intelligent User Interfaces Conference*, Jan, 8 p., http://www.cs.wpi.edu/~claypool/papers/iii/

COCKBURN, A., 2001, *Agile Software Development*, Addison-Wesley, 256 p.

COLLABNET, 2003, “SourceCast”, (http://www.collab.com).

CONSTANTINE, L., 1993, “Work Organization: Paradigms for Project Management and Organization”, *Communications of the ACM*, v. 36, n. 10, pp. 34-43.

COUTAZ, J., CROWLEY, J. L., DOBSON, S., GARLAN, D., 2005, “Context is key”, *Communications of the ACM*, v. 48, n. 3., pp. 49-53.

CUNNINGHAM, W., 2003, Portland Patterns Repository Wiki, (http://c2.com/cgi/wiki?WelcomeVisitors).

CUNNINGHAM, W., LEUF, B., 2001, *The Wiki Way. Quick Collaboration on the Web*. Addison-Wesley, 440 p.

DAVID, J. M. N., BORGES, M.R.S., 2001, “Selectivity of Awareness Components in Asynchronous CSCW Environments”, In: *Proceedings of the Seventh International Workshop on Groupware* (CRIWG'01), Darmstadt, Germany, IEEE Computer Society, pp. 115-124.

DEFRANCO-TOMMARELLO, J. F., DEEK, J., 2002, “Collaborative Software Development: A Discussion of Problem Solving Models and Groupware Technologies”, In: *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii, January, p. 41, http://doi.ieeecomputersociety.org/10.1109/HICSS.2002.993937.

DEMARCO, T., LISTER, T., 1998, “Human Capital”. *IEEE Software*, v. 15, n. 6, pp. 103-105.

DEWAN, P, RLEDL, J, 1993, “Toward computer supported concurrent software engineering”. *IEEE Computer*, Jan.

DIAS, M. S., 1998, *Um ambiente de suporte ao projeto cooperativo de software*. 1998. Tese de MSc., Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro.

DIAS, M. S., BORGES, M. R. S., 1999, "Development of Groupware Systems with the COPSE infrastructure", In: *Proceedings of International Workshop on Groupware*, Cancun, Mexico, Sep.

DOURISH, P., BELLOTTI, V., 1992, "Awareness and Coordination in Shared Workspaces", In: *Proceedings of the Conference on Computer Supported Cooperative Work*, Toronto, Canada, Nov.

DOURISH, P., 2001, *Where the action is*: the foundations of embodied interactions.

MIT, Cambridge.

DRIDI, F., NEUMANN, G., 1999, "How to implement Web-based Groupware Systems based on WebDAV", In: *Proc. of Intl. Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Stanford, USA, June.

D'SOUZA, D., WILLS, A. C., 1998, *Objects Components and Frameworks with UML: the Catalysis Approach*, Addison-Wesley.

DE SOUZA, C., FROEHLICH, J., DOURISH, P. 2005, "Seeking the Source: Software Source Code as a Social and Technical Artifact". In: *Proceedings of the ACM Conference Supporting Group Work GROUP*.

EBERT, C., NEVE, P. DE, 2001, "Surviving Global Software Development", *IEEE Software*, v.18, n.2 (March/April), pp. 62-69.

EICK, S. G., STEFFEN, J. L., SUMNER, E. E., 1992, "Seesoft-A Tool for Visualizing Line Oriented Software Statistics", *IEEE Trans. Software Eng.,* v.18, n.11 (Nov.), pp. 957-968.

ELLIS, C. A., GIBBS, S. J., REIN, G. L., 1991, "Groupware – Some Issues and Experiences", *Communications of the ACM*, v.34, n.1 (Jan.), pp. 39 - 58.

ENGELBART, D., ENGLISH, W., 1968, A Research Center for Augmenting Human Intellect, In: *Proceedings of Fall Joint Computing Conference*, v. 33, pp. 395-410, AFIPS.

ERICKSON, T., KELLOG, A., 2000, "Social Translucence: an Approach to Designing Systems that Support Social Processes", *Transactions on Computer-Human Interaction*, v.7, n.1 (March), pp. 59-83.

ESTUBLIER, J., 2000, "Software Configuration Management: a roadmap", In: *The Future of Software Engineering*, The Future of Software Engineering. A. Finkelstein (ed.), ACM Press, May 2000

ESTUBLIER, J., CÁRDENAS-GARCIA, S., VEGA, G., 2001, "Defining and Supporting Concurrent Engineering Policies in SCM", In: *Proceedings of the Software Configuration Management Conference*, Toronto, Canada, May.

FARSHCHIAN, B. A., 2001, "Integrating Geographically Distributed Development Teams Through Increased Product Awareness". *Information Systems Journal*, pp. 123 – 141, v. 26, n. 3, May.

FAVELA, J., *et al.*, 2004, "Empirical Evaluation of Collaborative Support for Distributed Pair Programming", In: *International Workshop on Groupware*, pp.215-222, Lecture Notes in Computer Science, San Jose, Costa Rica, set.

FELLER, J., FITZGERALD, B., 2000, "A Framework Analysis of the Open Source Software Development Paradigm", In: *Proceedings of the Proceedings of the Twenty-First International Conference on Information Systems*, Atlanta, USA, December.

FIELDING, R. T., WHITEHEAD JR., E. J., ANDERSON, K., BOLCER, G. A. , OREIZY, P., TAYLOR, R. N., 1998, "Web-Based Development of Complex Information Products", *Communications of the ACM*, v.41, n.8 (Aug), pp. 84-92.

FOGEL, K., BAR, M., 2001, *Open Source Development with CVS*, 2nd, Coriolis, 344p.

FREUND, J. E., PERLES, B. M.,1999, *Statistics*: a first course, Prentice Hall, 560 p.

FUKS, H., RAPOSO, A. B., GEROSA, M. A., 2002, "Engenharia de *Groupware*: Desenvolvimento de Aplicações Colaborativas", In: *Anais do Congresso da*

*Sociedade Brasileira de Computação, Jornada de Atualização Científica*, Florianópolis, julho.

