MAIS: UM MECANISMO PARA APOIO À PERCEPÇÃO APLICADO A MODELOS DE SOFTWARE COMPARTILHADOS

Marco Alexandre de Macedo Lopes

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 MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

_____

Profª. Cláudia Maria Lima Werner, D.Sc.

_____

Prof. Jano Moreira de Souza, Ph.D.

_____

Profª. Renata Mendes de Araújo, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

JANEIRO DE 2005

LOPES, MARCO ALEXANDRE DE MACEDO

MAIS: um Mecanismo para Apoio à Percepção aplicado a Modelos de Software Compartilhados [Rio de Janeiro] 2005

XI, 99 p., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2005)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Percepção
2. Trabalho Cooperativo Apoiado por Computador.
3. Engenharia de Software

I. COPPE/UFRJ II. Título (série)

DEDICATÓRIA.

Aos meus pais e ao meu irmão

Agradecimentos

Ao concluir este trabalho, me vem o desejo de agradecer a todos que, ao longo da minha vida, contribuíram para que este momento enfim chegasse. Muitas foram as pessoas que tiveram participação, ainda que involuntária, na realização deste trabalho. Para não ser injusto, vou agradecer nominalmente os que tiveram participação direta e ativa na realização deste trabalho.

Primeiramente, gostaria de agradecer a **minha mãe** (**Eliane**) e ao **meu pai** (**Marco Aurelio**) pelo incentivo, apoio, carinho, amor e atenção dispensados em toda minha vida. Realizei este trabalho com apoio total e incondicional deles, assim como o apoio do **meu irmão** (**Rafael**), que se sacrificou em vários momentos para que eu pudesse concluir este trabalho.

Agradeço a Professora **Cláudia Werner** por ter apostado no meu potencial para a realização do curso de mestrado. Sua maneira de conduzir a orientação deste trabalho me incentivou cada vez mais a trabalhar com determinação, vontade e prazer, muitas vezes abdicando de seu tempo de lazer para me ajudar neste trabalho.

Ao colega e amigo **Marco Mangan**, que me co-orientou durante todo o processo de realização desta dissertação. Sua participação neste trabalho foi essencial, expondo idéias, revisando textos, sendo um dos maiores críticos e defensores deste trabalho. Gostaria de agradecer principalmente a sua ajuda neste fim de trabalho, mesmo atarefado com a conclusão de sua tese de doutorado.

Aos Professores **Guilherme** e **Ana Regina** pelos ensinamentos passados durante as disciplinas.

Aos Professores **Jano** e **Renata Araújo** por terem aceitado fazer parte dessa banca, mesmo sendo pessoas muito atarefadas.

A Professora **Maria Luiza**, por me incentivar a realizar o curso de mestrado em um momento em que estava desacreditado.

Aos **amigos da COPPE**, que pude (e posso) contar e ter o privilégio de suas amizades: **Murta**, **Márcio**, **Alexandre Dantas**, **Gustavo**, **Marcelo**, **Beto**, **Ana Paula**, **Aline**, **Vaninha**, **Cristine**, **Hamilton**, **Isabella**, **Alexandre Correa**, **Regiane**, **Natanael**, **Luiz Gustavo**, **Gleison**, **Sômulo**, **Sávio**, **Rodrigo**, **Marcos**, **Léo** e **Luis Felipe**.

Aos amigos do curso de bacharelado em informática da UFRJ (**Info 951**): **Cardoso**, **Humberto**, **Victor**, **Sascha**, **Moringa**, **Erich**, **Marcus**, **Marcelo**, **André**, **Bruno**, **Lívia**, **Elisa** e **Bianca**.

Aos amigos da **Software Design**: **André Vicent**, **Cláudio**, **Marcelo Trannin**, **Carol**, **Luciana**, **Cátia**, **Eduardo** e **Léo**.

Aos amigos de sempre: **Gustavo**, **Ricardo** e **Marcelo**.

E a **Deus**, essa força que sempre estará comigo.

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

MAIS: UM MECANISMO PARA APOIO À PERCEPÇÃO APLICADO A MODELOS DE SOFTWARE COMPARTILHADOS

Marco Alexandre de Macedo Lopes

Janeiro/ 2005

Orientadora: Cláudia Maria Lima Werner

Programa: Engenharia de Sistemas e Computação

Mecanismos para apoio à percepção são uma alternativa para diminuir o isolamento entre membros de um grupo. Em particular, eles podem ser utilizados no contexto de modelagem concorrente de artefatos de software, em que uma equipe de desenvolvedores pode desconhecer como um artefato compartilhado está evoluindo. A concepção dos desenvolvedores sobre o modelo compartilhado pode ser continuamente atualizada com esse suporte, conforme as contribuições individuais vão ocorrendo. Geralmente, mecanismos para apoio à percepção são específicos e intrínsecos a uma ferramenta colaborativa, o que dificulta sua aplicação nas ferramentas de desenvolvimento de software. Esta dissertação apresenta uma proposta de mecanismo independente de ferramenta de modelagem, que realiza a coleta implícita e não intrusiva de informações para percepção de mudança em espaço de trabalho compartilhado. Essas informações são então apresentadas aos desenvolvedores como forma de reduzir os problemas de isolamento de uma equipe de desenvolvimento distribuída. O mecanismo tem como objetivo guiar os desenvolvedores em ações futuras, de maneira a atenuar o esforço de adaptação no momento de obter-se um estado global consistente do artefato compartilhado. As informações de mudança são classificadas, agrupadas e filtradas de forma a reduzir uma possível sobrecarga cognitiva. Dois conceitos são explorados para organizar a informação de mudança: relevância e ciência. Um estudo de caso, de foco qualitativo, foi realizado, visando obter indícios relativos à utilidade do mecanismo.

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

MAIS: AN AWARENESS MECHANISM APPLIED TO SHARED SOFTWARE MODELS

Marco Alexandre de Macedo Lopes

Janeiro/ 2005

Advisor: Cláudia Maria Lima Werner

Department: Computer and Systems Engineering

Awareness mechanisms are an alternative to reduce the isolation among group members. In particular, they can be used on concurrent modeling of software artifacts, where a software developer team may not know how a shared artifact is evolving. The developer's conception about the shared model can be continuously updated with this kind of support, meanwhile individual contributions are made. In general, awareness mechanisms are specific and intrinsic of a particular collaborative tool, in such a way that its application in a software development tool is a difficult task. This dissertation presents a modeling tool independent mechanism proposal, which collects workspace awareness change information in an implicitly and non-obtrusively way. These information are presented to the developers in the expectative of reducing some problems caused by the isolation inside a distributed software team. The mechanism's objective is to guide developers on future actions in some way to minimize the adaptation effort to get a consistent global state of the shared artifact. Change information are classified, grouped and filtered to reduce a possible cognitive overload. Two concepts are explored to organize change information: relevance and acknowledgement. A qualitative study case was performed, aiming to obtain some indications about the utility of the mechanism.

Índice:

Índice de Figuras

Índice de Tabelas

Capítulo 1 – Introdução

1.1 – Motivação

Com a globalização dos negócios, as organizações tiveram que repensar e reavaliar suas estruturas e procedimentos, para se manterem competitivas no mercado. O mercado de software também é afetado por essa característica. As organizações, para obter vantagens competitivas em relação às demais, podem buscar soluções externas (por exemplo, terceirização de parte do processo de desenvolvimento de software) em outras localidades (LAYZELL, BRERETON, FRENCH, 2000) (PRIKLADNICKI, AUDY, EVARISTO, 2003).

Em (HERBSLEB e MOITRA, 2001), são apresentados alguns fatores que motivam o desenvolvimento global de software: (i) necessidade de captar, no mercado global, recursos escassos com certos perfis; (ii) proximidade com o mercado consumidor do software; (iii) rápida formação de corporações e equipes virtuais, para explorar oportunidades de negócio; (iv) pressão para prover "tempo de mercado" (*time-to-market*), explorando a possibilidade de aumento de horas produtivas de trabalho com o fuso horário do período de expediente dos membros da equipe de desenvolvimento de software. Estes fatores, basicamente, estão relacionados com o aumento de produtividade das equipes de desenvolvimento e a redução de custos de implantação/execução de um processo de desenvolvimento de software. Desta forma, membros de uma equipe de desenvolvimento de software podem não estar disponíveis para interação entre si em um mesmo espaço físico ou em um mesmo intervalo de tempo.

Apesar desses fatores, existem tarefas, em um processo de desenvolvimento de software, nas quais interações entre indivíduos se fazem necessárias: *brainstorming* no planejamento, *pair-programming* na implementação e *peer review* na inspeção de software são algumas tarefas que costumam ser realizadas por mais de um indivíduo. No contexto de distribuição dos negócios de uma organização que desenvolve software, a realização deste tipo de tarefa necessita ser adaptada. Esta adaptação deve ocorrer de maneira a suprir a falta de alguns aspectos encontrados em interações presenciais, como a facilidade de comunicação face-a-face, além do comprometimento dos indivíduos com a tarefa a ser realizada.

A atividade de modelagem de software é uma tarefa, dentre as diversas que constituem um processo de desenvolvimento de software, que pode ser realizada em grupo. Além do fator de distribuição no tempo e no espaço dos desenvolvedores pertencentes a uma equipe, necessidades apresentadas no desenvolvimento de um software podem determinar que a modelagem de um dado artefato seja realizada de forma distribuída, por mais de um desenvolvedor. Atrasos no cronograma de desenvolvimento de um software, por exemplo, podem implicar na adoção desta prática, dividindo a atividade de modelagem entre desenvolvedores capacitados para tal. Visões sobre este modelo podem ser desenvolvidas por mais de um indivíduo, explorando o paralelismo entre as tarefas, para atingir o cronograma de entrega estipulado.

Dado que porções de um artefato de software podem ser modeladas individualmente, existe a necessidade de que estas, após a manipulação pelos desenvolvedores, componham um único artefato consistente. A Figura 1 descreve este cenário: desenvolvedores manipulam partes de um artefato de software "M". Após cada manipulação individual, as partes devem se compostas, reconstituindo o artefato. É desejável que não se despenda muito esforço com a adaptação das partes alteradas individualmente, para realizar a composição do artefato. Dependendo do esforço realizado na adaptação das partes, a vantagem de se dividir a tarefa de modelagem pode ser comprometida. Neste caso, o paralelismo obtido na manipulação concorrente das partes pode não reduzir o tempo necessário para a modelagem.

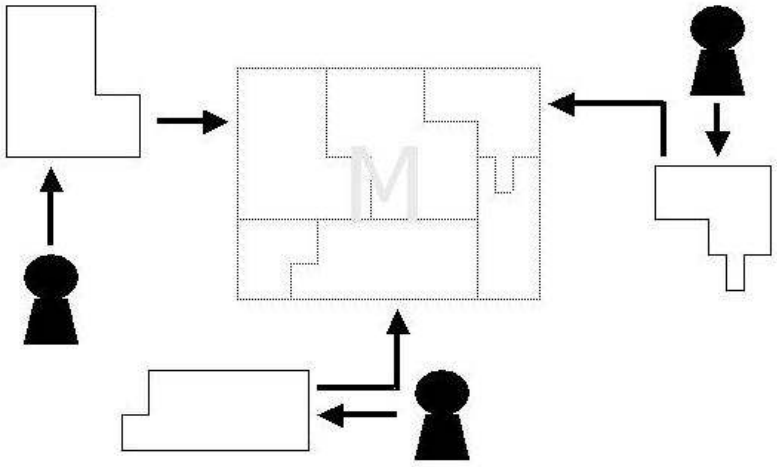

Figura 1 - Manipulação Concorrente de Modelos de Software

Para que a composição das partes seja realizada de maneira a não gastar, posteriormente, muito esforço com adaptações, é interessante que os desenvolvedores envolvidos tenham o conhecimento do trabalho realizado pelos demais. A Figura 2 apresenta a situação em que desenvolvedores manipularam individualmente partes do artefato "M", sem possuir conhecimento sobre o trabalho dos demais. É possível observar que as manipulações realizadas na parte "A" se sobrepõem às realizadas em "B", no artefato "M". A região "S" da figura define a intersecção entre estas partes. Neste caso, o desenvolvedor que manipulou "B" tinha uma concepção de como era o artefato "M" ao todo. Conforme as partes de "M" foram sendo modificadas, e este desenvolvedor não realizou mudanças relevantes em sua parte ("B"), a concepção global do artefato não mais se aplicava, devido a modificações ocorridas em "A".

Se esta concepção, por parte de cada desenvolvedor que manipulou o artefato compartilhado, fosse sendo sincronizada com o estado corrente deste (levando em consideração as contribuições de cada desenvolvedor), problemas como sobreposições das partes poderiam ter sido atenuados. Os desenvolvedores poderiam ter realizado suas modificações com base nas modificações realizadas pelos demais. Pressupõe-se que, na etapa de composição das partes do artefato, não teria sido necessário um grande esforço no reparo de inconsistências e sobreposições ocasionadas pelas modificações.

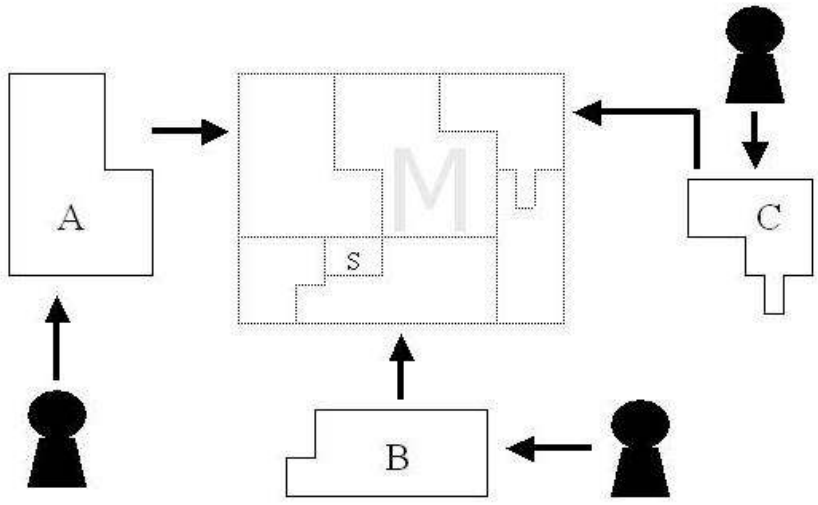

Figura 2 - Modelagem Concorrente sem Conhecimento Global

Advindo da área de Trabalho Cooperativo Assistido por Computador (*CSCW*), do inglês *Computer-Supported Cooperative Work* (ELLIS, GIBBS, REIN, 1991), o

conceito de percepção, definido inicialmente em (DOURISH e BELLOTTI, 1992) como "compreensão das atividades dos indivíduos envolvidos na colaboração, provendo um contexto para a atividade de determinado indivíduo", é útil para contextualizar os desenvolvedores, participantes de uma interação colaborativa de modelagem de um artefato de software, sobre o trabalho realizado pelos demais. Como percepção se refere a um estado mental de determinado indivíduo (SOHLENKAMP, 1998), estas informações são transmitidas aos indivíduos participantes de uma interação colaborativa (no caso, desenvolvedores de software) através de mecanismos. Desta forma, mecanismos que coletem, distribuem e apresentem informações, relativas à modelagem de partes de um artefato compartilhado de software, para os desenvolvedores envolvidos, são úteis neste contexto, pois permitem gerá-las e transmiti-las sem o esforço de um desenvolvedor (coleta implícita), já que a própria tarefa de modelagem abastece estes mecanismos com as informações necessárias.

Na medida que os desenvolvedores manipulam determinada parte do artefato compartilhado, algum suporte computacional (mecanismo para apoio à percepção) deve existir para inferir que mudanças foram realizadas nesta, sem a necessidade de informar explicitamente o que foi alterado. Assim, os desenvolvedores se concentram apenas na atividade de modelagem, sem a necessidade de execução da tarefa adicional de informar explicitamente as mudanças realizadas sobre o artefato compartilhado.

Desta forma, é interessante que um mecanismo para apoio à percepção de mudanças ocorridas em artefatos compartilhados de software pouco interfira no fluxo de trabalho dos desenvolvedores envolvidos. As informações de mudanças devem ser coletadas, distribuídas e apresentadas de maneira que os desenvolvedores não desviem o foco do trabalho que estão realizando. É desejável que este mecanismo seja de propósito geral, visando sua utilização em diversos contextos, permitindo ser acoplado a ferramentas já utilizadas pelos desenvolvedores para modelagem de um dado tipo de artefato de software.

1.2 – Objetivos da Dissertação

Esta dissertação tem como objetivo propor e implementar um protótipo de mecanismo, independente de ferramenta de modelagem, para apoio à percepção de mudanças em artefatos compartilhados de software, de nome ***MAIS*** (acrônimo para ***M****ulti-synchronous* ***A****wareness* ***I****nfra****S****tructure*). Mais especificamente, modelos

baseados na *UML* (OMG, 2004b) são tratados neste trabalho, visando explorar sua sintaxe para prover informações nem sempre aparentes aos usuários do mecanismo.

O mecanismo *MAIS* deve ter como responsabilidade coletar, distribuir e apresentar informações de mudanças, ocorridas em modelos compartilhados, sendo representadas como eventos (PRINZ, 1999). Esta representação é utilizada como informação para percepção do mecanismo em questão.

A coleta, distribuição e apresentação de eventos devem ser realizadas de forma não intrusiva, sem que a ferramenta utilizada pelos desenvolvedores para manipulação de modelos *UML* sofra adaptações relevantes. A coleta de eventos deve ser realizada de forma implícita, em que estes são gerados através da manipulação dos modelos compartilhados.

Este mecanismo deve atuar em conjunto com a ferramenta de modelagem de maneira passiva, isto é, agindo sobre estímulos gerados por esta. No caso de coleta de eventos, o mecanismo deve ser notificado pela ferramenta de modelagem, que informa qual mudança foi realizada no modelo compartilhado. Um evento relativo deve ser gerado e distribuído a todos os desenvolvedores, que são notificados sobre a existência de uma nova alteração. Dado que o evento é recebido, este deve ser apresentado a cada desenvolvedor. Esta apresentação deve ser realizada de maneira a fornecer uma visão global de como as cópias do modelo compartilhado estão evoluindo.

Desta forma, o mecanismo tem como propósito atender a etapa de convergência das partes do modelo, manipuladas individualmente pelos desenvolvedores, em um o estado global deste, armazenado em um repositório central. Conforme os desenvolvedores vão sincronizando suas concepções sobre o modelo compartilhado (isto é, vão percebendo a evolução do modelo através das modificações realizadas), as ações subsequentes realizadas sobre este poderão ser feitas visando reduzir o esforço de adaptação das partes do modelo na convergência em uma versão deste livre de inconsistências.

Os benefícios esperados, do ponto de vista do desenvolvedor, com a utilização do mecanismo *MAIS* são: (i) estar ciente do ritmo e foco do trabalho dos demais desenvolvedores, (ii) antever possíveis conflitos, (iii) identificar especialistas sobre determinados artefatos e partes destes e (iv) verificar as alterações entre o atual estado do artefato e da cópia local em que este trabalha.

1.3 – Contexto da Dissertação

A elaboração desta dissertação está no contexto do Projeto *Odyssey Share* (ODYSSEY, 2004), que é a proposta de criação de uma ambiente colaborativo sobre o ambiente *Odyssey* original com o uso de componentes de software colaborativos. O ambiente *Odyssey* define processos para construção e utilização de artefatos reutilizáveis, seguindo preceitos do Desenvolvimento Baseado em Componentes e das Engenharias de Domínio e de Aplicação.

Esta dissertação está contextualizada em duas áreas de pesquisa: em Engenharia de Software, o foco da pesquisa se encontra no **Desenvolvimento Global de Software** (HERBSLEB e MOITRA, 2001); e em *CSCW*, o trabalho refere-se a **Percepção** (DOURISH e BELLOTTI, 1992), mais especificamente em **Percepção de Mudanças** (TAM, MCCAFFREY, GREENBERG, 2000) de artefatos de software.

1.4 – Organização dos Capítulos

O restante desta dissertação está organizado da seguinte maneira:

O **Capítulo 2** apresenta o estudo realizado nas áreas de percepção em *CSCW* e de desenvolvimento globalizado de software. São descritas algumas abordagens que serviram de inspiração para o desenvolvimento desta dissertação.

No **Capítulo 3**, é apresentado o mecanismo *MAIS*, assim como suas características, funcionalidades e arquitetura, além da comparação de determinadas funcionalidades das abordagens pesquisadas com as relativas em *MAIS*.

O protótipo *MAIS* é descrito no **Capítulo 4**, apresentando as tecnologias utilizadas, além da descrição de como os elementos da arquitetura foram implementados. Um estudo de caso, de foco qualitativo, referente ao protótipo, é também descrito neste capítulo.

O **Capítulo 5** apresenta a conclusão deste trabalho, descrevendo as contribuições e limitações do mecanismo e do protótipo *MAIS*, vislumbrando possíveis melhorias nestes para trabalhos futuros, de maneira a dar continuidade na pesquisa sobre o tema abordado.

Os anexos apresentam documentos relativos ao estudo de caso realizado. O **Anexo A** apresenta o questionário de caracterização de participante. Já o **Anexo B** contém o a descrição da tarefa do estudo de caso. Por fim, o **Anexo C** apresenta o questionário referente à tarefa proposta.

Capítulo 2 – Desenvolvimento de Software e Percepção

2.1 – Introdução

Este capítulo apresenta uma revisão sobre o trabalho de desenvolvimento de software em equipe, em que membros não estão disponíveis para interação em um mesmo espaço físico ou em um mesmo instante de tempo. Em especial, é descrito um cenário onde membros desta equipe manipulam um artefato de software, que é compartilhado pelos integrantes da equipe de desenvolvimento.

É apresentado o conceito de percepção, advindo da área de Trabalho Cooperativo Assistido por Computador (*CSCW*), descrevendo alguns mecanismos existentes na literatura. Estes mecanismos servem de apoio aos integrantes das equipes distribuídas de desenvolvimento de software na atividade de manipulação concorrente de artefatos de software compartilhados. Mecanismos para apoio à percepção em espaços de trabalho compartilhados são apresentados como um ferramental para esse apoio, provendo os desesenvolvedores com informações que os auxiliem a convergir o artefato de software, manipulado concorrentemente, a um estado consistente.

Algumas abordagens existentes na literatura são descritas neste capítulo. Estas abordagens foram analisadas levando em consideração os seguintes aspectos :

- **Edição colaborativa.**
- **Apoio à modelagem de artefatos *UML*.**
- **Modos de interação entre membros da equipe.**
- **Importância de alterações realizadas sobre o artefato compartilhado.**
- **Ciência sobre alterações realizadas sobre o artefato compartilhado.**

2.2 – Desenvolvimento de Software

Desenvolver software é, essencialmente, um trabalho de equipe, onde pessoas com diferentes habilidades desempenham determinados papéis. Dependendo da complexidade do software a ser desenvolvido, as atividades inerentes a esse trabalho necessitam ser planejadas e organizadas, com a finalidade de se definir qual indivíduo da equipe faz o que e quando. Dividir o desenvolvimento de software em etapas ordenadas permite que sejam agrupados, em cada etapa, desenvolvedores com habilidades afins, que tomam como base para a execução da atividade artefatos

elaborados em etapas anteriores, produzindo outros, que serão utilizados em etapas subsequentes.

Podemos chamar esse conjunto de etapas ordenadas de processo (no caso, processo de desenvolvimento de software): uma sequência de passos que envolvem atividades, restrições e recursos, para produção de algum resultado (PFLEEGER, 2001). Como neste caso o resultado é um produto (software), este processo pode ser denominado ciclo de vida (de software). São exemplos de modelos de processos de desenvolvimento de software: (i) cascata, (ii) prototipação, (iii) incremental e (iv) espiral (PRESSMAN, 2000).

As etapas abaixo, geralmente, são comuns a todos os processos de desenvolvimento de software (PFLEEGER, 2001) :

- **Análise e definição de requisitos.**
- **Projeto de Sistemas.**
- **Projeto de Programas.**
- **Implementação.**
- **Testes de Unidade.**
- **Testes de Integração.**
- **Testes de Sistema.**
- **Entrega do Sistema.**
- **Manutenção.**

O problema abordado nesta dissertação se concentra nos estágios de projeto (tanto de Sistemas quanto de Programas), onde metáforas de modelos de software são utilizadas para descrever o problema que o software a ser desenvolvido deverá atender.

Algumas características observadas em desenvolvimento de software são apresentadas em (ALTMANN e POMBERGER, 1999): (i) aumento crescente na complexidade dos tipos de software a serem desenvolvidos; (ii) processos de software que não podem ser formalizados ou automatizados, já que a proporção de atividades em que é necessário a criação é muito maior do que as que requerem reprodução; (iii) alto potencial de risco envolvido, devido a mudanças ou indefinições de requisitos, aliado a dificuldade em decompor o desenvolvimento em sub-tarefas; (iv) grande necessidade de comunicação, especialmente a informal, entre os membros da equipe; (v) grande necessidade de documentação.

Estas características reforçam a idéia de que as etapas de processos de desenvolvimento de software precisam ser bem definidas e coordenadas. Um processo

para a realização de cada etapa é necessário. Como agravante, é possível que a realização de uma ou mais atividades de um processo de desenvolvimento de software seja distribuída pelo tempo e pelo espaço (ALTMANN e POMBERGER, 1999) (MANGAN, BORGES, WERNER, 2004) (BOULILA, DUTOIT, BRUEGGE, 2003) (SCHÜMMER e SCHÜMMER, 2001) (KOBYLINSKI et al., 2002) (VAN DER HOEK et al., 2004) (FROEHLICH e DOURISH, 2004) (HERBSLEB e MOITRA, 2001).

2.2.1 – Desenvolvimento Global de Software

O termo Desenvolvimento Global de Software (*Global Software Development*) é apresentado em (HERBSLEB e MOITRA, 2001) e engloba aspectos técnicos, sociais e econômicos de se desenvolver software com uma perspectiva distribuída. Esta distribuição tanto é geográfica, isto é, os indivíduos pertencentes a uma mesma equipe de desenvolvimento de software estão dispersos espacialmente, quanto temporal, isto é, estes indivíduos realizam, em horários alternados, uma mesma tarefa.

Em (FROEHLICH e DOURISH, 2004), se observa que, no contexto de desenvolvimento global de software, desenvolvedores lidam com duas fontes de complexidade: complexidade do desenvolvimento de um certo artefato e a referente ao processo distribuído de desenvolvimento deste. Neste sentido, é desejável que desenvolvedores sejam apoiados, computacionalmente, tanto no aspecto do desenvolvimento do artefato quanto na atividade referente à elaboração deste.

