GAW: Um Mecanismo Visual de Percepção de Grupo Aplicado ao Desenvolvimento Distribuído de *Software*

Isabella Almeida da Silva

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Informática.

Apresentado por:

_____

Isabella Almeida da Silva

Aprovado por:

_____

Profª Claudia Maria Lima Werner, D.Sc.
(Presidente)

_____

Prof. Marco Aurélio Souza Mangan, M.Sc.
(Co-orientador)

_____

Profª Claudia Lage Rebello da Motta, D.Sc.

_____

Prof. Jano Moreira de Souza, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2005

Agradecimentos

Antes de tudo, agradeço aos meus pais pelo carinho e apoio constantes durante toda a minha vida.

À minha orientadora, Cláudia Werner, que me incentivou ao longo da graduação e foi presença fundamental para a realização deste projeto. Ao meu co-orientador, Marco Mangan, pelas suas revisões detalhadas, que me ensinaram a ser cada vez mais perfeccionista e minuciosa. São dois exemplos de profissionais que, juntos, me proporcionaram uma orientação precisa e esclarecedora, alicerce para a finalização deste projeto.

Agradeço ainda a todas as pessoas que trabalharam comigo no laboratório de engenharia de *software*, pela troca de conhecimento e experiências; a todos os meus amigos e parentes, que estiveram presentes de alguma forma neste projeto e ao Marcelo, que esteve ao meu lado em todos os momentos.

RESUMO

GAW: Um Mecanismo Visual de Percepção de Grupo Aplicado ao Desenvolvimento Distribuído de *Software*

Isabella Almeida da Silva

Orientadora: D.Sc. Claudia Maria Lima Werner

Co-orientador: M.Sc. Marco Aurélio Souza Mangan

Este trabalho apresenta um mecanismo que representa graficamente informações úteis para incentivar a melhoria da percepção de grupo em equipes distribuídas de desenvolvimento de *software*, normalmente prejudicada pelo isolamento. O mecanismo foi implementado de forma a facilitar sua integração a ambientes de desenvolvimento utilizados por estas equipes. São apresentadas duas aplicações implementadas com base no mecanismo e integradas a diferentes tipos de ambientes de desenvolvimento.

ABSTRACT

GAW: A Group Awareness Widget Applied in Distributed Software Development

Isabella Almeida da Silva

Supervisor: D.Sc. Claudia Maria Lima Werner
Co-supervisor: M.Sc. Marco Aurélio Mangan

This work presents a mechanism that graphically represents useful information to improve group awareness in distributed software development teams, usually impaired by isolation. This mechanism was implemented in a way that facilitates its integration with development environments used by these teams. Two applications based in the mechanism were implemented and integrated with different kinds of development environments.

SUMÁRIO

Capítulo I - Introdução **1**
1.1. MOTIVAÇÃO 1
1.2. OBJETIVO 3
1.3. ORGANIZAÇÃO 4
Capítulo II - Equipes distribuídas e percepção **5**
2.1. DESENVOLVIMENTO DE *SOFTWARE*: LOCALIZADO E DISTRIBUÍDO 5
2.2. PERCEPÇÃO 8
2.3. INFORMAÇÕES DE PERCEPÇÃO 11
2.4. MECANISMOS DE APOIO À PERCEPÇÃO 13
2.4.1. Ensino à distância *14*
2.4.2. Modelagem de ritmo de trabalho *15*
2.5. CONSIDERAÇÕES FINAIS 16
Capítulo III - Abordagem proposta: GAW **18**
3.1. PROPOSTA 18
3.1.1 Tipo de informação de percepção *18*
3.1.2. Forma de explorar as informações *19*
3.1.3. Representação das informações *19*
3.1.4. Coleta e atualização de informações *19*
3.1.5. Nível de integração *20*
3.2. ARQUITETURA 20
3.2.1. Fonte *20*
3.2.2. Coletor *21*
3.2.3. Modelo *22*
3.2.4. Visão *22*
3.3. PRINCIPAIS FUNCIONALIDADES 23
3.3.1. Visualização Gráfica de Informações de Percepção *23*
3.3.2. Configuração da Visualização *25*
3.3.3. Filtros *26*
3.4. CONSIDERAÇÕES FINAIS 27
Capítulo IV - Implementação **29**
4.1. GAW 29
4.1.1. Modelo *30*
4.1.1. Visão *34*
4.1.3. Utilização *35*
4.2. *CVS-WATCH* 37
4.2.1. Contextualização *37*
4.2.2. Descrição *38*
4.2.3. Integração com o Eclipse *41*
4.2.4. Utilização *42*
4.2.5. Análise das informações *44*
4.3. *WORKRHYTHM* 46
4.3.1. Contextualização *46*
4.3.2. Descrição *47*
4.3.3. Integração com o Odyssey *50*
4.3.4. Utilização *51*
4.3.5. Análise das informações *51*
4.4. COMPARAÇÃO ENTRE *CVS-WATCH* E *WORKRHYTHM* 52
4.5. CONSIDERAÇÕES FINAIS 54
Capítulo V - Conclusão **56**
5.1. CONSIDERAÇÕES FINAIS 56
5.2. CONTRIBUIÇÕES 57
5.3. LIMITAÇÕES E TRABALHOS FUTUROS 57
Referências Bibliográficas **60**
Anexo A - Instalação do *CVS-Watch* no Eclipse **62**
Apêndice B - Instalação do *WorkRhythm* no *Odyssey* **68**

CAPÍTULO I
INTRODUÇÃO

1.1. Motivação

Com a popularização dos computadores pessoais e principalmente da Internet, surgiu a oportunidade de se conectar pessoas de todo o mundo através da rede. Não só o acesso a informações sobre qualquer assunto foi facilitado, como a comunicação entre pessoas se tornou mais rápida e, principalmente, mais barata.

Foram desenvolvidas diversas ferramentas de *software* por meio das quais grupos com interesses comuns podem trocar mensagens praticamente em tempo real e interagir a partir de suas próprias casas como se fossem vizinhos. O correio eletrônico, as salas de bate-papo, os mensageiros e os programas para teleconferências são exemplos dessas ferramentas. Estas ferramentas são conhecidas como *groupware* (ELLIS *et al.*,1991).

As facilidades trazidas por estas ferramentas permitem que sejam formadas equipes de trabalho com grupos em locações distintas que colaboram através do computador. As organizações têm assim um suporte maior para distribuir suas unidades pelo mundo e expandir seus negócios. O estudo de como os computadores podem auxiliar estas equipes é feito pela área de Trabalho Cooperativo Apoiado por Computador (CSCW – *Computer Supported Cooperative Work*) (ELLIS *et al.*,1991).

Entretanto, muitas destas ferramentas de *groupware* não obtiveram o desempenho esperado (GRUDIN, 1994). A participação dos membros do grupo era fraca e, conseqüentemente, a colaboração era muito pobre. Surgiu, então, a questão sobre se realmente a distância era impeditiva para a colaboração. De acordo com KREJINS e KIRSCHNER (2001), este desempenho insatisfatório se deve ao fato de que esses sistemas partiam do pressuposto de que a interação entre os usuários

ocorreria naturalmente. Porém, a simples existência do sistema não garantiu uma boa interação entre os usuários.

Na realidade, para se obter uma colaboração efetiva, os participantes devem receber tanto ou mais incentivo para interagir quanto aqueles que se encontram face-a-face. Os problemas de comunicação encontrados em grupos convencionais persistem, caso este mesmo grupo esteja distribuído, ou são agravados pela perda de detalhes na comunicação à distância.

Em locais de trabalho convencionais, o contato direto promove a interação social entre os participantes, que é essencial para que estes se conheçam melhor e possam colaborar mais facilmente. Em ambientes virtuais, porém, a percepção sobre as atividades dos demais participantes é prejudicada pela falta de convivência entre eles. Os sistemas de apoio à colaboração, devem, portanto, prover mecanismos para aumentar a percepção do grupo e, desta forma, melhorar sua eficácia. Os chamados mecanismos de percepção de grupo facilitam o acesso a informações que, normalmente, são perdidas pela equipe por causa da distância e oferecem, com isso, um maior incentivo à colaboração.

As equipes de desenvolvimento de *software* que se inserem neste contexto de trabalho distribuído sofrem, também, com os efeitos da distância. O *software* é um artefato construído colaborativamente. A falta de uma percepção homogênea sobre as atividades dos demais participantes dificulta essa colaboração e, portanto, prejudica o desenvolvimento de *software*. As diferentes partes da equipe ficam isoladas, o que traz dificuldades tanto para sincronização das diferentes atividades da equipe quanto para a localização de ajuda necessária para solucionar problemas encontrados durante o processo de desenvolvimento. Conseqüentemente, o tempo e o trabalho dos desenvolvedores são muitas vezes desperdiçados. Como estas equipes podem estar espalhadas em diferentes salas, prédios ou cidades, manter a percepção de grupo, quebrando o isolamento, é um desafio para essas empresas.

Neste trabalho, é apresentado um mecanismo de percepção para apoiar as equipes distribuídas de desenvolvimento de *software*. Em especial, exploramos uma abordagem que utiliza as informações geradas durante o próprio processo de desenvolvimento, para tentar amenizar os efeitos da deficiência de percepção citados anteriormente. As informações consideradas relevantes para o grupo são coletadas sem a necessidade de intervenção dos participantes da equipe e são apresentadas graficamente, com o intuito de tornar a análise destas mais ágil e fácil. Para incentivar o uso dessas informações rotineiramente durante o processo de desenvolvimento, a ferramenta de visualização das informações pode ser facilmente acoplada a ambientes de *software* utilizados por equipes de desenvolvimento.

1.2. Objetivo

Este trabalho destaca as dificuldades enfrentadas pelas equipes de desenvolvimento de *software* distribuídas e tem como objetivo apresentar um mecanismo para apoiar a percepção de grupo nestas equipes, facilitando a colaboração entre seus participantes. Este mecanismo se propõe a coletar informações consideradas relevantes para o aumento da percepção, que se encontram disponíveis nos locais de trabalho, e apresentá-las de uma maneira rápida de serem analisadas. Este mecanismo deve ser integrado a ambientes de desenvolvimento utilizados rotineiramente por equipes de desenvolvimento.

Para demonstrar como essa integração pode ser feita, são apresentadas duas aplicações baseadas no mecanismo proposto, uma integrada ao ambiente de desenvolvimento Eclipse (ECLIPSE FOUNDATION, 2005), e outra integrada ao ambiente de reutilização *Odyssey* (WERNER *et al.*, 2000). Tanto a implementação do mecanismo, quanto de suas aplicações derivadas são descritas neste trabalho.

1.3. Organização

Após esta breve introdução (Capítulo I), são destacados os principais conceitos relacionados com o desenvolvimento distribuído de *software*, a percepção de grupo e mecanismos de apoio à percepção, utilizados neste trabalho (Capítulo II). Em seguida, é apresentada a abordagem proposta para o desenvolvimento de um mecanismo de apoio à percepção para equipes de desenvolvimento distribuídas (Capítulo III). A implementação do mecanismo e de duas aplicações dele derivadas, integradas a diferentes ambientes de desenvolvimento, é detalhada em um capítulo à parte (Capítulo IV). Finalmente, são apresentadas as conclusões deste trabalho, incluindo suas contribuições, limitações e trabalhos futuros (Capítulo V).

CAPÍTULO II
EQUIPES DISTRIBUÍDAS E PERCEPÇÃO

Este capítulo apresenta os principais conceitos relacionados ao desenvolvimento distribuído de *software*, a percepção de grupo e mecanismos de apoio à percepção, utilizados ao longo deste trabalho.

Inicialmente, é feita uma comparação entre o trabalho colaborativo localizado e distribuído, destacando os desafios do último. Em seguida, são introduzidos conceitos relacionados com a percepção de grupo e apresentados os problemas que surgem pela falta desta percepção, além de alguns tipos de informação que podem ajudar a reduzi-los. Além disso, é descrito um mecanismo que utiliza essas informações para apoiar o trabalho das equipes distribuídas e são apresentados exemplos destas ferramentas encontrados na literatura. Por fim, são apresentadas as considerações finais deste capítulo.

2.1. Desenvolvimento de *software*: localizado e distribuído

O desenvolvimento de *software* é uma atividade colaborativa. Os artefatos gerados no processo de desenvolvimento são construídos através de contribuições feitas por diferentes indivíduos. Durante este processo, é necessário que estes indivíduos se comuniquem com freqüência para distribuir tarefas, organizar cronogramas, discutir possíveis soluções e resolver problemas, entre outras atividades.

Em um cenário típico de desenvolvimento de *software* localizado (Figura 1a), os participantes da equipe compartilham o mesmo ambiente físico e trabalham em computadores conectados a uma rede local através de um servidor. Neste servidor ficam armazenados os artefatos gerados pela equipe de desenvolvedores.

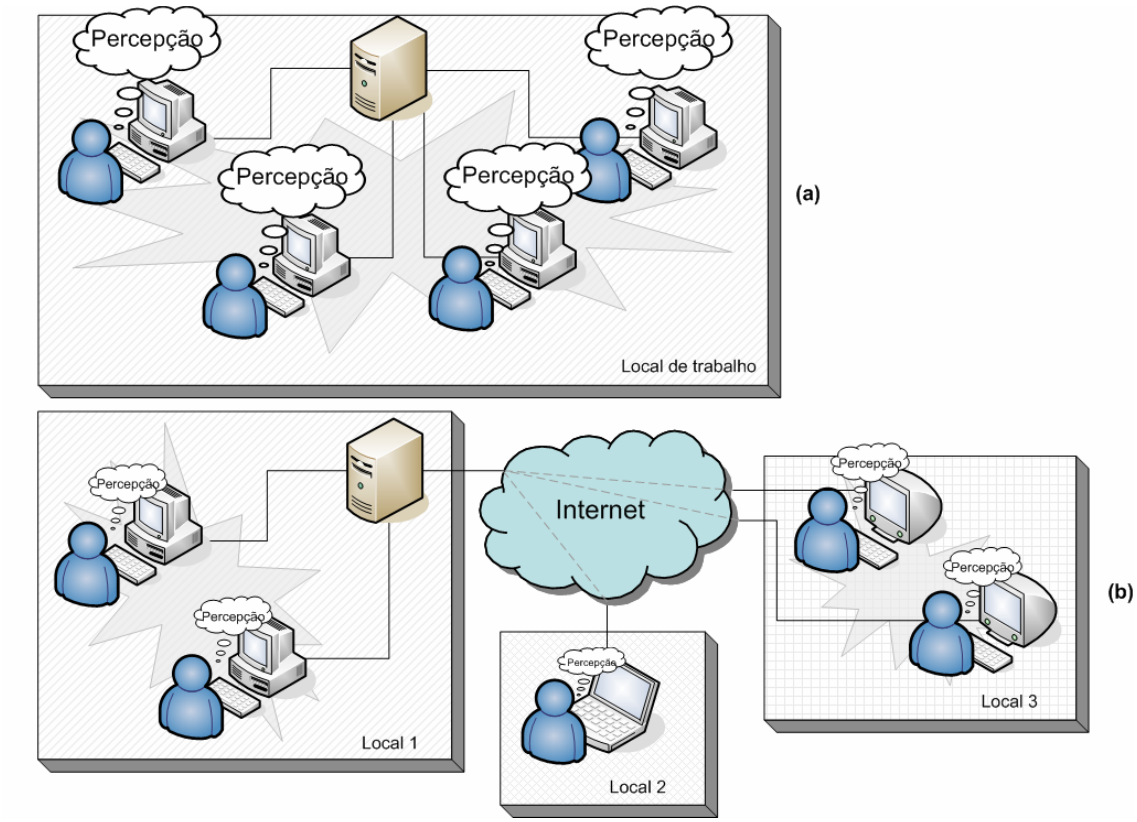

Figura 1 - Percepção e alcance da comunicação direta entre desenvolvedores no (a) desenvolvimento de *software* localizado e no (b) desenvolvimento de *software* distribuído

Quando um determinado desenvolvedor pretende fazer alterações em um artefato compartilhado, ele primeiramente faz uma cópia do artefato para o seu computador. Após fazer as alterações desejadas e se certificar de sua corretude, o desenvolvedor transfere a sua cópia local para o servidor, permitindo que os demais tenham acesso à nova versão do artefato. Esta atividade pode ser demorada e, durante este período, algum outro desenvolvedor, em paralelo, também pode fazer alterações no mesmo artefato, o que poderia gerar conflitos no momento de enviar a cópia deste de volta para o servidor. Entretanto, como os desenvolvedores estão em um mesmo ambiente físico, pode-se perguntar aos demais se alguém fez alguma alteração no artefato e, caso positivo, chamá-lo para discutir a junção das versões conflitantes. Mesmo que o conflito ocorra, é simples localizar e interagir com o colega e tentar resolver o problema. Essa simples política pode resolver problemas de integração, como esse.

Uma outra política muito utilizada para estes casos é o uso de travas (do inglês, *lock*) para os artefatos compartilhados. Enquanto o desenvolvedor está alterando um artefato, este fica travado, ou seja, não pode se alterado por mais ninguém. Somente ao término da alteração que o artefato é liberado novamente para os demais. Esta política, no entanto, não permite a edição concorrente dos artefatos, uma restrição que pode diminuir a produtividade da equipe, tão essencial para as empresas atualmente. Consideraremos, então, como cenário típico de desenvolvimento localizado, a política apresentada anteriormente, que não adota travas.