GELERNTER, D., 1985, "Generative communication in Linda", *ACM Transactions in Computer Languages*, v.7, n.1 (Jan.), pp. 80-112.

GIBSON, J. J., 1979, *The Ecological Approach to Visual Perception*. Hillsdale, New Jersey: Lawrence Erlbaum.

GIGASPACES, 2004, Gigaspaces Server, http://www.gigaspaces.com.

GEROSA, M. A., FUKS, H., RAPOSO, B. R.,. MITCHELL, L. H. R.G., 2002, "Using Groupware Tools to Extend the Organizational Memory with Collaboration Aspects", In: *Proceedings of the International Conference on CSCW in Design*, Rio de Janeiro, Sep.

GORTON, I., MOTWANI, S., 1996, "Issues in Co-operative Software Engineering Using Globally Distributed Teams", 1996, In: Information and Software Technology Journal, v.38, n.10, pp. 647-655.

GREENBERG, S., 1990, "Sharing Views and Interactions with Single User Applications", In: *Proceedings of ACM/IEE Conference on Office Information Systems*, Cambridge, USA.

GRINTER, R., 2001, "From local to global coordination: lessons from software reuse", In: *Proc. of the 2001 International ACM SIGGROUP Conference on Supporting Group Work*, Boulder, Colorado, USA. pp. 144-153.

GRISS, M.L., FAVARO, J., D'ALESSANDRO, M., 1998, "Integrating feature modelling with the RSEB". In: *Proceedings of Fifth International Conference on Software Reuse - ICSR5*, Victoria, British Columbia, Canada, pp. 76-85.

GROOVE NETWORKS, 2003, Groove, (http://www.groove.net/).

GRUDIN, J., 1994, "Groupware and social dynamics: eight challenges for developers", *Communications of the ACM*, v.37, n.1 (Jan), pp. 92 - 105.

GRUNDY, J. C., HOSKING, J. G., 2002, "Engineering plug-in software components to support collaborative work", *Software - Practice and Experience*, v.32, n.10, pp. 983-1013.

GRUNDY, J. C., MUGRIDGE, W. B., HOSKING, J. G., 1998, "Constructing Component-based Software Engineering Environments: Issues and Experiences", *Information & Software Tecnhology*, v.42, n.2, pp. 103-114.

GUNTHER, H., 2003, Mobility and affordance as the core of Person-Environment Studies. Estud. psicol. (Natal), v.8, n.2(May/Aug), p.273-280.

GUTWIN, C., GREENBERG, S., 1999, "Effects of awareness support on groupware usability". *ACM Transactions on Computer-Human Interaction*, v. 6, n. 3(Sep), pp. 243 – 281.

GUTWIN, C., GREENBERG, S., 1996, "Workspace Awareness for Groupware". In: Proceedings of Computer-Human Interaction Conference, pp. 208-209.

HAYEK, F., 1945, "The Use of Knowledge in Society", *American Economic Review*, v.25, n.4 (Sep), pp. 519-530.

HERBSLEB, J D., MOITRA, D. (eds.), 2001, "Global Software Development", *IEEE Software*, v. 18, n.2(Mar/Apr).

HERBSLEB, J. D., MOCHUS, A., FINHOLT, T., GRINTER, R., 2001, "An Empirical Study of Global Software Development: Distance and Speed", In: *Proceedings of*

*the International Conference on Software Engineering*, Toronto, Canada, May.

HERBSLEB, J. D., MOCHUS, A., 2003, "An empirical Study of Speed and Communication in Globally Distributed Software Development". *IEEE Transactions on Software Engineering*, v.29, n.6, pp.481-494.

HOLZ, H., MAURER, F., 2003, "Knowledge Management Support for Distributed Agile Software Processes", In: *Proc. of the Fourth International Workshop on Learning Software Organizations (LSO 2002)*.

HUDSON, J., CHRISTENSEN, J., KELLOG, W., ERICKSON, T., 2002, "'I'de be Overhelmed: but it's just one more thing to do': Availability and Interruption in Research Management", In: *Proceedings of the Computer-Human Interaction Conference*, Minneapolis, USA.

JAIL, R. K., 1991, *The art of Computer Systems Performance Analysis*, John Wiley.

JARVENPAA, S. L., LEIDNER, D. E., 1998, "Communication and Trust in Global Virtual Teams", *Journal of Computer-Mediated Communication*, v.3, n.4 (June).

KANTOR, M., REDMILES, D., 2001, "Creating an Infrastructure for Ubiquitous Awareness", In: *Proceedings of the Conference on Human Computer Interaction*, Tokyo, Japan, July, pp. 431-438.

KICZALES, G., *et al.*, 2001, "An Overview of AspectJ", In: Proc. of the *European Conference on Object Oriented Programming*, Budapest, Hungary, June.

KIRSH-PINHEIRO, M., LIMA, V. DE, BORGES, M. R. S., 2002, "A Framework for Awareness Support in Groupware Systems", In: *Proceedings of the International Conference on CSCW in Design*, Rio de Janeiro, Sep.

KOTLARSKY, J. M., KUMAR, K., HILLEGERSBERG, J., 2002, "Coordination and collaboration for globally distributed teams: the case of component-based/object-oriented software development", In: *Proceedings of the International Workshop on Global Software Development*, Orlando, USA, May.

KREIJN, K., KIRSCHNER, P. A., 2001, "The social affordances of Computer Supported Cooperative Learning Environments". In: *Proc. of the 31th ASEE/IEEE Frontiers in Education Conference*, Reno, pp. 12-17.

KRUCHTEN, P., 2000, *The Rational Unified Process*: an Introduction. Addison-Wesley. 2 ed.

LANDES, D., SCHNEIDER, K., HOUDEK, F., 1999, "Organizational learning and experience documentation in industrial software projects", *International Journal on Human-Computer Studies*, v.51, pp. 646-661.

LAURILLAU, Y., NIGAY, L., 2002, "Clover Architecture for Groupware", In: *Proceedings of the Conference on Computer Supported Cooperative Work*, New Orleans, November.