A Figura 3 apresenta um exemplo de situação descrita acima: fazem parte de uma mesma equipe de desenvolvimento de software os desenvolvedores "A", "B" e "C", que não estão disponíveis para trabalhar em um mesmo local ou em um mesmo tempo. Estes trabalham sobre cópias dos artefatos de software "X", "Y" e "Z". Um sistema de controle de versões, que utilize uma política de *check-out* e *check-in*[1] (MURTA, 2004), adequado para tratar os tipos de artefatos apresentados, é utilizado. O desenvolvedor "A" está trabalhando sobre uma cópia do artefato "X", que depende de "Y". Uma cópia do artefato "Y", que está sendo editado por "B", depende do artefato "Z". O desenvolvedor "C" trabalha sobre cópias dos artefatos "X" e "Z". O desenvolvedor "A" precisa saber, ao editar "X", quais foram as ações realizadas por

[1] Esta política baseia-se, em um processo não formal de gerência de configuração, no princípio de que cada usuário obtém uma cópia do artefato a manipular do repositório (*check-out*), realiza alterações sobre esta e a recoloca no repositório (*check-in*).

"C" sobre "X" e de "B" sobre "Y", já que "X" depende de "Y". Isto é útil para que "A" consiga antecipar futuros problemas de consistência na convergência das cópias de "X", no armazenamento desta no repositório de controle de versões, diminuindo o retrabalho.

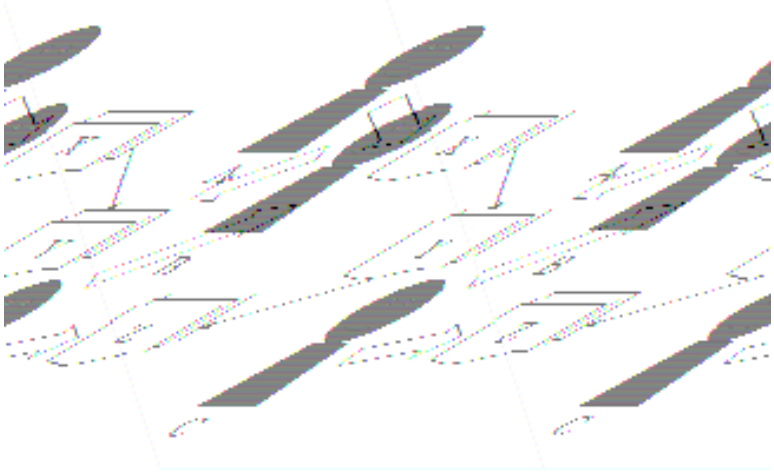

Figura 3 - Equipe Distribuída de Desenvolvimento de Software

2.3 – Percepção

No cenário apresentado na Figura 3, indivíduos de uma mesma equipe de desenvolvimento de software trabalham sobre cópias de artefatos compartilhados de software. Como esses indivíduos não estão disponíveis para interação em um mesmo local ou ao mesmo tempo, artefatos que influenciam no trabalho de mais de um desenvolvedor são elaborados sem que estes conheçam o que está sendo realizado pelos desenvolvedores afins. Estes desenvolvedores se baseiam apenas no estado das suas cópias dos artefatos para evoluí-las: alterações realizadas por outros desenvolvedores só são conhecidas no momento de unificação das cópias, ao retorná-las para o repositório de controle de versões. Assim, dependendo do que cada desenvolvedor realizou sobre a cópia relativa a um determinado artefato, um grande esforço em convergir essas cópias em um estado consistente do artefato pode ser necessário.

Desta forma, o isolamento dos indivíduos pertencentes a uma mesma equipe de desenvolvimento de software, com relação à manipulação concorrente de artefatos de software, pode prejudicar a interação onde esta é estabelecida, levando em consideração a falta de conhecimento, por parte de um desenvolvedor, das ações realizadas pelos demais sobre determinado artefato. É possível generalizar esse problema a qualquer equipe que realize uma tarefa colaborativa, onde os indivíduos não interagem face-a-face. Por exemplo, podemos considerar a atividade de reuniões à distância: gestos,

expressões faciais e o humor dos participantes desta poderiam ser representados, para tentar reproduzir o ambiente de uma reunião presencial. A expressão facial de determinado indivíduo pode externar sua opinião sobre determinado ponto abordado na reunião, permitindo aos demais participantes seguirem o curso da reunião de maneira apropriada, evitando constrangimentos ou desgastes. Para suprir essa necessidade, existe na literatura de Trabalho Cooperativo Assistido por Computador, do inglês *Computer Software Cooperative Work* (*CSCW*) (ELLIS, GIBBS, REIN, 1991), o conceito de **percepção,** derivado do inglês *awareness*. Este conceito pode ser utilizado no desenvolvimento de um software (*groupware*) que apóie determinadas etapas de um processo distribuído de desenvolvimento de software.

Percepção é definida em (DOURISH e BELLOTTI, 1992) como "a compreensão das atividades dos indivíduos envolvidos na colaboração, provendo um contexto para a atividade de determinado indivíduo". Este contexto permite identificar contribuições relevantes dos colaboradores para o grupo, bem como torna possível a avaliação das atividades individuais a respeito dos objetivos e progresso deste. Significa a compreensão do estado de um sistema (artefato comum em que indivíduos trabalham sobre, incluindo uma representação particular desses), englobando atividades passadas, estado atual e opções futuras (SOHLENKAMP, 1998).

Em atividades onde há trabalho em conjunto, para reconhecer que indivíduos fazem parte destas, é necessário que se estabeleça a idéia de um grupo de trabalho, formalizando canais de comunicação entre os seus membros. Para isso, é necessário identificar, em ações sobre o objeto compartilhado de trabalho, quais indivíduos são os autores, para serem reconhecidos como membro do grupo pelos demais participantes, permitindo que estes conheçam suas atividades no processo de cooperação (ARAÚJO, 2000). A percepção envolve saber quem é o grupo, qual é o seu objetivo e suas atividades, relacionando o conhecimento do que aconteceu, o que vem acontecendo e o que está acontecendo nessas atividades (PINHEIRO, 2001).

Abaixo estão listadas algumas características básicas, descritas em (GUTWIN e GREENBERG, 2001), relativas à percepção:

- Percepção é o conhecimento sobre o estado de um ambiente onde ocorre cooperação, sendo limitado em tempo e espaço.
- Como ambientes mudam a todo o tempo, a informação para percepção é um conhecimento que deve ser mantido atualizado.

- Indivíduos interagem entre si e exploram o ambiente, e a manutenção do estado de percepção é atingida por estas interações.
- Percepção é um objetivo secundário em uma tarefa; o objetivo maior não é prover informação para percepção e sim completar uma tarefa no ambiente compartilhado.

Em (GUTWIN e GREENBERG, 2001), o conceito de percepção é apresentado como o conhecimento criado através da interação entre um agente e seu ambiente. Esta definição estabelece que percepção pode ser encarada como um estado mental de determinado indivíduo sobre ações ocorridas no processo de interação entre membros de um grupo sobre uma atividade (SOHLENKAMP, 1998). Cada indivíduo percebe os fatos ocorridos em uma atividade colaborativa de maneira própria: no caso de uma reunião, uma afirmação de determinada pessoa pode levar dois indivíduos a tirarem diferentes conclusões sobre a opinião desta (por exemplo, um indivíduo pode achar que a pessoa que realizou a afirmação está feliz com o que disse, enquanto o outro pensa ao contrário).

Alguns benefícios de se prover informações para percepção em interações colaborativas são apresentados em (SOHLENKAMP, 1998): (i) **apoio à coordenação**, pois em alguns casos, o trabalho cooperativo é implicitamente coordenado observando as atividades dos outros membros do grupo ou os estados dos artefatos compartilhados; (ii) **apoio à comunicação informal**, já que grande parte da comunicação face-a-face entre indivíduos se dá dessa forma; (iii) **apoio à convenções**, definidas como acordos estabelecidos em um grupo para tornar possíveis interações entre indivíduos.

2.3.1 – Tipos de Informação para Percepção

Em (ARAÚJO, 2000), é considerado que, para que um indivíduo possa ter uma noção completa da interação do qual participa, é necessário que este seja capaz de perceber: (i) o **contexto social** dos grupos de onde é membro, (ii) o **contexto das atividades** que realiza e (iii) alterações realizadas sobre o **espaço de trabalho** no decorrer das interações. Para cada um destes tipos de informação para percepção, é acrescida a dimensão tempo, identificando o momento da ocorrência dessas informações na interação.

A Tabela 1 apresenta um resumo, extraído de (ARAÚJO, 2000), das características encontradas em cada um dos tipos de informação para percepção citados acima.

Tipos de Informação	Características
Percepção Social	• Composição dos Grupos • Localização Física • Presença • Proximidade • Disponibilidade • Emoção
Percepção de Atividades	• Estrutura • Status
Percepção de Espaço de Trabalho	• Status dos Objetos de Trabalho • Ações • Posicionamento dos Indivíduos na Interação

Tabela 1 - Características de Tipos de Informação para Percepção

2.3.1.1 – Percepção de Espaço de Trabalho

É destacado, nesta dissertação, o tipo de informação para percepção em espaço de trabalho compartilhado. A percepção de espaço de trabalho compartilhado é a compreensão imediata das ações de um indivíduo no espaço de trabalho compartilhado por um outro (GUTWIN e GREENBERG, 2001). Envolve saber onde os outros indivíduos estão trabalhando, o que estão fazendo e o que virão a fazer.

Em (PINHEIRO, 2001), é descrito que, em um espaço de trabalho compartilhado, o estado de percepção pode ser atingido por este ou pelos objetos compartilhados dispostos neste, onde os membros realizam suas atividades. Em (GUTWIN e GREENBERG, 2001), são apresentadas maneiras de como obter informações para percepção em espaços de trabalho compartilhados:

- **Comunicação por Consequência**: a informação para percepção é obtida como consequência de observações de ações de indivíduos no espaço de trabalho compartilhado.
- **Manipulação dos Artefatos**: a manipulação de artefatos fornece, a quem a realizou, um *feedback* sobre esta; essa informação pode ser disponibilizada aos demais participantes, para estes terem a noção do que está se passando na interação.
- **Comunicação intencional**: informações para percepção podem ser obtidas através de conversas explicitas entre pessoas sobre o que estão fazendo, escutando conversas de outras pessoas ou capturando comentários de terceiros, ao passo que estes realizam tarefas.

Com relação a obtenção de informações para percepção por manipulação de artefatos compartilhados, é descrito em (TAM, MCCAFFREY, GREENBERG, 2000) o conceito de **percepção de mudanças** como a habilidade dos indivíduos em reconhecer alterações realizadas em um artefato colaborativo por outro participante. Manter o histórico de alterações auxilia os indivíduos, que estão envolvidos em uma tarefa colaborativa, a recordar sobre ações passadas, além de contextualizar novos participantes sobre a atividade em andamento.

2.3.1.2 – Representação 5W+1H

Os tipos de informação para percepção apresentados na Tabela 1 categorizam, em diferentes perspectivas, as características necessárias para que indivíduos, que realizam uma atividade de interação colaborativa, sejam capazes de compreender esta atividade. Identificadas estas características, é possível o desenvolvimento de um software com características colaborativas (*groupware*) que atenda determinado tipo de interação, para resolução de um dado problema.

Alguns trabalhos encontrados na literatura (DOURISH e BELLOTTI, 1992) (SOHLENKAMP, 1998) (GUTWIN e GREENBERG, 2001) (PINHEIRO, 2001) (TAM e GREENBERG, 2004) se baseiam em um conjunto de seis questões (ou em um subconjunto deste) para identificar, em cada tipo de informação para percepção, as características necessárias para indivíduos alcançarem um estado de percepção. Estas questões são: "O que" (*What*), "Quem" (*Who*), "Quando" (*When*), "Onde" (*Where*), "Por que" (*Why*) e "Como" (*How*).

Por exemplo, são identificadas, em (PINHEIRO, 2001), características para um subconjunto dessas questões: (i) "O que" refere-se a quais informações devem ser fornecidas, ligadas a atividades e papéis; (ii) "Quem" destaca a noção de presença e das ferramentas de comunicação, que devem ser adaptadas ao tipo de sistema onde estão inseridas; (iii) "Quando" refere-se ao momento de geração das informações para percepção, do tempo de utilidade destas e quando estas são apresentadas; (iv) "Onde" envolve, especialmente, espaço de trabalho e objetos compartilhados neste; (v) "Como" se atribui à apresentação das informações para percepção e o balanceamento nesta apresentação, de modo a prover informações suficientes sem sobrecarregar os indivíduos. A questão "Por que" foi substituída pela questão "Quanto", que tem como objetivo definir a quantidade de informação, obtida pelas outras questões, de maneira

que seja suficiente e não sobrecarregue os indivíduos. Por exemplo, definir o quanto de informações geradas pela questão "Quem" deve ser avaliado.

Em (GUTWIN e GREENBERG, 2001), a identificação de elementos que permitam atingir percepção em espaço de trabalho compartilhado é baseada nessa representação. Neste caso, algumas dessas questões são respondidas sob a perspectiva de ações ocorridas no passado ou no presente.

2.3.1.3 – Informação de Contexto

Em (ROSA, BORGES, SANTORO, 2003), cita-se que o estado de percepção pode ser influenciado pelo contexto onde a colaboração se dá. Elementos deste contexto devem ser identificados e representados, visando induzir a um melhor estado de percepção. Estes elementos podem ser relacionados aos membros do grupo que realiza a interação colaborativa, ao grupo como um todo, as tarefas agendas e completas e ao ambiente onde a interação se dá. Isto auxilia os membros do grupo a se conhecerem e estarem cientes sobre seus objetivos e aspectos que os influenciam. Assim, a idéia de contextos pode ser utilizada para estruturar informações para percepção e oferece aos indivíduos participantes de uma interação colaborativa esta referência comum (GROSS e SPECHT, 2001)

O *framework* proposto em (ROSA, BORGES, SANTORO, 2003) define que esses elementos sejam agrupados em cinco categorias de informação: sobre indivíduos e grupos, sobre tarefas previamente estabelecidas, sobre o relacionamento entre pessoas e tarefas, sobre o ambiente onde a colaboração se dá e sobre tarefas concluídas. Este agrupamento é realizado para auxiliar a identificação dos elementos de contexto e implementação de mecanismos que permitem coletar e distribuir estes para os membros da colaboração.

2.3.2 – Tratamento de Informação para Percepção

Como apresentado anteriormente, percepção é um estado mental de um indivíduo, relacionado ao processo de interação colaborativo que este participa. Sistemas de *groupware* não são capazes de ofertar percepção, mas sim meios para que este estado possa ser atingindo. Assim, mecanismos para apoio à percepção devem ser desenvolvidos para que os membros de uma interação possam saber informações relacionadas a atividades que estão desempenhando, bem como ir além da reprodução do contexto de trabalho real dos indivíduos (ARAÚJO, 2000).

Esses mecanismos devem ser projetados tomando como base o modo de interação dos indivíduos (VIEIRA, 2003). Dependendo do tipo de interação realizada entre os indivíduos, cada tipo de informação para percepção tem mais ou menos importância (ARAÚJO, 2000). As informações para percepção devem ser obtidas, distribuídas e apresentadas aos participantes da interação por esses mecanismos (SOHLENKAMP, 1998).

O modo de como são obtidas as informações para percepção, ou informações de contexto (ROSA, BORGES, SANTORO, 2003), deve ser realizado de maneira a não privar os participantes de compreender o estado da interação que determinada informação pretende capturar. Por exemplo, é desejável que, se algum indivíduo deixou uma interação colaborativa, a informação de quem se retirou seja capturada e distribuída aos demais participantes, para que possam ter uma noção completa desta ação. Uma preocupação similar deve ser adotada com relação a como apresentar essas informações aos indivíduos de uma interação colaborativa: a forma com que estas devem ser apresentadas deve, além de representá-las por completo, ser a mais clara e objetiva possível, evitando que os indivíduos não fiquem cientes sobre essas informações.

2.3.2.1 – Mecanismos para Apoio à Percepção

Nesta dissertação, o termo **mecanismo para apoio à percepção** é usado para descrever como informações de percepção são geradas e levadas aos participantes de uma interação colaborativa. Esses mecanismos devem oferecer indicações sobre acontecimentos relevantes na interação, destacando o que, como, quando, onde e quem as realizou, procurando balancear o quanto deve ser apresentado desses acontecimentos (PINHEIRO, 2001).

Nem todo o acontecimento ocorrido numa interação colaborativa pode ser, para seus participantes, informação útil para os envolvidos (DAVID e BORGES, 2001); em uma edição colaborativa de documentos, por exemplo, os indivíduos envolvidos nessa atividade não precisam saber se determinado participante está falando ao telefone ao mesmo tempo em que realiza a atividade. Esta informação só seria interessante se o mediador dessa interação quisesse conhecer o motivo de uma eventual queda de produtividade deste indivíduo. Assim, os mecanismos para apoio a percepção devem levar em conta o papel que desempenha cada um dos participantes da interação (BORGES e PINO, 1999). Cabe ao projetista dos mecanismos selecionar, nas etapas de

coleta, distribuição e apresentação de informação para percepção, o que é relevante para o grupo em questão (SOHLENKAMP, 1998).

Em (VIEIRA, 2003), os mecanismos para apoio à percepção são classificados em três categorias principais :

- **Componentes visuais para percepção (*widgets*)**: visam apoiar a apresentação da informação para percepção em diferentes aplicações.
- **Mecanismos de percepção de domínio específico**: tratam da coleta, distribuição e apresentação das informações para percepção de uma aplicação em particular. São construídos de forma bastante acoplada às aplicações, sendo trabalhoso sua extração e transporte para outras aplicações.
- **Mecanismos de percepção de propósito genérico**: também atuam na coleta, distribuição e apresentação de informações para percepção, sendo projetados a atender diversas aplicações, em diferentes contextos. Estabelecem requisitos genéricos de mecanismos para apoio à percepção.

2.3.2.2 – Modos de Interação

Interações colaborativas podem ser observadas por duas dimensões: tempo e espaço. Grande parte da literatura de *CSCW* classifica as interações por essas duas dimensões: interações que ocorrem ao mesmo tempo são chamadas **síncronas**, enquanto as que ocorrem em tempos distintos são chamadas **assíncronas**; quando essas interações são realizadas em locais distintos, as interações **síncronas** e **assíncronas** são também **distribuídas** (ELLIS, GIBBS, REIN, 1991).

Em (MOLLI et al., 2002), é apresentada uma classificação de interações colaborativas, levando em consideração o(s) objeto(s) de trabalho destas (isto é, explorando mecanismos para apoio à percepção de mudanças). As próximas subseções apresentam esta classificação, sendo que a Tabela 2 apresenta um resumo desta.

	Mesmo Tempo		Tempos Diferentes	
	Mesmo Dado	Cópia do Dado	Mesmo Dado	Cópia do Dado
Mesmo Local	Face-a-Face	Multi-Síncrono	Assíncrono	Multi-Síncrono
Locais Diferentes	Síncrono Distribuído	Multi-Síncrono	Assíncrono Distribuído	Multi-Síncrono

Tabela 2 - Tipos de Interação em *Groupware*

2.3.2.2.1 – Interações Síncronas

Os indivíduos participantes desta interação trabalham ao mesmo tempo sobre o(s) mesmo(s) dado(s). Ações realizadas nestes dados compartilhados são identificadas, em tempo real, por todos os indivíduos. Características desse tipo de interação são apresentadas em (PINHEIRO, 2001), baseadas nas questões "O que", "Quando", "Onde", "Como" e "Quem":

- Informações sobre atividades devem ser fornecidas em um maior nível de detalhe possível, para que os indivíduos se contextualizem no momento que estas ocorrem.
- A apresentação das informações para percepção, que são relacionadas a ações do presente e do futuro, deve ser imediata, com tempo de utilidade baixo.
- A metáfora de "escritório virtual" pode ser usada, para tentar capturar detalhes com relação ao ambiente de trabalho compartilhado.
- Interfaces do tipo *WYSIWIS* (*What You See Is What I See*) e *WYSIWIS* relaxado, com recursos de múltiplas visões, podem ser utilizadas, já que esse tipo de interação requer que os participantes desta tenham uma noção uniforme do desenrolar da atividade em questão.
- Deve existir a noção de presença dos participantes da interação, pois está associada, por exemplo, à autoria de ações na atividade relativa à interação e a identificação se determinado participante está atento ou não à atividade.

2.3.2.2.2 – Interações Assíncronas

Os indivíduos participantes desta interação trabalham ao mesmo tempo ou em tempos diferentes sobre o(s) mesmo(s) dado(s). Ações realizadas nesses dados compartilhados são identificadas, em tempo real, por determinado indivíduo, se ele estiver participando da interação no momento da realização de determinada ação, ou até que esse retorne a interagir sobre esses dados. Características desse tipo de interação são apresentadas em (PINHEIRO, 2001), baseadas nas questões "O que", "Quando", "Onde", "Como" e "Quem":

- Informações sobre atividades podem ser fornecidas em um menor nível de detalhe em relação à interação síncrona, pois não existem garantias de quando determinado participante realizará a atividade. Neste caso, é mais interessante representar uma noção global dos acontecimentos relativos à interação, como: quem está responsável por qual tarefa, que parte do trabalho está sob os cuidados de quem, etc
- A apresentação das informações para percepção, que são relacionadas a ações do passado, as que começaram no passado e ainda estão ocorrendo e informações que podem vir a ocorrer no futuro, com tempo de utilidade maior do que as interações síncronas. Este tempo de utilidade pode ser determinado pelo tempo que os participantes tomarem ciência dessas informações.
- A metáfora de "sistemas mono-usuário" pode ser usada, para tentar capturar detalhes relacionados aos objetos de trabalho compartilhados.
- Interfaces do tipo *WYSIWIS* relaxado, com recursos de múltiplas visões, podem ser utilizadas, em casos onde o ponto central da atividade cooperativa é o conteúdo da atividade.
- A noção de presença dos participantes da interação aparece como uma oportunidade para colaboração, pois não existe uma necessidade intrínseca, nesta interação, do conhecimento da informação de presença pelos participantes.

Alguns mecanismos para apoio à percepção, em interações assíncronas, destinados ao papel de coordenador, são apresentados em (BORGES e PINO, 1999). Dentre estes mecanismos, podemos destacar: (i) **Participômetro**, que indica o grau de participação de um indivíduo na interação; (ii) **Densidade de Contribuições**, que indica o grau de contribuições de um indivíduo para realização da atividade; (iii) **Impactômetro**, que tenta mensurar o grau de impacto de uma dada contribuição no grupo.

2.3.2.2.3 – Interações Multi-Síncronas

Esta interação se caracteriza pelos seus participantes trabalharem com cópias do dado comum e, de tempos em tempos, as sincronizam com uma cópia armazenada eu um repositório central, usando uma política de *check-out* e *check-in* (MURTA, 2004). Dessa forma, se deseja explorar o isolamento positivo entre os participantes da

interação, isto é, aproveitar o paralelismo em realizar a atividade, que essa abordagem oferece (SARMA, NOROOZI, VAN DER HOEK, 2003).

Alguns sistemas de controle de versões de software (MURTA, 2004) utilizam esse tipo de interação em sua utilização, como é o caso dos sistemas *CVS* (FOGEL e BAR, 2001) e do *Subversion* (SUBVERSION, 2004). Esse tipo de interação pode ser encontrado na literatura com o nome de **colaboração autônoma** (EDWARDS e MYNATT, 1997) (BERGER, VOLKSEN, SCHILL, 1998).

2.3.2.3 – Geração de Informação para Percepção

Para que mecanismos para apoio à percepção possam coletar as informações relevantes em uma interação colaborativa, distribuí-las e apresentá-las aos participantes dessa, é necessário definir o modo de geração dessas informações. Por exemplo, em uma edição colaborativa de um documento texto, é necessário que, quando algum participante realiza alterações no documento compartilhado, estas sejam reconhecidas pelo mecanismo como informações relevantes.

Em (DOURISH e BELLOTTI, 1992), são apresentadas três formas de se extrair essas informações: (i) **informacional**, em que os indivíduos fornecem aos demais, explicitamente, informações sobre as atividades que estão realizando; (ii) **restrita a papéis**, em que indivíduos se limitam a fornecer informações sobre as atividades baseadas em um conjunto de ações, referentes a papéis na interação; (iii) ***feedback* compartilhado**, onde as informações para percepção são geradas a partir do *feedback* de operações realizadas pelos participantes.

Quando um participante da interação colaborativa informa aos demais, explicitamente, sobre suas atividades (seja ela informacional ou restrita a papéis), esta informação pode não ser provida de forma adequada. Dependendo do volume de ações realizadas em uma atividade colaborativa, os participantes podem se esquecer do que já realizaram, ou até fornecer informações que causem mal-entendidos (VIEIRA, 2003). O indivíduo que provê a informação é sobrecarregado, pois além de ter que realizar a atividade relativa à interação em si, deve intercalá-la com a tarefa de gerar a informação (DOURISH e BELLOTTI, 1992).

Essa sobrecarga de trabalho é atenuada na forma de obtenção da informação para percepção por meio de *feedback* compartilhado; o próprio ato de interagir já "produz" essas informações. Geralmente, essa informação é obtida através da

manipulação de artefatos compartilhados, onde um histórico de mudanças é criado e mantido para consulta pelos demais participantes da interação (VIEIRA, 2003) .

2.3.2.4 – Sobrecarga de Informações e Violação de Privacidade

Quando informações para percepção são apresentadas aos participantes de uma interação colaborativa, estas devem contextualizar, o melhor possível, estes sobre o acontecimento que originou esta informação. É desejável que essa contextualização seja feita interferindo o mínimo possível no fluxo de trabalho dos participantes; excesso de informações pode os distrair (SOHLENKAMP, 1998), enquanto a ausência de informações pode deixá-los perdidos na interação, sem um contexto para realização da atividade (PINHEIRO, 2001). Balancear a quantidade de informações a serem apresentadas aos participantes de uma interação colaborativa é um dos grandes desafios a serem enfrentados por mecanismos para apoio à percepção (ARAÚJO, 2000).

Uma alternativa interessante para atenuar o problema de sobrecarga de informações é utilizar o conceito de filtros, que possam selecionar informações relevantes e agrupá-las, de acordo com as necessidades dos participantes da interação (ARAÚJO, 2000) (DAVID e BORGES, 2001) (PINHEIRO, 2001).

Mesmo utilizando a metáfora de filtros na origem da informação e na apresentação desta (DAVID e BORGES, 2001), a interação é monitorada como um todo, para inferir qual ação ocorrida na interação deve ser disponibilizada aos participantes desta. Assim, pode existir, por parte dos membros da interação, a sensação de estarem sendo monitorados, controlados por outra pessoa (PINHEIRO, 2001). A ausência de informações também pode levar a esse sentimento de violação de privacidade; se determinado indivíduo requer a ajuda de outro, sem saber se este está disponível ou habilitado para tal, esta fato pode ser encarado como violação de privacidade e interrupção indesejada (ARAÚJO, 2000).