Neste contexto de equipes localizadas, as interações entre os participantes da equipe são predominantemente síncronas, ou seja, ocorrem quando os desenvolvedores estão trabalhando durante um mesmo período de tempo. Isto facilita não só a resolução de conflitos descrita anteriormente, como também a busca do "especialista" em um determinado artefato. Caso algum desenvolvedor necessite modificar um artefato sobre o qual ele não tem conhecimento, muitas vezes é útil conhecer o colega que criou ou que já trabalhou nesse artefato. O desenvolvedor pode, assim, consultar o colega caso tenha alguma dúvida durante o processo de modificação daquele artefato. Para encontrar um "especialista", basta perguntar aos colegas, que estão próximos, caso já não conheça a pessoas mais indicada, pela própria convivência.

Já no cenário típico de uma equipe distribuída (Figura 1b), os participantes se encontram em diferentes ambientes físicos. Neste contexto, podem existir desenvolvedores que acessam o servidor através da rede local, caso estejam próximos deste e outros que acessam o servidor de um local remoto através da Internet. Um grupo de desenvolvedores pode estar na sede da empresa (Local 1), um desenvolvedor pode estar trabalhando em sua própria casa (Local 2) e os demais em uma filial da empresa em outra cidade (Local 3). Neste caso, as interações dentro da equipe são, muitas vezes, assíncronas, pois os desenvolvedores podem ter horários distintos de trabalho.

Neste último contexto, geralmente, a equipe também compartilha os artefatos guardados no servidor, o que pode gerar conflitos como o descrito anteriormente. Entretanto, como a equipe está espalhada por diferentes ambientes, é mais difícil perguntar para todos os participantes, por exemplo, se alguém alterou um determinado artefato nos últimos dias. Também é complicado determinar o "especialista" em um determinado artefato, para solucionar possíveis dúvidas sobre este. A comunicação da equipe distribuída fica prejudicada.

Esta deficiência não se manifesta apenas nestes casos de resolução de conflitos de sincronização dos artefatos compartilhados. A elaboração de soluções coletivas, o treinamento de desenvolvedores da equipe através do contato com os mais experientes e demais tarefas que, normalmente, são feitas reunindo o conhecimento do grupo como um todo, também se tornam mais difíceis e, portanto, mais raras. Apesar de existirem meios de comunicação disponíveis, estes geralmente não são suficientes para diminuir o isolamento destas equipes e garantir a colaboração efetiva entre seus participantes (HERBSLEB *et al.*, 2001).

Também existem problemas de comunicação em grupos localizados, mas em equipes distribuídas esses problemas são agravados em conseqüência da distância. Alguns estudos (HERBSLEB *et al.*, 2001) (OLSON e OLSON, 2000) comprovam que tarefas de mesma complexidade demoram mais tempo e demandam mais pessoas para serem executadas quando a equipe está distribuída.

2.2. Percepção

A produtividade das equipes distribuídas é prejudicada principalmente pelos problemas de comunicação. Além de ser mais rápido e fácil falar com alguém que está próximo, conversar pessoalmente permite perceber expressões faciais, tons de voz e gestos que proporcionam uma comunicação rica e, conseqüentemente, um melhor entendimento entre as partes. O

compartilhamento de um mesmo ambiente físico também torna possível perceber, mesmo involuntariamente, muitos aspectos pessoais dos membros do grupo. Depois de alguns dias de convívio já é possível descobrir padrões de horários de chegada e saída dos colegas, assim como ter noção das suas atividades atuais e suas principais competências. Isto permite, por exemplo, que se consiga inferir quais são os melhores horários para encontrar um desenvolvedor "especialista" em determinado artefato, ou alguém que já tenha trabalhado numa tarefa similar e possa oferecer algum auxílio.

A perda desta percepção em equipes distribuídas contribui para o atraso destas em relação a equipes convencionais (HERBSLEB *et al.*, 2001). A **percepção** (do inglês, *awareness*) pode ser definida como "o entendimento das atividades dos demais que provê um contexto para a sua própria atividade" e é muito importante para a socialização, sendo um dos principais requisitos para que a colaboração seja efetiva (DOURISH e BELOTTI, 1992). Outros conceitos importantes são derivados da percepção e, muitas vezes, confundidos com a própria, como informação de percepção e mecanismo de apoio à percepção.

Percepção de grupo é o estado de ciência de um indivíduo em relação ao ambiente e aos demais, tendo um caráter subjetivo. I**nformação de percepção** é a informação que pode ser utilizada para melhorar o estado de percepção de um indivíduo. Por exemplo, o conhecimento sobre as atividades dos colegas e dos seus padrões de horário citado acima. **Mecanismo de apoio à percepção** é uma ferramenta que utiliza informações de percepção para tentar apoiar grupos em atividades colaborativas. Estes conceitos são explorados mais adiante neste capítulo.

As equipes distribuídas têm menos acesso às informações de percepção, e, portanto, a percepção de grupo de seus participantes é mais fraca do que a das equipes localizadas (Figura 1). Na figura, a mancha cinza representa o alcance da comunicação direta entre os participantes. Enquanto na equipe localizada, é possível estabelecer uma comunicação direta entre quaisquer participantes, na equipe distribuída esta comunicação é

limitada. Pode-se destacar, entre os problemas que as equipes distribuídas têm que enfrentar por esta deficiência de comunicação, as dificuldades na **coordenação de ações** e na **localização de ajuda**.

A baixa coordenação de ações é conseqüência direta da falta de percepção sobre as atividades dos demais participantes da equipe. É comum a criação de artefatos duplicados ou incompatíveis, como no caso de conflitos de versão, conforme exemplificado na seção anterior. Estes problemas fazem com que sejam desperdiçados tempo e esforço dos desenvolvedores. A resolução do conflito em um artefato é uma tarefa mecânica, que é realizada pelo desenvolvedor que percebe o conflito. Entretanto, a resolução do conflito de forma individual não alerta aos desenvolvedores envolvidos de que ocorreu um conflito e de que forma este foi resolvido. Essa falta de informação contribui para que no futuro, novos conflitos ocorram. Em contraste, a informação de percepção permite que os envolvidos entendam a situação e procurem evitá-la ou se preparar melhor para a sua ocorrência.

A localização de ajuda fica prejudicada pela falta de conhecimento sobre as competências e, mais uma vez, sobre as atividades do restante da equipe. Quando surgem dúvidas durante a modificação de um artefato, o participante de uma equipe distribuída encontra dificuldades em localizar o responsável pela criação ou manutenção do artefato. Também não é fácil saber quais desenvolvedores poderão ter seus trabalhos impactados por alterações neste artefato. Caso o desenvolvedor não consiga encontrar o "especialista" no artefato, nem sanar sua dúvida pesquisando sozinho em um tempo razoável, ele optará pela solução que considerar mais adequada. Desta forma, este desenvolvedor irá inserir uma modificação cujo impacto no restante do *software* ele desconhece. Se esta modificação gerar algum conflito com as contribuições feitas pelos demais desenvolvedores, tempo e trabalho adicionais serão necessários para removê-lo. Além disso, o desenvolvedor ao encontrar um "especialista" pode retomar as discussões, o histórico e a intenção do artefato sendo modificado.

2.3. Informação de percepção

Existem diferentes tipos de informação que podem ser utilizadas para melhorar a percepção de grupo e, deste modo, facilitar a colaboração entre os participantes de uma equipe distribuída. A **informação de presença**, por exemplo, indica quando um determinado usuário está utilizando um sistema. A **informação de localização**, por sua vez, indica onde o usuário se encontra, por exemplo, em casa, no escritório, na universidade etc. Já a **informação de atividade** se refere ao projeto em que o participante está alocado ou em qual tarefa ele está trabalhando no momento. Além do tipo da informação, pode-se variar o intervalo de tempo em que a informação é considerada. Neste caso, as informações coletadas podem ser exploradas de dois modos distintos: instantâneo e histórico.

A informação instantânea é atualizada em tempo real e indica o estado de uma determinada informação monitorada no momento atual. Por exemplo, informação de presença indica quais usuários estão potencialmente disponíveis para interação no momento. Neste modo, a informação de presença tem como finalidade o incentivo à interação síncrona. A informação de presença instantânea é comumente utilizada em *software* mensageiro. No mensageiro ICQ (ICQ, 2005), por exemplo, os contatos do usuário são agrupados de acordo com o estado de presença: os que estão presentes (do inglês, *on-line*) no sistema aparecem ao lado de uma pequena flor verde, enquanto os que não estão aparecem ao lado de uma flor vermelha (Figura 2).

Já a informação histórica é armazenada como um registro durante um período maior de tempo. Normalmente, esta é utilizada para encontrar padrões temporais que permitam fazer inferências sobre valores futuros. A informação de presença histórica pode ser utilizada para melhorar a percepção de determinadas características do ritmo de trabalho e da disponibilidade potencial de cada participante.

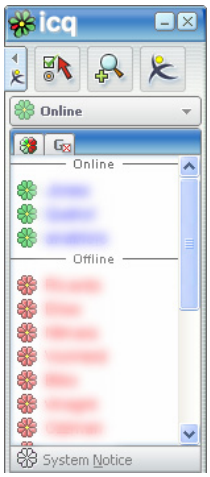

Figura 2 - O mensageiro ICQ explora a informação de presença instantânea dos usuários.

No modo histórico, a utilidade da informação de presença é mais variada. A primeira finalidade óbvia é monitorar a freqüência de uso do sistema por parte dos usuários ("O usuário X não está usando o sistema com a freqüência esperada"). Um outro uso seria apoiar o planejamento de atividades síncronas ("Nos últimos meses, o usuário Y está presente nas quartas e sextas, esses dias seriam mais indicados para tentar entrar em contato"). Seria possível também tentar prever o tempo de resposta de solicitações assíncronas ("Quando o usuário Z sai para o almoço, costuma retornar apenas depois das 14h. Este deve ser o melhor horário para contactá-lo"). Ainda outra utilidade é permitir que um usuário possa ter informações sobre o histórico de utilização do grupo, podendo obter informações sobre o que aconteceu no ambiente colaborativo enquanto esteve desconectado.

O mecanismo proposto neste trabalho permite representar as informações de percepção que dizem respeito às alterações feitas em artefatos gerados pelo processo de desenvolvimento de *software*, como documentos, modelos e código fonte. Quando um determinado artefato é alterado, podem ser guardadas informações tais como: o autor, a data e o “tamanho” (número de elementos modificados) da alteração. Este tipo de informação pode ser útil para responder uma série de perguntas que surgem rotineiramente quando é necessário modificar um artefato que foi desenvolvido colaborativamente (SILVA *et al.*, 2004):

- Quais participantes da equipe já trabalharam na construção deste artefato?
- Quem é o responsável pela última versão do artefato?
- Qual seria o participante mais indicado para esclarecer dúvidas sobre o artefato?

Quando a equipe se encontra distribuída, essas questões nem sempre são facilmente respondidas. O mecanismo descrito neste trabalho pretende auxiliar os participantes de equipes distribuídas a responder essas e outras questões que surgem durante o processo de desenvolvimento.

2.4. Mecanismos de apoio à percepção

Como visto anteriormente, mecanismos de apoio à percepção são ferramentas de *software* que têm como objetivo incentivar a colaboração em um grupo através de informações de percepção. Estas ferramentas são particularmente importantes para equipes distribuídas, que são prejudicadas pelo isolamento e conseqüentes deficiências na comunicação.

Voltando ao exemplo da equipe de desenvolvimento distribuída do início deste capítulo, onde os participantes têm a percepção de grupo prejudicada, pois estão divididos em três ambientes distintos (Figura 1b). Um mecanismo de apoio à percepção poderia repor algumas informações que os participantes perdem por estarem distantes. O desenvolvedor continuaria trabalhando normalmente em seu computador, acessando o servidor de artefatos através da *Internet*, só que durante suas atividades receberia informações para melhorar sua percepção sobre o trabalho dos demais (Figura 3).

Muitas das ferramentas de *groupware,* atualmente, apresentam algum tipo de mecanismo de apoio à percepção de grupo. Existem diferentes maneiras de oferecer este apoio, dependendo do tipo de informação de percepção coletada (vistos na seção anterior) e de como esta é apresentada pelo usuário. A

forma de apresentação pode ser textual, como uma descrição das atividades dos usuários no momento, pode também ser sonora, como uma campainha que avisa quando um determinado usuário está disponível no sistema. Entretanto, neste trabalho são explorados somente mecanismos que utilizam representações gráficas para as informações de percepção, como veremos a seguir.

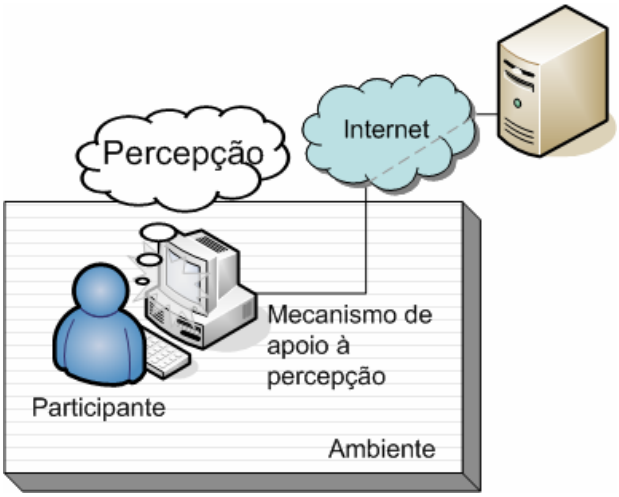

Figura 3 - Participante de uma equipe distribuída tem sua percepção incentivada por um mecanismo de apoio à percepção

Os mecanismos de percepção que utilizam esta representação são denominados "componentes visuais de percepção de grupo" e definem uma família de mecanismos que apóiam a percepção em atividades de colaboração (KREJINS e KIRSCHNER, 2001). Estes componentes coletam informações relevantes ao grupo e as representam graficamente para todos os participantes. Existem implementações desses componentes para diferentes contextos. Serão mostrados a seguir dois exemplos de componentes visuais de percepção encontrados na literatura, os quais se aplicam em contextos distintos.

2.4.1. Aplicação no ensino à distância

O componente visual de percepção proposto por KREJINS e KIRSCHNER (2001) explora informações de presença em um

sistema de *software* de apoio ao ensino à distância. Os alunos conseguem perceber quando os colegas de turma estão utilizando o sistema e, através dele, se comunicar. Na interface gráfica do componente (Figura 4), são listados todos os alunos de uma determinada turma e, ao lado do nome de cada um deles, são mostradas barras que representam suas sessões no sistema. Estas barras estão ordenadas do tempo mais atual para o passado, ou seja, as barras mais próximas dos nomes dos alunos são as mais recentes. A escala de tempo utilizada é logarítmica, fazendo com que as barras diminuam de largura com o passar do tempo e, desta forma, dando maior destaque às sessões mais atuais.

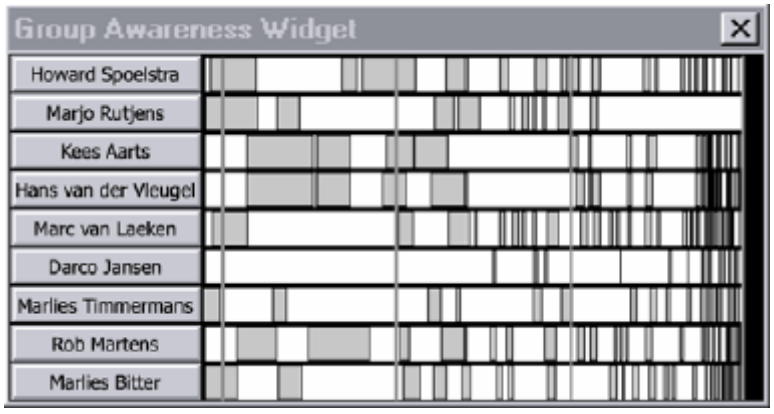

Figura 4 - Mecanismo de percepção que explora informações de presença para facilitar a comunicação em um sistema de ensino à distância (KREJINS e KIRSCHNER, 2001)

Analisando as sessões dos colegas, o aluno pode perceber os padrões de horário que estes utilizam o sistema e também saber quais deles estão utilizando o sistema no momento. Além disso, o aluno pode se comunicar através de mensagens de texto com os colegas de maneira síncrona ou assíncrona. Desta forma, os alunos conseguem tirar dúvidas ou discutir tópicos da matéria entre si, promovendo o aprendizado através da colaboração.

2.4.2. Aplicação no acompanhamento de ritmo de trabalho

O componente descrito por BEGOLE *et al*. (2002) tem como objetivo encontrar padrões na rotina de trabalho dos participantes

de uma mesma equipe. Informações de presença são coletadas em um *software* mensageiro e os padrões de horário são analisados. Nesta análise, o próprio componente tenta inferir, através das transições entre períodos de atividade e inatividade, os horários de eventos dos participantes do grupo, tais como a chegada no escritório, o almoço, as reuniões periódicas, entre outros (Figura 5).

Conhecendo os horários destes eventos, os colegas de trabalho podem ter uma boa noção de qual o melhor momento para entrar em contato ou para agendar reuniões, por exemplo. Esse conhecimento facilita a comunicação em equipes distribuídas ou para novos participantes de equipes localizadas que ainda desconhecem os padrões de horário dos colegas.