LAUWERS, J. C., LANTZ, K. A., 1990, "Collaboration Awareness in Support of Collaboration Transparency", In: *Proceedings of the ACM CHI'90 Conference on Human Factors in Computing Systems*, ACM Press, New York, pp. 303-311.

LI, D., LI, R., 2002, "Transparent Sharing and Interoperation of Heterogeneous Single-User Applications", In: *Proceedings of the CSCW'02*, New Orleans, USA, pp. 246-255.

LIKERT, R., 1932, *A Technique for the Measurement of Attitudes*, New York: McGraw-Hill.

LLOYD, P., WHITEHEAD, R. (eds), 1996, *Transforming Organizations through Groupware*, Springer Verlag, 197p.

LOPES, M.A.M., MANGAN, M. A. S., WERNER, C.M.L., 2004a, "MAIS: uma ferramenta de percepção para apoiar a edição concorrente de modelos de análise e projeto", In: Anais do *XVII Simpósio Brasileiro de Engenharia de Software - XI Sessão de ferramentas*, pp.61-66, Brasília, Brasil, out.

LOPES, M.A.M., MANGAN, M. A. S., WERNER, C.M.L., 2004b, "MAIS: um Mecanismo para Apoio à Percepção aplicado a Modelos de Software Compartilhados", In: Proc. of the LA-Web/WebMidia Joint Conference.

LOPES, M.A.M., 2005, MAIS: Um Mecanismo Multi-síncrono de Percepção aplicado ao Desenvolvimento de Software Baseado em Modelos. Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

LÓPEZ, P. G., SKARMETA, A. F. G., 2003, "ANTS Framework for Cooperative Work Environments", *Computer*, v. 36, n.3 (March), pp. 56-62.

LOTUS, 2002, ScreenCam, www.pcmag.com/article2/0,1759,1096928,00.asp.

MADHAVJI, N. H., 1992, "Environment Evolution: The Prism Model of Changes", *Transactions on Software Engineering*, v.18, n.5 (April), pp. 380-392.

MAIDANTCHICK, C., ROCHA, A.R.C. DA, 2002, "Managing a Worldwide Software Process", In: *Proceedings of the International Workshop on Software Development*, Orlando, USA, May.

MANGAN, M.A.S., 1998, Aspectos de Implementação de um Modelo para Gerência do Processo de Desenvolvimento de Software: Arquitetura e Protocolos para um Gerente de DesignFlow, CPGCC/UFRGS, Dissertação de Msc, 104p..

MANGAN, M. A. S., MURTA, L. G. P., SOUZA, J. M., WERNER, C.M.L., 2001, "Modelos de Domínio e Ontologias: Uma Comparação Através de um Estudo de Caso Prático em Hidrologia", In: Proceedings of the *IV International Symposium on Knowledge Management/Document Management (ISKM/DM'2001)*, pp.149-172, Curitiba, Paraná, Brasil, ago.

MANGAN, M. A. S., WERNER, C. M. L. ; BORGES, M. R. S. ; KALINOWSKI, M. ; ARAÚJO, R., 2002, "Towards the Evaluation of Awareness Information Support Applied to Peer Reviews", In: *Proceedings of the International Conference on Computer Supported Cooperative Work in Design*, Rio de Janeiro, 7, set., pp. 49-54.

MANGAN, M. A. S., WERNER, C.M.L., BORGES, M. R. S., 2002, "Componentes para Colaboração Síncrona em um Ambiente de Reutilização de Software", In: *Anais do Simpósio Brasileiro de Engenharia de Software, Ferramentas*, Gramado, outubro, pp. 373-377.

MANGAN, M. A. S., WERNER, C. M. L., BORGES, M. R. S., 2002, "Especificação de Componentes de Software em Sistemas Colaborativos", In: *Segundo Workshop de Desenvolvimento Baseado em Componentes*, Itaipava, ago.

MANGAN, M. A. S., 2003, Uma Arquitetura Transparente para um Ambiente Colaborativo de Software. Exame de Qualificação, COPPE/UFRJ, Rio de Janeiro, Brasil.

MANGAN, M. A. S., 2003, "Supporting Global Software Development through a Transparent Collaboration Architecture", *9th International Workshop on Groupware*, Autran, France, out.

MANGAN, M. A. S., SILVA, I.A., WERNER, C.M.L., 2004, "GAW: Uma Ferramenta de Percepção de Grupo Aplicada no Desenvolvimento de Software", In: Proc. of the XVIII Brazilian Symposium on Software Engineering, Tools Session, Brasília, DF, Brasil, out.

MANGAN, M. A. S., BORGES, M. R. S., WERNER, C.M.L., 2004, "A Middleware to Increase Awareness in Distributed Software Development Workspaces", In: Proc. of the *WebMedia & LA-Web Joint Conference*, pp. 62-64, Ribeirão Preto, SP, Brasil, out.

MANGAN, M. A. S., BORGES, M. R. S., WERNER, C.M.L., "Increasing Awareness in Distributed Software Development Workspaces", In: *International Workshop on Groupware*, pp.84-91, San Carlos, Costa Rica, setembro 2004.

MARK, G., BORDETSKY, A., 1998, "Structuring Feedback for Groupware Use: Memory Based Awareness", In: *Proc. of the Thirty-First Annual Hawaii International Conference on System Sciences-Volume 1*, 184-193, Jan.

MARQUÈS, J. M., NAVARRO, L., 2001, "WWG: a Wide-Area Infrastructure to Support Groups", In: *Proceedings of GROUP'01*, Boulder, USA, Sep.

MAURER, F., DELLEN, B., BENDECK, F., 2000, "Merging Product Planning and Web-Enabled Dynamic Workflow Technologies", *IEEE Internet Computing*, (May/June).

MAURER, F., MARTEL, S., 2002, "Process Support for Distributed Extreme Programming Teams", In: *Proceedings of the Workshop on Global Software Development*.

MEDVIDOVIC, N., ROSENBLUN, D.S., REDMILES, D.F, ROBBINS, J. E., 2002, "Modeling Software Architectures in UML", *ACM Transactions on Software Engineering and Methodology*, v. 11, n. 1, pp. 2-57. .....