2.4 – Percepção aplicada ao Desenvolvimento de Software

O cenário ilustrado pela Figura 3 descreve um exemplo de equipes de desenvolvimento de software, que apresentam o problema de, eventualmente, não poderem interagir em um mesmo local físico ou em um mesmo período de tempo. Esta interação é necessária pois estes indivíduos necessitam trabalhar sobre cópias de um mesmo artefato de software, que, de tempos em tempos, são sincronizadas com uma cópia armazenada em um repositório de controle de versões. Durante essa atividade, os

desenvolvedores necessitam saber informações relativas ao trabalho dos demais, para que, no momento de sincronismo com a cópia do artefato armazenada no repositório, não haja um esforço considerável para tornar esta consistente.

Neste cenário, o único momento em que um determinado desenvolvedor fica ciente sobre as atividades dos demais sobre o artefato de software compartilhado é quando este atualiza sua cópia com a cópia armazenada no repositório de controle de versões (*check-out*). No intervalo em que os desenvolvedores estão sem atualizar a cópia do repositório de controle de versão (*check-in*), as informações relacionadas à manipulação sobre o artefato compartilhado poderiam ser disponibilizadas a todos os membros da equipe de desenvolvimento envolvidos na atividade.

Conforme dito anteriormente, o conceito de **percepção** pode ser útil para atenuar a sensação de isolamento entre os participantes. Mais especificamente, a idéia de **percepção de mudanças** sobre o artefato compartilhado de software parece apropriada, já que o próprio ato de manipulá-lo pode oferecer informações relevantes aos desenvolvedores envolvidos na atividade, desde que mecanismos para apoio a este tipo de percepção sejam providos pela ferramenta utilizada no grupo de desenvolvedores em questão.

Nas subseções a seguir, serão apresentadas algumas abordagens pesquisadas na literatura que tratam de alguns aspectos encontrados em um contexto de desenvolvimento de software global, em que um dado artefato compartilhado de software é manipulado de maneira concorrente. Um resumo das abordagens pesquisadas é apresentado, destacando características que serviram de inspiração para a elaboração desta dissertação.

2.4.1 – COD2DE

No contexto de edição colaborativa de artefatos, a ferramenta CO2DE (MEIRE, BORGES, ARAÚJO, 2003) se apresenta como uma implementação de editor gráfico e síncrono de diagramas UML, fornecendo mecanismos de percepção da evolução "colaborativa" do artefato editado. Esse mecanismo é baseado no conceito de máscaras, onde versões do diagrama editado são criadas sobre outras previamente estabelecidas.

Novas versões do diagrama compartilhado são criadas colocando-se uma "lâmina" sobre a anterior, em que vão se dar mudanças inerentes à nova versão, dando a impressão de uma máscara. Utilizando esse conceito, para se obter versões anteriores a determinada versão do diagrama, basta remover as "lâminas" que estão sobre este.

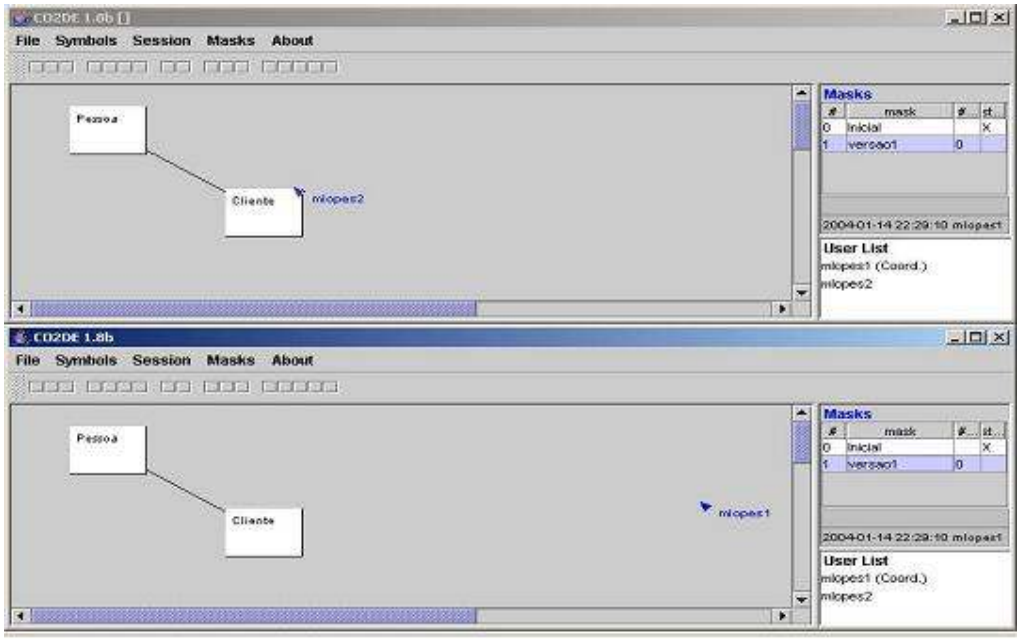

Figura 4 - Dois participantes sobre a mesma máscara (MEIRE, BORGES, ARAÚJO, 2003)

2.4.2 – D-UML

O *framework D-Meeting* (BOULILA, DUTOIT, BRUEGGE, 2003) destina-se a dar suporte a colaboração síncrona entre desenvolvedores de software. A idéia que permeia esse *framework* é que todos os componentes constituintes de uma sessão de trabalho sejam formalmente modelados como componentes de software, e que possam ser usados como estratégias "acopláveis". Os componentes pertencentes a este framework são: (i) Reunião, (ii) Percepção, (iii) Modelagem, (iv) Controle de Concorrência e (v) Construção de Raciocínio (*Design Rationale*).

É apresentada uma instância do *framework* (*D-UML*), baseada no padrão MVC distribuído. Um modelo *UML* é compartilhado por diversos desenvolvedores, que utilizam visões pré-estabelecidas para análise deste. Dentre as diversas informações providas por essas visões, pode-se destacar quais desenvolvedores estão participando da sessão, bem como qual deles está "agindo" no momento, além de ser possível repetir a seqüência de ações realizadas pelos desenvolvedores até o estágio corrente da sessão.

2.4.3 – SAMS

O ambiente *SAMS* (MOLLI et al., 2002) é resultado da combinação dos três tipos de interações em *groupware* citados anteriormente (síncrona, assíncrona e multi-síncrona), onde cada colaborador trabalha com uma cópia do dado compartilhado.

Operações realizadas no dado pelos demais colaboradores são enfileiradas, cabendo ao colaborador decidir se deseja trabalhar em modo síncrono. Se sim, os eventos recebidos e propagados são automaticamente integrados e propagados, respectivamente. Se não, cabe ao colaborador decidir o momento de propagação e integração. Quando um evento é propagado para algum colaborador que não esteja conectado ao ambiente, esses são armazenados em uma fila de mensagens para envio posterior (modo assíncrono).

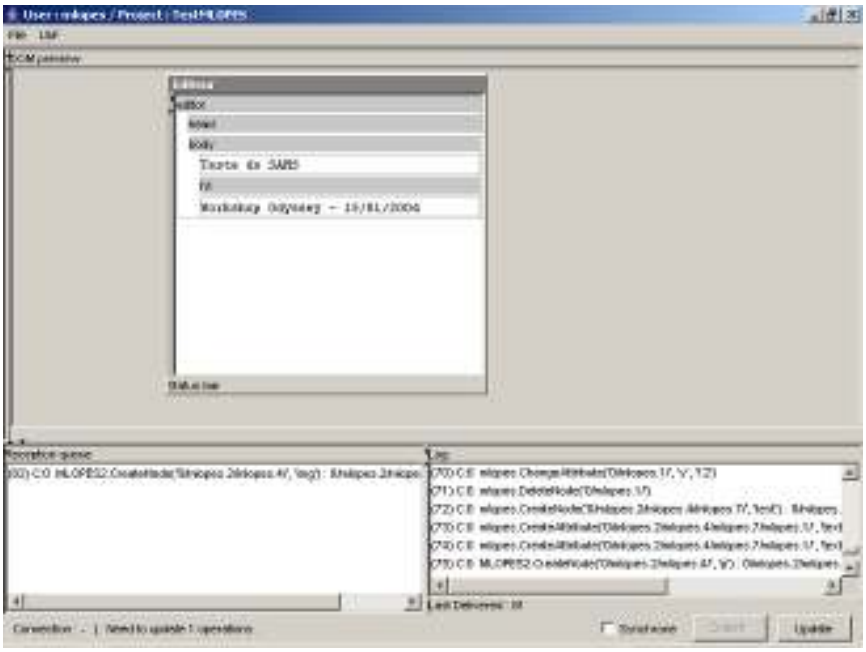

Figura 5 - Ambiente *SAMS*

2.4.4 – NetEdit

NetEdit (ZAFFER et al., 2001) é um editor colaborativo de documentos, acessível pela WEB. Utiliza uma arquitetura semi-replicada, com dados e processo distribuídos por todos os clientes da colaboração, coordenados por um servidor central. Provê percepção de colaboração através de visão de radar e teleapontador (GUTWIN E GREENBERG, 1999 APUD: MANGAN, 2003), tendo como metáfora de ação a utilização de mensagens, que são transmitidas entre clientes e servidor; ações (mensagens) locais são imediatamente processadas, enviadas ao servidor e retransmitidas aos demais clientes.

São utilizados em *NetEdit* mecanismos de sincronização, onde esta é obtida analisando as ações que ocorreram em uma par de estações de trabalho, através de um grafo de espaço de estados. Este conceito é expandido em um protocolo (utilizado pelo *NetEdit*) que utiliza múltiplas vezes a sincronização entre pares de estações clientes,

para obter a sincronização do documento compartilhado entre todos os participantes da colaboração.

Para preservar a finalidade de cada ação gerada, as mensagens, quando são recebidas pelos clientes ou pelo servidor *NetEdit*, sofrem transformações, para atingir a convergência das versões. Isto é realizado porque a ordem de execução das mensagens recebidas interfere no resultado gerado. Neste sentido, a execução de uma ação antes de uma outra pode interferir na última.

2.4.5 – Palantír

A ferramenta *Palantír* (SARMA, NOROOZI, VAN DER HOEK, 2003) destina-se a complementar sistemas de gerência de configuração, fornecendo aos desenvolvedores pertencentes à sessão de colaboração informações sobre os espaços de trabalho dos demais. Essas informações são conhecidas em três pontos específicos em sistemas de gerência de configuração tradicionais: quando o desenvolvendor obtém o artefato do repositório e o coloca no espaço de trabalho, quando (re) posiciona o artefato no repositório e quando consulta o repositório.

A proposta de *Palantír* é aumentar a percepção dos desenvolvedores, provendo o compartilhamento contínuo da informação. De uma forma simplista, é possível dizer que *Palantír* monitora as aplicações clientes de um sistema de gerência de configuração, capturando eventos relevantes e os distribuindo para os demais espaços de trabalho compartilhado.

Figura 6 - Ferramenta *Palantír*

2.4.6 – Tukan

TUKAN (SCHÜMMER e SCHÜMMER, 2001) é um ambiente síncrono e distribuído de programação *SMALLTALK*, onde são aplicados alguns resultados de pesquisas em *groupware* ao domínio de *eXtreme Programming*. Dentre essas funcionalidades, podemos destacar os mecanismos para apoio à percepção, onde elementos contidos no ambiente (por exemplo, classes e métodos) são dispostos em um "espaço", utilizando as metáforas de foco (item do espaço visualizado) e aura (itens ao "redor" do item visualizado).

Os elementos são dispostos no espaço de acordo com relacionamentos de estrutura (herança, composição), uso e versionamento. A aura dos elementos, isto é, a disposição estrutural dos elementos em relação a determinado elemento, é calculada dando-se um valor para distância entre pares destes. São utilizadas representações em cores para representar o quão perto o mais próximo usuário (isto é, o usuário que está trabalhando sobre algum elemento da aura do elemento focado) está do elemento focado. Além dessa representação, são utilizadas imagens que representam tempos meteorológicos para indicar o impacto de mudança em cada elemento da aura de um elemento focado sobre este.

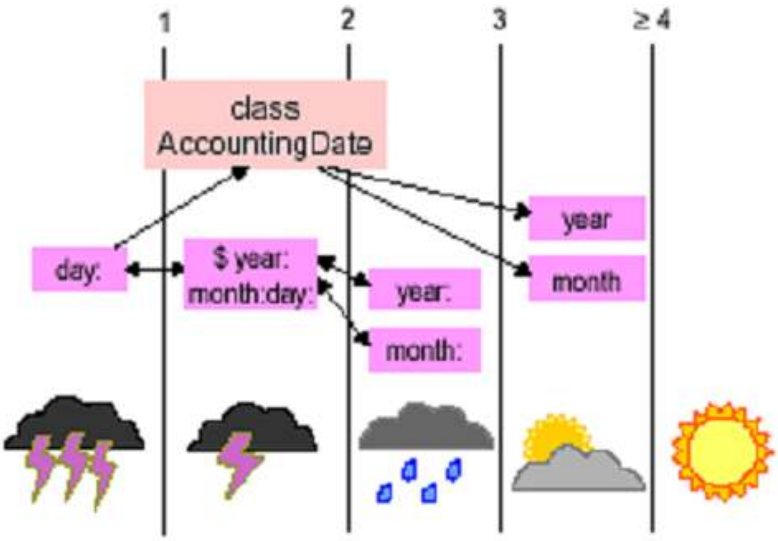

Figura 7 - Metáfora de Tempo Meteorológico (SCHÜMMER e SCHÜMMER, 2001)

2.4.7 – Abordagem Kobylinski

Um sistema de percepção que permite que colaboradores monitorem atividades de outros sobre artefatos de software é apresentado em (KOBYLINSKI et al., 2002).

Neste sistema, colaboradores podem se "inscrever" para serem notificados quando: um artefato é modificado; determinado colaborador realizou uma atividade; determinado colaborador realizou atividades relacionadas a determinado aspecto do software em desenvolvimento (para fins de contextualização da percepção).

A informação de percepção é coletada através de eventos gerados pelas ferramentas utilizadas pelos desenvolvedores e distribuída a quem for de interesse (isto é, os demais desenvolvedores interessados na informação atrelada aos eventos), sendo os eventos filtrados de acordo com a importância destes para cada desenvolvedor, que deve mostrar explicitamente seu nível de interesse em eventos relacionados a pessoas, comandos, artefatos, ferramentas (ou a combinação desses elementos) constituintes do processo de desenvolvimento de software. Para cada elemento desse, são associados um ou mais eventos, tornando possível expressar o interesse de um desenvolvedor em um elemento em particular, tornando a percepção contextualizada.

2.4.8 – iScent

O *framework iScent* (ANDERSON e BOUVIN, 2000) destina-se a construção de aplicações que provêem percepção e intersubjetividade entre indivíduos de uma equipe distribuída de desenvolvimento de software. Este *framework* é baseado na tecnologia *WEB*, utilizando também tecnologias do domínio de hipermídia e sistemas de notificação de eventos, tendo como premissa que as ações realizadas por um engenheiro de software são elementos de conversação, já que é do comportamento humano que a comunicação entre indivíduos deve ser realizada de maneira que cada parte saiba que está sendo compreendida (intersubjetividade).

A intersubjetividade entre os membros da equipe distribuída é atingida da seguinte forma: quando um colaborador A deseja ser notificado sobre um evento que o colaborador B gera, o primeiro cria um "gatilho" para o evento relativo (*watchdog*). Este "gatilho" busca no "rastro" de eventos do colaborador B (através do repositório de eventos deste) uma instância do evento desejado. Quando tal evento ocorre, o colaborador A é notificado, e o colaborador B é informado que A foi notificado.

2.4.9 – Resumo das Abordagens

Um resumo das abordagens pesquisadas é apresentado na Tabela 3, respondendo as seguintes perguntas sobre suas características:

- **Edição colaborativa**: A abordagem apresentada se trata de uma ferramenta de edição colaborativa de artefatos?
- **Apoio à modelagem de artefatos *UML***: O objeto compartilhado da interação colaborativa é um artefato de software representado pela linguagem *UML*?
- **Modos de interação**: Como é realizada a interação entre indivíduos na abordagem apresentada?
- **Importância de alterações**: É possível inferir o quão importante determinada informação é para os participantes da interação colaborativa?
- **Ciência sobre alterações**: É possível saber se determinado participante da interação está ciente sobre a ocorrência e do conteúdo de determinada informação?

	COD2DE	D-UML	SAMS	NetEdit	Palantir	Tukan	Abordagem Koblinsky	iScent
Editor Colaborativo	●	●	●	●	○	●	○	○
Artefatos UML	●	●	○	○	○	○	●	○
Abordagem não intrusiva	○	○	○	○	●	○	●	●
Interação Síncrona	●	●	●	●	●	●	●	●
Interação Assíncrona	●	○	●	○	●	○	●	●
Interação Multi-Síncrona	○	○	●	○	●	○	●	●
Relevância de Informações	○	○	○	○	●	●	●	○
Ciência sobre Informações	○	○	○	○	○	○	○	●

Legenda : ● Presente ○ Ausente

Tabela 3 - Resumo das Abordagens

2.5 – Considerações sobre Desenvolvimento de Software e Percepção

No contexto de desenvolvimento globalizado de software, utilizar mecanismos para apoio à percepção de mudanças em objetos compartilhados aparece como uma alternativa interessante para reduzir o isolamento negativo entre desenvolvedores de uma mesma equipe. Este isolamento é caracterizado por desenvolvedores, durante a atividade colaborativa de manipulação de um artefato compartilhado, desconhecerem o que outros estão fazendo sobre este (SARMA, NOROOZI, VAN DER HOEK, 2003). Este tipo de mecanismo para apoio à percepção pode interferir pouco no fluxo de trabalho dos desenvolvedores, já que a informação pode ser obtida pelo próprio histórico de mudanças realizadas sobre o artefato compartilhado (TAM e GREENBERG, 2004).

Oferecer a estes indivíduos informações originadas por alterações realizadas sobre um artefato compartilhado permite inferir sobre qual direção está se dando a atividade colaborativa de manipulação do artefato compartilhado; por exemplo, ações realizadas por um determinado desenvolvedor podem fornecer indícios sobre a compreensão deste sobre o problema a ser resolvido pela atividade em questão. Conhecer as ações dos demais desenvolvedores, participantes da interação, pode evitar que uma ação realizada por um determinado indivíduo seja comprometida por ações de outros, pois permite que os desenvolvedores se baseiem no histórico de modificações do artefato compartilhado para tomarem futuras decisões.

Esta oferta de informações para percepção pode não atingir seu objetivo se não houver um balanceamento de que informações são relevantes a cada indivíduo e em cada momento. Sobrecarregar os desenvolvedores com informações irrelevantes a eles possibilita que estes desviem o foco do trabalho que está sendo realizado, ou até mesmo abandone a utilização do mecanismo.

Capítulo 3 – Proposta de um Mecanismo para Apoio à Percepção Aplicado à Modelagem de Software

3.1 – Introdução

Conforme visto no capítulo anterior, mecanismos para apoio à percepção de mudanças em artefatos compartilhados podem ser utilizados para reduzir o isolamento negativo (SARMA, NOROOZI, VAN DER HOEK, 2003) entre desenvolvedores. Estes mecanismos visam atenuar o problema de ausência de informações, por parte de um desenvolvedor, sobre ações realizadas pelos demais sobre um mesmo artefato compartilhado de trabalho. A Figura 8 descreve o cenário apresentado no capítulo anterior, acrescentado da utilização de um mecanismo para apoio à percepção. Os artefatos compartilhados possuem dependência entre si (setas pretas finas), sendo manipulados (setas pretas grossas) por um grupo de desenvolvedores. Estes desenvolvedores obtêm informações sobre mudanças ocorridas nos artefatos compartilhados através do mecanismo (setas cinza claro), que é abastecido destas por mudanças realizadas nos artefatos (setas cinza escuro).

Neste cenário, um mecanismo para apoio à percepção pode ser utilizado da seguinte maneira: ações realizadas sobre as cópias dos artefatos compartilhados são capturadas, distribuídas e apresentadas aos desenvolvedores que trabalham sobre estes. Assim, cada alteração realizada sobre alguma cópia de determinado desenvolvedor torna-se de conhecimento de todos os demais que participam da interação. De posse dessas informações, os desenvolvedores podem se basear nestas para a realização de futuras ações. Desta forma, ter o conhecimento global das ações realizadas pelos desenvolvedores que estão interagindo sobre os mesmos artefatos (ou artefatos afins) torna possível que determinado desenvolvedor avalie cada ação, visando diminuir o esforço na convergência das cópias dos artefatos compartilhados.

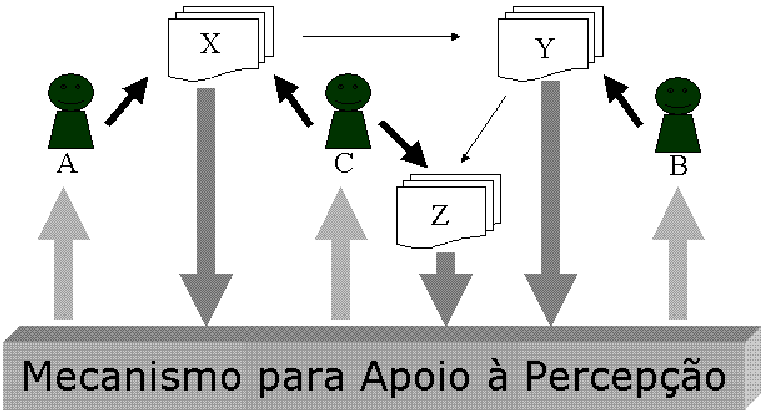

Figura 8 - Cenário de Uso com Mecanismo para Apoio à Percepção

Este capítulo apresenta a proposta do mecanismo para apoio à percepção ***MAIS*** (do inglês *Multi-Synchronous Awareness Infrastructure*). O mecanismo tem como objetivo principal apoiar desenvolvedores, que trabalham sobre cópias de um modelo compartilhado de software. Em particular, o mecanismo permite que um desenvolvedor, em um momento anterior à convergência (VIDOT et al., 2000), compare o estado da sua cópia local com o estado global do modelo, antevendo possíveis problemas de inconsistências ou conflitos. Este estado global contempla as contribuições individuais dos desenvolvedores, representando o modelo compartilhado em si, armazenado em um repositório centralizado.

O capítulo está organizado da seguinte forma: a Seção 3.2 apresenta as características gerais observadas para o desenvolvimento do mecanismo em questão. Na Seção 3.3 são descritos os requisitos identificados e restrições estabelecidas para a concepção do mecanismo proposto. Características, funcionalidades e arquitetura do mecanismo ***MAIS*** são apresentadas na Seção 3.4. Comparações das abordagens pesquisadas com o mecanismo ***MAIS***, bem como um resumo deste, são feitas na Seção 3.5.

3.2 – Decisões de Projeto Relacionadas ao Mecanismo

Para o desenvolvimento de um mecanismo que provê informações sobre mudanças em cópias de artefatos compartilhados, algumas decisões necessitam ser tomadas com relação a certas características que este deve apresentar. Estas características foram observadas em trabalhos existentes na literatura e serviram como

base para a definição do escopo da proposta de mecanismo para apoio à percepção apresentada nesta dissertação.

3.2.1.1 – Tipo de Artefato Compartilhado

Definir o tipo de artefato que o mecanismo deve dar suporte é necessário para fornecer informações mais específicas quanto ao seu estado. Desta forma, é possível prover informações com mais detalhes do que apenas informar que um determinado artefato foi alterado por alguém em um dado momento. Dentre os tipos de artefatos que são elaborados em um processo de desenvolvimento de software orientado a objetos, é possível dividi-los em três categorias: (i) documento texto, (ii) modelo de software e (iii) código-fonte.

Abordagens que tem como artefato suportado documentos texto, como *NetEdit* (ZAFFER et al., 2001) e *Microsoft Word* (MICROSOFT, 2004a), são interessantes no que diz respeito ao nível de abstração que tratam (por exemplo, especificação de requisitos). Geralmente, estas abordagens não exploram a semântica do artefato, pois se limitam apenas a tratar de modificações relacionadas ao texto que este possui. Em contrapartida, abordagens baseadas em código fonte, como *Coven* (CHU-CARROLL e SPRENKLE, 2000) e *Tukan* (SCHÜMMER e SCHÜMMER, 2001), oferecem o suporte à percepção se valendo da estrutura do artefato. Sendo assim, é interessante que o artefato escolhido esteja em um nível de abstração maior do que código fonte, mas que tenha uma estrutura semântica melhor definida do que documentos texto.

Um mecanismo que se baseasse em modelos de software surge como uma abordagem interessante para processos de desenvolvimento de software orientados a objeto. Neste paradigma, várias atividades são de elaboração de documentos baseados na *UML* (*Unified Modelling Language)* (OMG, 2004b). Aliado a esse fator, técnicas para transformação de modelos de software como o MDA (*Model Driven Architecture*) (OMG, 2004a) tem como uma das possíveis representações de modelos os baseados em *UML*.

3.2.1.2 – Consistência dos Artefatos Compartilhados

Um dos fatores predominantes para utilização de mecanismos para apoio à percepção de mudanças em artefatos compartilhados é vislumbrar conflitos entre cópias destes, que podem ocorrer no momento de convergência dessas cópias em um estado global. Através do conhecimento do que foi alterado em determinada cópia do artefato

compartilhado, é possível antever, comparando com outra cópia deste, que conflitos ocorrerão na convergência.

Para tratar o problema de consistência das cópias de um dado artefato compartilhado, políticas pessimistas e otimistas de **controle de concorrência** podem ser adotadas (GREENBERG e MARWOOD, 1994) (ESTUBILIER, 2001 APUD: MURTA, 2004): na abordagem pessimista, o artefato compartilhado (ou parte dele) é editado por um desenvolvedor durante um certo período de tempo, em que a manipulação do artefato compartilhado torna-se exclusiva dele. Cada desenvolvedor requisita o acesso ao artefato, trabalha em uma cópia deste e converge esta cópia em um estado global. Desta forma, as inconsistências ocasionadas por múltiplas edições simultâneas a cópias de um artefato não ocorrerão. Na abordagem otimista, as cópias do artefato compartilhado são manipuladas em paralelo por diversos desenvolvedores e, em um certo momento, são unificadas de modo a estabelecer um estado global do artefato.