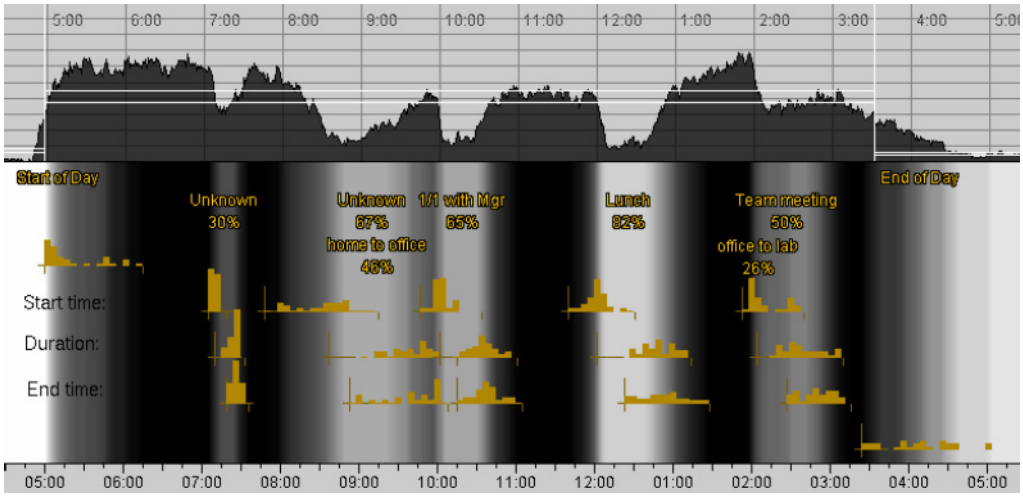

Figura 5 - Análise feita pelo mecanismo que explora informações de percepção de presença históricas para construir um modelo de ritmo de trabalho (BEGOLE *et al.*, 2002)

2.5. Considerações Finais

Uma decisão importante para o desenvolvimento de mecanismos de percepção é a definição de como estes devem coletar e apresentar as informações relevantes para os membros do grupo. De acordo com DOURISH e BELOTTI (1992), o ideal é que o mecanismo seja integrado ao ambiente de trabalho do usuário e que ele possa tanto passar suas informações quanto receber as dos colegas passivamente. Se o usuário precisar fazer

algum esforço para disponibilizar suas informações, provavelmente este mecanismo será subutilizado.

Outro ponto importante a considerar no desenvolvimento de mecanismos de apoio à percepção é a quantidade de informações coletadas e representadas. Com uma quantidade muito grande de informações, o usuário pode acabar tendo dificuldades de encontrar o que é relevante para ele no momento. Já quantidades pequenas podem não ser suficientes para melhorar a percepção do grupo. A possibilidade de filtrar ou classificar as informações de percepção de acordo com sua relevância pode ser útil, neste caso.

A privacidade dos usuários, também, é uma questão importante para estes mecanismos, principalmente para os que coletam informações sem necessidade de intervenção do usuário. Todos os participantes da equipe devem aprovar o uso das suas informações e ter a garantia de que estas somente serão utilizadas para os fins de melhoria da percepção de grupo.

No próximo capítulo, é mostrado como estas questões levantadas são consideradas na proposta de um mecanismo de percepção para equipes de desenvolvimento de *software* distribuídas.

CAPÍTULO III
ABORDAGEM PROPOSTA: GAW

Neste capítulo, apresentamos uma proposta de mecanismo de percepção para equipes de desenvolvimento de *software*: o ***G**roup **A**wareness **W**idget* (componente visual de percepção de grupo). Este mecanismo se propõe a apoiar o trabalho colaborativo distribuído através da ampliação do estado de percepção dos participantes da equipe.

Serão apresentadas, inicialmente, as principais características previstas para o mecanismo. Em seguida, é mostrada uma arquitetura de alto nível que satisfaz as características propostas e que foi utilizada como base para o desenvolvimento do mecanismo, detalhando cada um de seus módulos. Além disso, são descritas as principais funcionalidades esperadas para o mecanismo. Finalmente, são apresentadas as considerações finais do capítulo.

3.1. Proposta

Baseado nos diferentes tipos de mecanismos e informações de percepção apresentados no capítulo anterior, este trabalho apresenta uma abordagem de mecanismo de apoio à percepção para equipes de desenvolvimento de *software* distribuídas. Desta forma, as principais características previstas para o mecanismo, são:

3.1.1 Tipo de informação de percepção

Devem ser utilizadas informações relativas à evolução dos artefatos compartilhados pela equipe durante o processo de desenvolvimento. Entre essas informações se encontram, por exemplo, o autor e a data de cada modificação feita nos artefatos.

3.1.2. Forma de explorar as informações

O participante de uma equipe distribuída precisa de informações de percepção atuais, que possam ser exploradas de forma instantânea, para que ele possa estar ciente das atuais atividades dos demais. Entretanto, também é útil ter acesso ao histórico dessas informações para que análises mais profundas possam ser feitas. A visão instantânea permite, por exemplo, saber quais artefatos foram modificados recentemente. Já a visão histórica permite encontrar os desenvolvedores que mais contribuíram para a construção de um determinado artefato, permitindo encontrar os "especialistas" neste.

3.1.3. Representação das informações

O mecanismo deverá utilizar uma representação gráfica para as informações de percepção. A escolha desta forma de representação foi motivada pela capacidade humana de analisar cores e formas mais rapidamente do que textos, principalmente quando há uma grande quantidade de informações. Como, normalmente, existe um grande número de informações de percepção disponível, que só tende a crescer, este tipo de representação parece ser o mais apropriado.

3.1.4. Coleta e atualização de informações

As informações de percepção devem ser encontradas pelo mecanismo automaticamente, sem a necessidade de que os usuários as disponibilizem explicitamente. As informações sobre os artefatos compartilhados podem ser buscadas, por exemplo, diretamente no repositório de artefatos do servidor utilizado pela equipe.

A atualização das informações, por sua vez, pode ser automatizada ou não, dependendo do volume de novas informações que são geradas. Caso este volume seja muito grande, seria interessante que a atualização fosse automática,

evitando que o usuário precisasse solicitar a atualização das informações constantemente. Já no caso de poucas informações novas, a atualização automática seria desnecessária. Neste caso, se o acesso à Fonte de informações for custoso, a atualização automática poderia comprometer o desempenho do mecanismo.

A maneira de coletar e atualizar as informações de percepção depende de cada aplicação. Antes de determinar a melhor opção para uma determinada aplicação, deve-se analisar variáveis como o volume de novas informações e a velocidade de acesso à Fonte.

3.1.5. Integração com o ambiente ou ferramenta em uso

O mecanismo deve ser desenvolvido de forma a facilitar sua integração com ambientes de desenvolvimento utilizados pela equipe. Como dito anteriormente, este tipo de integração é muito importante para que os usuários utilizem o mecanismo.

3.2. Arquitetura

Nesta seção, apresentamos uma arquitetura de alto nível para o mecanismo de percepção GAW, projetada para refletir as características apresentadas na seção anterior. O mecanismo é formado por quatro módulos principais (Figura 6): Fonte, Coletor, Modelo e Visão. Detalharemos cada um destes módulos em seguida.

3.2.1. Fonte

Este módulo representa o sistema onde estão armazenadas as informações de percepção que são utilizadas pelo mecanismo. Como Fonte, podemos ter, por exemplo, bancos de dados ou arquivos texto, como *logs*. A Fonte é um módulo externo e independente, localizado normalmente em um servidor utilizado pela equipe distribuída para compartilhar artefatos. As

informações armazenadas podem estar pouco estruturadas ou com uma estrutura diferente da utilizada pelo mecanismo, necessitando de tratamento antes que possam ser utilizadas.

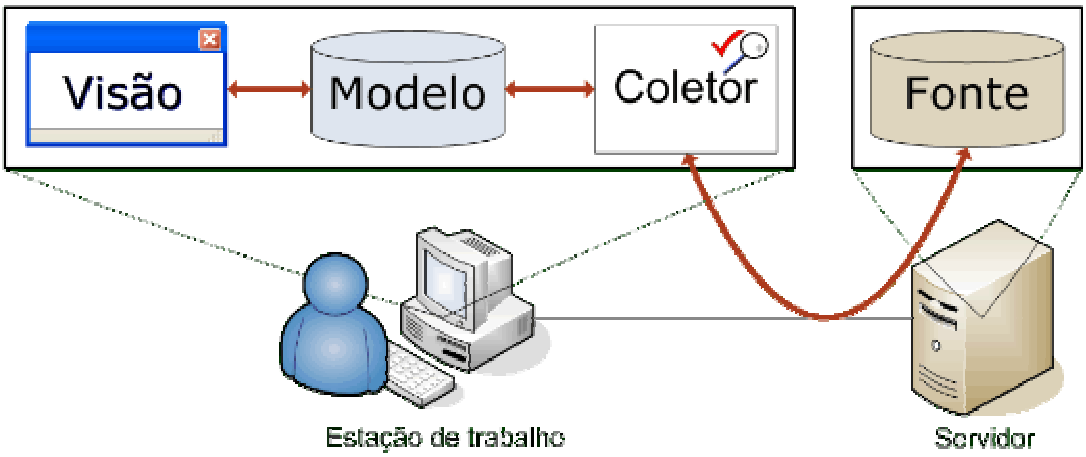

Figura 6 - Arquitetura geral do mecanismo GAW

3.2.2. Coletor

O módulo Coletor é responsável por buscar as informações armazenadas na Fonte, fazer o tratamento necessário e entregá-las ao Modelo. O papel de Coletor pode ser desempenhado pela ferramenta a qual o mecanismo está integrado, por um módulo do próprio mecanismo, ou pela combinação de ambos. Neste último caso, a ferramenta pode ficar responsável pela busca das informações na Fonte, enquanto o módulo, propriamente dito, faz o tratamento destas para o armazenamento no Modelo. O tipo de tratamento necessário para as informações coletadas irá depender do tipo de informações encontradas na Fonte.

O Coletor fica responsável, também, pela atualização periódica das informações de percepção, ou simplesmente por esperar que o usuário solicite a atualização das informações explicitamente. Para fazer a atualização automática, este módulo pode monitorar a Fonte, consultando-a em intervalos regulares para buscar novas informações quando estas estiverem disponíveis.

Este módulo, normalmente, se encontra na estação de trabalho dos usuários e deve se comunicar com o servidor através de uma rede. No entanto, caso seja necessário, a implementação de um Coletor remoto também é possível. A implementação deste módulo deverá ser feita por cada aplicação, pois as características deste dependem das necessidades específicas de cada uma delas.

3.2.3. Modelo

O Modelo tem o papel de armazenar as informações de percepção na estação de trabalho e mantê-las em memória local para um melhor desempenho do mecanismo. As informações devem ser organizadas de forma a facilitar sua visualização. Além disso, o Modelo deve ter a capacidade de agrupar e filtrar as informações tanto de forma estática quanto dinâmica, para que o usuário possa restringir a quantidade de informações visualizadas. As configurações de usuário também são guardadas no Modelo.

Este módulo abrange as funcionalidades básicas para o armazenamento e organização de informações para visualização, que podem ser utilizadas por diferentes aplicações. Entretanto, este pode ser modificado para se adaptar às necessidades específicas de cada aplicação, se necessário.

3.2.4. Visão

A Visão é a interface do mecanismo com o usuário e, portanto, é responsável pela representação gráfica das informações de percepção. Em tempo de execução, a Visão deve ser atualizada automaticamente quando houver mudanças no Modelo como, por exemplo, a chegada de novas informações ou a aplicação de um filtro.

Assim como o Modelo, a Visão abrange as principais funcionalidades necessárias para o desenvolvimento de uma

aplicação baseada no mecanismo GAW e não deve ser necessário alterá-la para cada aplicação.

3.3. Principais Funcionalidades

Conforme dito anteriormente, o mecanismo proposto tem a função de coletar e apresentar informações de percepção relevantes para equipes distribuídas de desenvolvimento de *software*. O GAW deve conter as funcionalidades básicas comuns a aplicações que se encaixam à arquitetura proposta na seção anterior. Porém, antes de ser utilizado para alguma aplicação específica, este deve ser adaptado.

Nesta seção, detalharemos as funcionalidades básicas que o mecanismo deve oferecer. Os demais passos necessários para adaptá-lo e integrá-lo a outras ferramentas serão discutidos somente no próximo capítulo.

3.3.1. Visualização gráfica de informações de percepção

Representada pelo módulo Visão da arquitetura proposta na seção anterior, a interface gráfica do mecanismo GAW (Figura 7), deve permitir que o participante da equipe obtenha informações sobre as alterações feitas nos artefatos compartilhados durante o processo de desenvolvimento.

A interface gráfica foi definida baseando-se no mecanismo para ensino à distância proposto por KREIJNS e KIRSCHNER (2001), apresentado na Seção 2.4.1. Nesta interface, as informações de percepção são representadas por barras coloridas agrupadas por usuário. As barras são apresentadas em ordem cronológica inversa, ou seja, as informações mais recentes aparecem mais à esquerda, perto do botão com o nome do usuário. A largura da barra pode variar, representando a duração daquela informação de percepção, por exemplo, no caso da informação de presença, esta representa a duração da sessão do participante em um determinado sistema. Algumas informações

não têm duração relevante e, neste caso, todas as barras apresentam largura constante. As cores das barras são usadas para identificar cada participante da equipe. A figura a seguir é uma reprodução em tons de cinza de cinco cores diferentes.

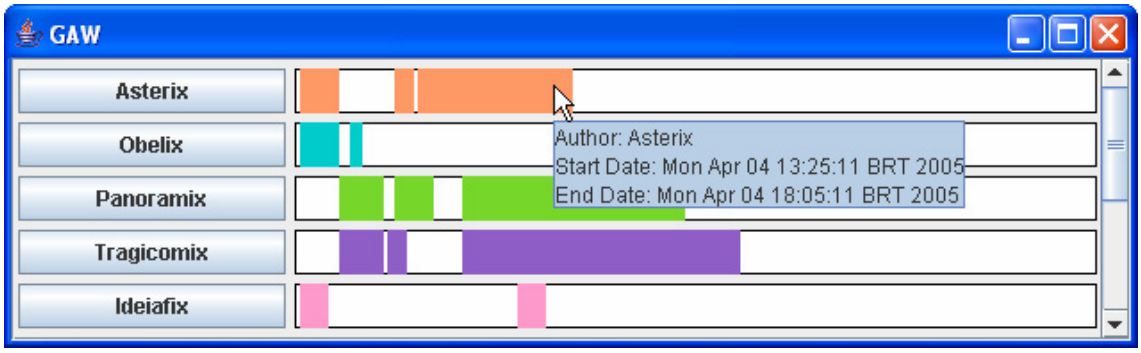

Figura 7 - Protótipo de interface gráfica do mecanismo GAW, utilizando informações de presença fictícias.

Foram definidas ainda formas de representação adicionais, não presentes no mecanismo de KREIJNS e KIRSCHNER (2001), para os atributos das informações de percepção. No GAW, além da largura, a altura da barra também pode variar de acordo com algum critério, como a importância da informação para o grupo (não mostrada na figura). Essa opção foi estabelecida para que as informações possam ser classificadas de acordo com a sua relevância. A importância de uma informação pode ser calculada de diferentes maneiras e, por isso, deve ser implementada para cada aplicação específica. No caso das informações sobre alterações em artefatos compartilhados, esta medida de importância pode ser calculada, por exemplo, de acordo com a quantidade de artefatos impactados pela mudança.

Para aumentar o grau de detalhamento das informações mostradas, é possível mostrar outros atributos das informações como, por exemplo, sua data e hora. Essas informações podem ser visualizadas através de uma dica (*tooltip*) que aparece ao se posicionar o ponteiro do *mouse* sobre a barra de informação na interface gráfica do mecanismo. Isto permite que sejam apresentados quaisquer atributos das informações de percepção adicionados para uma determinada aplicação, sem a necessidade de modificar a interface gráfica.

Para facilitar o entendimento sobre a visualização de informações de percepção no mecanismo GAW, analisaremos o exemplo da Figura 7, onde são mostradas informações fictícias de presença de um sistema utilizado pela suposta equipe distribuída. À esquerda, na vertical, se encontra a lista com todos os participantes desta equipe. Ao lado, na horizontal, estão as barras que representam, neste caso, os intervalos de tempo em que os respectivos participantes estavam presentes no sistema. A dica que aparece ao manter o ponteiro em cima de uma barra mostra a data do início e do fim da sessão do usuário.

Ao analisar a disposição das barras dos usuários, é possível tentar extrair algumas informações sobre os horários de trabalho do grupo mostrado na figura. Por exemplo, é fácil perceber que os usuários *Panoramix* e o *Tragicomix* têm estado presentes no sistema em horários parecidos. Neste caso, estes seriam bons candidatos para dividir tarefas que precisam de muita interação síncrona. Já o usuário *Obelix*, provavelmente, não seria o mais apropriado para dividir estas atividades com os dois, já que o seu horário de trabalho quase não tem interseção com o deles. Os horários de entrada e saída do sistema, também, podem ser úteis quando é necessário inferir o melhor momento para entrar em contato com determinados usuários. Pela figura, é possível ver que o usuário *Asterix*, por exemplo, só trabalhou na parte da tarde, portanto talvez não fosse possível encontrá-lo pela manhã em outros dias. Obviamente, apenas com a observação de curtos períodos de tempo não se pode chegar a conclusões precisas. Porém, ao longo do tempo, alguns padrões passam a ser facilmente percebidos e a probabilidade de se chegar a inferências corretas aumenta.

3.3.2. Configuração da visualização

O mecanismo deve oferecer algumas configurações que podem ser alteradas em tempo de execução para uma maior adequação às necessidades do usuário. Estas configurações alteram a maneira como o usuário visualiza as informações de

percepção na interface gráfica do mecanismo, ou seja, interferem na Visão. O Modelo, no entanto, é quem deve ser responsável pelo armazenamento dessas configurações.

Uma dessas configurações é a cor das barras de informação dos participantes da equipe, que pode ser alterada de acordo com a preferência do usuário. Outra configuração é a da escala de tempo, utilizada para determinar a posição das barras de informação na linha do tempo e limitar a quantidade de informações mostradas. Se for escolhida a escala por hora, por exemplo, só serão mostradas as informações de percepção datadas de até uma hora atrás.