MEIRE, A., BORGES, M.R.S., ARAUJO, R., 2003, Supporting Collaborative Drawing with the Mask Versioning Mechanism. In: *Proceedings of the 9th International Workshop on Groupware*, Lecture Notes in Computer Science, 2806. Springer-Verlag, Berlin Heidelberg New York, pp. 208-223.

MICROSOFT, 2003, Windows XP Messenger, (http://www.microsoft.com).

MILLER, N., 2001, *A Engenharia de Aplicações no Contexto da Reutilização baseada em Modelos de Domínio*, tese de M.Sc., COPPE/UFRJ.

MOCKUS, A., FIELDING, R.T., HERBSLEB, J., 2000, "A Case Study of Open Source Software Development: The Apache Server", In: *Proceedings of the International Conference on Software Engineering*, Limerick, Ireland, June, pp. 263-272.

MOCKUS, A., HERBSLEB, J. D., 2002, "Expertise browser: a quantitative approach to identifying expertise", In: *Proceedings of the 24th International Conference on Software Engineering*, Orlando, USA, pp. 503-512.

MOLLI, P., SKAF-MOLLI, H., OSTER, G., JOURDAIN, S., 2002, "SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments", In: *Proceedings of the International Conference on Computer Supported Cooperative Work in Design*, Rio de Janeiro, Brasil, Sep, pp. 80-84.

MORENO, M., VIVACQUA, A., SOUZA, J. M. DE, 2003, *An Agent Framework to Support Opportunistic Collaboration*, Relatório Técnico ES 599, PESC, UFRJ.

MOORE, J. M., BAILIN, S. C., 1991, "Domain Analysis: Framework for reuse", *IEEE*

*Computer Society Press Tutorial*, pp. 179-202.

MURTA, L.G.P., 2002, Charon: Uma Máquina de Processos Extensível baseada em Agentes Inteligentes, Dissertação de M.Sc., COPPE, UFRJ.

MURTA, L. G. P., VASCONCELOS, A. P. V., BLOIS, A.P.T.B., LOPES, M.A.M., MELO JR, C. R. S., MANGAN, M. A. S., WERNER, C.M.L., 2004, "Run-time Variability through Component Dynamic Loading", In: *XVIII Brazilian Symposium on Software Engineering , Tools Session*, Brasília, DF, Brazil, outubro.

OCHOA, S. F., GUERRERO, L. A., PINO, J. A., COLLAZOS, C. A., 2004, Reusing Groupware Applications, In: *Proc. of the 10th International Workshop (CRIWG)*, San Carlos, Costa Rica, pp. 262-270, Sept..

OLIVEIRA, R. F., BLOIS, A.P.T.B., VASCONCELOS, A. P. V., WERNER, C.M.L., 2005, "Representação de Variabilidades em Componentes de Negócio no Contexto da Engenharia de Domínio", In: *Proc. of the V Workshop em Desenvolvimento Baseado em Componentes*, pp.73-80, Juiz de Fora, MG, Brasil, nov.

OMG – OBJECT MANAGEMENT GROUP, 2003, "OMG Unified Modeling Language Specification", http://www.omg.org/technology/documents/vault.htm. Acesso em janeiro de 2003.

PERLMAN, C. (2002). "An Introduction to Performance Assessment Scoring Rubrics". In: C. Boston (Eds.), *Understanding Scoring Rubrics*, pp. 5-13. University of Maryland, MD: ERIC Clearinghouse on Assessment and Evaluation.

PERRY, M., AGARWALD, D., MCPARLAND, C., 2002, "Use Scenarios for Collaborative Editing in Scientific Collaborations", In: *Proc. of the 4th International Workshop on Collaborative Editing, CSCW 2002 Conference*, New Orleans, Louisiana, USA, Nov.

PIMENTEL, M. G., FUKS, H. LUCENA, C. J. P. DE, 2003, "Co-text Loss in Textual Chat Tools", In: *Proceedings of the CONTEXT'2003,* pp. 483-490.

PRESSMAN, R., 2002, *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 880 p.

PRIKLADNICKI, R., AUDY, J., EVARISTO, J.R., 2003, "Requirement Management in Global Software Development: Preliminary Findings from a Case Study in a SW-CMM Context", In: *Proceedings of Global Software Development*, Portland, USA, May.

REICH, 1992, *The Work of Nations*, Vintage, 352 p.

REPENNING, A., IOANNIDOU, A., PAYTON, M. YE, W, ROSCHELLE, J., 2001, "Using Components for Rapid Distributed Software Development", *IEEE Software*, v.18, n.2 (March/April), pp. 38-46.

RATIONAL, 2003, "Rational® XDE™ Professional v2002: Java™ Platform Edition". (http://www.rational.com/products/xde/javaed/index.jsp).

RISING, L., JANOFF, N. S., 2000, "The Scrum Software Development Process for Small Teams", *IEEE Software*, v.17, n.4 (July), pp. 11-13.

ROBSON, C., 2002, *Real World Research*, 2nd ed., Blackwell Publishing.

RODDEN, T. R., 1996, "Populating the Application: A Model of Awareness for Cooperative Applications", In: *Proceedings of the Conference on Computer-Supported Cooperative Work*, Boston, USA, pp. 87-96.

RUSSEL, C., 2002, *Java Data Objects (JDO) Specification*, http://jcp.org/aboutJava/communityprocess/final/jsr012.

SAYÃO, M., LEITE, J. C. S. P., 2005, "Uso de agentes no processo de requisitos em ambientes distribuídos de desenvolvimento". In: *Proc of the Workshop de Engenharia de Requisitos*. http://wer.inf.puc-rio.br/WERpapers/

SAMPAIO, A., VASCONCELLOS, A., SAMPAIO, P. F., 2004, "Design and Empirical Evaluation of an Agile Web Engineering Process", In: *Anais do Simpósio Brasileiro de Engenharia de Software*, Brasília.

SANTANCHÉ, A., 2003, "RDF: Fundamentos para a Web Semântica". In: *Anais do Simpósio Brasileiro de Web e Mídia*, apostila de curso.