Apesar da abordagem pessimista evitar que inconsistências geradas por múltiplas edições simultâneas sobre um artefato compartilhado não ocorrerão, o tempo que cada desenvolvedor terá que esperar para ter acesso de manipulação ao artefato compartilhado pode interferir no seu fluxo de trabalho (GREENBERG e MARWOOD, 1994). Além disso, conflitos gerados por inconsistências entre cópias de um artefato compartilhado são oportunidades para os usuários envolvidos (isto é, os desenvolvedores de software) interagirem entre si (BORGES e PINO, 2000). Por exemplo, uma inconsistência ocasionada em um modelo *UML* compartilhado pode indicar que dois desenvolvedores compreenderam determinado requisito do software que está sendo modelado de maneiras diferentes.

Desta forma, a escolha de uma abordagem otimista para controle de concorrência é adequada para ser utilizada em um mecanismo de apoio à percepção de mudanças em artefatos compartilhados, pois oferece flexibilidade aos desenvolvedores com relação a como manipular determinado artefato compartilhado. Além disso, a própria utilização do mecanismo pode atenuar problemas de inconsistências que possam ocorrer, já que os desenvolvedores possuem a informação do que está sendo realizado pelos demais.

3.2.1.3 – Especificidade de Ferramenta de Modelagem

O trabalho dos desenvolvedores é realizado ou registrado com o uso de ferramental que permite a edição dos modelos de software. Mesmo que o modelo seja

impresso e alterado, a alteração só será registrada com o uso de uma ferramenta. A representação desse modelo permanece armazenada em um repositório centralizado.

Quanto à utilização de um mecanismo para apoio a percepção, em conjunto com ferramentas de modelagem, é possível destacar dois cenários: (i) as ferramentas usadas pelo desenvolvedor são mantidas e acrescidas de suporte à colaboração ou (ii) as ferramentas são substituídas por ferramentas colaborativas acrescidas de funcionalidades para apoio à tarefa de modelagem.

A mudança de estado, ocasionada por manipulações de um grupo de desenvolvedores, de determinado artefato compartilhado deve ser capturada, interpretada (isto é, identificar que tipo de mudança ocorreu e quais propriedades do artefato foram alteradas) e distribuídas aos demais. Na prática, a ferramenta em que o artefato compartilhado é manipulado pode não possuir a funcionalidade de identificar mudanças ocorridas na cópia local de um certo desenvolvedor.

O suporte a realização de tarefas, em ferramentas colaborativas, é menos completo do que às ferramentas similares de uso individual (LI e LI, 2002 APUD: MANGAN, 2003). Ferramentas que contemplam a manipulação de modelos *UML* (IBM, 2004) (BORLAND, 2004b) (GENTLEWARE, 2004) podem apresentar esta deficiência.

A troca de ferramentas de uso individual de manipulação de modelos *UML,* utilizadas em uma equipe de desenvolvimento de software, por outras que realizam esta tarefa e ainda possuem suporte colaborativo, pode prejudicar o andamento das atividades deste grupo. O tempo de aprendizado da utilização de novas ferramentas pode ser inadequado com as expectativas da organização onde está inserida esta equipe. Atrasos nos cronogramas estipulados, nesta situação, não compensam o benefício que o mecanismo traria.

Seria interessante que desenvolvedores pudessem trabalhar com ferramentas nas quais estão habituados e ainda ter o suporte à percepção de mudanças em artefatos compartilhados. A utilização do mecanismo para apoio à percepção seria independente da ferramenta em que os desenvolvedores trabalham. Este suporte seria introduzido de maneira transparente ao sistema (BEGOLE, ROSSON, SHAFFER, 1999) (LI e LI, 2002), isto é, sem a necessidade de alterar a ferramenta onde este será utilizado.

3.2.1.4 – Organização da Informação para Percepção

Dentre as modificações que são realizadas, algumas são de maior importância para cada desenvolvedor envolvido na interação. Dependendo da tarefa que determinado desenvolvedor esteja realizando, em um certo momento, sobre o artefato compartilhado (por exemplo, refinamento de um modelo de classes), as ações que estão de alguma maneira relacionadas com esta tarefa podem ser mais importantes do que as demais.

Conforme as informações sobre mudanças vão sendo geradas, é necessário que cada desenvolvedor identifique, dentre todas as mudanças geradas pelos demais desenvolvedores, quais as que realmente lhe interessam. Conforme o volume de mudanças realizadas aumenta, a identificação destas, por parte dos desenvolvedores, torna-se mais trabalhosa. Se mudanças ocorrem com alta freqüência, esta identificação pode prejudicar a manipulação das cópias do artefato compartilhado. Por exemplo, em um modelo de classes *UML*, se determinado desenvolvedor trabalha apenas sobre uma classe "A", este deve pesquisar, dentre todas as modificações realizadas por todos os desenvolvedores, quais as que se relacionam com a classe "A".

Neste procedimento de identificação de modificações que sejam importantes para determinado desenvolvedor, este irá encontrar modificações que, em um momento anterior, já tinha tomado conhecimento. Dependendo de quanto tempo dure a interação dos desenvolvedores sobre as cópias do artefato compartilhado, ou do volume de modificações realizadas, informações sobre mudanças já conhecidas pelos desenvolvedores podem ser a maioria no total de modificações.

Para atenuar este problema, mecanismos que provêm esse tipo de informação para percepção necessitam classificar as informações sobre mudanças, no momento de apresentá-las. Assim, dependendo dos critérios adotados para classificar as informações de mudanças, os desenvolvedores não são sobrecarregados com a quantidade de informação gerada (DAVID e BORGES, 2001). Além disso, devem ser registradas informações que desenvolvedores já tenham tomado conhecimento para, além de reduzir a sobrecarga cognitiva de informações, servir como fonte de consulta sobre o estado de percepção dos desenvolvedores envolvidos em uma interação de manipulação de um artefato compartilhado (BORGES e PINO, 2000) (DAVID e BORGES, 2001).

3.3 – Requisitos e Restrições

Os requisitos definidos para concepção do mecanismo de apoio à percepção proposto nessa dissertação foram identificados com base na revisão da literatura e nas considerações expostas na seção anterior. São eles:

- **Controle de Concorrência**: a abordagem **otimista** de controle de concorrência deve ser adotada no mecanismo proposto, baseando-se nas considerações apresentadas na Seção 3.2.1.2. Desta forma, evita-se restrições quanto ao acesso simultâneo ao artefato compartilhado, o que pode comprometer o **fluxo de trabalho** da equipe de desenvolvimento de software em questão.
- **Especificidade de Ferramenta de Modelagem**: o mecanismo não deve ser implementado visando à utilização em uma ferramenta específica. O objetivo é que o mecanismo possa ser acoplado em ferramentas de edição de artefatos de software que forneçam um conjunto de serviços que torne sua utilização viável. Em linhas gerais, o mecanismo deve ser reutilizável. Desta forma, o mecanismo deve atuar de maneira **não-intrusiva**, isto é, a ferramenta de edição de artefatos de software não deve necessitar ser reimplementada para ter o suporte do mecanismo. Caso a ferramenta de modelagem em questão não possua o conjunto de serviços que o mecanismo espera, a extensão desta, para contemplar estes serviços, deve ser facilmente realizada. A característica do mecanismo de ser não-intrusivo, em conjunto com a abordagem otimista, é uma tentativa de reproduzir o modo como os desenvolvedores realizariam o trabalho de modelagem em um contexto individual.
- **Coleta Passiva de Informações de Mudança**: relacionado com a característica do mecanismo de não comprometer o fluxo de trabalho dos desenvolvedores, as informações de mudança não devem ser explicitamente informadas, cabendo ao mecanismo coletá-las sem o auxílio dos desenvolvedores.
- **Organização da Informação para Percepção**: as informações para percepção oferecidas pelo mecanismo (isto é, mudanças ocorridas nos artefatos compartilhados de software) devem ser apresentadas aos desenvolvedores de maneira a evitar sobrecarga de informações.

Classificações e filtros devem ser adotados, visando guiar os desenvolvedores a buscar informações que, em determinado momento da interação sobre o artefato compartilhado, lhes são mais apropriadas. O mecanismo deve ser capaz de prover meios para identificar se desenvolvedores tiveram conhecimento sobre a realização de determinadas mudanças.

Algumas restrições foram estabelecidas para delimitar a abrangência da abordagem proposta:

- **Utilização de Modelos *UML***: as informações de mudança que o mecanismo deve tratar são as relacionadas com modelos *UML*. Além dos benefícios apresentados na Seção 3.2.1.1, aliado ao fato desta linguagem ser utilizada em grande escala na indústria de software, esta restrição foi imposta para ser possível associar informações de mudança, com o intuito de classificá-las. Por exemplo, definir que uma associação entre classes foi criada significa que a informação relativa a esta ação deve contemplar as classes envolvidas. Assim, é possível organizar as informações de mudanças geradas com base nessa informação, que diz respeito com a estrutura do artefato compartilhado.
- **Consistência dos Artefatos Compartilhados**: o mecanismo não deve tratar as inconsistências que modificações possam gerar sobre o artefato compartilhado, sejam estas inconsistências locais a cópias dos desenvolvedores ou projetadas em uma futura convergência destas. Cabe ao mecanismo organizar as informações de mudanças, deixando a identificação de inconsistências como uma tarefa relacionada à ferramenta de modelagem utilizada pelos desesenvolvedores, que geralmente dispõe desse tipo de serviço.
- **Persistência de Informações de Mudança**: para tornar possível obter e organizar informações sobre mudanças no artefato compartilhado, geradas em momentos anteriores ao da apresentação, deve ser configurável o tempo em que estas informações devem persistir. Estas informações devem possuir um nível de detalhe que seja possível realizar consultas gerenciais relativas ao processo de interação colaborativa que o mecanismo está apoiando.

3.4 – Abordagem *MAIS*

Nesta seção, é apresentada uma proposta de mecanismo para apoio à percepção, de nome ***MAIS*** (acrônimo de ***M****ulti-synchronous* ***A****wareness* ***I****nfra****S****tructure*) (LOPES, MANGAN, WERNER, 2004a; LOPES, MANGAN, WERNER, 2004b). As informações para percepção, neste mecanismo, são obtidas através de manipulação de artefatos compartilhados (TAM e GREENBERG, 2004), que são **coletadas**, **distribuídas** e **apresentadas** aos desenvolvedores que estão interagindo sobre este (SOHLENKAMP, 1998).

A abordagem *MAIS* trata de **modelos compartilhados de software**, representados em conformidade com a versão 1.4 da ***UML*** (OMG, 2004b). O mecanismo é baseado em informações para percepção obtidas pela manipulação de elementos de um modelo de software e trata das características apresentadas em 3.2 –.

O modo de interação entre os desenvolvedores, para manipulação de modelos compartilhados, no *MAIS* é o **multi-síncrono**. Esta forma de interação foi escolhda pois permite que os desenvolvedores trabalhem de maneira independente. O modelo compartilhado é manipulado através de cópias deste, locais a cada desenvolvedor. Desta maneira, é possível manipular este artefato de forma paralela. O mecanismo *MAIS*, desta maneira, atua mais especificamente na fase de **convergência** das cópias do modelo em um estado global, cuja cópia é armazenada em um repositório central. Neste trabalho, assume-se que a convergência das cópias do modelo compartilhado ocorre quando os desenvolvedores envolvidos na interação sobre estas desejam que suas contribuições sejam confirmadas.

A especificação do mecanismo *MAIS* é independente de ferramenta de modelagem; trata-se de um componente de software reutilizável (GRUNDY e HOSKING, 2002), sendo passível de acoplamento em editores ou ambientes que satisfaçam requisitos básicos de extensibilidade. Entendem-se como estes requisitos, nesta dissertação, funcionalidades (como por exemplo, uma *API* de programação) que ferramentas de modelagem devem possuir para que o mecanismo *MAIS* consiga inferir sobre mudanças de estado em determinado modelo.

O mecanismo proposto trata como representação de informação para percepção mudanças ocorridas em modelos compartilhados, utilizando a metáfora de **eventos**. Utiliza a idéia de **sistemas de notificação de eventos** (PRINZ, 1999) (FARSHCHIAN, 2001) (DE SOUZA, BASAVESWARA, REDMILES, 2002), em que desenvolvedores

são registrados para serem notificados quando determinado evento (isto é, alguma ação sobre o artefato compartilhado) ocorrer. Assim, os desenvolvedores que interagem sobre um mesmo artefato compartilhado de software agem de forma passiva na utilização do mecanismo, isto é, somente utilizam o mecanismo quando estimulados, em detrimento da ocorrência de um evento. Desta maneira, é realizada a distribuição de informações para percepção entre os desenvolvedores.

A coleta de informações é realizada através de **sensores**, que são providos pela ferramenta em que os artefatos compartilhados são manipulados e estão relacionados a elementos deste (no caso de um modelo *UML*) (PRINZ, 1999). Esses sensores capturam as informações de alteração de estado de elementos do artefato compartilhado e repassam às ferramentas "interessadas" em obter esse tipo de informação (MANGAN, 2003).

Esta Seção está organizada da seguinte forma: na sub-seção 3.4.1 são apresentados os passos para utilização do mecanismo *MAIS*. A sub-seção 3.4.2 descreve como os eventos do mecanismo são representados neste. Já a sub-seção 3.4.3 apresenta quais foram os conceitos usados na organização dos eventos gerados pelo mecanismo, para a apresentação aos desenvolvedores. A arquitetura do mecanismo *MAIS* é descrita na sub-seção 3.4.43.4.4 –.

3.4.1 – Cenário de Utilização do *MAIS*

Esta sub-seção apresenta os passos que compreendem a utilização do mecanismo por uma equipe de desenvolvimento de software, cujos membros interagem sobre um mesmo artefato compartilhado (modelo) e estão dispersos no espaço e no tempo. Esses passos são divididos nas macro-atividades de coleta, distribuição e apresentação de eventos para os desenvolvedores.

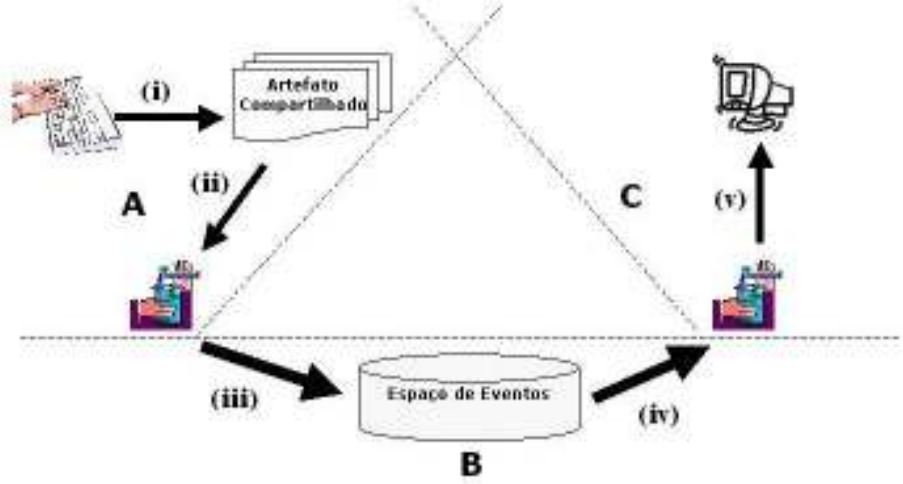

Figura 9 – Cenário de Utilização do Mecanismo *MAIS*

Na etapa de **coleta** dos eventos (**A)** , são realizados os passos:

(i) Os desenvolvedores realizam suas tarefas sobre cópias do artefato compartilhado de software.

(ii) As ações realizadas sobre as cópias resultam na produção de eventos.

A **distribuição** dos eventos (**B**) ocorre da seguinte maneira:

(iii) Um evento, relativo a mudança de estado de um elemento do artefato compartilhado, é disponibilizado no espaço de eventos do grupo.

(iv) Nesse instante, todos os desenvolvedores são notificados que determinado evento ocorreu. Essa mensagem de notificação contém o novo evento gerado.

Por fim, a **apresentação** dos eventos (**C**) é feita da seguinte forma:

(v) Dado que os desenvolvedores recebem o novo evento gerado, os eventos já recebidos (inclusive o novo) são organizados e (re)apresentados aos desenvolvedores.

3.4.2 – Representação de Eventos

Um evento, na abordagem proposta, é representado por uma **ação** (***H**ow*) sobre um **elemento** (*What*) do **modelo compartilhado** (*Where*). As ações são representadas em um **instante de tempo** (*When*), realizadas por determinado **desenvolvedor** (*Who*). Relações de pertinência também são representadas pelos eventos no *MAIS*, isto é, se

determinado evento é relacionado a um elemento que pertence a um outro, esta informação é associada ao evento.

O motivo da ação que gerou determinado evento não é representado nos eventos do *MAIS*. Essa informação, usualmente, é informada de maneira explícita pelos indivíduos que realizam modificações em artefatos compartilhados (TAM e GREENBERG, 2004). Se os desenvolvedores fornecerem, de maneira explícita, esse tipo de informação, o fluxo de trabalho destes pode ser comprometido. Inferir os motivos pelos quais ações foram realizadas pelos desenvolvedores sobre o modelo compartilhado de software está fora do escopo deste trabalho.

A Figura 10 representa o modelo de classes conceitual de como são representados os eventos no mecanismo *MAIS*. As especializações de "Evento" e "Elemento" são definidas no momento do projeto de implementação do mecanismo. Na Figura 10, estão representadas algumas das possíveis ações que podem ocorrer sobre modelos de software, bem como possíveis elementos de modelos *UML* que podem ser monitorados.

O modelo compartilhado onde ocorrem os eventos ("Onde") não é descrito nesta representação de eventos, pois a abstração deste modelo é representada, no *MAIS*, em uma porção do repositório de eventos (espaço de eventos) do grupo.

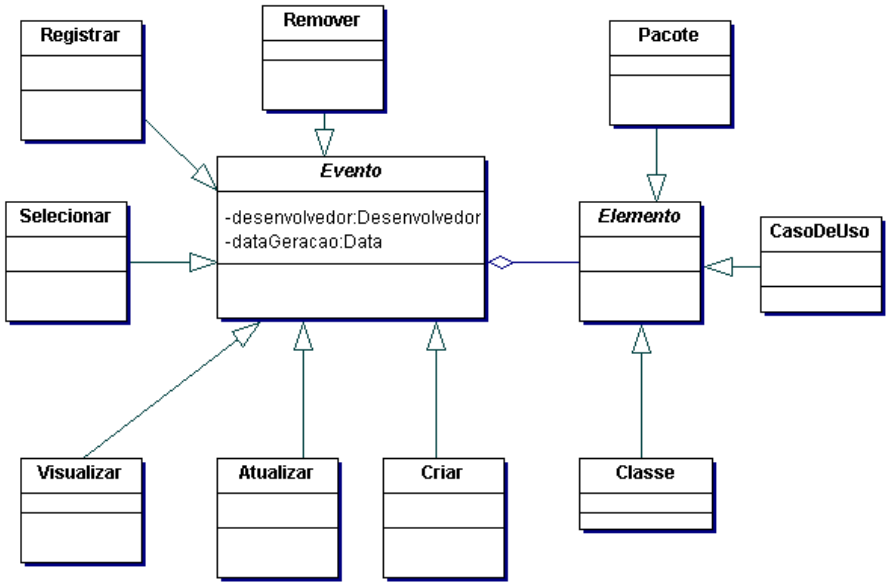

Figura 10 - Modelo Conceitual de Eventos

3.4.3 – Organização de Eventos

Os eventos são organizados e apresentados aos desenvolvedores com o intuito de evitar **sobrecarregá-los cognitivamente** e **facilitar a busca de informações pertinentes**, relativas a determinado momento do trabalho de um desenvolvedor. Assim, o mecanismo *MAIS* utiliza os conceitos de **agrupamento**, **relevância** e **filtragem** de eventos para auxiliar os desenvolvedores a encontrarem as informações desejadas, além de tentar evitar que o fluxo de trabalho desses seja comprometido, devido a um possível desvio de atenção na realização da tarefa, causado pela quantidade de informações apresentadas.

Estes três conceitos são descritos a seguir.

3.4.3.1 – Agrupamento

Como o mecanismo para apoio à percepção *MAIS* trata de modificações realizadas sobre elementos de cópias de um modelo *UML* compartilhado, nem todas as modificações interessam aos desenvolvedores, em certo momento da interação sobre essas. Dependendo da tarefa que determinado desenvolvedor está realizando sobre uma cópia do modelo compartilhado, este pode estar interessado em observar as modificações dos demais desenvolvedores segundo alguma característica específica do modelo. Por exemplo, em um modelo de classes compartilhado, certo desenvolvedor trabalha sobre uma classe "C"; em determinado momento, este desenvolvedor pode estar interessado em observar as modificações dos demais relativas a classes que estão associadas a "C".

Desta forma, *MAIS* permite que os eventos relativos a modificações no modelo compartilhado sejam **agrupados**, segundo alguma característica do modelo *UML* que está sendo tratado. Por exemplo, é possível agrupar eventos por casos de uso, isto é, agrupar eventos relativos a casos de uso, onde cada agrupamento está associado a um caso de uso. Seguindo essa linha, os eventos podem ser agrupados por classes, por diagramas de classe, por pacotes, etc. O mecanismo oferece serviços básicos para ser possível estender a quantidade de agrupamentos tratados.

3.4.3.2 – Relevância

Mesmo agrupando eventos seguindo algumas características, é interessante que os desenvolvedores possuam algum indicador para perceberem quais eventos podem ser importantes, em um dado momento. Esse indicador seria uma tentativa de classificação

desses eventos, atribuindo-os valores que determinariam a importância destes para o desenvolvedor em questão.

Com isso, aplica-se ao *MAIS* o conceito de **relevância** de eventos, medida que é calculada na geração de cada novo evento para todos os eventos recebidos por determinado desenvolvedor. Assim, é possível avaliar a importância dos eventos e representá-la para determinado desenvolvedor.

Esta medida de relevância é calculada da seguinte forma: seja ***Er*** a quantidade de eventos que um desenvolvedor "D" gerou, relativos ao elemento "e" do modelo compartilhado em questão, e ***Eg*** a quantidade total de eventos gerados por "D". O grau de relevância ***Gr***, relativo ao evento "e" para "D" é calculado pela seguinte fórmula:

$$Gr(e,D) = Er(e,D)/Eg(D)*100$$

Por exemplo, se um desenvolvedor "D1" gerou 40 eventos de mudança, sendo que 5 destes foram relativos à classe "C", eventos relativos à classe "C" tem o grau de relevância igual a 12,5% para "D1".

3.4.3.3 – Filtragem

Para os desenvolvedores conhecerem os eventos que lhes interessam, gerados por todos os que interagem sobre um mesmo modelo compartilhado de software, sem que o fluxo de trabalho não seja comprometido, é necessário que não se despenda muito tempo buscando essas informações. Em outras palavras, quanto menor a quantidade de eventos do conjunto a se pesquisar, mais facilmente pode-se encontrar eventos de importância para determinado desenvolvedor, em um certo momento na interação. A utilização dos conceitos de agrupamento e relevância auxilia na redução do número de eventos a se analisar.

Dependendo do volume de eventos gerados, pode ser grande a quantidade de eventos agrupados por determinada categoria. Muitos dos eventos apresentados podem ter sido percebidos pelos desenvolvedores; reapresentá-los apenas sobrecarregaria estes desenvolvedores com informações redundantes. A filtragem de eventos, para apresentação, pode atenuar esse problema.

Sendo assim, o mecanismo *MAIS* realiza filtragem de eventos, seguindo determinados critérios, escolhidos no momento do projeto do mecanismo. Dentre os possíveis filtros a serem utilizado, vislumbra-se o uso do conceito de **ciência** de eventos, que representa a informação que determinado desenvolvedor tomou conhecimento da existência de um evento, para filtragem de informações. Eventos

podem ser marcados como "percebidos" por cada desenvolvedor. Desta forma, o mecanismo *MAIS* tem como utilizar essa informação para filtrar ou não eventos para apresentação.

3.4.4 – Arquitetura do *MAIS*

Esta sub-seção apresenta a arquitetura proposta para o mecanismo *MAIS*, em que cada funcionalidade destacada no processo de utilização pode ser observada em um ou mais componentes desta. Os componentes relevantes são descritos nas subseções seguintes.

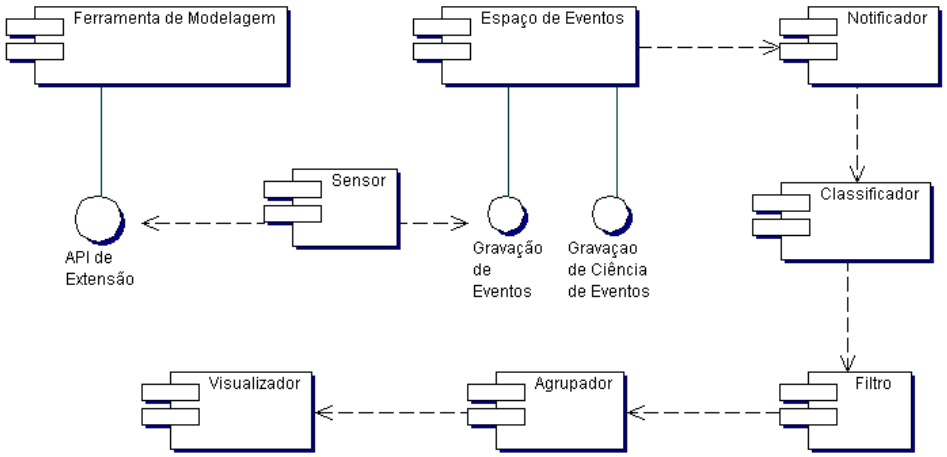

Figura 11 - Arquitetura do Mecanismo *MAIS*

Em linhas gerais, o funcionamento da arquitetura é o seguinte: os desenvolvedores acionam o mecanismo, que está acoplado às ferramentas de modelagem ("**Ferramenta de Modelagem"**). Nesse momento, é registrado no repositório de eventos do grupo ("**Espaço de Eventos**"), através da interface "**Gravação de Eventos**", que determinado desenvolvedor está interagindo sobre um modelo compartilhado *UML*. Para registrar a informação de ciência sobre determinado evento, é utilizada a interface "**Gravação de Ciência de Eventos**".

As mudanças realizadas no modelo compartilhado são capturadas por sensores ("**Sensor**"), que obtém essas informações através de mecanismos de extensão ("**API de Extensão**") providos pelas ferramentas de modelagem. Dado que uma informação de mudança é capturada, um evento associado a esta é gerado por "**Sensor**" e armazenado em "**Espaço de Eventos**".