O intervalo de atualização da Visão também pode ser configurado, determinando o intervalo de tempo esperado para que a interface gráfica do mecanismo seja redesenhada na tela. Este intervalo foi definido para evitar problemas de desempenho caso a quantidade de informações visualizadas seja muito grande. Durante este intervalo, mesmo que novas informações de percepção estejam disponíveis no Modelo, estas não serão mostradas ao usuário. É importante não confundi-lo com o intervalo de atualização do Modelo. Este último representa o intervalo com que o Coletor verifica se existem novas informações de percepção na Fonte para atualizar o Modelo. Este intervalo é definido na implementação do Coletor, mas pode ser necessário alterá-lo em tempo de execução de acordo com as condições de execução e vontade do usuário.

3.3.3. Filtros

A quantidade de informações de percepção aumenta rapidamente no decorrer do tempo de uso do mecanismo. A sobrecarga de informações dificulta a recuperação de pequenas partes das informações que são relevantes para o usuário, sem a definição de filtros (MOTTA e BORGES, 2000). No mecanismo GAW, o Modelo é responsável pela filtragem dessas informações (Figura 9) para facilitar sua visualização. Quando um filtro é aplicado ao mecanismo, o Modelo continua armazenando todas as

informações de percepção, mas passa a disponibilizar para a Visão somente as selecionadas pelo filtro. Isto permite que, ao desativar os filtros, todas as informações de percepção sejam mostradas novamente ao usuário, sem a necessidade de consultar novamente a Fonte.

Existem duas formas de se aplicar filtros ao mecanismo: dinâmica e estática. O filtro aplicado dinamicamente pode ser utilizado para permitir que o usuário reduza as informações mostradas na Visão do mecanismo, facilitando a análise dos dados. Um filtro dinâmico por intervalo de datas, por exemplo, permite a visualização das informações de um período de tempo restrito, escolhido pelo usuário.

Os filtros estáticos são aqueles configurados pelo desenvolvedor do mecanismo e não podem ser modificados em tempo de execução. Esses filtros servem para que o Modelo consiga agrupar e organizar as informações que serão entregues à Visão, de acordo com algum critério pré-determinado. Filtros estáticos podem ser úteis, também, para restringir o acesso a determinadas informações de percepção, protegendo a segurança de dados ou a privacidade de determinados usuários.

3.4. Considerações Finais

O GAW serve de base para a implementação de mecanismos de apoio à percepção, para equipes de desenvolvimento de *software* distribuídas. Esta base é formada pelos módulos Modelo e Visão da arquitetura proposta. Estes módulos foram projetados para poderem ser utilizados sem, ou com pouca necessidade de alterações por aplicações derivadas do GAW. Já os módulos Coletor e Fonte, por serem muito dependentes das necessidades de cada aplicação, devem ser desenvolvidos ou, caso já existam, escolhidos especificamente para cada uma destas.

No capítulo seguinte, será detalhada a implementação do mecanismo GAW, assim como de duas aplicações dele derivadas,

mostrando como o mecanismo pode ser adaptado a diferentes contextos.

CAPÍTULO IV
IMPLEMENTAÇÃO

Neste capítulo, são apresentados os principais aspectos da implementação do mecanismo GAW, assim como de duas aplicações dele derivadas: a aplicação *CVS-Watch*, integrada ao ambiente de desenvolvimento Eclipse e a aplicação *WorkRhythm*, integrada ao ambiente *Odyssey*. Após o detalhamento das implementações destas, é feita uma comparação entre elas, mostrando como o GAW pode ser utilizado por mecanismos aplicados em diferentes contextos. Em seguida, são apresentadas as considerações finais deste capítulo.

4.1. *Group Awareness Widget* (GAW)

O mecanismo GAW foi implementado na plataforma Java (SUN, 2005a), uma plataforma independente de sistema operacional que é usada também para a implementação dos ambientes de desenvolvimento integrados com as aplicações. A implementação do mecanismo não inclui os módulos Fonte e Coletor, como visto anteriormente. O primeiro é um módulo externo ao mecanismo, que representa o sistema de onde são coletadas informações de percepção. Já o segundo, é muito dependente da aplicação em questão, inclusive da Fonte escolhida, e necessita ser implementado a cada nova aplicação criada a partir do GAW. Nesta seção, são apresentados os principais aspectos da implementação dos demais módulos descritos no capítulo anterior: o Modelo e a Visão e, em seguida, uma breve descrição de como utilizar as principais funcionalidades do mecanismo.

O GAW é o módulo básico usado para o desenvolvimento das aplicações *CVS-Watch* e *WorkRhythm*. A implementação destes compreendem, principalmente, a adaptação do GAW na interface de usuário do sistema e o acesso à Fonte apropriada por meio de um Coletor específico para o ambiente integrado.

4.1.1. Modelo

O Modelo, como visto no Capítulo III, é o módulo do mecanismo responsável por armazenar as informações de percepção em memória. Além disso, o Modelo armazena as configurações globais de usuário, permite que sejam aplicados filtros estáticos e dinâmicos e informa a Visão de mudanças em seus atributos. As classes que implementam essas funcionalidades são mostradas de forma simplificada na Figura 8.

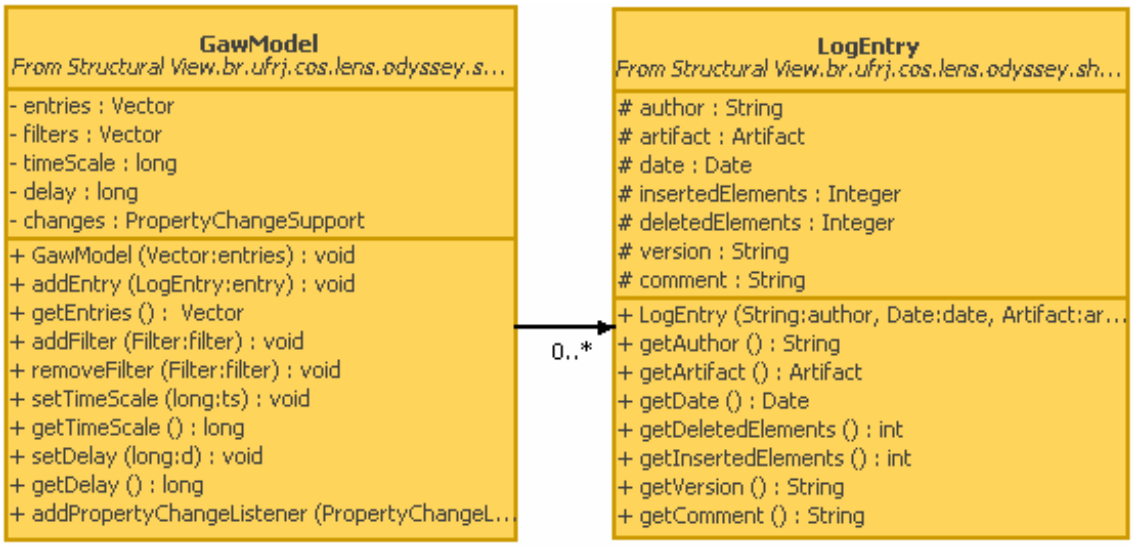

Figura 8 - Diagrama de classes simplificado que representa o Modelo na arquitetura do mecanismo GAW

A classe *LogEntry* agrega os atributos básicos para caracterizar uma alteração feita em um artefato compartilhado pela equipe, ou seja, ela representa uma informação de percepção. Seus principais atributos são:

- ***author***: Nome do participante da equipe responsável pela alteração em questão.
- ***artifact***: Nome do artefato alterado. Dependendo da aplicação, pode ser útil armazenar o caminho até o diretório onde se encontra o artefato, facilitando sua localização.

- ***date***: Data e hora em que a alteração foi registrada pela Fonte. Pode representar a hora exata em que a alteração ocorreu ou a hora em que a alteração foi enviada ao repositório de artefatos compartilhados, dependendo da aplicação.

- ***insertedElements*** / ***deletedElements***: Número de elementos do artefato que foram criados / apagados. Este atributo depende da granularidade considerada pela aplicação específica. Este pode representar, por exemplo, tanto o número de linhas de código quanto o número de arquivos criados / apagados.

- ***version***: Versão do artefato. Como, muitas vezes, as versões são denominadas por um conjunto de letras e números, esta é guardada por uma cadeia de caracteres e não simplesmente um número.

- ***comment***: Comentário escrito pelo autor da alteração. Normalmente, utilizado para facilitar o entendimento sobre o que foi alterado para esta versão do artefato.

Todos estes atributos podem ser inicializados no construtor de *LogEntry* e possuem um método de acesso de leitura (*get<atributo>())* para consultar seu valor. Caso alguma aplicação necessite adicionar novos atributos sobre a informação de percepção, o desenvolvedor deve alterar a classe *LogEntry*.

A classe *GawModel* é responsável pelas funcionalidades do Modelo:

- **Armazenamento das informações de percepção:** o atributo *entries* armazena uma coleção de instâncias de *LogEntry*, ou seja, armazena as informações de percepção. Os métodos *getEntries()* e *addEntry()* são utilizados para consultar e atualizar esta coleção.

- **Armazenamento das configurações globais do usuário**: a escala de tempo e o intervalo de atualização são armazenados nos atributos *timeScale* e *delay,*

respectivamente. Existem métodos do tipo *get<atributo>()* e *set<atributo>()* para consultar e atualizar o valor destes atributos. A cor das barras de informação mostradas na Visão não são armazenadas no Modelo, por serem específicas para cada participante da equipe e não configurações globais do mecanismo.

- **Atualização da Visão**: o *GawModel* adota um sistema de notificação a respeito de mudanças com base na classe *PropertyChangeListener* do modelo de componentes *JavaBeans* (SUN, 2005b). É desta forma que a Visão é informada de alguma mudança no Modelo. Quando alguma configuração é alterada pelo usuário ou alguma nova informação de percepção é adicionada pelo Coletor, a Visão é notificada e pode ser atualizada de acordo.

- **Filtros**: a coleção de filtros é armazenada no atributo *filters*. Esta coleção engloba tanto os filtros estáticos quanto os dinâmicos, descritos no capítulo anterior. Assim como os outros atributos da classe *GawModel*, este também possui métodos para consulta e atualização. A implementação dos filtros será mais detalhada a seguir.

Um diagrama simplificado da hierarquia de classes dos filtros é mostrado na Figura 9. A implementação destas classes foi feita de maneira a permitir a criação de novos filtros, de acordo com as necessidades da aplicação. Para isso, todos os filtros foram implementados como filhos da classe abstrata *Filter*. Esta classe apresenta o método *apply()* que recebe uma coleção de informações de percepção e retorna outra coleção, já filtrada. A maneira com que uma informação será selecionada para integrar a coleção de retorno é determinada pelo método *select()*, que é abstrato na classe *Filter*, mas que deve ser implementado por todos os filtros filhos. Cada filtro, então, determina o seu próprio critério para seleção de uma informação de percepção. Deste modo, pode-se criar filtros com os mais diversos critérios, dando grande flexibilidade para as aplicações baseadas no mecanismo GAW.

Os filtros apresentados na Figura 9 são simples e restringem as informações de acordo com apenas um atributo da informação de percepção por vez. Entretanto, quando aplicados simultaneamente, estes funcionam como um filtro composto e restringem ainda mais as informações visualizadas. Filtros complexos, então, podem ser criados a partir de filtros simples como esses.

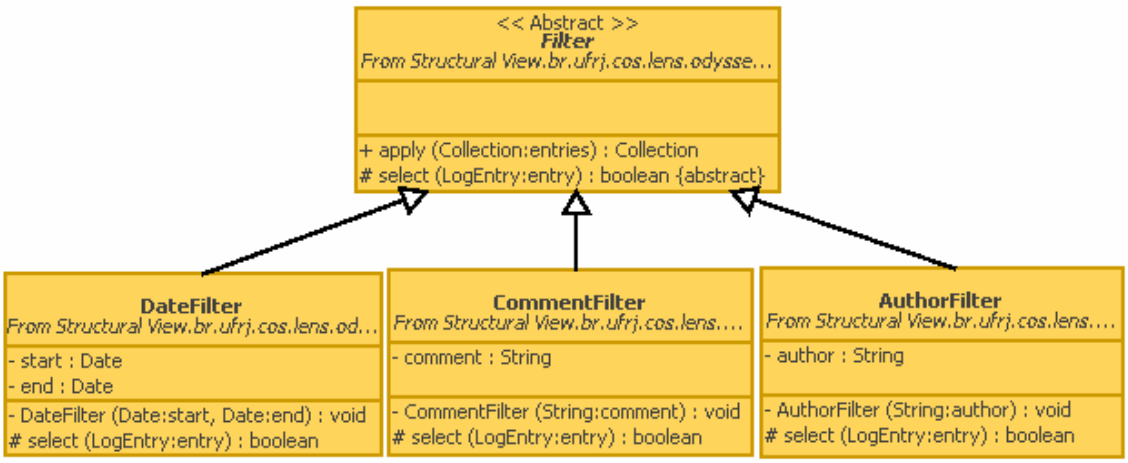

Figura 9 - Diagrama de classes simplificado dos filtros implementados para o mecanismo GAW e aplicações derivadas.

Na implementação do próprio mecanismo GAW, foi utilizado um filtro, implementado pela classe *AuthorFilter*. O filtro é aplicado estaticamente para fazer a seleção das informações de percepção de um dado autor. Na Visão do mecanismo, cada barra está associada ao filtro do autor respectivo. Desta forma, ao desenhar o histórico de um autor, não é necessário se preocupar com as informações de outros autores.

Os outros dois filtros mostrados na Figura 9 foram desenvolvidos para aplicações baseadas no mecanismo GAW, não tendo sido utilizados na implementação do mecanismo propriamente dito.

4.1.1. Visão

A Visão do mecanismo foi desenvolvida utilizando a biblioteca gráfica Swing/AWT (SUN, 2005c), muito utilizada em aplicativos Java. Inicialmente, uma versão alternativa do mecanismo foi desenvolvida utilizando uma outra biblioteca gráfica, também muito utilizada, a SWT (ECLIPSE FOUNDATION, 2005). A manutenção de duas versões do mecanismo era trabalhosa, mas permitia que este conseguisse ser integrado a ambientes que utilizassem qualquer uma dessas duas bibliotecas. Versões recentes da biblioteca SWT, entretanto, permitem que componentes gráficos da biblioteca Swing/AWT sejam integrados a através do seguinte trecho de código:

```
Composite container = new Composite(parent, SWT.EMBEDDED);
Frame frame = SWT_AWT.new_Frame(container);
```

Na primeira linha, é criado um *Composite* da biblioteca SWT, que nada mais é que um painel ao qual os componentes visuais SWT são normalmente adicionados. Na segunda linha, é criado um *Frame*, que faz este mesmo papel de painel na biblioteca Swing/AWT. No entanto, a criação do *Frame* é feita a partir do método *new_Frame()* da classe SWT_AWT e não pelo construtor padrão deste. Este método permite que o *Frame* Swing/AWT seja adicionado ao *Container* SWT, recebido como parâmetro. Desta forma, o *Frame* e todos os componentes visuais Swing/AWT que forem adicionados a ele aparecem dentro do *Composite* SWT, na interface gráfica.

A Figura 10 ilustra a integração de um *Frame* Swing/AWT e seus componentes visuais Swing/AWT a diferentes ambientes. No caso de ambientes SWT (Figura 10a), a integração é feita através de um *Container* SWT, como descrito no trecho de código acima. Já no caso de ambientes Swing/AWT (Figura 10b), a integração é direta, sem a necessidade de código ou componentes adicionais.

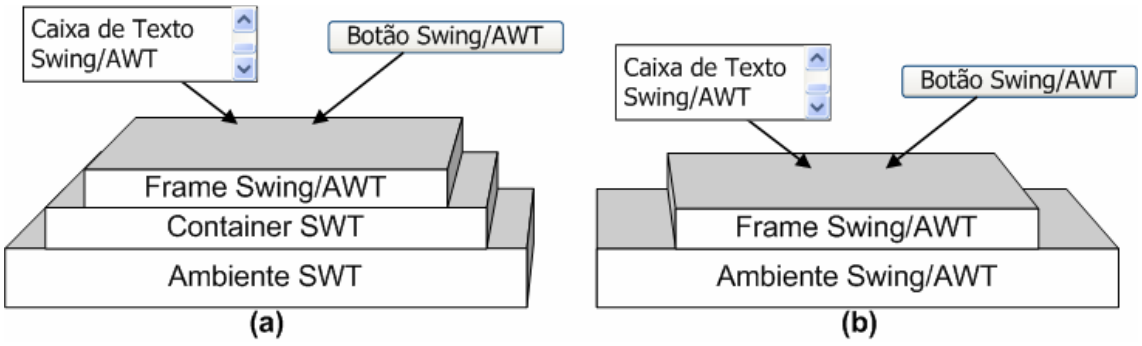

Figura 10 - Integração de um *Frame* e seus componentes visuais Swing/AWT (a) a um ambiente SWT e (b) a um ambiente Swing/AWT.

Com isso, a versão do GAW que utilizava componentes visuais da biblioteca SWT foi descontinuada, mantendo somente a versão Swing/AWT. Isto facilitou a manutenção do mecanismo, sem que se perdesse a possibilidade de integração com ambientes baseados em ambas bibliotecas.

4.1.3. Utilização

A implementação das funcionalidades básicas do mecanismo GAW foi detalhada nas seções anteriores. Nesta seção, será mostrado como estas funcionalidades estão acessíveis ao usuário através da interface gráfica.