SARMA, A., NOROOZI, Z., VAN DER HOEK, A., 2003, "Palantír: Raising Awareness among Configuration Management Workspaces", In: *Proceedings of the International Conference on Software Engineering*, Portland, USA, May.

SARMA, A., VAN DER HOECK, A., 2002, "Palantír: Increasing Awareness in Distributed Software Development", In: *Workshop on Global Software Development*, Orlando, Florida, May.

SCHLICHTER, KOCH, BÜRGER, 1997, "Workspace Awareness for Distributed Teams", In: *Proc. Workshop Coordination Technology for Collaborative Applications*, Singapore.

SCHUCKMANN, C., KIRCHNER, L., SCHÜMMER, J., HAAKE, J., 1996, "Designing object-oriented synchronous groupware with COAST", In: *Proceedings of the 1996 ACM conference on Computer Supported Cooperative Work*, Boston, Massachusetts, USA.

SCHÜMMER, T., HAAKE, J. M., 2001, "Supporting Distributed Software Development by Modes of Collaboration", In: *Proceedings of the European Conference on Computer Supported Cooperative Work*, Bonn.

SCHUMMER, T., 2001, "Lost and Found in Software Space", In: *Proc. of the 34th Annual Hawaii International Conference on System Sciences*, IEEE, Maui, Hawaii.

SHAW, M., GARLAN, D., 1996, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 242p.

SHULL, F., BASILI, V., CARVER, J. *et al.*, 2002, "Replicating Software Engineering Experiments: Addressing the Tacit Knowledge Problem", In: *Proceedings of International Symposium on Empirical Software Engineering (ISESE'02)*, Nara, Japan.

SILVA, I.A., MANGAN, M. A. S., WERNER, C.M.L., 2004, "CVS-Watch: A Group Awareness Tool Applied to Collaborative Software Development", In: *La WEB/WebMidia 2004*, Ribeirão Preto, SP, Brasil, out.

SILVA, I. A., 2005, *GAW: Um mecanismo visual de percepção de grupo aplicado ao desenvolvimento distribuído de software*, IM/UFRJ, Rio de Janeiro, Brasil, Projeto Final.

SOUZA, R. *et al.*, 2001, "Software Components Reuse through Web Search and Retrieval", In: *International Workshop on Information Integration on the Web Technologies and Applications*, abril.

STEINFIELD, C., JANG, Y., PFAFF, B., 1999, "Supporting Virtual Team Collaboration: The TeamSCOPE System", In: *Proceedings of GROUP Conference*,

Stockholm, Sweden, November.

SUN, 2003, Project JXTA, http://www.jxta.org/.

SUN, 2003b: SUN MICROSYSTEMS, JavaTM Virtual Machine Profiler Interface, http://java.sun.com/products/jdk/1.2/docs/guide/jvmpi/index.html.

SUN, C., ELLIS, C., 1998, "Operational Transformation in real-time group editors: issues, algorithms, and achievements", In: *Proceedings of the CSCW*, Seattle, USA, December.

SVEIBY, K. E., 1999, "Tacit Knowledge", 1999, In: *The Knowledge Management Yearbook 1999-2000*.

TAYLOR, P., ANDERSON, T.J., NICHOLL, P., 2000, "Information Required for the Successful Geographically Distributed Software Engineering of Components and Component-based Systems", In: *Proceedings of the International Conference and Workshop on the Engineering of Computer-Based Systems*, Edinburgh, Scotland.

TECHSMITH, 2002, Camtasia, www.techsmith.com/.

TEIXEIRA, H. V., 2003, *Geração de Componentes de Negócio a Partir de Modelos de Análise*, COPPE/UFRJ, Tese de M.Sc.

TRAVASSOS, G. H., GUROV, D., AMARAL, D., 2002, Introdução à Engenharia de Software Experimental, Relatório Técnico ES-590/02, PESC/COPPE/UFRJ.

VA SOFTWARE, 2003, "SourceForge". http://www.vasoftware.com/products/.

VÄLIMÄKI, M., 2005, The Rise of Open Source Licensing: A Challenge to the Use of Intellectual Property in the Software Industry, Turre Publishing.

VIEIRA, V., MANGAN, M. A. S., WERNER, C.M.L., MATTOSO, M., 2004, "An Awareness Mechanism for Shared Databases", In: *International Workshop on Groupware*, pp.92-104, Lecture Notes in Computer Science, San Jose, Costa Rica, set.

VIEIRA, V., MANGAN, M. A. S., WERNER, C.M.L., MATTOSO, M., 2003, "Ariane: Um Serviço de Percepção em Bases de Dados Compartilhadas", In: *Anais do Simpósio Brasileiro de Multimídia e Web*, pp.429-436, Salvador, Brasil, novembro.

VIEIRA, V., 2003, *Ariane: Um Mecanismo de Apoio à Percepção em Bases de Dados Compartilhadas*. Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

WALENSTEIN, 2002, *Cognitive Support in Software Engineering Tools: A distributed cognition framework*, University of Alberta, Tese de DSc.

WALLNAU, K. C., HISSAM, S. A., SEACORD, R. C., 2001, *Building Systems from Commercial Components*, SEI Series in Software Engineering, Addison-Wesley, 316p.

WEBDAV, 2003, WebDAV papers and Articles, Disponível em http://www.webdav.org/papers/, último acesso em junho de 2003.

WERNER, C. M L., MANGAN, M. A. S., *et al.*, 2003, "OdysseyShare: an Environment for Collaborative Component-Based Development". In: *Information Reuse and Integration Conference*, Las Vegas, Nevada, pp. 61-68.

WERNER, C. M L., MANGAN, M. A. S., MURTA, L. G. P. *et al.*, 2002, "Odyssey Share: Um ambiente para o Desenvolvimento Cooperativo de Componentes", In: *XVI Simpósio Brasileiro de Engenharia de Software, Caderno de Ferramentas*, Gramado, outubro, pp. 444-449.