Desta forma, os desenvolvedores registrados "recebem" cada evento relacionado ao modelo compartilhado, através do componente "**Notificador**", que distribui os eventos para o grupo de desenvolvedores em questão. Cada evento pode ser filtrado por "**Filtro**", de acordo com regras pré-estabelecidas (por exemplo, um evento no qual determinado desenvolvedor já está ciente pode não ser apresentado a este). Eventos recebidos são classificados ("**Classificador**") e agrupados ("**Agrupador**") segundo determinados critérios, sendo por fim apresentados aos desenvolvedores através do componente "**Visualizador**".

3.4.4.1 – Sensor

Dado que o modelo compartilhado sofre alteração em seu estado, por manipulações realizadas em "**Ferramenta de Modelagem**", cabe ao componente "**Sensor**" capturar as informações de mudanças relativas a esta alteração de estado. Esta coleta de informações é realizada de forma passiva; à medida que determinada informação de mudança é gerada em "**Ferramenta de Modelagem**", este componente repassa esta informação para "**Sensor**", pela interface "**API de Extensão**". Cabe ao componente "**Sensor**" transformar as informações de mudança identificadas em eventos do mecanismo *MAIS*. Estes eventos são armazenados em "**Espaço de Eventos**" através da interface "**Gravação de Eventos**".

Vale ressaltar que o componente "**Sensor**" é o único componente da arquitetura do mecanismo *MAIS* que depende de determinada ferramenta de modelagem. Isto ocorre porque cada ferramenta de modelagem (isto é, cada implementação de "**Ferramenta de Modelagem**") pode prover interfaces de extensão ("**API de Extensão**") distintas. Este fato pode afetar o componente "**Sensor**", pois este depende desta interface de "**Ferramenta de Modelagem**".

3.4.4.2 – Espaço de Eventos

A informação de que desenvolvedores estão (ou estiveram) interagindo sobre determinado modelo compartilhado deve ser registrada, para tornar possível distribuir eventos gerados na manipulação das cópias deste. O componente responsável por tratar essa informação é "**Espaço de Eventos**".

Este componente apresenta as seguintes funcionalidades:

- **armazenamento de eventos**: Os eventos gerados por desenvolvedores que utilizam o mecanismo são persistidos por esse componente. Isto é

feito para que seja possível fornecer, a um desenvolvedor novato na interação sobre o modelo compartilhado, um histórico de ações realizadas pelo grupo.

- a**rmazenamento da informação de ciência de eventos**: quando determinado desenvolvedor informa, explicitamente, que conhece determinado evento, esta informação é armazenada em "**Espaço de Eventos**" através da interface "**Gravação de Ciência de Eventos"**. Quando este desenvolvedor voltar a interagir sobre o modelo compartilhado, o evento do qual está ciente pode não lhe ser apresentado.
- **repassar informação sobre novos eventos**: a cada novo evento gerado e armazenado em "**Espaço de Eventos**", este componente deve identificar quais desenvolvedores devem receber este evento e repassá-lo ao componente que realiza a notificação de desenvolvedores sobre a existência deste. Esta identificação é feita com base na informação de quais desenvolvedores estão registrados para receber eventos relativos a um modelo compartilhado.

3.4.4.3 – Notificador

O componente "**Notificador**" tem como responsabilidade notificar às instâncias do mecanismo, atreladas aos desenvolvedores que interagem sobre um mesmo modelo compartilhado, sobre a existência de novos eventos armazenados em "**Espaço de Eventos**". Estas instâncias do mecanismo atuam de forma passiva quanto ao tratamento de novos eventos, isto é, só realizam manipulações sobre este quando são estimuladas por "**Notificador**".

3.4.4.4 – Classificador

Antes dos eventos serem repassados ao componente "**Filtro**", esses podem ser classificados para apresentação. Essa classificação é, por exemplo, o cálculo da relevância do novo evento recebido. Todos os eventos são (re)classificados para que novos valores sejam atribuídos e recebidos por determinado desenvolvedor.

Sendo assim, o componente "**Classificador**" classifica todos os eventos recebidos por uma instância do mecanismo *MAIS*, associada a um desenvolvedor. Desta forma, propriedades podem ser estabelecidas a eventos, como por exemplo relevância, e atualizadas ao passo que os eventos são criados. No mecanismo *MAIS*, apenas a

informação de relevância é tratada por "**Classificador**"; a escolha de representar essa manipulação, de uma maneira mais genérica, auxilia em futuras extensões do mecanismo.

3.4.4.5 – Filtro

Conforme o volume de eventos gerados, relacionados a determinado modelo compartilhado, aumente, pode ser desejável que estes sejam filtrados. Esta filtragem é realizada pelo componente "**Filtro**", que realiza esta tarefa de acordo com regras pré-estabelecidas. A utilização da informação de ciência de eventos, para realização de filtragem de eventos, pode ser útil, pois é razoável que determinado desenvolvedor não deseje visualizar determinado evento que já está ciente.

Vale ressaltar que a filtragem de eventos no mecanismo *MAIS* acontece no nível de agrupamento e apresentação de eventos; isto é feito para tornar possível classificar os eventos levando em consideração todos os eventos gerados por todos os desenvolvedores do grupo.

3.4.4.6 – Agrupador e Visualizador

O componente "**Agrupador**" organiza os eventos "recebidos" por determinado desenvolvedor, agrupando-os seguindo determinados critérios, estabelecidos no projeto do mecanismo *MAIS*. Estes critérios têm como base elementos relacionados a eventos, como por exemplo classes de um modelo compartilhado.

É de responsabilidade do componente "**Visualizador**" apresentar eventos aos desenvolvedores. Este componente, geralmente, é construído baseado na plataforma em que o mecanismo está sendo projetado (por exemplo, sistemas de janelas, páginas HTML dinâmicas, etc).

3.5 – Considerações sobre a Proposta

Uma equipe de desenvolvimento de software, cujos membros necessitam interagir sobre um mesmo artefato, mas não estão disponíveis para interação em um mesmo local físico e/ou em um mesmo instante de tempo, necessita ter apoio computacional para realização desta tarefa. O artefato compartilhado de software, nesta situação, é manipulado através de cópias, pertencentes a cada participante da interação. As contribuições relevantes realizadas nessas cópias são, de tempos em tempos, unificadas, gerando um estado global do artefato compartilhado, armazenado em um

repositório central. Sendo assim, cada desenvolvedor necessita conhecer as contribuições realizadas pelos demais, para facilitar a obtenção de um estado consistente e coerente do artefato compartilhado, na unificação das contribuições. É desejável que esse estado global seja obtido com um mínimo de esforço, por parte dos desenvolvedores.

Mecanismos para apoio à percepção de mudanças em artefatos compartilhados podem atender essa necessidade. Para concepção de um mecanismo que apóie a situação ilustrada acima, alguns questionamentos devem ser realizados, especificamente quanto: (i) ao tipo de artefato que o mecanismo irá atender; (ii) à consistência do artefato compartilhado; (iii) à utilização de ferramental específico para realização de manipulações em artefatos compartilhados; (iv) à maneira como as informações de mudanças devem ser apresentadas. Estes questionamentos devem ser feitos tendo como premissa que o mecanismo deve interferir o mínimo possível no fluxo de trabalhos dos desenvolvedores que o utilizam.

A proposta do mecanismo *MAIS* é coletar, distribuir e apresentar informações de mudanças em modelos compartilhados de software, descritos pela *UML*. O processo de interação dos desenvolvedores sobre o modelo compartilhado é o multi-síncrono, isto é, a manipulação do modelo compartilhado é realizada através de cópias destes. Está além da proposta do mecanismo verificar se o modelo compartilhado está consistente, com base nas modificações realizadas, deixando a cargo dos desenvolvedores envolvidos utilizarem protocolos sociais para tratar esse tipo de situação.

A utilização deste mecanismo é feita através do acoplamento destes em ferramentas de manipulação de modelos *UML*. Este acoplamento é realizado através de um conjunto de serviços que estas ferramentas devem fornecer (ou possuir meios de facilmente estende-las para fornecer esses serviços). Desta forma, os desenvolvedores não precisariam migrar para outras ferramentas para utilizarem o mecanismo.

A coleta de informações de mudanças é feita por intermédio de sensores, implementados nas ferramentas de manipulação de modelos *UML*. Determinada informação de mudança é detectada por esses sensores e esta informação é repassada ao mecanismo *MAIS*. A informação sobre determinada mudança é armazenada no espaço de eventos do grupo, que dispara o processo de notificação dos desenvolvedores que estão interagindo sobre o artefato. Assim, a informação de mudanças, que no mecanismo *MAIS* é representada pela metáfora de evento, é agrupada por características atribuídas ao tipo de modelo *UML* em questão.

3.5.1 – Comparação com Trabalhos Relacionados

A Tabela 4 apresenta uma comparação das características encontradas nas abordagens apresentadas no capítulo anterior com as referentes ao mecanismo *MAIS*.

	COD2DE	D-UML	SAMS	NetEdit	Palantir	Tukan	Abordagem Koblinsky	iScent	MAIS
Editor Colaborativo	●	●	●	●	○	●	○	○	○
Artefatos UML	●	●	○	○	○	○	●	○	●
Abordagem não intrusiva	○	○	○	○	●	○	●	●	●
Interação Síncrona	●	●	●	●	■	●	■	■	■
Interação Assíncrona	●	○	●	○	●	○	●	●	■
Interação Multi-Síncrona	○	○	●	○	■	○	■	■	■
Relevância de Informações	○	○	○	○	■	●	●	○	■
Ciência sobre Informações	○	○	○	○	○	○	○	●	■

Legenda : ●/■ Presente ○ Ausente

Tabela 4 - Comparação das Abordagens com *MAIS*

Para comparar as abordagens pesquisadas com o mecanismo *MAIS*, não é suficiente apenas descrever se determinada característica (como por exemplo, ciência sobre informações de mudança) está ou não presente nas abordagens ou no *MAIS*. Sendo assim, para diferenciar como determinada característica é encontrada nas abordagens (e no *MAIS*), foram criadas duas notações (● e ■). Estas notações dividem as abordagens em subgrupos, de maneira a diferenciar como essas características foram implementadas. Abaixo serão apresentadas as características que apresentam essas diferenças, bem como a descrição das diferenças entre estas para cada característica.

- **Interação Síncrona**: é possível identificar, nas abordagens descritas na Tabela 4, que a interação síncrona pode ocorrer em visões do artefato compartilhado (notação ●), isto é, o grupo de desenvolvedores interage em partes deste artefato, conhecendo apenas informações de mudanças relativas

a estas partes. Existem abordagens em que é possível interagir e perceber mudanças de todo o artefato compartilhado (notação ■). A proposta do mecanismo *MAIS* não está focada em interações síncronas, pois cada desenvolvedor interage sobre uma cópia de um artefato compartilhado independentemente da interação simultânea de outros. Como a propagação dos eventos gerados ocorre no momento em que estes são armazenados no espaço de eventos, sendo que este armazenamento é realizado a medida que mudanças de estado, de uma cópia do modelo compartilhado, ocorram, é possível afirmar que o mecanismo *MAIS* trata o modo síncrono de interação.

- **Interação Assíncrona**: no modo assíncrono de interação, quando determinada informação de mudança é propagada para os desenvolvedores, estes podem estar interessados no estado, do elemento do modelo compartilhado atingido pela mudança, anterior e posterior a esta. Esta informação pode ser útil aos desenvolvedores para situá-los do contexto em que foi gerado o evento relativo a mudança em questão. Desta maneira, podemos classificar, dentre as abordagens apresentadas, interações assíncronas cujos eventos apresentam (notação ■) ou não (notação ●) informações sobre os estados anterior e posterior dos elementos envolvidos.
- **Interação Multi-Síncrona**: neste modo de interação, como os eventos são produzidos com base no estado da cópia local do modelo compartilhado que o gerou. Sendo assim, a operação que resultou no evento, isto é, o conjunto de passos realizado por um desenvolvedor na ferramenta de manipulação do artefato compartilhado para alterar o estado este de forma a obter tal evento, pode (notação ●) ou não (notação ■) ser propagada junto ao evento, para por exemplo, ser aplicada ou não nas cópias dos demais desenvolvedores.
- **Relevância de Informações**: dentre as abordagens apresentadas, foi identificado que o modo de atribuir-se relevância à informações de mudanças (eventos) pode ser definido no momento anterior ao suporte colaborativo em questão (notação ●) ou durante o processo de interação sobre o artefato compartilhado (notação ■). As abordagens que tratam o cálculo relevância nesta última forma o fazem baseadas no histórico de modificações (eventos) geradas pelos desenvolvedores.
- **Ciência de Informações**: foram identificadas duas maneiras de se representar ciência de um determinado desenvolvedor sobre um evento, nas

abordagens apresentadas; (i) os desenvolvedores devem, explicitamente, expressar que desejam conhecer os eventos que determinados estão cientes (notação ●) ou (ii) a informação de ciência de eventos é disponibilizada para todos os desenvolvedores do grupo (notação ■).

É possível destacar como contribuição do mecanismo *MAIS*, em relação às demais abordagens, o fato de que se trata de uma proposta que obtém as informações de mudanças, associadas a um modelo *UML* compartilhado, de maneira implícita e não intrusiva. Devido à semântica fornecida pela notação *UML*, é possível inter-relacionar as informações obtidas pelas manipulações concorrentes no mecanismo *MAIS*, para inferir determinados comportamentos do grupo de desenvolvedores em relação à interação que participam. Analisando a Tabela 4, observa-se que estas características se assemelham com as apresentadas na abordagem descrita em (KOBYLINSKI et al., 2002). Vale ressaltar que este trabalho é apresentado em termos de requisitos. Esta abordagem não é específica a um tipo de artefato de software, nem descreve como as informações coletadas podem ser inter-relacionadas.

A implementação do mecanismo *MAIS*, em que são definidos os elementos a serem monitorados e os agrupamentos para apresentação de eventos, é necessária para por em prática as idéias apresentadas até agora, tentando obter indícios de quão útil o mecanismo pode ser em determinadas situações. Cenários onde o mecanismo poderia ser aplicado necessitariam ser reproduzidos, além de reportar como se deu esta utilização.

Capítulo 4 – Protótipo *MAIS*

4.1 – Introdução

Este capítulo apresenta a implementação de um protótipo (de nome ***MAIS***) do mecanismo para apoio à percepção *MAIS*, baseado na arquitetura proposta no capítulo anterior. A utilização deste protótipo se dá no contexto do ambiente ***Odyssey Share*** (ODYSSEY, 2004), onde os eventos são coletados, distribuídos e apresentados a desenvolvedores que interagem sobre um mesmo modelo *UML* compartilhado.

O *Odyssey Share* é um ambiente cooperativo para desenvolvimento de componentes, que vem sendo desenvolvido por meio da combinação de técnicas como Desenvolvimento Baseado em Componentes, Trabalho Cooperativo e Banco de Dados. Dentre os diversos artefatos que podem ser desenvolvidos por este ambiente, temos alguns descritos pela linguagem *UML*, com o diagrama de casos de uso e modelo de classes.

A plataforma **Java** (SUN, 2004c) foi utilizada para implementação do protótipo. O idioma adotado no projeto e na codificação do protótipo foi o inglês, entrando em conformidade com o utilizado na tecnologia escolhida. A representação de diagramas de classes usadas neste capítulo está em conformidade com a ***UML*** 1.4 (OMG, 2004b) descritos pela ferramenta *Together* (BORLAND, 2004a).

O protótipo *MAIS* não foi concebido visando atender determinado tipo de processo de desenvolvimento de software. Assume-se que o modo em que sua utilização comporá determinado processo de software é definido pela organização onde está inserido.

As seções desse capítulo estão organizadas da seguinte maneira: a Seção 4.2 descreve como os componentes propostos na arquitetura apresentada no capítulo anterior foram implementados. Cabe a Seção 4.3 apresentar os requisitos necessários ao protótipo para ser utilizado em conjunto com o ambiente *Odyssey Share*, bem como as modificações realizadas neste para tornar possível a utilização do protótipo. Um estudo de casos relativo à utilização do protótipo é descrito na Seção 4.4. A Seção 4.5 apresenta algumas considerações sobre o protótipo *MAIS*.

4.2 – Implementação dos Elementos da Arquitetura

Esta seção descreve como foram implementados, no protótipo apresentado, os elementos da arquitetura do mecanismo *MAIS*, descrita no capítulo anterior. Para tal, os elementos são apresentados por categorias. A classificação utilizada refere-se a elementos responsáveis pela: (i) coleta de informações de mudanças e geração de eventos; (ii) distribuição de eventos gerados para as estações de trabalho dos desenvolvedores; (iii) tratamento, agrupamento e apresentação dos eventos para os desenvolvedores.

4.2.1 – Coleta de Eventos

A tarefa de obter informações de modificações, de um modelo compartilhado *UML*, e transformá-las em eventos, cabe ao componente "Sensor". Como essas informações são obtidas de forma passiva, além da premissa de que o mecanismo *MAIS* deve ser independente de ferramenta de manipulação de modelos *UML*, existe a necessidade de se desenvolver um dispositivo adaptador, referente ao componente "Sensor", para que este possa utilizar a interface "API de Extensão" do componente "Ferramenta de Modelagem". Isto é necessário para que instâncias de "Sensor" sejam notificadas pela ferramenta de modelagem utilizada sobre modificações ocorridas em cópias do modelo compartilhado e as convertam em eventos próprios do mecanismo. Desta forma, o mecanismo mantém-se independente de determinada ferramenta de manipulação de modelos *UML*.

No ambiente *Odyssey Share*, o dispositivo para obtenção de informações de mudanças e transformação destas em eventos do protótipo *MAIS* respeita o protocolo imposto pelo Serviço de Notificação de Eventos do *Odyssey Share*, descrito na seção 4.3.2. Neste protótipo, cabe a classe *MaisNotificationTool*, que implementa o padrão de projeto *Adapter* (GAMMA et al., 1995), desempenhar o papel deste dispositivo. Esta classe recebe notificações desencadeadas por ações sobre elementos de diversos tipos de artefato, representados no ambiente *Odyssey Share* (Figura 12); apenas algumas ações sobre determinados elementos de modelos *UML* (mais precisamente, alguns elementos pertencentes a modelos de classes) são contemplados neste protótipo.

As sub-seções a seguir apresentam quais elementos de modelos *UML* são monitorados pelo protótipo, além de descrever como eventos são representados e como é realizada a geração de eventos, respectivamente.

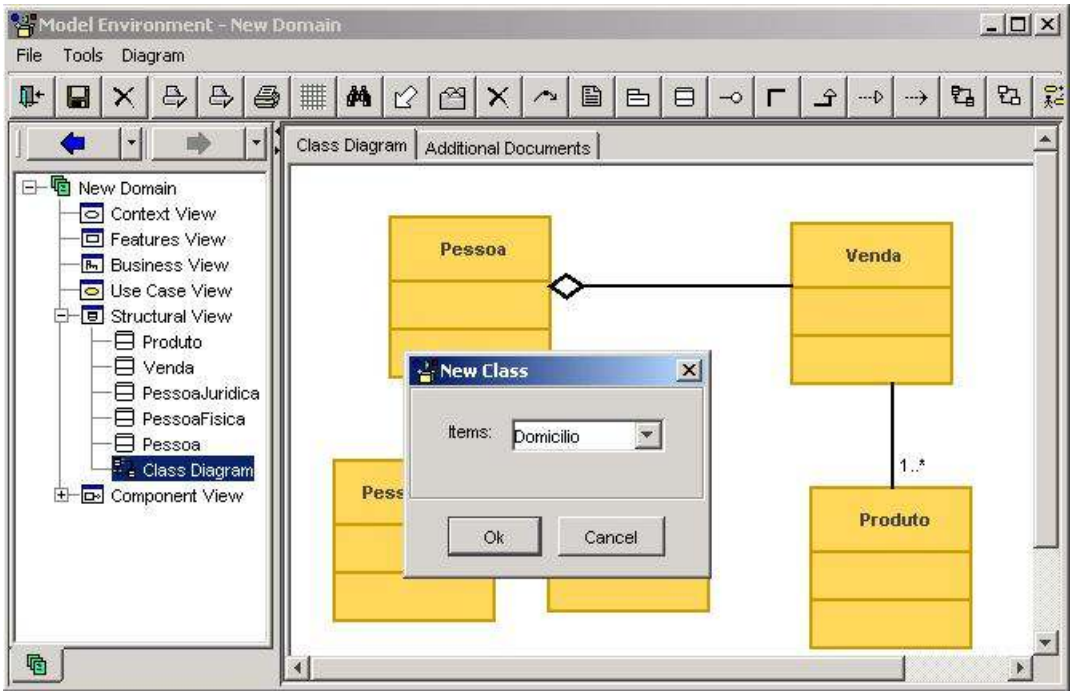

Figura 12 - Manipulação de Artefatos no *Odyssey Share*

4.2.1.1 – Representação de Eventos no Protótipo

No protótipo *MAIS*, os eventos relativos a mudanças em modelos compartilhados *UML* são representados através da combinação de uma **ação** sobre algum **elemento** do modelo em questão. A Figura 13 ilustra o modelo de classes que descreve esta representação: a classe *Event* representa uma ação realizada por determinado desenvolvedor que, associada a um elemento representado pela interface *Element*, formam a representação de eventos no protótipo *MAIS*. Esta classe descreve qual desenvolvedor foi o responsável pela criação do evento, a data de criação deste e seu grau de relevância. Este último é (re)definido no momento em que determinado desenvolvedor é notificado sobre a existência de um novo evento.

As ações implementadas neste protótipo são: (i) **registro** de usuários do mecanismo (*Register*); (ii) **criação** (*Create*), **remoção** (*Delete*) e **atualização** (*Update*) de elementos do modelo compartilhado. Quando ocorre alguma atualização, em determinado elemento do modelo compartilhado, as modificações são representadas através de instâncias da classe *Change*. Esta classe descreve qual propriedade do elemento representado na ação foi atingido por esta, representando o valor anterior e posterior a esta ação.

Os elementos de modelos de classes, definidos nesse protótipo são: (i) **classes** do modelo (*Class*); (ii) **relacionamentos** entre classes (*Relationship*); **atributos** de classes (*Attribute*); **métodos** de classes (*Method*). Cada elemento pode estar relacionado a um ou mais elementos do modelo; esta relação está associada com a pertinência de determinados elementos em outros. Por exemplo, o elemento *Attribute* está relacionado a um elemento *Class*, já que, necessariamente, um atributo deve pertencer a uma classe, em modelos *UML*.

A escolha das ações e elementos, que o protótipo deveria atender, não seguiu nenhum critério formal, que apresentasse indícios sobre quais opções eram mais apropriadas para utilização na implementação do mecanismo. Os elementos e ações descritos no protótipo *MAIS* foram selecionados através de observações sobre a manipulação de modelos de classes *UML*. Estas observações foram uma tentativa de inferir que ações e elementos de modelos de classes *UML* ocorrem com mais frequência em manipulações destes.

Vale destacar que o conjunto de elementos e ações do protótipo *MAIS* é extensível. Além disso, a representação de elementos do modelo compartilhado (no caso, modelo de classes *UML*) difere da utilizada na implementação do *Odyssey Share*; desta forma, é possível manter independente a implementação do mecanismo da ferramenta onde este é utilizado.

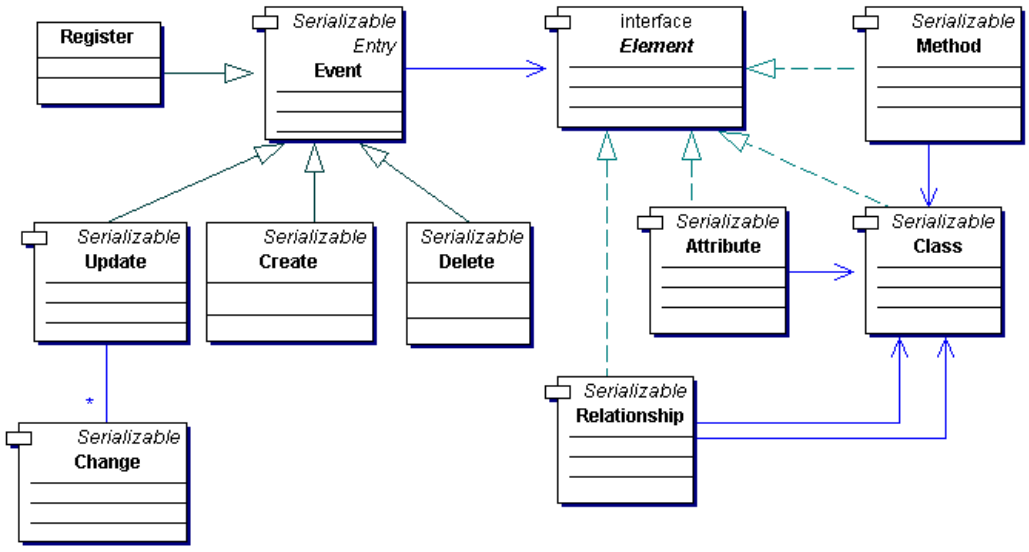

Figura 13 - Modelo de Eventos

4.2.1.2 – Geração de Eventos

Os eventos utilizados no protótipo *MAIS* são gerados através de dispositivos adaptadores, criados para a utilização em determinada ferramenta de manipulação de modelos *UML*. No caso do ambiente *Odyssey Share*, a classe *MaisNotificationTool* é responsável por essa tarefa.

Na Figura 14, a classe *MaisNotificationTool* recebe notificações sobre mudanças, do ambiente *Odyssey Share*. Cabe a esta classe converter a representação dos elementos *UML*, referentes à modificação, para a correspondente no protótipo *MAIS*. Por exemplo, no momento de criação de uma determinada classe no *Odyssey Share*, o protótipo *MAIS* é notificado. Nesta notificação, o protótipo obtém a referência da instância da classe *NoClasse*, representação interna de uma classe no ambiente, referente à classe que está sendo criada. De posse dessa referência, a classe *MaisNotificationTool* obtém as informações relevantes ao protótipo, criando uma instância da classe *Create*, combinada com a classe *Class*.

Gerado o evento (isto é, instanciada uma classe descendente de *Event*), este é repassado a classe *MaisClient*, que é a abstração utilizada para representar uma instância do protótipo *MAIS* associada a um desenvolvedor. De posse do evento gerado, esta classe inicia o procedimento de distribuição deste para os demais desenvolvedores.