- **Cor das barras de informação**: Ao iniciar o mecanismo, as cores das barras são escolhidas aleatoriamente, mas podem ser alteradas pelo usuário através da palheta de cores. É possível acionar esta palheta de duas formas: através do botão com o nome do usuário, ou através do item "*Color Chooser*" do menu de contexto da barra de eventos. A cor, escolhida aleatoriamente ou definida pelo usuário, não é armazenada de forma persistente entre diferentes execuções do mecanismo. Ao executar novamente o mecanismo, todas as cores serão geradas novamente. Isto vale para todas as configurações de usuário feitas no mecanismo, já que o GAW mantém as configurações somente em memória principal. A escolha de cores para um mesmo desenvolvedor pode variar entre

sessões da ferramenta. Desta forma, a cor não deve ser usada como referência ao usuário e seu histórico. O nome do usuário deve ser usado para identificar o mesmo usuário em sessões de uso diferentes. Essa limitação evita que seja necessária a definição de um servidor de dados para armazenar a configuração global do cadastro dos usuários. Apenas os dados da Fonte são suficientes.

- **Escala de tempo e intervalo de atualização**: Ambos podem ser configurados através do item "*Options*" do menu de contexto, presente no painel principal do mecanismo GAW. O painel de configuração é mostrado na Figura 11. A escala pode receber valores variando desde um milisegundo até uma década, enquanto o intervalo de atualização recebe valores de milisegundo a hora.

- **Filtros**: No GAW, não existe uma tela de configuração de filtros para o usuário. Para que uma aplicação específica disponibilize a funcionalidade de filtros dinâmicos aos usuários, é necessário que esta tela seja implementada. Um exemplo de tela de filtro será mostrado na próxima seção.

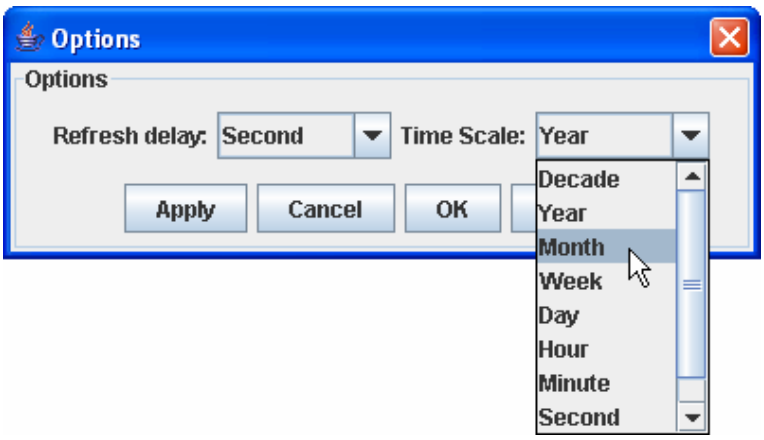

Figura 11 - Painel de opções do mecanismo GAW.

A seguir, serão apresentadas duas aplicações desenvolvidas a partir do mecanismo GAW. Estas aplicações utilizam a Visão e o Modelo implementados no mecanismo, porém têm Fonte e Coletor próprios. Cada aplicação foi integrada a um ambiente de

desenvolvimento distinto utilizado por equipes de desenvolvimento. A implementação destas aplicações ilustra a utilização do mecanismo GAW em diferentes contextos e sua integração em ambientes utilizados para desenvolver artefatos de *software* de diferentes níveis de abstração.

Ambas aplicações serão descritas da seguinte forma. Inicialmente, será feita uma breve contextualização para o desenvolvimento da aplicação. Em seguida, será feita uma descrição geral sobre sua implementação e sua integração com um ambiente de desenvolvimento. Logo após, serão apresentadas as instruções básicas para sua utilização e uma descrição de como pode ser feita a análise das informações de percepção apresentadas aos usuários.

4.2. *CVS-Watch*

Nesta seção, apresentamos a primeira aplicação desenvolvida a partir do mecanismo GAW: o *CVS-Wach*.

4.2.1. Contextualização

Muitas equipes de desenvolvimento de *software* utilizam repositórios para armazenar os artefatos gerados durante o processo de desenvolvimento. Estes repositórios ficam localizados em servidores aos quais a equipe tem acesso. Como visto no Capítulo II, antes de começar a modificar um determinado artefato, é necessário trazê-lo do servidor para a estação de trabalho e só então fazer as alterações necessárias. Ao terminar, o artefato deve ser enviado de volta para o repositório, para que toda a equipe compartilhe a mesma versão deste.

As modificações feitas nos artefatos ao longo do tempo são, muitas vezes, gerenciadas por sistemas de controle de versões e apóiam os processos de Gerência de Configuração de *Software* (ESTUBLIER, 2000). Podem existir, ainda, funcionalidades adicionais dependendo do sistema de versionamento utilizado. A

maioria, porém, permite encontrar conflitos, restaurar trechos modificados erroneamente, comparar diferentes versões de um mesmo artefato, entre outras opções.

O envio de uma nova versão do artefato ao repositório é chamado de ação de *commit*. Para cada ação de *commit* efetuada, são gravadas informações como: os nomes dos artefatos alterados, o autor das modificações, a data do *commit*, a quantidade de elementos modificados (no caso de código fonte, o número de linhas de código inseridas e removidas) entre outras. Entretanto, estes dados se encontram espalhados entre vários diretórios. Estas informações, então, não são devidamente aproveitadas pela equipe. No entanto, se estes dados, que são gerados automaticamente quando artefatos são alterados, se tornarem de fácil acesso e entendimento para os outros, estes passam a ter maior utilidade para o aumento da percepção de grupo.

4.2.2. Descrição

O *CVS-Watch* é um mecanismo derivado do GAW, que utiliza, como Fonte, os registros de atividade de um sistema de controle de versão comumente utilizado pelas equipes de desenvolvimento, o CVS (*Concurrent Versioning System*). Este sistema contém todas as funcionalidades de versionamento descritas no item anterior.

O *CVS-Watch* foi integrado ao ambiente de desenvolvimento Eclipse. O Eclipse é um ambiente extensível: a maioria de suas funcionalidades, com exceção das mais básicas que se encontram em seu núcleo, são implementadas por módulos adicionais (GAMMA e BECK, 2003). O Eclipse tem seu ambiente de trabalho dividido em painéis, chamados de visões. O *CVS-Watch* foi implementado como um desses módulos adicionais e integrado ao ambiente como uma dessas visões, como se pode observar na Figura 12. O *CVS-Watch* foi construído de acordo com o Guia de Interfaces de Usuário do Eclipse (EDGAR *et al.*, 2004), reutilizando ícones e janelas já existentes para funcionalidades

semelhantes, facilitando a integração visual do mecanismo ao ambiente.

Existe um módulo do Eclipse que integra as funcionalidades do CVS, permitindo que os usuários mantenham os projetos nele desenvolvidos sob controle de versão. A ferramenta *CVS-Watch* utiliza as funções existentes deste módulo para acessar os registros de atividade do CVS e, assim, utilizá-los como fonte de informação de percepção. Portanto, o próprio módulo do ambiente de desenvolvimento faz parte do Coletor do *CVS-Watch*. No entanto, as informações coletadas pelo Eclipse ainda precisam ser armazenadas no Modelo do *CVS-Watch*, que nada mais é do que o Modelo implementado no GAW. Para isso, foi implementado um pequeno Coletor que faz a adaptação das informações de percepção entre o Eclipse e o *CVS-Wacth*.

No contexto do *CVS-Watch*, os atributos das informações de percepção armazenadas no Modelo ganham uma semântica específica:

- ***author***: autor da versão enviada ao repositório. Corresponde ao nome de *login* cadastrado no repositório CVS;
- ***date***: data e hora do envio. Registro feito pelo CVS, no formato: “dia da semana” “mês” “dia do mês” “hora”:”minuto”:”segundo”. Como o repositório do CVS se encontra normalmente em um servidor centralizado, não deve haver conflitos de data e horário;
- ***artifact***: caminho completo do arquivo enviado. Corresponde ao nome do artefato dentro da hierarquia de diretórios;
- ***comment***: comentário escrito pelo autor sobre as alterações feitas.
- ***revision***: número da versão do arquivo enviado no repositório. A seqüência de números é controlada pelo próprio CVS;

- ***inserted / deleted elements***: número de linhas adicionadas / removidas no artefato desde a última versão enviada. Estes atributos só fazem sentido para artefatos com formato de texto, como código fonte ou documentação. Artefatos considerados binários para o CVS como, por exemplo, imagens, não apresentam esse atributo.

A informação sobre linhas adicionadas e removidas em uma versão de um determinado artefato está disponível no registro de modificações do CVS. Entretanto, o módulo do Eclipse que coleta essas informações ignora este atributo. Para ter acesso a essas informações, foi necessário implementar algumas alterações em classes internas do módulo CVS. Estas classes, no entanto, são utilizadas somente pelo próprio módulo do *CVS-Watch*, não tendo impacto no funcionamento do restante do ambiente. Também não é necessária nenhuma alteração na instalação ou configuração do ambiente para o uso dessas classes.

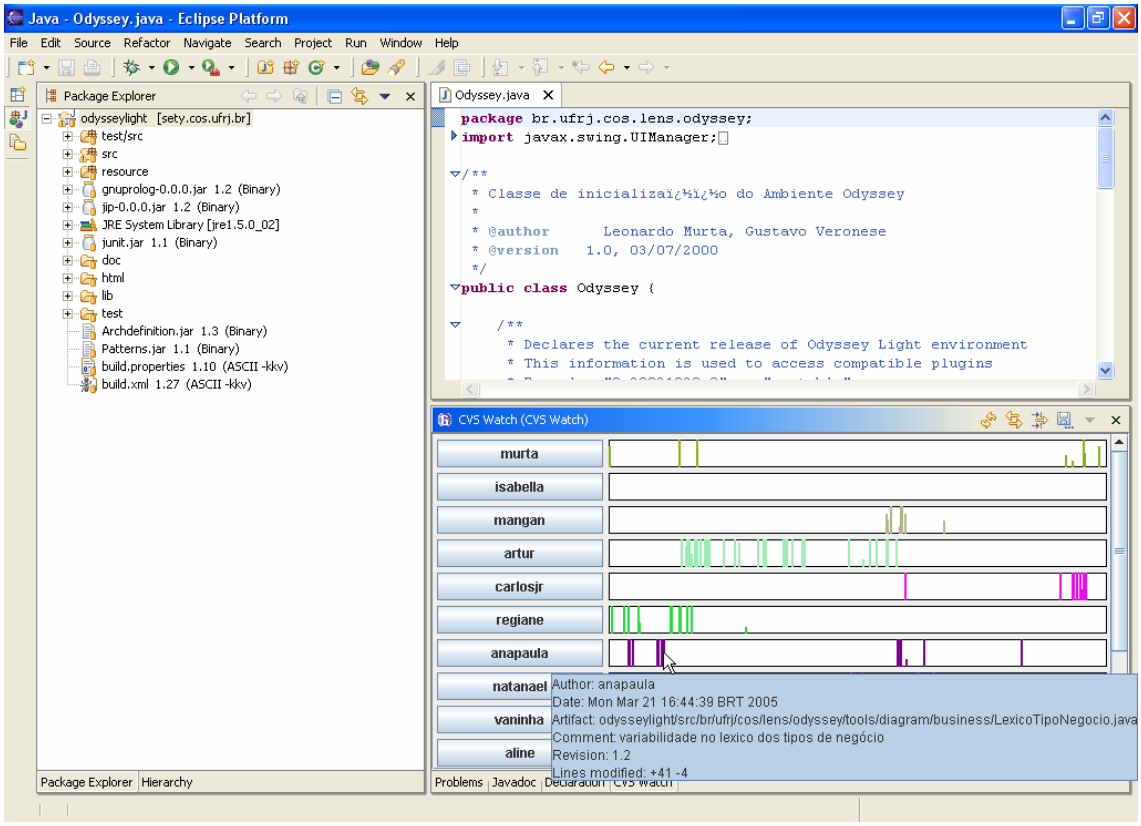

Figura 12 - Ambiente de trabalho do Eclipse com a visão do *CVS-Watch*

Na Figura 12, é mostrada visão do Eclipse do *CVS-Watch*. Cada membro da equipe que modificou o arquivo selecionado aparece listado ao lado de seus *commits*, representados pelas barras de informação do GAW da forma descrita na seção 3.3.1. No caso desta ferramenta, a altura dessas barras representa o número de linhas modificadas (soma do número de linhas inseridas e removidas) dos artefatos sob controle de versão. A largura das barras é sempre a mesma, pois o evento mostrado no caso é o envio de uma nova versão para o repositório, ação esta que tem duração irrelevante. A dica que aparece ao se posicionar o ponteiro sobre uma determinada barra de informações, mostra todos os atributos conhecidos desta.

4.2.3. Integração com o Eclipse

O Eclipse, como dito anteriormente, é um ambiente de desenvolvimento extensível. Sua interface gráfica é desenvolvida com a biblioteca SWT, o que torna necessária a adaptação descrita na Seção 4.1.1 para que o GAW possa ser integrado.

O ambiente oferece pontos de extensão para seus menus, barras de ferramenta e visões, entre outros. Os módulos devem declarar quais os pontos de extensão serão utilizados em um descritor XML, como mostrado abaixo.

```
<extension point="org.eclipse.ui.views">
      <view
       class="br.ufrj.cos.lens.odyssey.share.CvsWatch.views.CvsWatchView"
       icon="icons/cvs_watch.gif"
       category="org.eclipse.team.ccvs.ui"
       name="CVS Watch"
       id=" CvsWatch.views.CvsWatchView"/>
</extension>
```

Neste trecho, é definido o ponto de extensão que representa a visão do *CVS-Watch* no Eclipse. É indicada a classe que implementa essa visão, o ícone que a representa no ambiente, a categoria a qual esta pertence (no caso, a categoria de ferramentas integradas ao CVS), além do seu nome por extenso e seu identificador dentro do Eclipse.

No descritor do módulo, são também declaradas as dependências a outros módulos do ambiente (não mostrado no trecho anterior). Além desse descritor, cada módulo deve conter uma subclasse de *AbstractUIPlugin*, que contém os métodos necessários para a interação entre os módulos e o ambiente.

O módulo é empacotado em um *Java Archive* (JAR) (SUN, 2005d) com seu descritor. Basta copiar este arquivo para a pasta *plugins* da instalação do Eclipse, para que o módulo seja instalado e carregado no ambiente. Maiores informações sobre como baixar e instalar o módulo encontram-se no Anexo A.

4.2.4. Utilização

É necessário ter na área de trabalho do Eclipse, pelo menos, um projeto sob controle de versão do CVS para utilizar esta aplicação. O *CVS-Watch* pode ser acionado de duas maneiras distintas: uma através de um menu de contexto e outra através de um gesto de "arrastar e soltar". Ao clicar com o botão direito em um determinado arquivo, diretório, pacote ou projeto aparecerá um menu de contexto. Neste menu basta escolher a opção *Team* ▸ *CVS Watch* para que a visão da ferramenta apareça com as informações listadas anteriormente. A outra forma de ativação é arrastar o item desejado e soltá-lo sobre a visão da ferramenta no ambiente de trabalho do Eclipse. Para utilizar essa alternativa, o painel do *CVS-Watch* deve estar visível no ambiente. Caso não esteja, é necessário acessar o menu do Eclipse *Window* ▸ *Show View* ▸ *Other*... e selecionar, na lista que aparecerá, a opção *CVS* ▸ *CVS Watch*.

É possível, também, selecionar múltiplos itens simultaneamente, inclusive de tipos distintos (por exemplo: um pacote e um arquivo) e obter as informações sobre estes agrupadas pela ferramenta. A seleção múltipla é uma função que não encontra similar no módulo de integração com o CVS do Eclipse. Para distinguir a que item selecionado pertence um determinado *commit*, basta posicionar o ponteiro do *mouse* em cima da barra e consultar na dica o nome do artefato que o gerou.

Outras funcionalidades podem ser acessadas através da barra de ferramentas da visão do *CVS-Watch* (Tabela 1).

Tabela 1 - Opções da barra de ferramentas do *CVS-Watch*

	*Refresh*	Atualiza os dados mostrados
	*Link with Editor*	Mostra informações do arquivo corrente no editor
	*Filter*	Configura filtros para os eventos mostrados
	*Save Log As*	Grava as informações em um arquivo texto

O botão *Refresh* aciona o Coletor para atualizar as barras de eventos de acordo com as últimas informações de percepção encontradas nos registros do CVS. São adicionadas novas barras de eventos, caso tenham sido feitas ações de *commits* dos artefatos selecionados desde a última vez que o usuário consultou os registros. Ao pressionar o botão *Link with Editor*, é estabelecida uma ligação entre o editor do Eclipse e o *CVS-Watch*. Isto quer dizer que o *CVS-Watch* passa a mostrar as informações sobre o arquivo que estiver aberto no editor do ambiente de trabalho. Ao abrir outro arquivo no editor, o *CVS-Watch* busca e mostra as informações referentes a este novo arquivo. Para desativar essa função, basta pressionar o botão novamente.

O botão *Filter* abre uma janela (Figura 13) com campos para a configuração de um filtro dinâmico para as informações de percepção. Através desta janela, é possível fazer a filtragem pelo autor do *commit* (*author*), por palavras contidas nos comentários (*comment containing*) e por período de tempo. Para este último, é necessário informar a data de início (*from date*) e de fim (*to date*) do período desejado.

O botão *Save Log As* permite que o usuário exporte os dados mostrados para o formato de texto, como estão originalmente representados no arquivo de registro do CVS. Ao pressionar o botão, uma janela é aberta para que se escolha onde salvar o arquivo texto gerado. Este arquivo pode ser utilizado para fazer

análises mais detalhadas das informações de percepção, inclusive com o uso de outras ferramentas que sejam de interesse do usuário, como o *Microsoft Excel* (MICROSOFT, 2005) ou um programa estatístico.