WERNER, C.M.L, *et al.*, 2000, "Odyssey: Estágio Atual", In: *Caderno de Ferramentas*

*do XV Simpósio Brasileiro de Engenharia de Software (SBES'2000)*, João Pessoa, Outubro.

WERNER, C.M.L., BORGES, M. R. S., MATTOSO, M., BRAGA, R. M. M., CAMPOS, F., MANGAN, M. A. S., VIEIRA, V., 2003, "OdysseyShare: Desenvolvimento Colaborativo de Componentes", In: *Anais do Simpósio Brasileiro de Multimídia e Web*, pp.469-476, Salvador, Brasil, nov.

WIEGERS, K. E., 2002, *Peer Reviews in Software*: a practical guide, Addison-Wesley, 232 p.

WOHLIN, C., *et al.*, 2002, Experimentation in Software Engineering: An Introduction.

WORLD WIDE WEB CONSORTIUM, 2005a, *Resource Description Framework*, http://www.w3.org/RDF/.

WORLD WIDE WEB CONSORTIUM, 2005b, *Semantic Web*, disponível em http://www.w3.org/2001/sw/.

WULF, V., 1999, "Evolving Cooperation when Introducing Groupware - A Self-Organization Perspective", *Cybernetics and Human Knowing*, v. 6, n. 2, pp. 55 - 75.

YE, Y., FISCHER, G., 2000, "Promoting Reuse with Active Reuse Repository Systems", In: *Proceedings of the 6th International Conference on Software Reuse*, Viena, Austria.

YVORY, M. Y., HEARST, M. A., 2001, "The state of the art in automating usability evaluation of user interfaces", *ACM Computing Surveys*, v.33, n.4 (Dec), pp. 470-516.

# Anexo I – Instrumentos de Avaliação

| T | Avaliação de Percepção de Grupo<br>Procedimento | ID |
|---|---|---|

## INSTRUÇÕES

O estudo é gravado. Você utilizará um conjunto de fone de ouvido e microfone durante a tarefa. Para iniciar a gravação aperte F9. Para suspender a gravação aperte F9 e F9 novamente para retomar. Aperte F10 e informe o nome do arquivo para encerrar a gravação. O nome do arquivo é formado pelo seu número de participante. Exemplo: p01.avi.

Pergunte e comente tudo que achar necessário.

## PASSOS

Preencher o questionário de identificação.

Ler o documento de treinamento. (Nota: não apresentado nesta tese.)

Preencher o questionário 1.

Realizar o teste de gravação.

Ler a descrição da tarefa e responder as questões com o auxílio do Eclipse.

Preencher o questionário 2.

| ID | **Avaliação de Percepção de Grupo**<br>Questionário de Identificação de Participante | **ID** |
|---|---|---|

## ESCLARECIMENTOS

Este questionário representa um acordo entre você (Participante) e a equipe de avaliação.

Durante esta avaliação, você será designado para realizar uma tarefa individual. O objetivo desta avaliação é obter dados que permitam a melhoria da qualidade das ferramentas utilizadas e o entendimento da tarefa solicitada. Neste sentido, os autores do sistema contam com seus comentários. Comunique qualquer sugestão, dúvida, opinião ou comentário durante ou depois da avaliação.

Toda informação coletada nesta pesquisa será utilizada com o propósito único e exclusivo de avaliar os serviços de espaço compartilhado em questão. O seu desempenho individual não está sendo avaliado para fins acadêmicos ou profissionais. Os dados nesta folha não serão divulgados nos resultados da avaliação. Sua identificação pessoal nos registros aparece no canto superior direito desta folha.

Lembre-se:

- Concentre-se para realizar a tarefa;
- Comunique tudo que achar necessário;
- Você não está sendo avaliado.

## DADOS DE IDENTIFICAÇÃO

Informe:

1. Nome: [ ]
[ ]
2. Endereços de correio eletrônico: [ ]
[ ]
3. Telefones: [ ]
[ ]

## DECLARAÇÃO

Autorizo a utilização e divulgação dos dados coletados e das minhas opiniões emitidas durante a pesquisa. Compreendo que meu desempenho individual não está sendo avaliado.

Participante Local e Data

| Q1 | **Avaliação do Espaço Compartilhado**<br>Primeiro Questionário do Avaliador (32 questões) | **ID** |
|---|---|---|

**Instruções: Leia com atenção cada pergunta. Assinale com um "X" todas as alternativas entre parênteses que julgar apropriadas. Preencha os campos de respostas entre colchetes.**
**Utilize as margens e o verso desta folha para anotar comentários se necessário.**

## DADOS DEMOGRÁFICOS

Informe:
1. Ano de nascimento (exemplo: 1978): [ ] [ ] [ ] [ ]
2. Sexo: ( ) Masculino ( ) Feminino

## FORMAÇÃO

Informe os dados do curso de mais alto nível que ingressou.
3. Nível do curso: ( ) Graduação ( ) Mestrado ( ) Doutorado
4. Área de especialização, caso se aplique:
[ ]
5. Ano de ingresso: [ ] [ ] [ ] [ ]
6. Ano de conclusão ou previsão de conclusão: [ ] [ ] [ ] [ ]

## EXPERIÊNCIA PROFISSIONAL

Responda esta seção apenas se exerce alguma função remunerada.
7. Sua única remuneração é uma bolsa de estudos? ( ) Sim ( ) Não
8. Principais atividades: [ ]
[ ]
9. Instituição em que exerce função: [ ]
[ ]
10. Ano de ingresso: [ ] [ ] [ ] [ ]

## SOBRE DESENVOLVIMENTO CONCORRENTE E DISTRIBUÍDO

11. Participou de alguma tarefa concorrente ou distribuída nos últimos seis meses? ( ) Sim ( ) Não

Nas questões seguintes, considere uma das tarefas distribuídas.
12. Durante a tarefa foram manipulados:
( ) Código fonte ( ) Interface de usuário ( ) Modelos e diagramas
( ) Outros: [ ]
13. Os objetos manipulados eram produtos de autoria:
( ) de minha própria autoria ( ) da equipe ( ) de terceiros