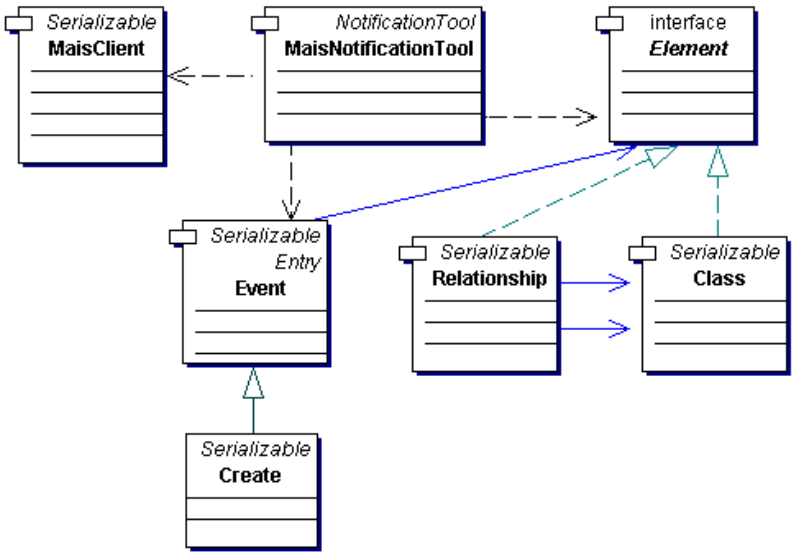

Figura 14 - Geração de Eventos

4.2.2 – Distribuição de Eventos

Como visto no capítulo anterior, o processo de distribuição de eventos é realizado da seguinte maneira: dado que um evento é gerado, este é disponibilizado no espaço de eventos referente, que delega ao componente "Notificador" a tarefa de notificar os desenvolvedores registrados a receber eventos associados a determinado modelo compartilhado *UML*. Nesta notificação, o evento gerado é transmitido junto com a "mensagem" de notificação.

A Figura 15 refere-se ao modelo de classes do protótipo *MAIS* relativo à distribuição de eventos, onde é possível observar as classes *ChannelAccessor* e *Channel*. Estas abstrações foram concebidas para manter independente a implementação do protótipo *MAIS* com a referente ao componente "Espaço de Eventos". A classe *Channel* representa a porção de "Espaço de Eventos" destinada ao armazenamento de eventos de um modelo compartilhado *UML*. *ChannelAcessor* representa o ponto de interação entre *Channel* e o resto da aplicação. A interação com "Espaço de Eventos" é realizada por intermédio de uma especialização de *Channel* (no caso, *MaisChannel*). Essas classes foram concebidas de maneira a permitir que outras aplicações do *Odyssey Share* possam reutilizar esta infraestrutura de espaço de eventos; cada aplicação especializa a classe *Channel* conforme o tipo de objeto que deseja armazenar em "Espaço de Eventos". Por exemplo, não necessariamente uma aplicação, no contexto do ambiente *Odyssey Share*, deseja usar como abstração de eventos objetos da classe *Event*.

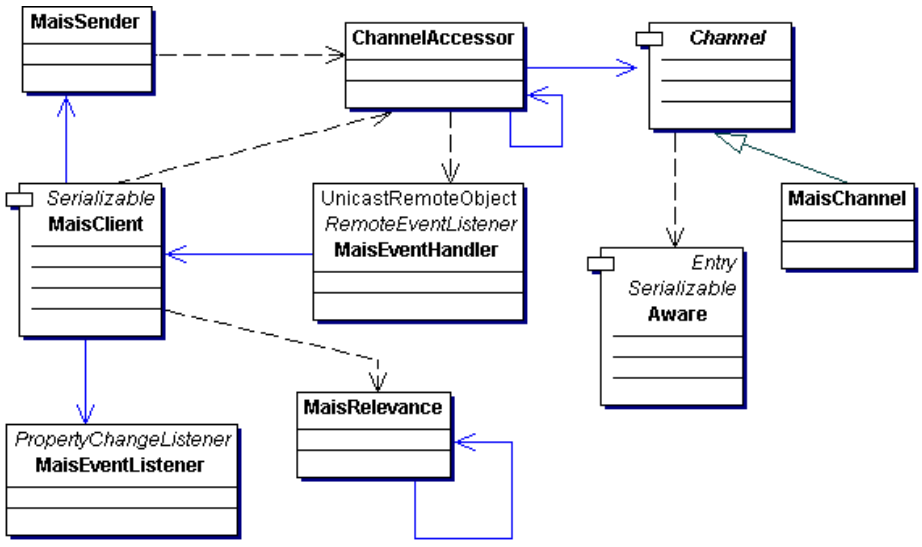

Figura 15 - Modelo de Distribuição de Eventos

4.2.2.1 – Envio de Novos Eventos

Dado que um novo evento é gerado e repassado a *MaisClient*, esta classe realiza o envio desse para "Espaço de Eventos" através da classe *MaisSender*, como apresentado na Figura 15. Esta classe envia o evento para *ChannelAccessor*, que identifica qual instância de *Channel* deve ser utilizada (no caso, *MaisChannel*) e realiza os procedimentos para armazená-lo em "Espaço de Eventos".

4.2.2.2 – Notificação sobre Eventos

Para ser notificada, uma instância de *MaisClient* deve ser registrada em "Espaço de Eventos" para tal. Este registro é realizado através da chamada, por uma instância de *MaisClient*, do método *ChannelAccessor.register()*.

Como este componente deve ser projetado visando atender outras aplicações do *Odyssey Share*, o mecanismo de notificação deve ser genérico quanto a que tipo de componente que deve notificar, não sendo dependente de alguma implementação em específico (por exemplo, *MaisClient*). Sendo assim, a alternativa adotada, no protótipo, baseia-se na implementação de um dispositivo adaptador, análogo ao adotado na utilização do Serviço de Notificação de Eventos do *Odyssey Share*, que atenda as exigências impostas por "Espaço de Eventos", para ser notificado.

Desta forma, foi implementada a classe *MaisEventHandler* (conforme visto na Figura 15), que funciona como um *Wrapper* (GAMMA et al., 1995) da classe *MaisClient* para receber notificações de "Memória Compartilhada". A associação entre *MaisClient* e *MaisEventHandler* é realizada no momento em que o procedimento de registro de determinado desenvolvedor em "Espaço de Eventos" é realizado. No caso, uma instância de *MaisEventHandler* é armazenada neste componente, para ser notificada quando um novo evento (isto é, instâncias da classe *Event*) for armazenado em "Espaço de Eventos".

4.2.2.3 – Recepção de Eventos

No protótipo *MAIS*, eventos são recebidos por *MaisClient*, para serem classificados, filtrados (caso isso seja necessário) e repassados a componentes de agrupamento e visualização, em dois momentos distintos : (i) quando um desenvolvedor é registrado em "Espaço de Eventos" e (ii) a cada novo evento gerado. Os eventos, neste caso, são repassados um a um para *MaisClient*. Dado que *MaisClient* recebe eventos individualmente, o procedimento, nas duas situações apresentadas, é o mesmo: todos os

eventos já recebidos pela instância de *MaisClient* podem ser classificados por classes que implementam o componente "Classificador".

No momento que uma instância de *MaisEventHandler* é registrada em "Espaço de Eventos", esta faz uma requisição a esse componente (via *ChannelAccessor*), para obter todos os eventos disponíveis, relacionados a manipulações sobre determinado modelo *UML* compartilhado (isto é, em um determinado *Channel*). *ChannelAcessor* retorna os eventos com uma "marcação" individual, informando se o desenvolvedor, associado à instância de *MaisClient* em questão, está ou não ciente destes. Esta informação de ciência é disponibilizada da seguinte forma: quando algum desenvolvedor informa ao protótipo *MAIS* que está ciente sobre a existência de determinado evento (detalhes deste procedimento na seção 4.2.3.1), uma instância da classe *Aware* é criada (ou obtida de "Espaço de Eventos", no caso em que este desenvolvedor já tenha utilizado este procedimento, para outros eventos, anteriormente), representando os eventos que o desenvolvedor em questão já está ciente. Instâncias da classe *Aware* são armazenadas em "Espaço de Eventos", de onde são consultadas para a realização de filtragem de eventos.

Quando um novo evento é gerado, as instâncias de *MaisEventHandler*, armazenadas em "Espaço de Eventos", são notificadas sobre este acontecimento. Esta classe repassa o evento recebido em conjunto com a notificação para *MaisClient*, que dispara o processo para reclassificação de todos os eventos já recebidos (comportamento do componente "Classificador"). No protótipo *MAIS*, esta classificação está associada apenas ao cálculo de relevância de eventos, que é realizado pela classe *MaisRelevance*.

Após o processo de classificação, *MaisClient* repassa, para as classes de agrupamento e visualização, apenas os eventos que o desenvolvedor relacionado a esta classe não está ciente (comportamento do componente "Filtro"). O novo evento recebido é repassado a classes que realizam agrupamento e visualização de eventos. Desta forma, a classe *MaisEventListener* inicia a realização desses passos. Esta classe foi concebida para tornar independente os processos de agrupamento e apresentação de eventos do processo de distribuição destes, funcionando como um adaptador entre esses.

4.2.2.4 – Espaço de Eventos

A implementação do componente "Espaço de Eventos" é baseada no conceito de **espaço de tuplas** (CARRIERO e GELERNTER, 1989), advindo da área de Inteligência

Artificial para tratar processos distribuídos. Em linhas gerais, este conceito representa um repositório central de armazenamento de objetos, que contêm algum tipo de computação específica, podendo ser lidos, escritos e retirados por dois ou mais processos distribuídos, em uma rede de comunicação, conforme apresentado pela Figura 16. Esta representação para "Espaço de Eventos" foi escolhida, em detrimento de outras pesquisadas na literatura, pois oferece uma maneira simples de comunicação entre processos distribuídos, uma característica fundamental do protótipo.

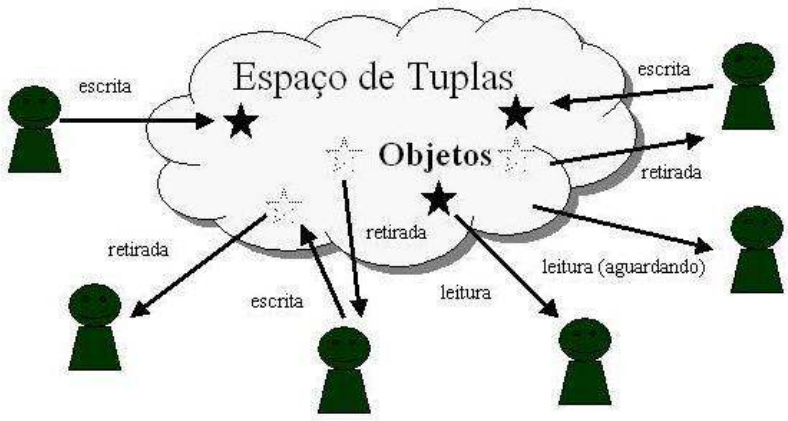

Figura 16 - Espaço de Tuplas – Adaptado de (SUN, 2004d)

A tecnologia ***Javaspaces*** (SUN, 2004d) foi utilizada, no protótipo *MAIS*, para representar o espaço de tuplas. Esta tecnologia define uma *API* para programação baseada em espaços de tuplas na plataforma *Java*. Esta tecnologia implementa o comportamento do componente "Notificador", isto é, permite que quando determinado objeto (no caso, um evento) for escrito no espaço de tuplas, sejam notificados outros objetos (no caso, abstrações de desenvolvedores). A implementação dessa *API* utilizada no protótipo foi o ***Gigaspaces*** (GIGASPACES, 2004).

4.2.3 – Apresentação de Eventos

Na arquitetura proposta para o mecanismo *MAIS*, os eventos devem ser agrupados, seguindo alguns critérios pré-estabelecidos, antes de serem apresentados aos desenvolvedores, para evitar sobrecarga de informação e tentar acelerar a identificação de eventos relevantes a esses. Os componentes "Agrupador" e "Visualizador", responsáveis pelo agrupamento e apresentação de eventos, respectivamente, foram implementados, no protótipo *MAIS*, utilizando classes *Java Foundation* (*Swing*) (SUN, 2004b).

A Figura 17 apresenta o modelo de classes desenvolvido para tratar de agrupamento e apresentação de eventos. Dado que uma instância de *MaisClient* repassa a *MaisEventListener* um dado evento para apresentação, descrevendo a seguir as demais classes envolvidas:

- ***UIWindow***: Interface que representa uma janela onde o evento em questão deve ser apresentado. Para a visualização no *Odyssey Share*, foi criada a classe *UIWindowOdyssey*, que implementa esta interface.
- ***UIEventPanel***: Painel contido em *UIWindowOdyssey*, onde o evento será visualizado. Reserva áreas para visualização de eventos: (i) gerados pelo desenvolvedor corrente e (ii) gerados pelos demais desenvolvedores.
- ***UITabbedPaneManager***: Responsável por fornecer painéis a *UIEventPanel* associados aos tipos de agrupamentos de eventos disponíveis.
- ***UIListModel***: Representa a abstração genérica de agrupamentos. Para que um (novo) tipo de agrupamento seja utilizado no protótipo *MAIS*, é necessário implementar uma classe que estenda de *UIListModel*, associando-a a *UITabbedPaneManager*. As implementações disponíveis em *MAIS*, relacionadas a **agrupamento de eventos**, são: (i) **por classe** (*UIByClassListModel*), (ii) **por usuário** (*UIByUserListModel*), (i) **por relevância** (*UIByRelevanceListModel*), (ii) **por proximidade** (*UIByProximityListModel*) e (i) listagem de **todos os eventos** (*UIAllListModel*).
- ***UIList***: Classe responsável por apresentar os eventos, dado um agrupamento, no formato de árvore.

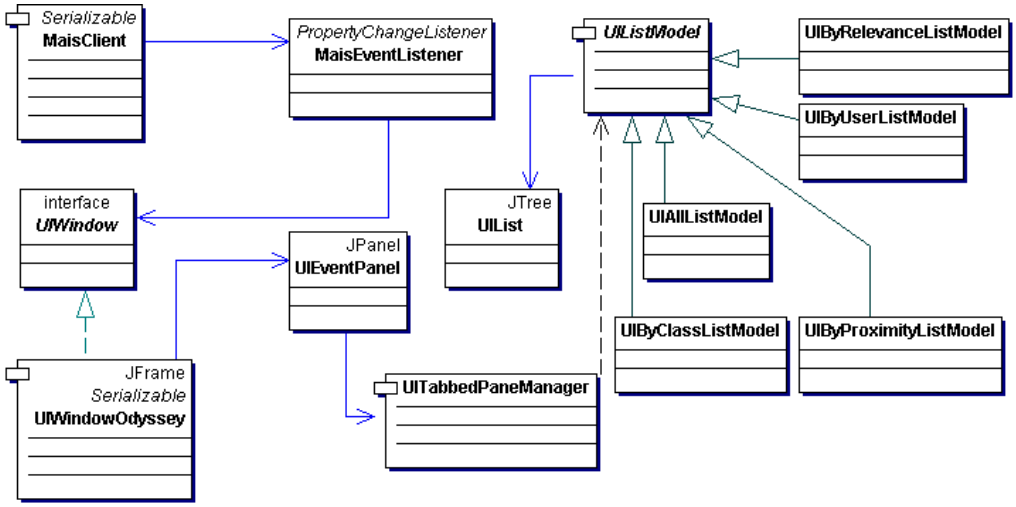

Figura 17 - Modelo de Apresentação de Eventos

A próxima sub-seção descreve como os eventos são visualizados por desenvolvedores que utilizam o protótipo *MAIS*, fornecendo mais detalhes relacionados ao agrupamento desses.

4.2.3.1 – Visualização de Eventos em *MAIS*

No protótipo *MAIS*, os eventos são visualizados de forma textual, conforme apresentado na Figura 18. Nessa descrição, são informados: (i) qual a ação que gerou o evento, (ii) qual tipo de elemento envolvido na ação, (iii) o nome do elemento envolvido, (iv) o desenvolvedor que realizou a ação sobre o modelo *UML* compartilhado e (v) a data de geração do evento.

Os eventos são dispostos em duas listas para visualização; a lista da esquerda contém os eventos gerados pelo desenvolvedor corrente, enquanto a referente à direita agrupa os eventos gerados pelos demais usuários. Cada lista de eventos pode ser visualizada através da perspectiva de determinado agrupamento. As "abas" associadas às listas representam, cada uma, um tipo diferente de agrupamento de eventos.

Figura 18 - Visualização de Eventos

Em determinadas situações, pode ser interessante que os desenvolvedores queiram saber detalhes sobre determinado evento. Isto se torna mais comum quando o evento é referente a uma ação de atualização. Assim, o protótipo *MAIS* permite que desenvolvedores visualizem detalhes referentes a um determinado evento, conforme apresentado na Figura 19. Neste exemplo, o evento a ser detalhado refere-se a uma atualização, realizada pelo desenvolvedor "Mangan", em propriedades da classe "Venda", contida em um modelo de classes do ambiente *Odyssey Share*. Além das informações apresentadas na visualização do evento na lista de eventos, são informados os nomes das propriedades alteradas e os valores anteriores e posteriores à atualização.

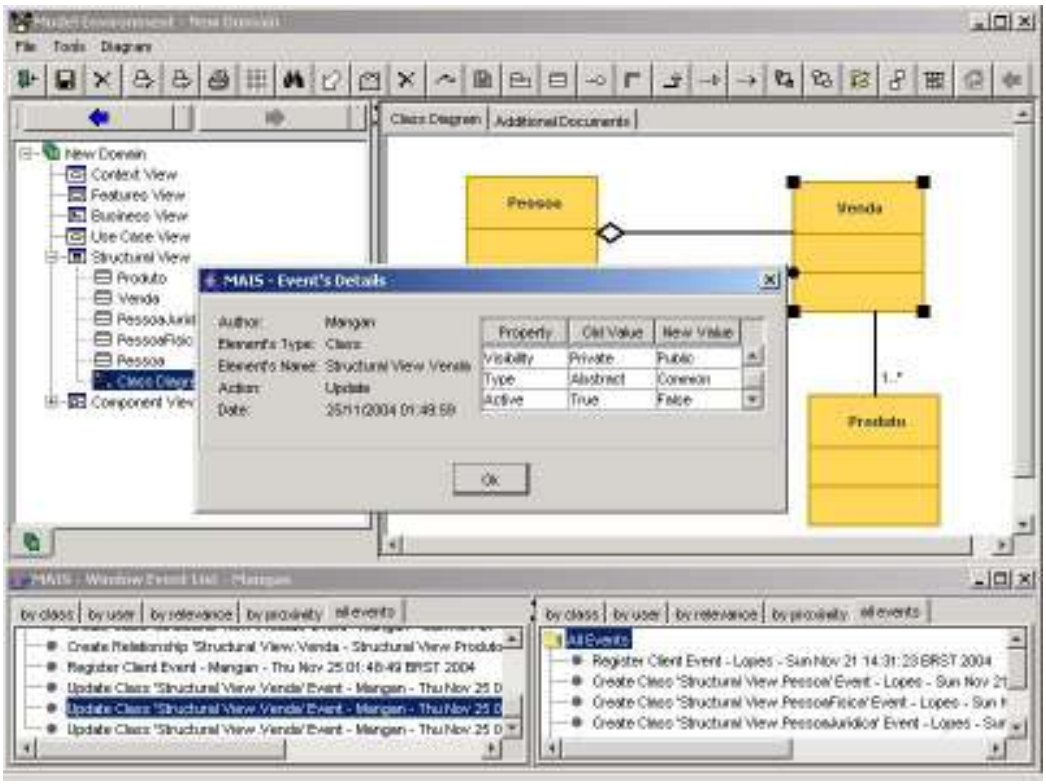

Figura 19 - Detalhes sobre Eventos

A Figura 20 descreve com é realizado o procedimento de ciência de eventos: quando um desenvolvedor desejar informar, ao protótipo *MAIS*, que está ciente sobre a ocorrência de determinado evento, esse deve, ao clicar com o botão direito do *mouse* sobre o evento, selecionar a opção "*I'm Aware*". Desta forma, o evento em questão não mais será apresentado para esse desenvolvedor.

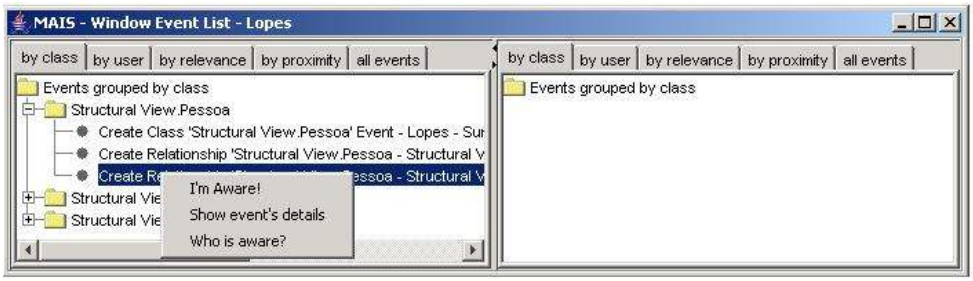

Figura 20 - Ciência de Eventos

A Figura 21 apresenta como obtém-se a informação de quem está ciente sobre determinado evento. Ao clicar com o botão direito do *mouse* sobre o evento, o desenvolvedor deve selecionar a opção "*Who is Aware?*". Feito isso, é apresentada uma lista de desenvolvedores que estão cientes da ocorrência do evento selecionado.

Figura 21 - Informação sobre Ciência

4.2.3.1.1 – Visualização por Tipo de Agrupamento

Esta seção apresenta como eventos são visualizados pelos desenvolvedores, envolvidos na modelagem concorrente de um modelo de classes compartilhado, de acordo com os agrupamentos implementados no protótipo *MAIS*. A Figura 22 apresenta um modelo de classes *UML* compartilhado, descrito no ambiente *Odyssey Share*. Este modelo representa o estado global do artefato em questão, contemplando as contribuições dos desenvolvedores "Lopes" e "Mangan".

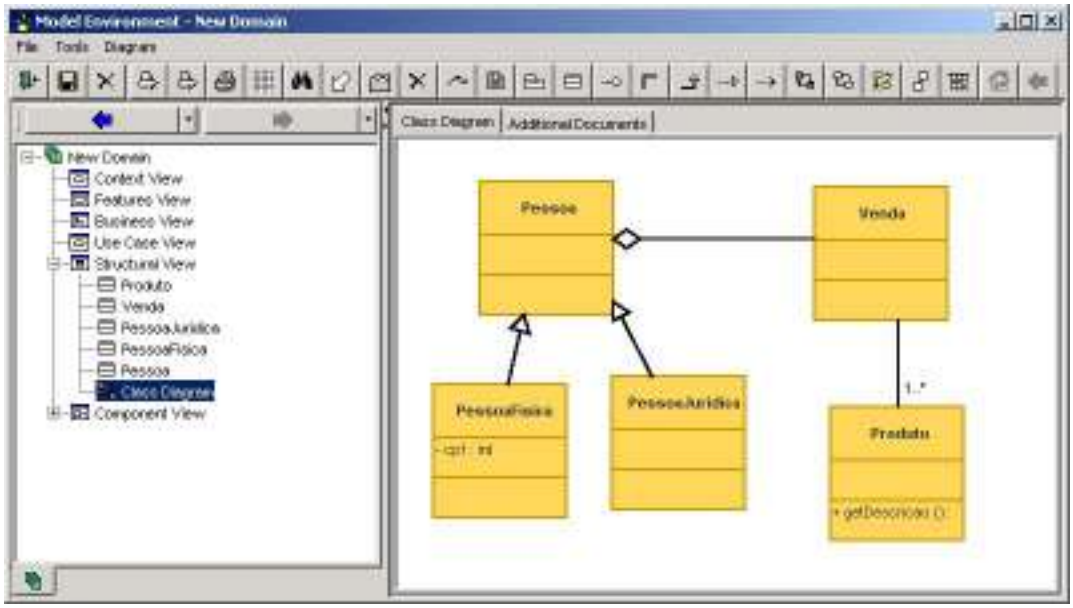

Figura 22 - Modelo de Classes Utilizado para Agrupamentos

O agrupamento por classes, descrito pela Figura 23, apresenta os eventos gerados na perspectiva das classes que estes atingem. Eventos relativos a propriedades de uma classe, seus atributos, métodos e relacionamentos são agrupados por esta. No caso de relacionamentos entre classes, os eventos relativos a estes são agrupados, neste caso, de acordo com a classe definida como origem do relacionamento. No exemplo, eventos associados ao relacionamento entre “Pessoa” e ”PessoaFisica” são agrupados, neste caso, pela classe “Pessoa”. Na Figura 23, é possível perceber que o desenvolvedor “Lopes” não modelou, em sua cópia local, as classes “Venda” e “Produto”, criadas por “Mangan”. Esta informação é inferida pois não existem eventos agrupados pelas classes “Venda” e “Produto” na lista de eventos do usuário “Lopes” (lista situada à esquerda, já que a figura representa uma instância do protótipo usada por “Lopes”), isto é, este usuário não criou estas classes em sua cópia local.

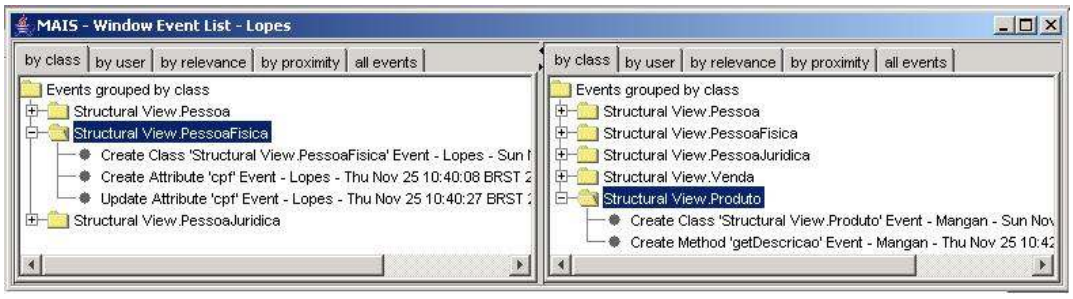

Figura 23 - Agrupamento por Classe

Eventos agrupados por desenvolvedores (usuários) são apresentados na Figura 24. Esta forma de visualização permite que sejam analisadas e avaliadas as contribuições realizadas por “Lopes” e “Mangan” nas cópias do modelo de classes compartilhado, bem como fornece um indicador do volume de contribuições individuais realizadas.