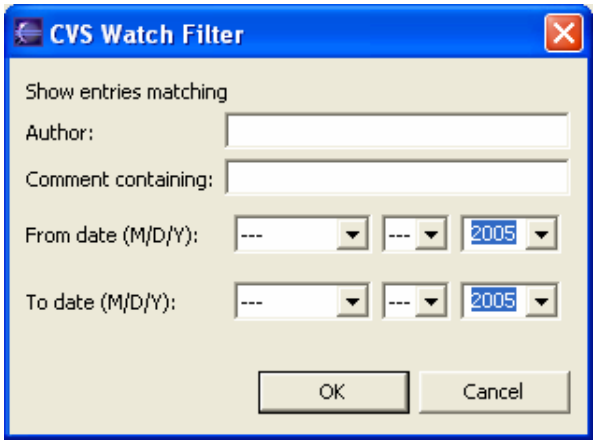

Figura 13 - Janela de filtro do *CVS-Watch*

4.2.5. Análise das informações

Pela análise das informações de percepção, o *CVS-Watch* permite que os participantes da equipe de desenvolvimento obtenham conhecimento sobre as atividades dos colegas. Explorando essas informações como um histórico, pode-se encontram os "especialistas" em determinado artefato ou saber o que foi alterado desde a última vez que se trabalhou em determinado artefato ou módulo do *software*. Como, neste caso, as informações sobre as alterações só são disponibilizadas depois que a nova versão do artefato foi mandada para o servidor e não quando estas realmente acontecem, a exploração dos dados de forma instantânea fica prejudicada.

Para melhor demonstrar como é possível analisar as informações mostradas pelo *CVS-Watch*, responderemos as questões propostas na Seção 2.2 deste trabalho.

- **Quais participantes da equipe já trabalharam na construção deste artefato?**
  Para responder a esta pergunta, basta olhar para a lista

de botões com o nome dos usuários que aparecem no *CVS-Watch* para o artefato desejado. Apenas os usuários que enviaram pelo menos uma versão deste artefato para o repositório aparecerão nesta lista.

- **Quem é o responsável pela última versão do artefato?**
  Como a linha do tempo no mecanismo é ordenada do presente para o passado, as barras posicionadas mais à esquerda correspondem às informações mais recentes. Portanto, para saber quem enviou a última versão para o repositório do artefato selecionado, basta encontrar a barra que se encontra mais perto dos botões e ver qual o nome do usuário.

- **Qual seria o participante mais indicado para tirar minhas dúvidas sobre o artefato?**
  A resposta para essa pergunta depende do que se considera como "o participante mais indicado". Caso seja o que criou o artefato, basta ver qual a barra de evento mais antiga (mais à direita) para aquele artefato e verificar quem é o autor daquela ação de *commit*. Já quando se considera quem fez o maior número de modificações no artefato, basta ver quem possui o maior número de barras de eventos. Pode ser também considerado o que fez mudanças mais significativas no artefato. Neste caso, pode-se verificar qual usuário possui as barras de eventos mais altas ou o maior número de linhas de código modificadas (através da dica). Outra possibilidade ainda é ver quem é o responsável pela última versão do artefato, como na pergunta anterior.

Uma análise mais abrangente pode ser feita ao se selecionar um projeto, ao invés de somente um arquivo na área de trabalho. A partir das informações do projeto inteiro, ou de vários projetos simultaneamente, pode-se saber quem é o usuário mais ativo (através da análise do número de barras), ou quais são as épocas de maior atividade do grupo (através da análise da concentração das barras em relação à linha de tempo). Análises mais detalhadas, que necessitem de processamento dos dados, podem

ser feitas a partir do arquivo texto exportado, como descrito no item anterior.

O próprio GAW e o *CVS-Watch* não exploram a automação da análise das informações, pois existem fatores subjetivos na sua interpretação. O usuário pode ter preferência em contactar um desenvolvedor em relação a outro. Outros fatores não presentes na informação de percepção coletada, como o histórico de trabalho entre os desenvolvedores em outros artefatos, também podem também interferir na escolha. Além disso, a informação mais recente ou mais regular pode ser valorizada de forma diferente de acordo com as preferências do usuário que faz a consulta. Entretanto, a exportação dos dados permite que usuários com interesses específicos realizem análises mais detalhadas.

4.3. *WorkRhythm*

A segunda aplicação derivada do mecanismo GAW, o *WokRhythm*, é mais limitada em relação ao número de funcionalidades do que a anterior. Isto se deve ao fato de que o desenvolvimento desta teve como principal objetivo mostrar a adaptabilidade da ferramenta GAW a diferentes contextos e ambientes de desenvolvimento. Por isso, o mecanismo só apresenta as funcionalidades básicas implementadas pelo próprio GAW.

4.3.1. Contextualização

As equipes de desenvolvimento não lidam somente com código fonte. Durante o processo de desenvolvimento, artefatos de *software* em diversos níveis de abstração são gerados. Da mesma forma como a implementação, os modelos evoluem, sejam estes de classes, arquiteturais ou de casos de uso. Isto significa que os membros da equipe estão atualizando estes modelos constantemente, muitas vezes paralelamente.

Existem técnicas de controle de versões já bastante utilizadas para resolver os problemas do desenvolvimento concorrente de artefatos implementacionais como: ferramentas para identificar as diferenças entre duas versões do mesmo código fonte (*diff*) e ferramentas para combinar versões de forma manual ou automática (*merge*). Entretanto, o suporte para o versionamento para os níveis de abstração mais altos de *software* ainda é muito limitado. Grande parte dos sistemas de GCS não oferece técnicas de *diff* e *merge* para modelos de *software*. A solução utilizada mais comumente é, então, a comparação por inspeção visual das diferentes versões do modelo para tentar identificar as diferenças e depois a junção manual das mesmas. Esta solução, além de ser lenta, é muito suscetível a erros.

Deve-se, portanto, tentar reduzir os problemas gerados por conflitos entre diferentes versões de um mesmo modelo que necessitem desta abordagem penosa de integração para serem solucionados. Para tanto, pode-se adotar uma medida preventiva: notificar instantaneamente o desenvolvedor sobre alterações que estão sendo feitas no modelo pelos demais. Desta forma, o desenvolvedor pode entrar em contato com o responsável pela alteração, caso seja necessário, e atualizar o seu modelo antes, evitando possíveis conflitos futuros.

Além disso, o desenvolvedor pode querer saber quais foram as mudanças feitas em um determinado modelo enquanto ele esteve ausente da área de trabalho. Ao retornar, o desenvolvedor deveria ser notificado das modificações que ocorreram desde a última vez em que ele visualizou tal modelo, para conseguir acompanhar a evolução e avaliar possíveis incompatibilidades com a sua percepção do modelo.

4.3.2. Descrição

Neste contexto de edição colaborativa de modelos de *software* está inserido o *WorkRhythm*, a adaptação da ferramenta GAW que mostra as mudanças feitas em diagramas compartilhados pela equipe. Todos os membros da equipe podem saber quando

um determinado elemento do diagrama foi modificado, por quem e quando a alteração ocorreu. As informações sobre as modificações feitas nos diagramas são mostradas de forma semelhante à do *CVS-Watch*.

O *WorkRhyhtm* está integrado ao *Odyssey*, um ambiente de reutilização baseado em modelos de domínio. O ambiente permite representar o conhecimento de um domínio, através de extensões de diagramas UML (OMG, 2004), e permite que estes sejam reutilizados para facilitar a construção de novas aplicações. O *Odyssey* é, também, um ambiente extensível e permite que novos módulos sejam encontrados e baixados pela Internet e adicionados ao ambiente em tempo de execução, através de um mecanismo de carga dinâmica (MURTA *et al*., 2004). O *WorkRhythm* foi implementado como um módulo de extensão do ambiente *Odyssey*. Os detalhes sobre a integração com o *Odyssey* serão apresentados em uma seção posterior.

Um dos outros módulos adicionais do *Odyssey*, o MAIS (LOPES, 2004) é utilizado como Coletor de informações de percepção do *WorkRhythm*. O MAIS coleta e distribui as informações sobre as modificações feitas nos modelos de classe. Estas informações são persistidas em um espaço de tuplas, que serve como Fonte para o *WorkRhythm*. Este espaço representa "um repositório central de armazenamento de objetos, que contém algum tipo de computação específica, podendo ser lidos, escritos e retirados por dois ou mais processos distribuídos, em uma rede de comunicação" (LOPES, 2004).

Assim como no *CVS-Watch*, foi necessário implementar uma parte do Coletor do *WorkRhythm* que ficasse responsável por fazer a adaptação das informações de percepção recebidas o Modelo da aplicação.

A Figura 14 apresenta a janela do *WorkRhythm* no ambiente *Odyssey*. Quando um determinado desenvolvedor faz alterações em um modelo, os demais são notificados através da interface gráfica da ferramenta. Cada notificação de mudança aparece como uma barra vertical na linha do tempo da barra de eventos. Os

eventos são agrupados mais uma vez pelo nome do autor da modificação que gerou a notificação e formam um histórico das modificações feitas no modelo em questão.

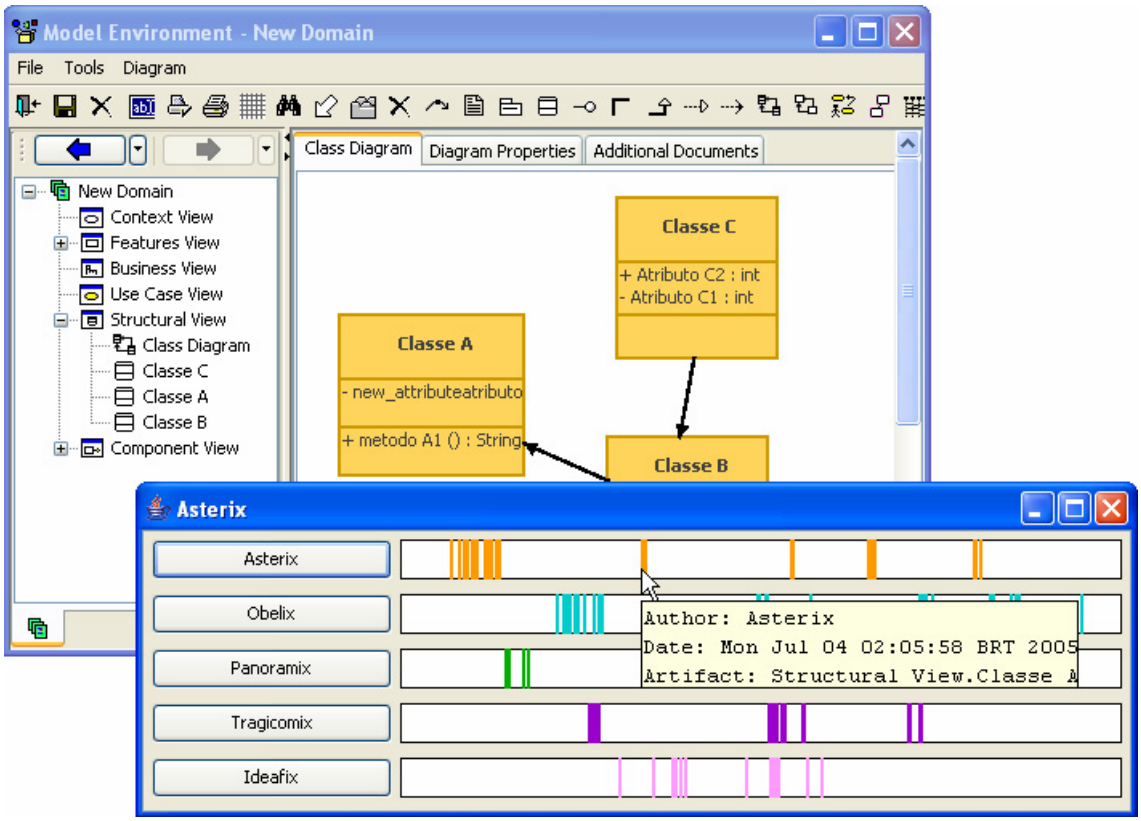

Figura 14 - Ambiente de trabalho do *Odyssey* com a janela do *WorkRhythm*.

Os atributos das informações de percepção mostradas na dica (Figura 14) são apenas as de autor, data e artefato. Isso ocorre, pois os demais atributos do Modelo do GAW não fazem sentido no contexto do *WorkRhythm*. Como as informações são coletadas diretamente da estação de trabalho dos usuários, não existe a ação de *commit*, portanto, não existe o conceito de versão ou de comentário do autor. As barras também são sempre da mesma altura e largura, pois cada barra representa uma única alteração em um diagrama, como “renomear atributo X” ou “criar classe Y”. O número de elementos modificados é sempre igual a uma unidade e a ação de modificação não tem uma duração relevante.

A aplicação é limitada ao monitoramento de diagramas de classes UML e seus elementos básicos: classe, método e atributo,

devido a restrições do próprio módulo MAIS. Entretanto, caso este módulo passe a coletar informações sobre outros diagramas, não haverá necessidade de modificar o *WorkRhythm* para que as alterações nos novos elementos sejam mostradas pelo mecanismo.

4.3.3. Integração com o *Odyssey*

Como dito anteriormente, o *Odyssey* é um ambiente extensível e o *WorkRhythm* foi implementado como um módulo de extensão deste ambiente. Como a interface gráfica do *Odyssey* também é desenvolvida com a biblioteca SWING/SWT, não é necessário fazer adaptações ao GAW para que este se integre ao ambiente.

Todos os módulos do *Odyssey* têm que implementar a interface *Tool* (Figura 15), que define os possíveis pontos de interação com o ambiente (MURTA *et al.*, 2004).

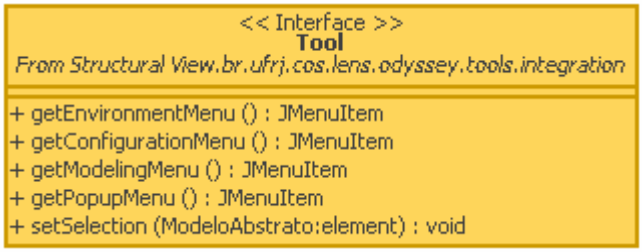

Figura 15 - Inteface *Tool*

Dentre estes pontos de interação, o *WorkRhythm* contribui apenas para o menu da área de modelagem do *Odyssey*. Esta contribuição é implementada pelo método *getModelingMenu()*, onde é criado um item de menu que instancia o *WorkRhythm* e abre sua janela sobre o ambiente.

O módulo é empacotado em um arquivo tipo JAR, que deve possuir o arquivo MANIFEST.MF com meta-dados úteis para que o *Odyssey* identifique qual classe implementa a interface *Tool*. É necessário, ainda, indicar onde o módulo empacotado se encontra, assim como o nome dos módulos dos quais o *WorkRhythm*

depende, para que também possam ser baixados e instalados pelo *Odyssey*. Essas informações se encontram em um descritor XML. O trecho abaixo, extraído deste descritor, define o *WorkRhythm* como um módulo do Odyssey empacotado no arquivo *workrhythm-1.1.0.jar*, assim como o endereço de armazenamento deste arquivo. Além disso, é definida a dependência entre o *WorkRhythm* e o módulo MAIS.

```
<component type="plugin" name="workrhythm-1.1.0.jar"
description="WorkRhythm"
location="http://reuse.cos.ufrj.br/releases/components">
      <dependency name="mais-1.2.0.jar"/>
</component>
```

Maiores informações sobre como fazer a carga dinâmica do *WorkRhythm* no *Odyssey* se encontram no Apêndice B.

4.3.4. Utilização

Antes de iniciar o uso do GAW, é necessário iniciar o uso do MAIS, devido a dependência entre os módulos. Para isso é necessário acessar o menu *Tools* ▸ *MAIS InfraStructure* ▸ *Create Channel*. Em seguida, para ativar o *WorkRhythm*, basta acessar o menu *Tools* ▸ *Gaw*. As duas ferramentas são instaladas através da rede, como módulos adicionais do *Odyssey*.

Como dito anteriormente, esta é uma implementação mais simples do GAW do que o *CVS-Watch*. As funcionalidades da barra de ferramentas, dos filtros e de salvar em arquivo não estão disponíveis. Apenas as funcionalidades básicas do GAW estão presentes como o painel de opções e as dicas.

4.3.5. Análise das informações

A análise das informações mostrados pelo *WorkRhythm* pode ser feita de maneira semelhante à do *CVS-Watch*, (apresentada na Seção 4.2.4.) explorando-as como um histórico. Porém, como neste caso o mecanismo é notificado assim que as alterações nos

modelos de *software* são feitas, o *WorkRhythm* também permite a exploração das informações de modo instantâneo. Desta forma, os desenvolvedores ficam cientes das alterações que estão sendo feitas pelos demais no momento em que estas ocorrem. Como dito anteriormente, isso permite que sejam percebidos conflitos antes que as alterações sejam enviadas ao servidor.

Por exemplo, na Figura 14, o usuário *Asterix* pode estar seguro de que nenhum outro usuário realizou alterações nos artefatos selecionados. Se houver uma operação após as realizadas pelo *Asterix*, aparecerá uma barra à esquerda da última barra pertencente a ele.

4.4. Comparação entre *CVS-Watch* e *WorkRhythm*

As duas aplicações derivadas do mecanismo GAW são muito semelhantes, pois compartilham das funcionalidades básicas oferecidas por este. Apesar disso, as aplicações propõem abordagens diferentes para melhorar a percepção do grupo. Uma comparação entre elas se encontra resumida na Tabela 2.