14. A iniciativa para a seleção de um colega para trabalhar foi:
( ) minha ( ) de um colega ( ) de um superior hierárquico
( ) outro, explique:
15. Quantas pessoas participaram da tarefa (informe o número)? [ ][ ]
16. Quanto tempo durou a tarefa (informe o total de horas)? [ ]
17. Descreva resumidamente objetivos e principais passos da tarefa realizada:
[ ]
[ ]
[ ]
18. Em uma dessas tarefas, foi utilizada uma técnica de trabalho em grupo ou ferramenta de apoio (repositório, banco de dados ou um sistema de apoio ou mediação)? Quais?
[ ]
[ ]
[ ]
19. Cite um artigo, livro ou autor que considere importante na área de programação concorrente ou globalizada, caso conheça:
[ ]
[ ]
[ ]
20. A sua experiência com esse tipo de tarefa é derivada:
( ) de estudo em cursos ou livros
( ) de exercícios em cursos
( ) da prática no meu trabalho
21. No contexto de desenvolvimento de software, considero a tarefa de seleção de colaborador como:
( ) uma das tarefas menos complexas que conheço
( ) uma das tarefas de complexidade média
( ) uma das tarefas mais complexas que conheço

## SOBRE *O ECLIPSE IDE E PROGRAMAÇÃO*

21. Você já programou na linguagem Java? ( ) Sim ( ) Não
22. Indique as ferramentas que utiliza na programação:
[ ]
[ ]
23. Você já utilizou o *Eclipse IDE*? ( ) Sim ( ) Não

24. Descreva resumidamente o domínio de aplicação de um dos sistemas realizados:

[ ]
[ ]
[ ]

25. Cite um artigo, livro ou autor que considere importante na área de programação:

[ ]
[ ]

## SOBRE O USO DE PROGRAMAS E DA INTERNET

26. Quantas horas por dia você utiliza a Internet? [ ][ ]

27. Em uma sessão de trabalho típica você:

( ) trabalha com apenas uma janela de aplicação de cada vez

( ) abre várias janelas de aplicações diferentes e alterna sua atenção entre elas

28. Você costuma utilizar o botão direito do *mouse*? ( ) Sim ( ) Não

29. Indique, em ordem de preferência, até seis programas que utiliza com maior freqüência no seu dia-a-dia: (a – programa de maior preferência)

a. [ ] d. [ ]
b. [ ] e. [ ]
c. [ ] f. [ ]

30. Indique alguns dos melhores programas que você já utilizou.

[ ]

## SOBRE TRABALHO EM GRUPO

31. Você prefere trabalhar individualmente ou em grupo? Por quê?

[ ]
[ ]
[ ]

32. Sua experiência de trabalho em grupo poderia ser considerada:

( ) péssima ( ) ruim ( ) razoável ( ) boa ( ) ótima

Obrigado por sua colaboração!

| T | Avaliação de Percepção de Grupo<br>Descrição da Tarefa | ID |
|---|---|---|

INSTRUÇÕES

O seu empregador firmou um acordo de cooperação com o projeto do *Apache*. Uma equipe local está sendo formada para interagir com a equipe do *Ant*, um dos diversos subprojetos do Apache. O Ant oferece diversos comandos para a automação da manutenção de um projeto. Cada comando é descrito por meio de uma *tarefa* (*Ant Task*). Cada tarefa é descrita em uma ou mais classes Java.

A equipe local deve programar novas *tarefas* para automatizar o uso de um novo produto da sua empresa que deve substituir o atual CVS. Essas tarefas farão parte de uma versão futura do Ant: Termite 2.0. O Termite 2.0 substituirá o Ant com uma série de vantagens. Entretanto, o projeto ainda é um segredo do pessoal do Apache. Existe pouca documentação sobre as alterações realizadas e a informação é fornecida sob demanda. Por sorte, a equipe do Termite é a mesma do Ant atual.

Você foi alocado para trabalhar com nessa equipe local. Sua responsabilidade, no momento, é descobrir quem trabalha na equipe do Termite/Ant e indicar pessoas que estejam cientes do estado atual do projeto.

Seria tudo mais fácil se você pudesse viajar até a sede da Apache e encontrar a equipe pessoalmente. O problema é a equipe não se reúne regularmente em uma mesma sala. Bom, a sugestão do pessoal do Apache é que você procure as pessoas que estão trabalhando no repositório CVS do Ant, nas classes relacionadas com o SSH e com o CVS. A sua gerência local elaborou algumas questões que podem ajudar você a realizar uma melhor indicação.

Para completar sua primeira participação no projeto, responda as questões a seguir:

## QUESTÕES

Por favor, leia e responda cada questão da melhor forma possível antes de tentar responder a próxima questão. Marque um asterisco no início da resposta quando estiver em dúvida ou não souber responder. Utilize qualquer ferramenta ou opção do Eclipse que julgar necessária. A resposta pode ser alterada depois, se você mudar de idéia. Preencha suas respostas na página da questão correspondente.

***(1)*** Verifique se o seu ambiente está configurado para utilizar o projeto Ant e anote a configuração utilizada:

a. Versão do Eclipse [ ]
b. Plataforma [ ]
c. Versão do JDK [ ]
d. Repositório CVS [ ]
e. Projeto [ ]

*(2)* Localize a classe `Task`. Liste subclasses da hierarquia de herança da classe `Task` que estejam relacionadas com a integração com o SSH ou com o CVS. Nesta lista é que seu trabalho vai se concentrar. O projeto será um sucesso se você incluir as tais novas classes corretamente nesta hierarquia. Mas por enquanto, basta encontrar quem anda trabalhando com estas classes, o que nos leva para a próxima questão.

*(3)* O próximo passo é listar as pessoas que contribuíram com essas classes. Tenha um pouco de paciência e liste todo o pessoal envolvido. Coloque a lista em ordem alfabética para evitar repetições de nomes. Dica: use o programa que foi apresentado no treinamento para descobrir os nomes dos programadores. Você será informado das atividades nos últimos cinco anos de cada arquivo.

*(4)* Considerando as classes escolhidas na questão 2, indique as três últimas subclasses que foram adicionadas, ou seja, as classes que foram criadas recentemente.