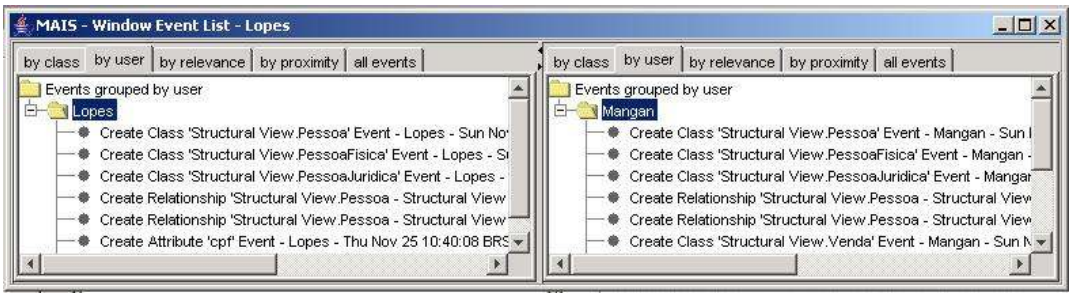

Figura 24 - Agrupamento por Usuário

Dado que é atribuído, a cada novo evento gerado, um valor de relevância para cada evento já recebido por um desenvolvedor, os eventos podem ser agrupados por esse valor. A Figura 25 apresenta este tipo de agrupamento: os eventos são agrupados em quatro faixas de valor de relevância (0-25%, 25-50%, 50-75%, 75-100%). Os valores de relevância são calculados com base nos eventos gerados pelo desenvolvedor corrente. Por exemplo, na Figura 25, é possível observar, na lista de eventos gerados pelos outros desenvolvedores (lista à direita), que eventos relativos a classe “Produto” estão agrupados na faixa de 0-25% de valor de relevância, para o desenvolvedor “Lopes”. Mais precisamente, esses eventos têm valor de relevância igual a zero para “Lopes”, pois este desenvolvedor ainda não interagiu sobre a classe “Produto”.

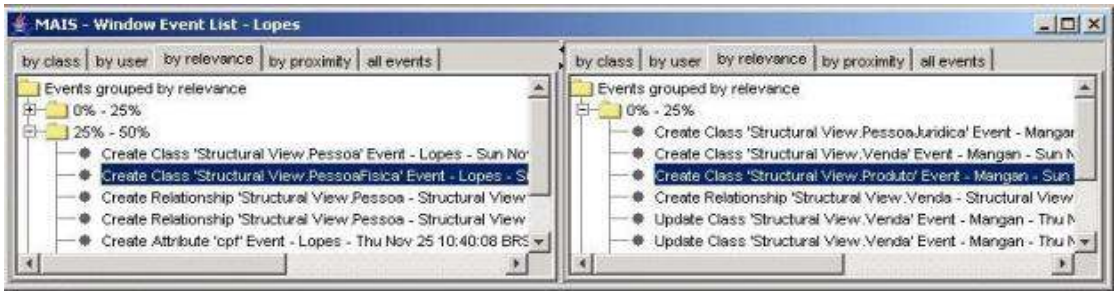

Figura 25 - Agrupamento por Relevância

Uma tentativa de obter informações através do relacionamento entre classes é apresentada no agrupamento por proximidade. Na visualização desse agrupamento,

apresentada na Figura 26, são explorados os relacionamentos estruturais entre classes. Para cada classe criada, são agrupados eventos de acordo com as classes que se relacionam com esta, na perspectiva do desenvolvedor corrente ou dos outros desenvolvedores. Na Figura 26, é possível verificar, na lista de eventos do desenvolvedor "Lopes", que a classe "PessoaJuridica" está relacionada com a classe "Pessoa", listando os eventos desta. Na lista dos demais desenvolvedores, observa-se que algum desenvolvedor (no caso, "Mangan") relacionou a classe "Produto" com a classe "Venda".

Neste tipo de agrupamento, apenas são consideradas as classes que se relacionam diretamente uma com as outras. Este critério foi adotado para evitar que os desenvolvedores fiquem sobrecarregados com informações. Por exemplo, se o modelo de classes compartilhado representa uma árvore (cujos nós são descritos pelas classes), o agrupamento por proximidade perde a utilidade, já que todos os eventos seriam apresentados em cada grupo.

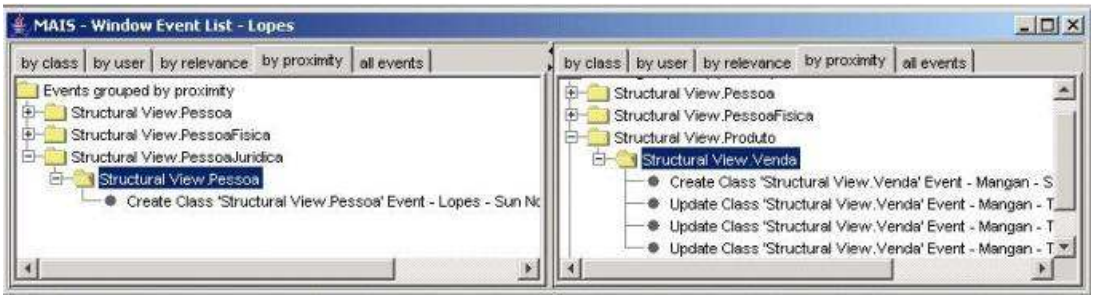

Figura 26 - Agrupamento por Proximidade

4.3 – Adaptação do Protótipo ao *Odyssey Share*

Para utilizar o protótipo *MAIS*, em conjunto com o ambiente *Odyssey Share*, foi necessário contemplar, em seu desenvolvimento, algumas atividades para tornar isto possível. Dentre as funcionalidades apresentadas, relativas ao mecanismo *MAIS*, duas em especial merecem destaque, para que fosse possível a utilização deste em conjunto com alguma ferramenta de manipulação de artefatos *UML* (no caso, o ambiente *Odyssey Share*): seu uso independer de alguma ferramenta específica e sua atuação de forma passiva na coleta de eventos. As próximas sub-seções apresentam mecanismos que foram utilizados e desenvolvidos para atender estas necessidades.

4.3.1 – Carga Dinâmica de Componentes

A utilização de ferramentas dentro do contexto do ambiente *Odyssey Share* é realizada por um mecanismo chamado **carga dinâmica de componentes** (MURTA et al., 2004). Neste procedimento, ferramentas desenvolvidas fora do contexto do ambiente *Odyssey Share* são descarregadas, de um servidor onde estão armazenadas, e instaladas em tempo de execução. Desta forma, cada instância do ambiente possui uma lista atualizada de todos os módulos disponíveis para instalação.

Em linhas gerais, para tornar possível a utilização do protótipo *MAIS* no ambiente *Odyssey Share*, foram necessárias as seguintes adaptações:

- A interação entre o ambiente *Odyssey Share* e as ferramentas instaladas é feita através de classes, empacotada junto com o protótipo, que implementam a interface *Tool*. Desta forma, foi necessário criar uma classe deste tipo no protótipo *MAIS*, que basicamente oferece, ao ambiente *Odyssey Share*, alguns tipos de menus de opção, utilizados na interação com os usuários deste.
- O empacotamento do protótipo *MAIS* foi realizado através de arquivos *Java Archive* (*JAR*) (SUN, 2004a). O arquivo MANIFEST.MF do arquivo *JAR* referente ao *MAIS* deve conter, no valor da propriedade *Tool-Class*, o nome da classe que implementa a interface *Tool* nesta ferramenta. A Figura 27 representa um exemplo de arquivo MANIFEST.MF com as características apresentadas.