Tabela 2 - Comparação entre as aplicações derivadas do mecanismo GAW

	CVS-Watch	***WorkRhythm***
Ambiente	Eclipse	*Odyssey*
Artefatos	Arquivos e pastas	Elementos do diagrama de classes
Confirmação	Informações confirmadas	Informações ainda não confirmadas
Origem das informações	Servidor	Estações
Atualização	Usuário	Automática

A primeira diferença é em relação ao tipo de ambiente em que foram integradas as aplicações. Enquanto o *CVS-Watch* foi integrado a um ambiente de implementação, o *WorkRhythm* foi acoplado ao *Odyssey*, um ambiente de modelagem. Por serem de

natureza diferente, estes ambientes trabalham com abstrações distintas de representação do *software*. No Eclipse, são exploradas as diferentes versões dos arquivos do projeto, incluindo código fonte, documentos textuais, entre outros. Já no *Odyssey*, são acompanhadas as alterações feitas em elementos de modelos UML. Se o CVS for usado para armazenar o arquivo de dados do *Odyssey*, por exemplo, apesar de se conseguir perceber as diferentes versões de modelos UML com o *CVS-Watch*, não é possível através dele saber o que realmente foi mudado de uma versão do modelo para outra, pois o CVS somente distingue os arquivos sob controle de versão como textuais ou binários. Já no *WorkRhythm*, é possível lidar com uma granularidade mais fina e distinguir mudanças feitas em classes, atributos e métodos do modelo.

Outros pontos que distinguem as duas variantes são em relação à confirmação das informações apresentadas. O *CVS-Watch* busca dados sobre as versões dos arquivos que teoricamente já foram validados e disponibilizados para que toda equipe possa utilizá-los. Neste caso, os usuários têm um papel ativo em relação ao envio da nova versão desenvolvida ao servidor de onde serão coletadas as informações de percepção. No caso do *WorkRhythm*, as informações apresentadas são coletados diretamente da estação de trabalho dos participantes da equipe. Isto significa que, enquanto o usuário está fazendo modificações em um determinado modelo, todos os outros irão receber notificações sobre essas modificações. As informações coletadas, neste caso, são sobre uma versão que ainda está sendo alterada e que, provavelmente, não será definitiva. Essa abordagem tem a vantagem de permitir que se percebam possíveis conflitos antes que uma nova versão seja disponibilizada para o resto do grupo.

O último ponto de comparação entre as variantes se refere à atualização das informações mostradas ao usuário através da interface gráfica. Quando uma nova modificação é feita em um determinado artefato, o *WorkRhythm* é atualizado automaticamente para mostrar as informações da modificação, já o *CVS-Watch* depende da intervenção do usuário para se atualizar. Este

comportamento se deve à diferença entre o volume de informações recebidas pelas aplicações e pela forma propagação de mudanças oferecida pela Fonte adotada. Por receber informações sobre as modificações sendo feitas em artefatos nas estações de trabalho de todos os membros da equipe, o *WorkRhythm* foi implementado de maneira a não necessitar que o usuário solicite a atualização das informações. Já o *CVS-Watch*, só busca informações no servidor e, portanto, recebe um volume menor de informações. A atualização das informações, então, não precisa ser feita tão freqüentemente e, caso seja desejada, o usuário deve solicitá-la. Do ponto de vista de implementação, a notificação no CVS exigiria a implementação de um mecanismo adicional, o que foi evitado.

4.5. Considerações Finais

Neste capítulo, foi apresentada a implementação do mecanismo GAW e de duas aplicações dele derivadas, assim como a integração dessas aplicações a diferentes ambientes de desenvolvimento. A Figura 16 ilustra as relações entre o mecanismo, as aplicações derivadas e os diferentes ambientes, módulos e fontes de informação descritos neste trabalho.

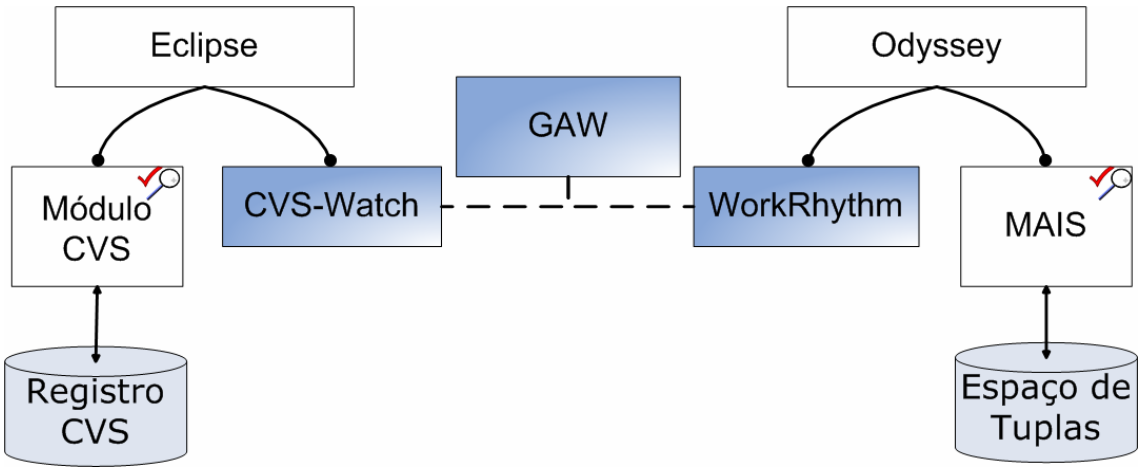

Figura 16 - Relação entre o mecanismo GAW, suas aplicações derivadas *CVS-Wacth* e *WorkRhythm* e os Coletores, Fontes e Ambientes a que estão integrados.

O mecanismo de percepção GAW contém as funcionalidades básicas comuns utilizadas pelas aplicações derivadas *CVS-Watch*

e *WorkRhythm*. Estas duas estão integradas, como módulos adicionais, aos ambientes de desenvolvimento extensíveis Eclipse e *Odyssey*, respectivamente. Estas aplicações utilizam esta integração para se comunicarem com outros módulos do ambiente, que fazem parte do papel de Coletores de informações de percepção. O *CVS-Watch* utiliza o Módulo de CVS do Eclipse para obter as informações no registro de modificações de arquivos sob controle de versão pelo CVS. Enquanto que o *WorkRhythm* utiliza o módulo MAIS, do ambiente *Odyssey*, para consultar o Espaço de Tuplas, onde ficam armazenadas as informações sobre as alterações feitas em diagramas compartilhados pela equipe.

Apesar do *CVS-Watch* e do *WorkRhythm* estarem integrados a ambientes de desenvolvimento, nada impede que sejam desenvolvidas aplicações independentes baseadas no mecanismo GAW. Ambas foram implementadas desta forma, pois, como dito anteriormente, essa integração acaba facilitando o uso destas aplicações.

CAPÍTULO V
CONCLUSÃO

Neste último capítulo são apresentadas as considerações finais sobre este trabalho, assim com suas principais contribuições, limitações e perspectivas de trabalhos futuros.

5.1. Considerações finais

Este trabalho descreveu o problema de isolamento das equipes de desenvolvimento de *software* distribuídas, que necessitam de mais tempo para realizar suas tarefas do que as equipes localizadas. O conceito de percepção de grupo foi introduzido como base para a resolução deste problema. O uso de mecanismos de percepção permite que os participantes fiquem cientes das atividades dos demais, melhorando, possivelmente, a coordenação de esforços da equipe e facilitando a localização de ajuda, o que torna o processo de desenvolvimento mais ágil.

Neste trabalho, foi proposto um mecanismo de percepção que utiliza informações sobre alterações feitas em artefatos compartilhados pelas equipes distribuídas de desenvolvimento. Esse tipo de informação já se encontra em arquivos de *log*, bancos de dados ou outras Fontes localizadas no servidor utilizado pela equipe, não necessitando que seus participantes realizem ações adicionais para gerá-las. O mecanismo permite que estas informações sejam visualizadas através de uma representação gráfica, facilitando sua análise.

Este mecanismo foi implementado e serve de base para outras aplicações, permitindo que este seja integrado a ambientes de desenvolvimento utilizados pela equipe. Foram desenvolvidas duas aplicações integradas, ilustrando o uso do mecanismo em diferentes contextos e sua adaptação com ambientes e condições distintas.

Apesar ter sido desenvolvido para o uso no contexto do desenvolvimento distribuído de *software*, o mecanismo proposto também pode ser útil para equipes localizadas. Por exemplo, um novo desenvolvedor, quando chega à equipe, tem pouco conhecimento sobre o histórico do *software* em desenvolvimento e ainda não conhece as competências de seus colegas. Neste caso, o mecanismo poderia auxiliar este novo desenvolvedor a se integrar mais rapidamente à equipe.

5.2. Contribuições

Entre as principais contribuições deste trabalho, podemos destacar:

- A proposta de um mecanismo de percepção para equipes de desenvolvimento de *software* distribuídas, baseada em informações sobre alterações feitas em artefatos compartilhados.
- A descrição de uma arquitetura de alto nível para mecanismos que utilizem esta abordagem.
- A implementação do mecanismo de percepção GAW, que pode ser utilizado como base do desenvolvimento de outras aplicações.
- A implementação das aplicações *CVS-Watch* e *WorkRhythm*, derivadas do mecanismo GAW, integradas ao ambiente de desenvolvimento Eclipse e ao ambiente de reutilização *Odyssey*, respectivamente. Estas aplicações ilustram a flexibilidade e a capacidade de integração do GAW.

5.3. Limitações e Trabalhos Futuros

O mecanismo proposto neste trabalho tem a função de representar, de forma gráfica, as informações de percepção

coletadas. A análise das informações, apesar de ser facilitada por este tipo de representação, é feita pelo próprio usuário. Seria interessante que o mecanismo o auxiliasse nesta tarefa. As informações poderiam ser utilizadas para tentar encontrar padrões e fazer inferências, auxiliando a determinar o "especialista" em determinado artefato ou fornecendo informações sobre produtividade da equipe. Para tal, técnicas de mineração de dados poderiam ser utilizadas pelo mecanismo.

Além disso, o mecanismo foi projetado para indicar que ocorreram mudanças nos artefatos compartilhados, mas este consegue apenas apresentar o "tamanho" da modificação feita de uma versão para a seguinte, seja em número de linhas de código ou número de elementos inseridos ou removidos. Para saber o que exatamente mudou, o próprio usuário pode ter que comparar as duas versões. O mecanismo poderia indicar mais precisamente as alterações feitas, assim como possíveis conflitos entre diferentes versões do artefato, facilitando a convergência das diferentes versões.

Uma das principais limitações da implementação do GAW, em relação à sua capacidade de adaptação, é que os atributos das informações de percepção são definidos diretamente no código do Modelo do mecanismo. Apesar de já estarem incluídos os atributos básicos para representar essas informações, alguma futura aplicação pode necessitar incluir outros atributos. Neste caso, o desenvolvedor deverá incluir estes atributos diretamente no código do GAW. Seria interessante, como trabalho futuro, permitir que a lista de atributos fosse extensível.

Outra limitação desta implementação é ter apenas uma única maneira de visualizar as informações de percepção, representadas pelas barras coloridas, ordenadas em uma linha do tempo decrescente. Outros tipos de representação poderiam ser incorporados ao mecanismo para facilitar diferentes tipos de análise das informações.

Outra limitação no GAW é que as configurações selecionadas por um usuário não são mantidas entre diferentes execuções do

mecanismo. Por exemplo, as cores das barras de usuário, poderiam ser mantidas, facilitando sua identificação na interface gráfica entre sessões de trabalho.

As aplicações derivadas do GAW apresentam suas próprias limitações. O *WorkRhythm* pode ser aprimorado para apresentar funcionalidades já presentes no *CVS-Watch*, como a tela de configuração de filtros dinâmicos. O *CVS-Watch*, por sua vez, poderia explorar mais funcionalidades do próprio CVS, como, por exemplo, mostrar as diferenças inseridas de uma versão para outra em um determinado artefato.

Além disso, outras aplicações poderiam ser desenvolvidas a partir do mecanismo GAW, para integrá-lo a outras ferramentas extensíveis utilizadas pelas equipes de desenvolvimento, como o *IBM Rational Software Modeler* e o *NetBeans IDE*. Para tanto, seria necessário identificar a Fonte de dados adequada e desenvolver o Coletor e as adaptações necessárias para a integração.

REFERÊNCIAS BIBLIOGRÁFICAS

BEGOLE, J., TANG, J. C., SMITH, R. B. *et al.*, 2002, "Work rhythms: analyzing visualizations of awareness histories of distributed groups". In: *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, Nova Orleans, USA, novembro.

DOURISH, P., BELLOTTI, V., 1992, "Awareness and Coordination in Shared Workspaces". In: *Proceedings of the Conference on Computer Supported Cooperative Work*, pp. 107-114, Toronto.

ECLIPSE FOUNDATION, 2005, "Eclipse". In: http://www.eclipse.org, Acessado em julho de 2005.

EDGAR, N., HAALAND, K., LI, J. *et al.*, 2004, "Eclipse User Interface Guidelines", In: http://www.eclipse.org/articles/Article-UI-Guidelines/Contents.html. Acessado em abril de 2005.

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

ESTUBLIER, J., 2000, "Software Configuration Management: a roadmap". In: *Proceedings of the Conference on the Future of Software Engineering*, pp. 279-289, Limerick, Irlanda.

GAMMA, E., BECK, K., 2003, *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*. 1 ed. USA, Addison Wesley.

GRUDIN, J., 1994. "Groupware and social dynamics: eight challenges for developers", *Communications of the ACM,* v. 37, n. 1 (janeiro), pp. 92-105.

HERBSLEB, J. D., MOCHUS, A., FINHOLT, T. *et al.*, 2001, "An Empirical Study of Global Software Development: Distance and Speed". In: *Proceedings of the International Conference on Software Engineering*, pp. 81-90, Toronto, Canada.

ICQ, 2005, "ICQ". In: http://www.icq.com, Acessado em julho de 2005.

KREIJNS, K., KIRSCHNER, P. A., 2001, "The Social Affordances of Computer-Supported Collaborative Learning Environments". In: *Proceedings of the 31st ASEE/IEEE Frontiers in Education Conference*, Nevada, USA, outubro 10-13.

LOPES, M.A.M., MANGAN, M. A. S., WERNER, C.M.L., 2004, "MAIS: uma ferramenta de percepção para apoiar a edição concorrente de modelos de análise e projeto", In: *XVII Simpósio Brasileiro de Engenharia de Software - XI Caderno de ferramentas*, pp.61-66, Brasília, Distrito Federal, outubro.

MICROSOFT, 2005, "Microsoft Excel". In: http://office.microsoft.com/pt-br/FX010858001046.aspx, acessado em 08/2005.

MOTTA, C.L.R., BORGES, M.R.S., 2000, "A Cooperative Approach for Information Recommendation and Filtering," In: *6th International Workshop on Groupware - CRIWG'00*, p. 42.

MURTA, L. G. P., VASCONCELOS, A. P. V., BLOIS, A.P.T.B., *et al*., 2004, "Run-time Variability through Component Dynamic Loading", In: *XV Simpósio Brasileiro de Engenharia de Software – Caderno de Ferramentas*, Brasília, Distrito Federal, outubro.

OLSON, G., OLSON, J., 2000, "Distance Matters", *Human-Computer Interaction*, v.15, n. 2, pp.139-178.

SILVA, I.A., MANGAN, M. A. S., WERNER, C.M.L., 2004, "CVS-Watch: A Group Awareness Tool Applied to Collaborative Software Development", In: *La WEB/WebMidia 2004*, Ribeirão Preto, SP, Brasil, outubro.

SUN, 2005a, "Java Technology". In: http://java.sun.com/, acessado em 08/2005.

SUN, 2005b, "JavaBeans". In: http://java.sun.com/products/javabeans/, acessado em 08/2005.

SUN, 2005c, "Java Foundation Classes (JFC/Swing)". In: http://java.sun.com/products/jfc/index.jsp, acessado em 08/2005.

SUN, 2005d, "Java Archive". In: http://java.sun.com/docs/books/tutorial/jar/, acessado em 08/2005.

WERNER, C. M. L., BRAGA, R. M. M., MATTOSO, M. *et al*., 2000, "Infra-estrutura Odyssey: Estágio Atual", In: *XV Simpósio Brasileiro de Engenharia de Software – Caderno de ferramentas*, pp. 366-369, João Pessoa, Paraíba, outubro.

APÊNDICE A
INSTALAÇÃO DO *CVS-WATCH* NO ECLIPSE

É possível baixar e instalar módulos do Eclipse através da interface gráfica do próprio ambiente. O Eclipse permite o desenvolvimento de sítios de atualização, que nada mais são endereços que indicam ao ambiente de onde baixar os módulos pela Internet.

Em seguida, serão listados os passos necessários para utilizar o sítio de atualização do *CVS-Watch* para instalar este módulo no Eclipse (versões 3.x):

- Acionar o menu *Help* ▸ *Software Updates* ▸ *Find and Install...* no ambiente de trabalho do Eclipse (Figura 17).