*(5)* Considerando as classes escolhidas na questão 2, aproximadamente, quantas contribuições cada classe recebeu, em média?

*(6)* Considerando as classes escolhidas na questão 2, em média, quantas pessoas diferentes trabalharam em cada classe?

*(7)* Entre as alternativas abaixo, a pessoa mais indicada para responder sobre uma classe é: (Considere que todos que contribuíram ainda participam do projeto.)

( ) o autor.

( ) o programador que realizou o maior número total de contribuições.

( ) o programador que realizou o maior número das revisões mais recentes.

( ) eu utilizaria outra regra: (explique)

*(8)* Indique os três desenvolvedores com maior número de contribuições nas classes concretas derivadas de `AbstractCvsTask`.

***(9)*** Finalmente, indique dois desenvolvedores do Ant que você escolheria para pedir ajuda ou informações sobre como programar novas *tarefas* para automatizar o uso do produto da sua empresa?

| Q2 | **Avaliação de Percepção de Grupo**<br>Segundo Questionário do Avaliador (34 questões) | **ID** |
|---|---|---|

## SOBRE A UTILIZAÇÃO

**Assinale uma das opções da escala ao lado de cada afirmativa. A escala varia de 1 discordo totalmente a 5 concordo plenamente. Assinale NA caso considere que a pergunta não se aplica no caso.**

| | | |
|---|---|---|
| 01 | Levando em consideração a ferramenta como um todo, estou satisfeito com a facilidade de uso. | 1 2 3 4 5 NA |
| 02 | A utilização dos componentes foi simples. | 1 2 3 4 5 NA |
| 03 | Eu consigo efetivamente realizar o meu trabalho utilizando estes componentes. | 1 2 3 4 5 NA |
| 04 | Eu sou capaz de realizar o meu trabalho rapidamente utilizando estes componentes. | 1 2 3 4 5 NA |
| 05 | Eu sou capaz de realizar o meu trabalho com eficiência utilizando estes componentes. | 1 2 3 4 5 NA |
| 06 | Sinto-me confortável utilizando a ferramenta. | 1 2 3 4 5 NA |
| 07 | Foi fácil aprender a utilizar a ferramenta. | 1 2 3 4 5 NA |
| 08 | Acredito que possa me tornar rapidamente mais produtivo utilizando a ferramenta. | 1 2 3 4 5 NA |
| 09 | A ferramenta fornece mensagens de erro que me informam claramente como resolver problemas. | 1 2 3 4 5 NA |
| 10 | Ao cometer um erro, consigo voltar a trás de forma fácil e rápida. | 1 2 3 4 5 NA |
| 11 | Informações (ajuda on-line, mensagens na tela e outros tipos de documentação) fornecidas pela ferramenta são claras. | 1 2 3 4 5 NA |
| 12 | É fácil encontrar as informações necessárias para a tarefa. | 1 2 3 4 5 NA |
| 13 | A informação que devo fornecer à ferramenta é de fácil compreensão. | 1 2 3 4 5 NA |
| 14 | A informação fornecida é efetiva na ajuda em completar a tarefa. | 1 2 3 4 5 NA |
| 15 | As informações estão organizadas na tela de forma clara. | 1 2 3 4 5 NA |
| 16 | A interface dos componentes é agradável. | 1 2 3 4 5 NA |
| 17 | Eu gostei de utilizar a interface dos componentes. | 1 2 3 4 5 NA |

| | | | |
|---|---|---|---|
| 18 | Os componentes possuem todas as funcionalidades por mim esperadas. | 1 2 3 4 5 | NA |
| 19 | Ao todo, estou satisfeito com os componentes. | 1 2 3 4 5 | NA |

20. Liste aspectos positivos:

[ ]
[ ]
[ ]
[ ]

21. Liste aspectos negativos:

[ ]
[ ]
[ ]
[ ]

## SOBRE A EXPERIÊNCIA DE UTILIZAÇÃO DOS COMPONENTES

| | | | |
|---|---|---|---|
| 22 | Em geral, considero a minha experiência satisfatória. | 1 2 3 4 5 | NA |
| 23 | Os objetivos da tarefa foram atingidos. | 1 2 3 4 5 | NA |
| 24 | Conseguiu efetivamente realizar minha tarefa. | 1 2 3 4 5 | NA |
| 25 | Estou satisfeito com a lista dos participantes da equipe. | 1 2 3 4 5 | NA |
| 26 | Estou satisfeito com as escolhas que realizei na seleção de colaboradores em potencial. | 1 2 3 4 5 | NA |
| 27 | Estou totalmente satisfeito com a tarefa. | 1 2 3 4 5 | NA |
| 28 | Eu usaria a ferramenta para realização de seleção de colaboradores na prática. | 1 2 3 4 5 | NA |

## SOBRE OS RECURSOS UTILIZADOS

29. Você já havia utilizado este recurso em outro programa?

| | | |
|---|---|---|
| Indicação de presença de usuário | ( ) Sim | ( ) Não |
| Histórico de presença individual | ( ) Sim | ( ) Não |
| Histórico de diversos participantes | ( ) Sim | ( ) Não |

30. Indique quais programas computador apresentam pelo menos um dos recursos acima.

[ ]
[ ]
[ ]

31. Resuma sua opinião sobre a aplicação dos recursos na tarefa realizada:

[ ]
[ ]
[ ]

## SOBRE OS PROCEDIMENTOS DA AVALIAÇÃO

32. Você considera que o treinamento aplicado para o uso da ferramenta foi suficiente? Você teria sugestões?
[ ]
[ ]
[ ]

33. Você considera que existiu alguma dificuldade na interpretação dos desenhos e informações apresentados pela ferramenta? Se sim, por favor, exemplifique.
[ ]
[ ]
[ ]

34. As informações oferecidas sobre o sistema a ser utilizado, as técnicas propostas e sua aplicação foram de fácil compreensão? Você sentiu falta de alguma informação nesta descrição?
[ ]
[ ]
[ ]

Novamente, obrigado por sua colaboração!