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.5.4
Created-By: 1.4.2_01-b06 (Sun Microsystems Inc.)
Main-Class: br.ufrj.cos.lens.odyssey.tools.mais.Mais
Tool-Class: br.ufrj.cos.lens.odyssey.tools.mais.MaisTool
Notification-Class: br.ufrj.cos.lens.odyssey.tools.mais.MaisNotificationTool
Class-Path: jini-core.jar mahalo-dl.jar outrigger-dl.jar
            JSpaces.jar jini-ext.jar reggie-dl.jar log4j-1.2.8.jar
```

Figura 27 - Arquivo MANIFEST.MF

4.3.2 – Serviço de Notificação de Eventos

Para que a coleta de eventos fosse realizada de forma passiva, o ambiente *Odyssey Share* precisava prover meios de notificar ferramentas, instaladas através do mecanismo de carga dinâmica, sobre alterações em modelos *UML*. Essas notificações deviam fornecer as informações necessárias para atender ferramentas como o *MAIS*,

relacionadas a quem realizou a ação no modelo *UML* compartilhado que gerou determinado evento, onde, quando e o que foi realizado.

Entretanto, esta funcionalidade não se encontrava disponível no ambiente *Odyssey Share*. Desta forma, foi necessária a concepção de um serviço que visasse atender a necessidade de notificação sobre mudanças ocorridas na manipulação, no ambiente, de determinado artefato de software. Este serviço tinha como premissa: (i) ser genérico, para atender não apenas ao protótipo *MAIS*, mas a qualquer ferramenta que tenha interesse em explorar esse tipo de informação; (ii) notificar a todas as ferramentas que o utilizam sobre o estado dos elementos envolvidos na ação que gerou determinado evento.

Sendo assim, foi desenvolvido o **Serviço de Notificação de Eventos** do ambiente *Odyssey Share*, que captura eventos realizados por determinado usuário deste sobre algum artefato, informando ferramentas interessadas sobre a ocorrência e os estados dos elementos associados a estas ações. Este serviço é uma extensão do mecanismo de carga dinâmica já oferecido pelo ambiente *Odyssey Share*.

4.3.2.1 – Visão Geral da Solução

Nesta seção é apresentada, em linhas gerais, a implementação do Serviço de Notificação de Eventos do *Odyssey Share*. As classes envolvidas são apresentadas na Figura 28.

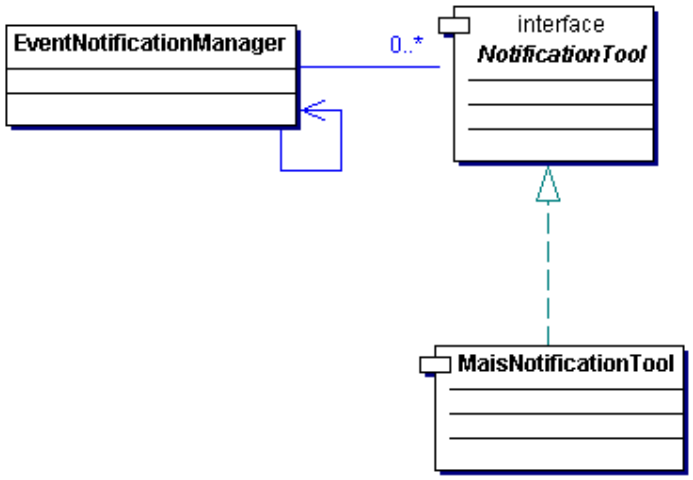

Figura 28 - Serviço de Notificação de Eventos

No momento em que um módulo é carregado dinamicamente no *Odyssey Share*, é verificado se este está configurado para ser notificado sobre eventos. Esta verificação é feita obtendo o valor do atributo *Notification-Class* do arquivo MANIFEST.MF do

módulo. Este atributo refere-se ao nome da classe do módulo que deve ser notificada quando determinado evento ocorrer. Esta classe implementa a interface *NotificationTool*. Na Figura 28, é possível observar que a classe *MaisNotificationTool* refere-se ao módulo *MAIS*, descrevendo métodos que representam os eventos referentes ao Serviço de Notificação de Eventos. Por exemplo, o método *NotificationTool.editAssociationEvent(LigAssociacao oldLig, LigAssociacao newLig)* está associado ao evento de edição de associação. Neste exemplo, os parâmetros *oldLig* e *newLig* são da classe *LigAssociacao*, que no *Odyssey Share* representa o relacionamento de associação entre classes descrito na *UML*.

A classe *EventNotificationManager*, que implementa o padrão de projeto *Singleton* (GAMMA et al., 1995), dispara as notificações para os módulos cadastrados a receber notificação de eventos. Nos pontos do código fonte do *Odyssey Share*, onde uma dada ação sobre um elemento ocorre, é chamada a operação correspondente em *EventNotificationManager*.

Este serviço é baseado no padrão de projeto *Observer* (GAMMA et al., 1995); quando se obtém o valor do atributo *Notification-Class*, a classe descrita por este valor é instanciada e associada a *EventNotificationManager*. Assim, todo evento gerado pelo serviço é "repassado" a todos os módulos associados.

4.4 – Estudo Qualitativo de Utilização do Protótipo

O protótipo *MAIS* se apresenta como um ferramental que permite que membros de uma equipe recebam informações sobre a evolução de um modelo compartilhado durante a edição concorrente de suas cópias. Estas informações de mudanças (informações para percepção) são importantes para que seja possível gerar um estado consistente do modelo, resultado da união das contribuições individuais, em uma tentativa de se diminuir o volume de retrabalho posterior.

Como o conceito de percepção está associado a um estado mental de determinado indivíduo (SOHLENKAMP, 1998), é desejável observar a utilização do protótipo *MAIS*, para obter informações que possam indicar o quão aplicável é esta solução em determinado cenário. Novos requisitos, bem como deficiências, são passíveis de ser identificados neste processo de utilização do protótipo.

Esta seção apresenta um estudo, de foco qualitativo, realizado sobre a utilização do protótipo *MAIS*, na atividade de modelagem concorrente. No final desta seção, é apresentada uma discussão do que foi observado durante a realização do mesmo.

4.4.1 – Premissas para Utilização do Protótipo *MAIS*

Pressupõe-se, na utilização do protótipo *MAIS*, que todos os desenvolvedores estão trabalhando sobre uma mesma versão do modelo compartilhado. Não é finalidade do protótipo fornecer mecanismos para se estabelecer uma cópia consistente do modelo compartilhado, que contemple as contribuições geradas por cada desenvolvedor. Este apenas provê informações que permitam que desenvolvedores manipulem suas cópias do artefato com base nas alterações dos demais, sendo esta versão gerada com a utilização de ferramental apropriado (por exemplo, um sistema de controle de versões).

Todos os usuários do protótipo *MAIS* possuem o mesmo papel (desenvolvedor). Não existe um papel de coordenador/moderador do processo de modelagem concorrente. Conflitos entre os participantes são resolvidos utilizando protocolos sociais, com o apoio de ferramentas de comunicação como, por exemplo, *NetMeeting* (MICROSOFT, 2004b) e *ICQ* (ICQ, 2004).

4.4.2 – Planejamento

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.

Sendo assim, o estudo em questão tem como hipótese a redução do tempo na identificação de alterações entre versões evolutivas subseqüentes do modelo compartilhado, além do incremento da percepção de alterações encontradas por cada desenvolvedor. Este estudo visa obter também informações subjetivas quanto à utilização do protótipo, como satisfação dos participantes no uso deste.

A definição dos objetivos deste estudo é descrito pela estrutura a seguir, apresentada em (TRAVASSOS, GUROV, AMARAL, 2002):

> **Analisar** o mecanismo *MAIS* em uma atividade que envolve a revisão informal de um documento em relação a alterações realizadas em um modelo relacionado.

Com o propósito de caracterizar a utilidade do mecanismo, referente à satisfação dos usuários na utilização e aplicabilidade na tarefa.

Com respeito à satisfação em decorrência do ganho de rapidez e eficiência na realização da atividade de modelagem.

No ponto de vista do desenvolvedor

No contexto do ambiente *Odyssey Share*.

4.4.3 – Participantes

Para a realização deste estudo, foram convidados quatro voluntários. Dois são alunos do curso de ciência da computação da Universidade Federal do Rio de Janeiro. Os outros dois são alunos do Programa de Engenharia de Sistemas e Computação, da Coordenação dos Programas de Pós-Graduação em Engenharia.

Os quatro voluntários possuem alguma experiência anterior em modelagem de software e no uso do ambiente *Odyssey Share*. Porém, nenhum destes possui experiência no uso do protótipo *MAIS*.

4.4.4 – Instrumentos

Os instrumentos usados na execução do estudo foram adaptados do planejamento proposto em (MANGAN et al., 2002):

- Um termo de compromisso assinado.
- Dois questionários, para serem preenchidos pelos participantes.
- Um documento descrevendo a tarefa.
- Gravação da realização da tarefa, por cada participante, em vídeo.

Todos os quatro voluntários utilizaram a versão 1.1.0 do ambiente *Odyssey Share*. Dois voluntários foram selecionados para utilizar esta versão do ambiente combinada com a versão 1.1.0 do protótipo *MAIS*.

4.4.5 – Procedimento

O procedimento adotado para execução no estudo é resumido pelos passos a seguir listados. A ordem de execução destes é definida pela ordem de apresentação dos mesmos.

- Assinatura do termo de compromisso por cada participante.
- Preenchimento do questionário de caracterização de participante.
- Leitura, por parte de cada participante, da descrição da tarefa.

- ❖ Explicação oral sobre o uso do protótipo e do *Odyssey Share* (caso necessário).
- ❖ Teste de gravação de vídeo.
- ❖ Realização da tarefa e gravação desta.
- ❖ 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 e (ii) os que não o utilizaram. 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 *Odyssey Share* 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 *Odyssey Share* 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.

4.4.6 – Resultados

Basicamente, os participantes se mostraram nivelados quanto a experiências anteriores relacionadas à modelagem concorrente de artefatos descritos pela notação

UML e utilização de aplicações colaborativas. Esta informação foi inferida pelo preenchimento do questionário de caracterização de participante.

A Tabela 5 apresenta as informações observadas na realização do estudo. Foram totalizadas 19 (dezenove) mudanças entre as versões do modelo, referente à tarefa em questão.

Dentre as questões respondidas pelos participantes, através do preenchimento do questionário de realização de tarefa, foram emitidas as seguintes opiniões, quanto ao ferramental oferecido: realização da tarefa com (i) **eficiência** e (ii) **rapidez**, (iii) possibilidade de determinado participante ter rapidamente **aumento de produtividade**, devido à utilização deste ferramental, e (iv) a **satisfação** na utilização deste ferramental. Para expressar as opiniões dos participantes sobre estas afirmações, são usados valores definidos em um intervalo de cinco valores, onde 1 representa discordar totalmente e 5 concordar plenamente.

	Participante 1	Participante 2	Participante 3	Participante 4
Escolaridade	Estudante de Graduação	Estudante de Graduação	Estudante de Doutorado	Estudante de Mestrado
Disponibilidade do Protótipo	Sim	Não	Sim	Não
Modificações Encontradas (%)	95%	79%	63%	44%
Duração da tarefa (min)	51 min	35 min	37 min	16 min
Eficiência	4	1	3	2
Rapidez	4	1	3	2
Aumento de Produtividade	5	1	4	1
Satisfação	4	2	4	2

Tabela 5 - Dados Obtidos no Estudo

O participante 2 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 4 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 1 ter levado mais tempo que o participante 2 para identificar as mudanças, o primeiro as descreveu em um nível de detalhe maior do que o segundo. Além disso, o participante 1 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 1 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 à determinado evento fossem apresentados ao clicar com o botão direito do *mouse*.

Por fim, o participante 3 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 suporte 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.

4.4.7 – Conclusões

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. Acreditamos 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 suporte do protótipo.

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 diferenciação do questionário de execução de tarefa entre os grupos de participantes que usaram ou não o protótipo avaliado, explorando detalhes destes dois cenários.

Este estudo serviu ainda como base para a realização de experimentos melhor controlados, no contexto de uma tese de doutorado, em fase final de elaboração.

4.5 – Considerações sobre o Protótipo

O protótipo apresentado neste capítulo foi concebido baseando-se na proposta do mecanismo *MAIS* para apoio à percepção de mudanças. Desta forma, o protótipo deve fornecer informações a um grupo de desenvolvedores de software, que manipulam concorrentemente cópias de um dado modelo de classes *UML*, de forma que estes percebam individualmente, durante a realização desta tarefa, o que está sendo realizado por todos.

A utilização deste protótipo torna-se mais interessante quando realizada em conjunto com sistemas de controle de versões de artefatos *UML*, como o *Odyssey-VCS* (OLIVEIRA, MURTA, WERNER, 2004). Este tipo de sistema de controle de versões tem como unidade de versionamento itens de um modelo, enquanto em sistemas como o *CVS*, esta unidade é representada por arquivos do sistema operacional. Dado que uma versão de um modelo é estabelecida por esse tipo de ferramenta, os desenvolvedores podem utilizar o protótipo *MAIS* durante a elaboração da próxima versão a ser criada. O conhecimento, por parte de um desenvolvedor, do que está sendo realizado pelos demais, sobre o modelo compartilhado, pode guiar as ações deste, de maneira a reduzir o esforço empregado com resolução de conflitos causados na unificação das contribuições individuais dos desenvolvedores.

Os tipos de agrupamentos foram escolhidos com base em observações informais, relacionadas à manipulação de modelos de classes *UML*. No protótipo *MAIS*, a obtenção de características como relevância e proximidade foi apresentada como uma proposta inicial desta. No caso de proximidade, por exemplo, esta se baseia apenas no relacionamento estrutural. Para esse agrupamento, seria interessante levar em consideração características relacionadas à semântica dos elementos de um modelo de classes.

As informações geradas na utilização do protótipo *MAIS* podem servir como base para abordagens que tratem de aspectos relacionados ao andamento do trabalho em grupo. Por exemplo, uma ferramenta para analisar o ritmo de trabalho de um grupo de desenvolvedores (MANGAN, SILVA, WERNER, 2004) pode utilizar as informações geradas pelo *MAIS* como fonte de dados a apresentar.

O estudo de caso, de foco qualitativo, realizado para observação da utilização do protótipo se mostrou restrito quanto à configuração do ambiente de execução e treinamento. Mesmo assim, foi possível identificar a utilidade do protótipo *MAIS*,

devido ao nível de detalhe de informações de mudanças relatadas, na realização da tarefa proposta pelo estudo com a utilização deste protótipo. Os participantes que não utilizaram o protótipo relataram que era necessário um ferramental para melhor realizar a tarefa proposta. Porém, foram identificadas, na realização do estudo, deficiências do protótipo *MAIS*, tal como a insuficiência de informações relativas a determinado tipo de evento.

Capítulo 5 – Conclusões

5.1 – Considerações sobre a Dissertação

Esta dissertação apresentou o problema decorrente do isolamento de indivíduos, pertencentes a uma equipe de desenvolvimento de software globalizada, na realização da tarefa de modelagem de software. Estes indivíduos estão dispersos no tempo e no espaço, sendo esta condição motivada por fatores como: (i) busca de recursos especializados em outros países e (ii) pressões para se ter "tempo de mercado" (HERBSLEB e MOITRA, 2001). O isolamento desses desenvolvedores ocorre pela indisponibilidade de interações face-a-face. Cada desenvolvedor trabalha sobre uma cópia de um determinado modelo de software, sem possuir informações sobre o trabalho realizado sobre as demais cópias.

O conceito de percepção foi apresentado como técnica para a redução deste isolamento. Para tal, informações relativas a mudanças de estado do modelo compartilhado são propagadas, através de mecanismos, para os desenvolvedores que participam desta interação colaborativa.

Aplicações colaborativas podem contemplar esse tipo de mecanismo. Porém, o suporte à tarefas oferecido por esse tipo de aplicação é menos completo do que os similares de uso individual (LI e LI, 2002). O efeito causado pela mudança de utilização de uma determinada ferramenta de modelagem, de uso individual, por uma equivalente de uso colaborativo, pode ser negativo. O ritmo de trabalho de uma equipe de desenvolvimento de software, bem como a produtividade de seus integrantes, pode ser prejudicada com esta mudança. Por exemplo, o tempo de aprendizado para utilização de novas ferramentas pode ser inadequado com as expectativas da organização onde está inserida esta equipe

Baseado nestes fatores, foram apresentados como produtos desta dissertação uma proposta de mecanismo para apoio à percepção de modelos compartilhados e um protótipo correspondente, de nome *MAIS*. Tanto a proposta do mecanismo quanto o protótipo são de uso geral, isto é, não são específicos a uma ferramenta de modelagem. Os modelos que este mecanismo trata são representados pela notação *UML*.

O mecanismo para apoio à percepção *MAIS* realiza o monitoramento de ações nos ambientes locais de trabalho de cada usuário, de maneira não intrusiva, e divulga

essas informações entre os desenvolvedores utilizando a metáfora de eventos (PRINZ, 1999). As informações são apresentadas considerando características que minimizem a sobrecarga de informações. Ao monitorar a manipulação das cópias de um modelo compartilhado em determinada ferramenta de modelagem, o mecanismo *MAIS* é capaz de perceber ações que ainda não estão disponíveis na cópia armazenada no repositório centralizado, que representa o modelo compartilhado em si. Dessa forma, é possível que desenvolvedores, de posse desta informação, sejam alertados para futuros conflitos de edição, permitindo que sejam tomadas ações preventivas (SARMA, NOROOZI, VAN DER HOEK, 2003).

Para apresentar as informações sobre a evolução do modelo compartilhado, três procedimentos são propostos pelo mecanismo *MAIS*: (i) classificação, (ii) agrupamentos e (iii) filtragem de eventos. Dentre as formas de classificação, é apresentado o conceito de relevância de eventos, que atribui um destaque maior ou menor a cada evento, na tentativa de ressaltar a importância de determinada informação no ponto de vista de cada desenvolvedor, com base em seu histórico de ações. O agrupamento de eventos é útil para facilitar a busca de informações relevantes, oferecendo perspectivas sobre a evolução do artefato. Através da quantidade de eventos gerados por um desenvolvedor que são associados a determinado agrupamento, tem-se o indicador de como está o ritmo de trabalho deste desenvolvedor (por exemplo, um agrupamento por usuário), além do foco de trabalho deste (por exemplo, pelo agrupamento de classes de um modelo de classes).

Para reduzir a sobrecarga de informações relativas a manipulações sobre o artefato compartilhado, filtros foram estabelecidos. O conceito de ciência de eventos é utilizado também neste contexto; a informação de que determinado desenvolvedor está ciente sobre certo evento deve ser obtida, podendo ser usada na filtragem de exibição deste, para o desenvolvedor em questão. Como a informação de ciência sobre determinada modificação é disponibilizada para todos os desenvolvedores que interagem sobre um mesmo modelo compartilhado, torna-se possível que novas ações sejam realizadas baseadas nesse conhecimento.

Foi ainda realizado um estudo de caso, de caráter qualitativo, com o objetivo de observar indícios sobre a utilidade do protótipo *MAIS*. Neste estudo, foi proposta uma tarefa na qual um participante realiza a comparação entre duas versões consecutivas de um modelo de classes, com o objetivo de identificar quais modificações foram realizadas. Os participantes deste estudo foram divididos em dois grupos; um deles

utilizou o protótipo e o outro realizou a tarefa sem contar com o mecanismo. O estudo tem como hipótese que a utilização do protótipo reduz o tempo necessário para realização da tarefa e aumenta o número de modificações percebidas.

Apesar do planejamento do estudo apresentar deficiências, como por exemplo, a falta de treinamento adequado para utilização do protótipo, algumas características interessantes foram observadas: (i) o tempo observado para identificação das mudanças não foi reduzido com a utilização do protótipo; (ii) o número de modificações percebidas foi maior quando o protótipo foi usado de forma efetiva; (iii) as modificações foram relatadas em um nível de detalhe maior quando o protótipo foi usado de forma efetiva.

5.2 – Contribuições

Podemos destacar como principais contribuições deste trabalho:

a) Proposta de mecanismo não-intrusivo, independente de ferramentas de modelagem *UML*, que realiza coleta implícita, distribuição e apresentação de informações de mudanças sobre cópias de modelos compartilhados. Implementação do protótipo deste mecanismo, também independente de ferramenta de modelagem.
b) Indicação de como está focado o trabalho, assim como o ritmo deste, dos desenvolvedores. Esta indicação é apresentada pelos agrupamentos de eventos.
c) Indicação de possíveis especialistas sobre determinado modelo. Esta indicação pode ser inferida, por exemplo, pelo volume e pela frequência de eventos gerados por determinados desenvolvedores. Cabe a cada desenvolvedor analisar se os eventos gerados por um desenvolvedor candidato a especialista o credenciam para tal.
d) As informações sobre relevância de eventos são obtidas no processo de interação entre desenvolvedores e modelo compartilhado, não sendo necessária a mudança na forma de trabalho destes.
e) Utilização da tecnologia de espaço de tuplas no contexto de aplicações colaborativas. Dentre a literatura revista, este tipo de tecnologia apenas era encontrado no contexto da área de Inteligência Artificial.
f) Geração de base histórica de informações relativas a interações de desenvolvedores sobre determinados modelos compartilhados, ao passo

que esses utilizem o mecanismo ao longo do tempo. Esta base é construída pelo armazenamento de eventos, que utilizam a representação 5W+1H, no espaço de tuplas.

g) Definição e realização de um estudo de caso, de foco qualitativo, sobre a utilização do protótipo *MAIS* para a verificação de mudanças entre duas versões de um modelo de classes, tendo sido realizado no contexto do ambiente *Odyssey Share*.

h) Acoplamento do protótipo junto ao ambiente *Odyssey Share*, ampliando a capacidade de oferta de percepção deste. Este acoplamento foi realizado através de mínimas adaptações para entrar em conformidade com o Serviço de Notificação de Eventos deste ambiente.

5.3 – Limitações e Trabalhos Futuros

O trabalho realizado nesta dissertação apresenta algumas limitações. A seguir, algumas delas são apresentadas, bem como trabalhos que podem ser realizados futuramente, como extensões deste trabalho:

a) Quando um desenvolvedor percebe uma nova alteração, realizada por um outro, para replicá-la em sua cópia do modelo, este tem que realizá-la de forma manual. Está previsto contemplar, na representação do evento, o procedimento realizado na ferramenta de modelagem para gerá-lo. Isto é necessário para reproduzir o evento na estação local de um desenvolvedor.

b) A informação de ciência de eventos é relatada de forma explícita. Acreditamos ser possível inferir, por determinadas ações realizadas por um desenvolvedor, que este está ciente sobre a ocorrência de certa mudança.

c) Dado que uma interação se estabeleça, relativa à manipulação concorrente de um modelo de software, todos os desenvolvedores devem utilizar a mesma ferramenta de modelagem. É possível estabelecer uma interação em que as ferramentas de modelagem sejam heterogêneas, utilizando o conceito de eventos semânticos (LI e LI, 2002) (MANGAN, 2003). Estes eventos seriam obtidos através de gestos dos desenvolvedores sobre a interface da ferramenta de modelagem. Cada

ferramenta define um modelo semântico que descreve, para determinado evento, uma sequência de gestos.

d) Os eventos são apresentados, pela interface protótipo *MAIS*, na forma de texto. Seria interessante explorar novas formas de apresentação destes, associadas aos locais onde estes ocorreram (por exemplo, alguma forma de apresentar mudanças de uma classe na representação gráfica da ferramenta de modelagem relativa ao diagrama de classes onde esta está inserida). Além disso, os eventos poderiam ser apresentados de modo a guiar os desenvolvedores no processo de busca de informações pertinentes, sugerindo ações que os auxiliem na modelagem do artefato.

Abaixo, são listadas algumas características que necessitariam ser exploradas em trabalhos futuros:

e) Explorar a base histórica de eventos, associada ao espaço de tuplas, para identificar padrões nas informações de mudança, úteis para a tomada de decisão. Estes padrões podem auxiliar a gerência do processo de software, determinando competências entre os membros da equipe e fornecendo informações sobre produtividade e qualidade dos artefatos produzidos.

f) Mecanismos para detecção de conflitos de forma automática e semi-automática precisam ser pesquisados e implementados. A informação de existência de conflitos entre ações geradas pelos desenvolvedores os auxiliariam no momento de convergência das alterações realizadas sobre o modelo compartilhado em um estado global deste. Um indicador de quão grave um conflito é poderia ser adotado.

g) Existe a necessidade de pesquisa sobre uma melhor forma de se calcular relevância de eventos.

h) Refinar o protótipo desenvolvido, para que este contemple características como: (a) definição do tipo de eventos no momento da instalação e (b) identificar correlação entre eventos na lista do usuário corrente e na dos demais.

Referências Bibliográficas

ALTMANN, J., POMBERGER, G., 1999, "Cooperative Software Development: Concepts, Model and Tools" *Proceedings of the Technology of Object-Oriented Languages and Systems*, pp. 194-209, Santa Barbara, California, United States, August.

ANDERSON, K. M., BOUVIN, N. O., 2000, "Supporting project awareness on the WWW with the iScent framework", *SIGGROUP Bull.*, v. 21, n. 3, pp. 16-20

ARAÚJO, R. M., 2000, *Ampliando a Cultura de Processos de Software – Um Enfoque Baseado em Groupware e Workflow*, Dsc Thesis, COPPE-UFRJ, Rio de Janeiro, RJ, Brazil.

BEGOLE, J., ROSSON, M. B., SHAFFER, C. A., 1999, "Flexible collaboration transparency: supporting worker independence in replicated application-sharing systems", *ACM Trans.Comput.-Hum.Interact.*, v. 6, n. 2, pp. 95-132

BERGER, M., VOLKSEN, G., SCHILL, A., 1998, "Supporting Autonomous Work and Reintegration in Collaborative Systems" *Coordination Technology for Collaborative Applications - Organizations, Processes, and Agents [ASIAN 1996 Workshop]*, pp. 177-198.

BORGES, M. R. S., PINO, J. A., 2000, "Requirements for Shared Memory in CSCW applications". In: *Proceedings of WITS'2000, 10th Annual Workshop On Information Technologies and System*, pp. 211-216, Brisbane, Australia.

BORGES, M. R. S., PINO, J. A., 1999, "Awareness Mechanisms for Coordination in Asynchronous CSCW". In: *Proceedings of 9th Workshop on Information Technologies and Systems*, pp. 69-74, Charlotte, North Caroline, United States, December.

BORLAND, 2004a, "Together Control Center". In: http://www.borland.com/together, Accessed in 11/11/2004.

BORLAND, 2004b, "Together Editon for Eclipse". In: http://www.borland.com/together/eclipse/index.html, Accessed in 11/11/2004.

BOULILA, N., DUTOIT, A. H., BRUEGGE, B., 2003, "D-Meeting: an Object-Oriented Framework for Supporting Distributed Modelling of Software" *International Workshop on Global Software Development, International Conference on Software Engineering*, pp. 34-38, Portland, Oregon, United States, May.

CARRIERO, N., GELERNTER, D., 1989, "Linda in context", *Communications of ACM*, v. 32, n. 4, pp. 444-458

CHU-CARROLL, M. C., SPRENKLE, S., 2000, "Coven: brewing better collaboration through software configuration management". In: *Proceedings of 8th ACM*

SIGSOFT International Symposium on Foundations of Software Engineering, pp. 88-97, San Diego, California, United States.

DAVID, J. M. N., BORGES, M. R. S., 2001, "Selectivity of Awareness Components in Asynchronous CSCW Environments". In: *Proceedings of 7th International Workshop on Groupware (CRIWG 2001)*, pp. 115-124, Darmstadt, Germany, September.

DE SOUZA, C. R. B., BASAVESWARA, S. D., REDMILES, D. F., 2002, "Supporting Global Software Development with Event Notification Servers". In: *Proceedings of 24th International Conference on Software Engineering, International Workshop on Global Software Development*, pp. 9-13, Orlando, Florida, United States, May.

DOURISH, P., BELLOTTI, V., 1992, "Awareness and coordination in shared workspaces" *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pp. 107-114, Toronto, Ontario, Canada.

EDWARDS, W. K., MYNATT, E. D., 1997, "Timewarp: techniques for autonomous collaboration" *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 218-225, Atlanta, Georgia, United States.

ELLIS, C. A., GIBBS, S. J., REIN, G., 1991, "Groupware: some issues and experiences", *Communications of ACM*, v. 34, n. 1, pp. 39-58, ACM Press.

FARSHCHIAN, B. A., 2001, "Integrating geographically distributed development teams through increased product awareness", *Information Systems.*, v. 26, n. 3, pp. 123-141

FOGEL, K. F., BAR, M., 2001, *Open Source Development with Cvs*. 2nd ed., Coriolis Group Books.

FROEHLICH, J., DOURISH, P., 2004, "Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams". In: *Proceedings of 26th International Conference on Software Engineering*, pp. 387-396, Edinburgh, United Kingdom, May.

GAMMA, E., HELM, R., JOHNSON, R., et al., 1995, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley.

GENTLEWARE, 2004, "Poseidon for UML". In: http://www.gentleware.com/products/index.php3, Accessed in 11/11/2004.

GIGASPACES, 2004, "Gigaspaces Grid Server". In: http://www.gigaspaces.com/docs/doc/index.htm, Accessed in 23/11/2004.

GREENBERG, S., MARWOOD, D., 1994, "Real time groupware as a distributed system: concurrency control and its effect on the interface". In: *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, pp. 207-217, Chapel Hill, North Carolina, United States.

GROSS, T., SPECHT, M., 2001, "Awareness in Context-Aware Information Systems" *Mensch & Computer - 1. Fachuebergreifende Konferenz* , pp. 173-182, Bad Honnef, Germany.

GRUNDY, J., HOSKING, J., 2002, "Engineering plug-in software components to support collaborative work", *Software: Practice and Experience*, v. 32, n. 10, pp. 983-1013

GUTWIN, C., GREENBERG, S., 2001, "A Descriptive Framework of Workspace Awareness for Real-Time Groupware", *Computer Supported Cooperative Work*, v. 11, pp. 411-446, Kluwer Academic Publishers.

HERBSLEB, J. D., MOITRA, D., 2001, "Guest Editors' Introduction: Global Software Development,", *IEEE Softw*, v. 18, n. 2, pp. 16-20, IEEE Computer Society Press.

IBM, 2004, "Rational Rose Technical Developer". In: http://www-306.ibm.com/software/awdtools/developer/technical/, Accessed in 11/11/2004.

ICQ, 2004, "ICQ". In: http://www.icq.com, Accessed in 25/11/2004.

KOBYLINSKI, R., CREIGHTON, O., DUTOIT, A., et al., 2002, "Building awareness in global software engineering: using issues as context" *International Workshop on Global Software Development*, pp. 18-22, Orlando, Florida, United States.

LAYZELL, P., BRERETON, O. P., FRENCH, A., 2000, "Supporting collaboration in distributed software engineering teams". In: *Proceedings of the Seventh Asia-Pacific Software Engineering Conference*, pp. 38-45, Singapore.

LI, D., LI, R., 2002, "Transparent sharing and interoperation of heterogeneous single-user applications". In: *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, pp. 246-255, New Orleans, Louisiana, United States.

LOPES, M. A. D. M., MANGAN, M. A. S., WERNER, C. M. L., 2004a, "MAIS: um Mecanismo para Apoio à Percepção aplicado a Modelos de Software Compartilhados". In: *Proceedings of 1th Brazilian Workshop on Technologies for Collaboration (WCSCW), LA-Web/Webmedia*, Ribeirão Preto, SP, Brazil, October.

LOPES, M. A. M., MANGAN, M. A. S., WERNER, C. M. L., 2004b, "MAIS: uma ferramenta de percepção para apoiar a edição concorrente de modelos de análise e projeto". In: *Proceedings of 18th Brazilian Symposium on Software Engineering , Tools Session*, pp. 61-66, Brasília, DF, Brazil, October.

MANGAN, M. A. S., 2003, *Uma Arquitetura Transparente para um Ambiente Colaborativo de Desenvolvimento de Software*, Qualifier Exam, COPPE - UFRJ, Rio de Janeiro, RJ, Brazil.

MANGAN, M. A. S., ARAÚJO, R. M., KALINOWSKI, M., et al., 2002, "Towards the Evaluation of Awareness Information Support Applied to Peer Reviews". In:

Proceedings of 7th Conference on Computer Supported Cooperative Work in Design (CSCWD´2002), pp. 49-54, Rio de Janeiro, Brazil, September.

MANGAN, M. A. S., BORGES, M. R. S., WERNER, C. M. L., 2004, "Increasing Awareness in Distributed Software Development Workspaces". In: *Proceedings of 10th Groupware: Design, Implementation, and Use* , v. 3198, pp. 84-91, San Carlos, Costa Rica, September.

MANGAN, M. A. S., SILVA, I. A. D., WERNER, C. M. L., 2004, "GAW: Uma Ferramenta de Percepção de Grupo Aplicada no Desenvolvimento de Software". In: *Proceedings of 18th Brazilian Symposium on Software Engineering , Tools Session*, pp. 25-30, Brasília, DF, Brazil, October.

MEIRE, A., BORGES, M. R. S., ARAÚJO, R. M., 2003, "Supporting Collaborative Drawing with the Mask Versioning Mechanism". In: *Proceedings of 9th Groupware: Design, Implementation, and Use (CRIWG 2003)*, v. 2806, pp. 208-223, Autrans, France, September.

MICROSOFT, 2004a, "Microsoft Word". In: http://office.microsoft.com/pt-br/FX010857991046.aspx, Accessed in 14/12/2004.

MICROSOFT, 2004b, "Windows NetMeeting". In: http://www.microsoft.com/windows/netmeeting/, Accessed in 25/11/2004.

MOLLI, P., SKAFA-MOLLI, H., OSTER, G., et al., 2002, "SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments". In: *Proceedings of 7th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2002)*, pp. 80-84, Rio de Janeiro, RJ, Brazil, October.

MURTA, L. G. P., PIRES, A. P. V., BLOIS, A. P. T. B., et al., 2004, "Run-time Variability through Component Dynamic Loading". In: *Proceedings of XVIII Brazilian Symposium on Software Engineering , XI Tools Session*, pp. 67-72, Brasília, Brazil, October.

MURTA, L. G. P., 2004, *Odyssey-SCM: Uma Abordagem de Gerência de Configuração de Software para o Desenvolvimento Baseado em Componentes*, Qualifier Exam, COPPE-UFRJ, Rio de Janeiro, RJ, Brazil.

ODYSSEY, 2004, "Odyssey Share Project". In: http://www.cos.ufrj.br/~odyssey/share/, Accessed in 16/11/2004.

OLIVEIRA, H., MURTA, L. G. P., WERNER, C. M. L., 2004, "Odyssey-VCS: Um Sistema de Controle de Versões para Modelos Baseados no MOF". In: *Proceedings of 18th Brazilian Symposium on Software Engineering , Tools Session*, pp. 85-90, Brasília, DF, Brazil, October.

OMG, 2004a, "MDA Guide Version 1.0.1". In: http://www.omg.org/docs/omg/03-06-01.pdf, Accessed in 05/11/2004.

OMG, 2004b, "Unified Modelling Language". In: http://www.omg.org/cgi-bin/apps/doc?formal/04-07-02.pdf, Accessed in 25/11/2004.

PFLEEGER, S. L., 2001, *Software Engineering: Theory and Practice*. 2nd ed., Prentice Hall PTR.

PINHEIRO, M. K., 2001, *Mecanismo de Suporte à Percepção em Ambientes Cooperativos*, Msc. Dissertation, PPGC/UFRGS, Porto Alegre, Brasil.

PRESSMAN, R. S., 2000, *Software Engineering: A Practitioner's Approach*. 5rd ed., McGraw-Hill Higher Education.

PRIKLADNICKI, R., AUDY, J., EVARISTO, R., 2003, "Requirements Management in Global Software Development: Preliminary Findings from a Case Study in a SW-CMM context". In: *Proceedings of 25th International Conference on Software Engineering, International Workshop on Global Software Development*, pp. 53-58, Portland, Oregon, USA, May.

PRINZ, W., 1999, "NESSIE: an awareness environment for cooperative settings". In: *Proceedings of the 6th European Conference on Computer Supported Cooperative Work*, pp. 391-410, Copenghagen Denmark.

ROSA, M. G. P., BORGES, M. R. S., SANTORO, F. M., 2003, "A Conceptual Framework for Analyzing the Use of Context in Groupware". In: *Proceedings of 9th Groupware: Design, Implementation, and Use*, v. 2806, pp. 300-313, Autrans, France, October.

SARMA, A., NOROOZI, Z., VAN DER HOEK, A., 2003, "Palantír: raising awareness among configuration management workspaces". In: *Proceedings of 25th International Conference on Software Engineering (ICSE)*, pp. 444-454, Portland, Oregon, United States, May.

SCHÜMMER, T., SCHÜMMER, J., 2001, "Tools for XP Development - Support for Distributed Teams in eXtreme Programming". In Succi, G. and Marchesi, M., *eXtreme Programming Examined*, Addison Wesley.

SOHLENKAMP, M., 1998, *Supporting Group Awareness in Multi-User Environments through Perceptualization*, Msc Dissertation - Fachbereich Mathematik, Informatik der Universität - Gesamthochschule - Paderborn.

SUBVERSION, 2004, "Subversion project". In: http://www.subversion.org, Accessed in 12/10/2004.

SUN, 2004a, "JAR Files". In: http://java.sun.com/docs/books/tutorial/jar/, Accessed in 20/11/2004.

SUN, 2004b, "Java Foundation Classes (JFC/Swing)". In: http://java.sun.com/products/jfc/index.jsp, Accessed in 23/11/2004.

SUN, 2004c, "Java Technology". In: http://www.java.sun.com, Accessed in 16/11/2004.

SUN, 2004d, "JavaSpaces Principles, Patterns, and Practice". In: http://java.sun.com/developer/Books/JavaSpaces/, Accessed in 23/11/2004.

TAM, J., GREENBERG, S., 2004, "A Framework for Asynchronous Change Awareness in Collaboratively-Constructed Documents". In: *Proceedings of 10th Groupware: Design, Implementation, and Use*, v. 3198, pp. 67-83, San Carlos, Costa Rica, September.

TAM, J., MCCAFFREY, L., GREENBERG, S., 2000, *Change Awareness in Software Engineering Using Two Dimensional Graphical Design and Development Tools*. Report 2000-670-22, Departament of Computer Science, University of Calgary, Alberta, Canada.

TRAVASSOS, G. H., GUROV, D., AMARAL, E. A. G. G., 2002, *Introdução à Engenharia de Software Experimental*. Report ES-590/02-April, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, RJ, Brazil.

VAN DER HOEK, A., REDMILES, D., DOURISH, P., et al., 2004, "Continuous Coordination: A New Paradigm for Collaborative Software Engineering Tools" *Workshop on Directions in Software Engineering Environments, International Conference on Software Engineering (ICSE 2004)*, pp. 29-36, Edinburgh, United Kingdom, May.

VIDOT, N., CART, M., FERRIÉ, J., et al., 2000, "Copies convergence in a distributed real-time collaborative environment". In: *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, pp. 171-180, Philadelphia, Pennsylvania, United States.

VIEIRA, V., 2003, *Ariane: Um Mecanismo de Apoio à Percepção em Base de Dados Compartilhadas*, Msc Dissertation, COPPE - UFRJ, Rio de Janeiro, RJ, Brazil.

ZAFFER, A., SHAFFER, C., EHRICH, R., et al., 2001, *NetEdit: A Collaborative Editor*. Report TR-01-13, Computer Science, Virginia Tech, United States.

Anexo A - Questionário de Caracterização

Q1	AVALIAÇÃO DE MODELAGEM CONCORRENTE DE ARTEFATOS *UML* **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: [ ]
[ ]
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 Modelagem Concorrente e Distribuída

11. Participou de alguma atividade concorrente ou distribuída nos últimos seis meses?

( ) Sim ( ) Não

Nas questões seguintes, considere a tarefa distribuída mais complexa já realizada por você.

12. Durante a tarefa foram manipulados:

( ) Código fonte ( ) Interface de usuário ( ) Modelos e diagramas

( ) Outros: [ ]

13. Os objetos manipulados são produtos de autoria:

( ) de minha própria autoria ( ) da equipe ( ) de terceiros

14. A iniciativa para a seleção de um colega para colaborar é:

( ) minha ( ) do colega ( ) de um superior hierárquico

15. Foi utilizada uma técnica de trabalho em grupo ou um sistema de apoio ou mediação? Qual?

[ ]

[ ]

[ ]

16. Quantas pessoas participaram da tarefa (informe o número)? [ ][ ]

17. Quanto tempo durou a tarefa (informe o total de horas)? [ ][ ]

18. Descreva resumidamente objetivos e principais passos da tarefa realizada:

[ ]

[ ]

[ ]

19. Cite um artigo, livro ou autor que considere importante na área de programação concorrente ou distribuída:

[ ]

[ ]

[ ]

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 a Linguagem *UML* e o Ambiente O*dyssey Share*

21. Você já utilizou a linguagem *UML*? ( ) Sim ( ) Não

22. Indique as ferramentas que utiliza para modelagem de artefatos descritos pela linguagem *UML*:

[ ]

[ ]

23. Você já utilizou o *Odyssey Share* para modelagem de artefatos *UML*?

( ) 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 que?

[ ]

[ ]

[ ]

32. Sua experiência de trabalho em grupo poderia ser considerada:

( ) péssima ( ) ruim ( ) razoável ( ) boa( ) ótima

Obrigado por sua colaboração!

Anexo B – Descrição da Tarefa

T	Avaliação de Modelagem Concorrente de Artefatos *UML* Descrição da Tarefa	ID

Instruções

O diagrama de classes visualizado no Odyssey SDE foi criado com base em uma especificação informal, relacionada com a representação de eventos no protótipo *MAIS*.

Você possui uma descrição informal e resumida das classes que representam este diagrama, além de duas versões deste. Estas versões representam visões de estados do modelo de classes, que representa eventos no protótipo MAIS, antes e depois de ser aplicado um conjunto de mudanças.

Sua tarefa é revisar estes diagramas e destacar as alterações que diferenciam as versões destes. Estas alterações deverão ser escritas na seção 'Alterações Observadas'.

Especificação Resumida das Classes do Diagrama de Eventos

1) Um evento, no protótipo *MAIS*, é a composição de uma instância de alguma subclasse da classe *Event* com alguma classe que implementa a interface *Element*.
2) A classe *Event* representa a ação relativa a um evento. No protótipo *MAIS* estas ações são: (i) registro no sistema (classe *Register*); (ii) criação de elementos (classe *Create*); (iii) remoção de elementos (classe *Delete*); (iv) atualização de elementos (classe *Update*).
3) A interface *Element* descreve o protocolo a ser seguido por uma classe, referente a um elemento do modelo de classes. Os elementos são: (i) classe (classe *Class*); (ii) atributo (classe *Attribute*); (iii) metodo (classe *Method*); relacionamento entre classes (classe *Relationship*).
4) A classe *MaisRelevance* realiza operações sobre eventos, com o propósito de atribuir pesos a estes, tentando indicar o quão importante determinado evento é para um dado desenvolvedor.

Alterações observadas

Anexo C – Questionário sobre a Tarefa

Q2	Avaliação de Modelagem Concorrente de Artefatos *UML* **Segundo Questionário do Avaliador (32 questões)**	**ID**

Sobre A Utilização Dos Componentes

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 do mecanismo foi simples.	1 2 3 4 5 NA
03	Eu consigo efetivamente realizar o meu trabalho utilizando este mecanismo.	1 2 3 4 5 NA
04	Eu sou capaz de realizar o meu trabalho rapidamente utilizando este mecanismo.	1 2 3 4 5 NA
05	Eu sou capaz de realizar o meu trabalho com eficiência utilizando este mecanismo.	1 2 3 4 5 NA
06	Sinto me confortável utilizando o mecanismo.	1 2 3 4 5 NA
07	Foi fácil aprender a utilizar o mecanismo.	1 2 3 4 5 NA
08	Acredito que possa me tornar rapidamente mais produtivo utilizando o mecanismo.	1 2 3 4 5 NA
09	A ferramenta de modelagem 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 de modelagem 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 de modelagem é de fácil compreensão.	1 2 3 4 5 NA

14	A informação fornecida é efetiva na ajuda em completar as tarefas e os cenários.	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 do mecanismo é agradável.	1 2 3 4 5	NA
17	Eu gostei de utilizar a interface do mecanismo.	1 2 3 4 5	NA
18	O mecanismo possui todas as funcionalidades por mim esperadas.	1 2 3 4 5	NA
19	Ao todo, estou satisfeito com o mecanismo.	1 2 3 4 5	NA

20. Liste o(s) aspecto(s) positivos:

[ ]
[ ]
[ ]
[ ]
[ ]

21. Liste o(s) aspecto(s) negativos:

[ ]
[ ]
[ ]
[ ]
[ ]

Sobre a Experiência de Utilização do Mecanismo

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 totalmente satisfeito com a tarefa.	1 2 3 4 5	NA
26	Eu usaria o mecanismo para realização modelagem concorrente na prática.	1 2 3 4 5	NA

Sobre os Recursos Utilizados

27. Você já havia utilizado este recurso em outro programa?

Indicação de presença de usuário ( ) Sim ( ) Não

Histórico de contribuições individuais ( ) Sim ( ) Não

Histórico de relações entre contribuições ( ) Sim ( ) Não

28. Indique quais programas computador apresentam pelo menos um dos recursos acima.

[ ]
[ ]
[ ]

29. Resuma sua opinião sobre a aplicação dos recursos na tarefa realizada:

[ ]
[ ]
[ ]
[ ]
[ ]

Sobre os Procedimentos da Avaliação

30. Você considera que o treinamento aplicado para o uso da ferramenta de modelagem e do mecanismo foi suficiente? Você teria sugestões?

[ ]
[ ]
[ ]
[ ]
[ ]

31. Você considera que existiu alguma dificuldade na interpretação dos textos e informações apresentados pela ferramenta de modelagem e pelo mecanismo? Se sim, por favor, exemplifique.

[ ]
[ ]
[ ]
[ ]
[ ]

32. As informações disponibilizadas sobre o sistema a ser gerenciado, as técnicas propostas e sua aplicação foram de fácil compreensão? Você sentiu falta de alguma informação nesta descrição?

[ ]
[ ]
[ ]
[ ]
[ ]

Novamente, obrigado por sua colaboração!