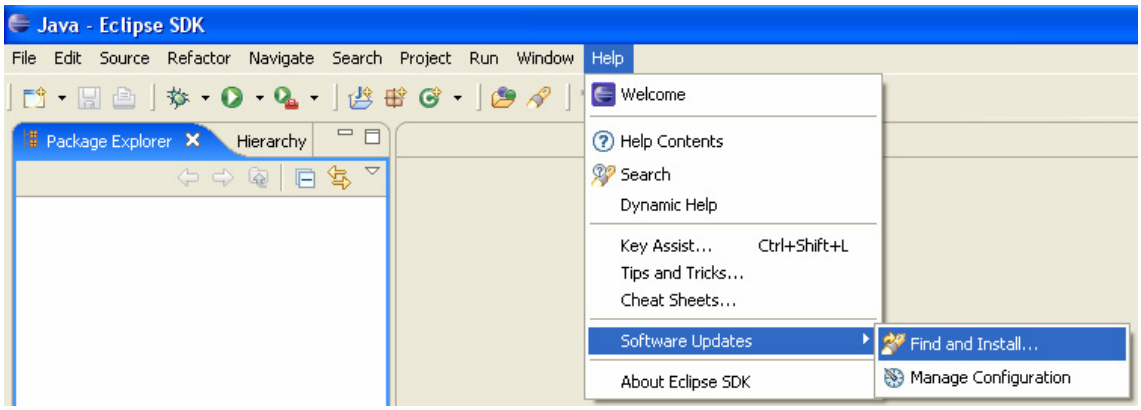

Figura 17 - Menu do Eclipse de instalação de novos módulos

- Aparecerá a janela *Install* (Figura 18). Escolher a opção *Search for new features to install*, para instalar novos módulos, ao invés de atualizar os módulos já instalados e clicar no botão *Next*.

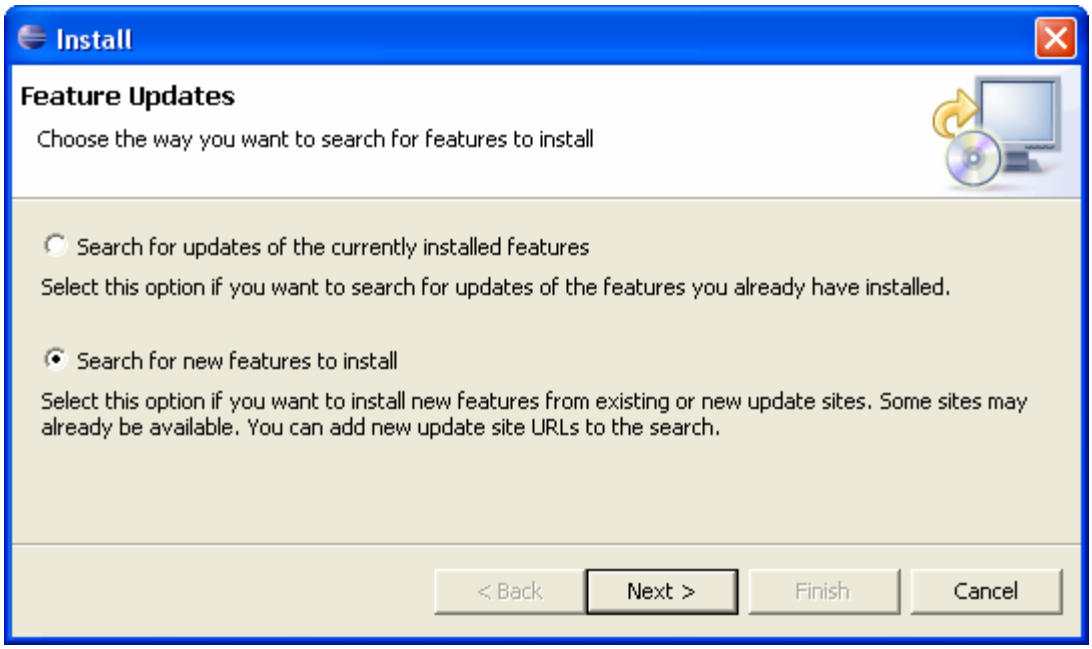

Figura 18 - Janela *Install*

- Ainda na janela *Install*, aparecerá a lista de sítios de atualização cadastrados, com o sítio do próprio Eclipse (Figura 19). Clicar no botão *New Remote Site...*, para adicionar o sítio do *CVS-Watch* à lista.

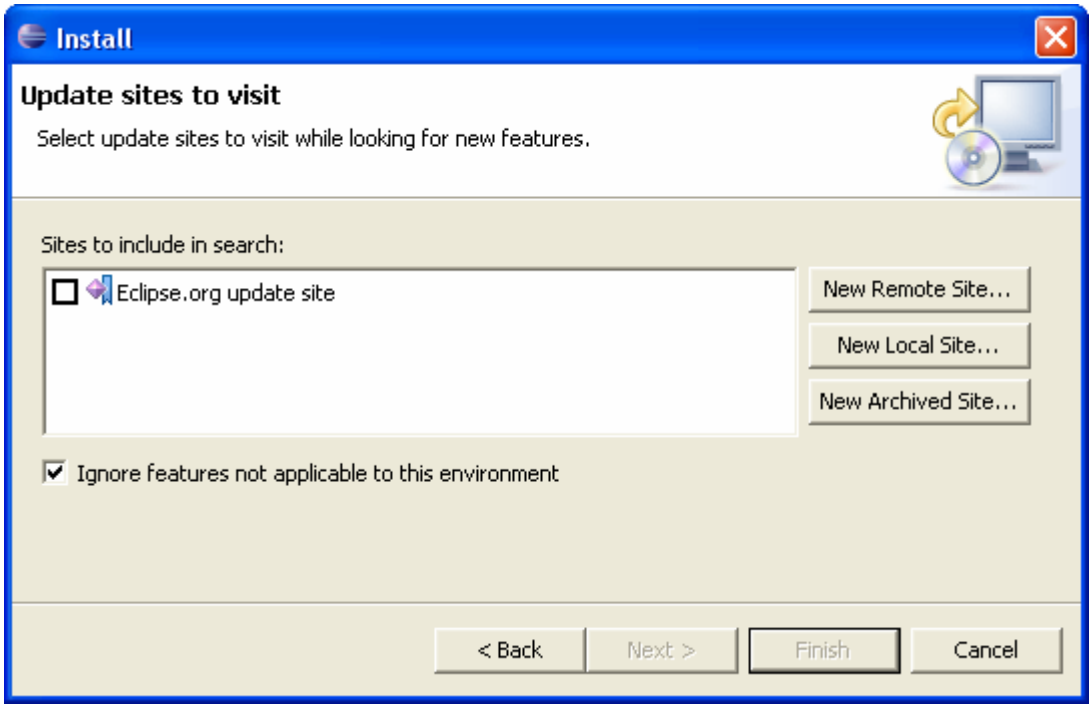

Figura 19 - Lista de sítios de atualização cadastrados

- Aparecerá uma janela *New Update Site* que deve ser preenchida de acordo com a Figura 20.

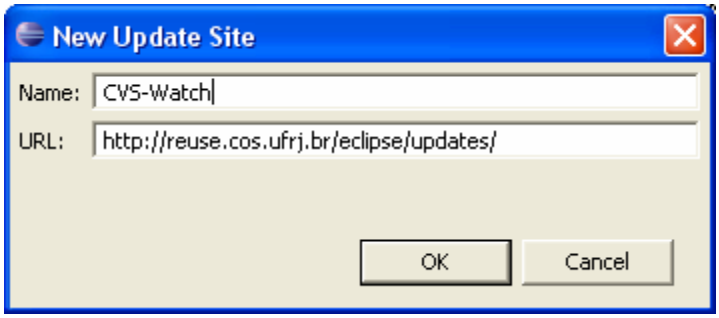

Figura 20 - Configuração do sítio de atualização

- Na lista de sítios, agora, aparecerá o do *CVS-Watch* já marcado (Figura 21). Clicar no botão *Finish*.

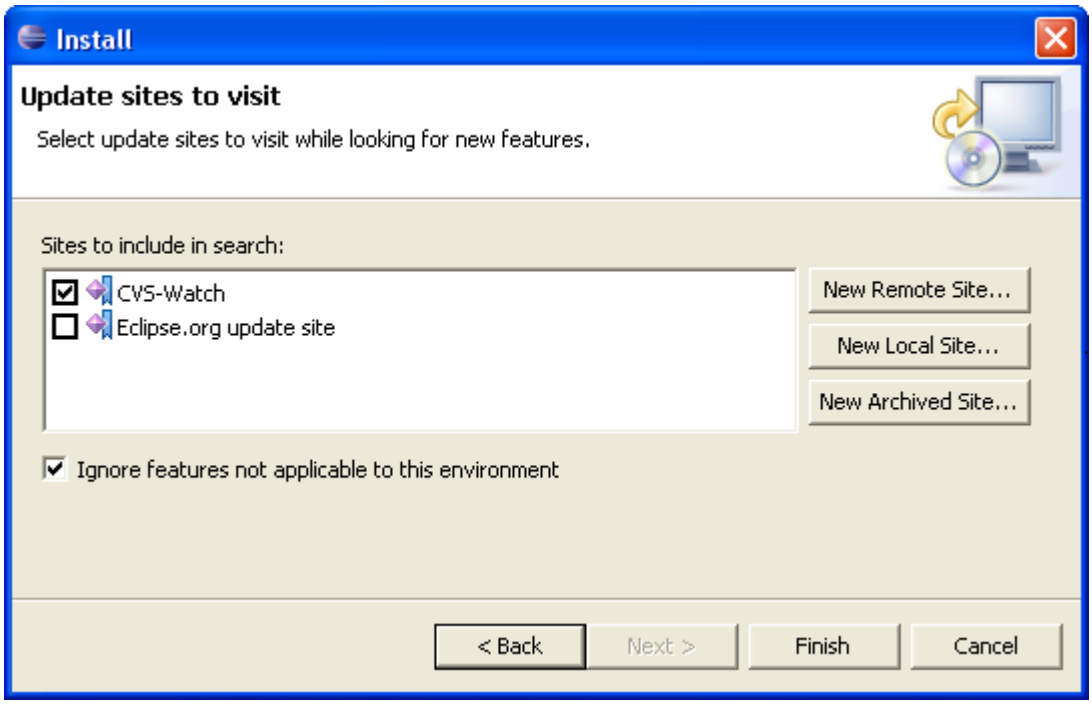

Figura 21 - Sítio do *CVS-Watch* na lista de sítios

- Aparecerá a janela *Updates*, mostrando o módulo *CVS-Watch* como disponível para instalação (Figura 22). Marcar a opção *CVS Watch 6.0.0* ou versão mais recente.

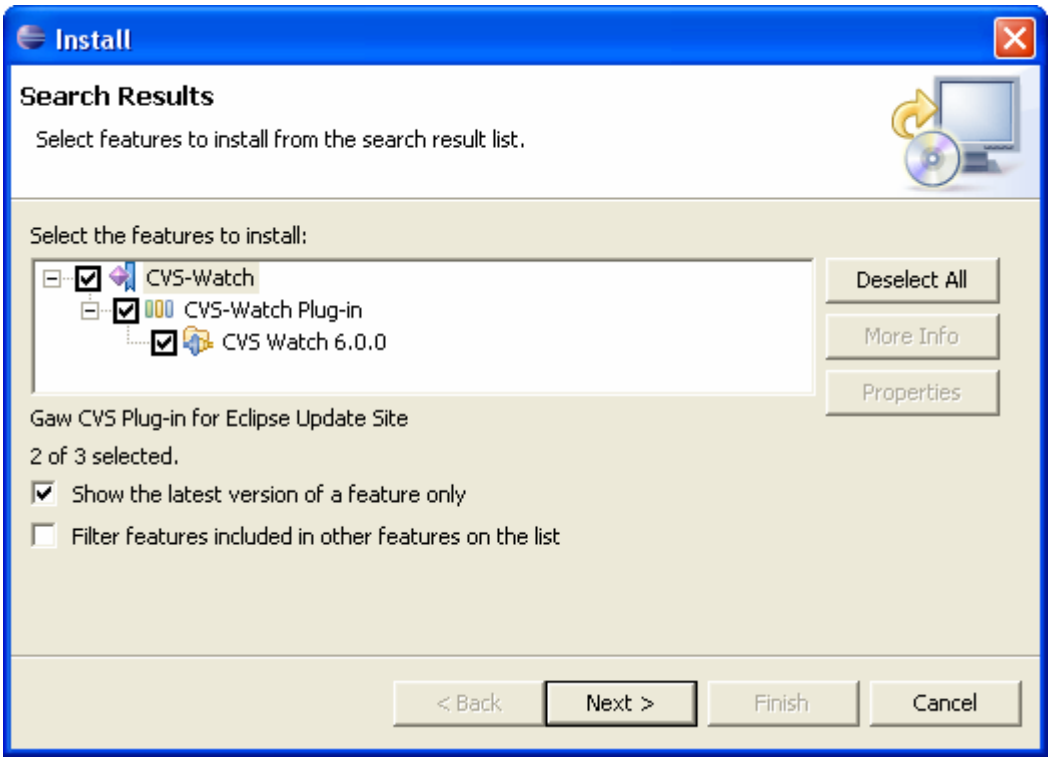

Figura 22 - Tela *Updates* com a lista de módulos disponíveis para instalação

- Aparecerá, então, uma janela com a licença de uso do módulo (Figura 23). Marcar a opção *I accept the terms in the license agreement* para aceitá-la e clicar no botão *Next*.

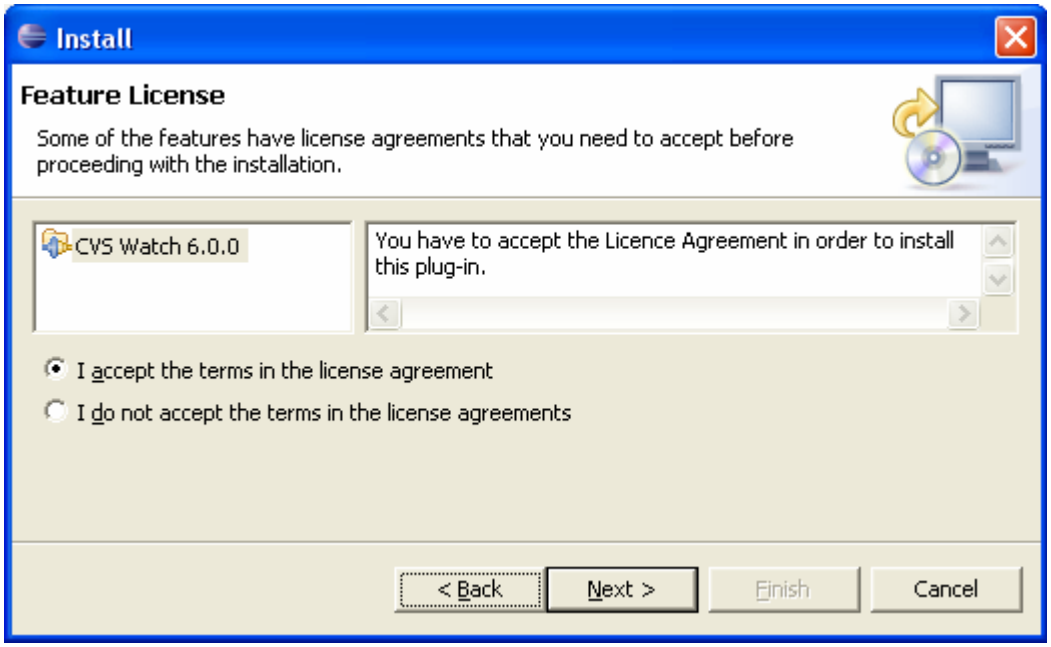

Figura 23 - Licença de uso do módulo *CVS-Watch*

- A janela de definição do local de instalação é mostrada (Figura 24). O local padrão de instalação é o diretório do Eclipse, caso se deseje alterá-lo, clicar no botão *Change Location...* e escolher o local desejado. É recomendado manter o local padrão, no entanto.

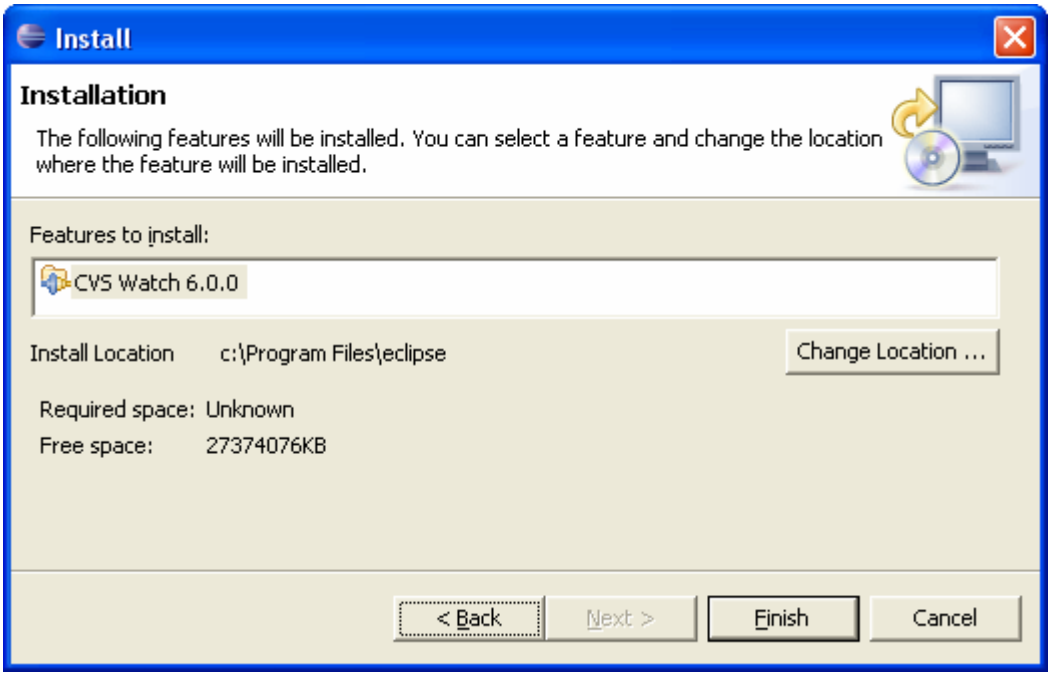

Figura 24 - Definição do local de instalação do módulo

- Em seguida, o módulo será baixado pelo Eclipse e será requisitada a confirmação da instalação do *CVS-Watch* (Figura 25).

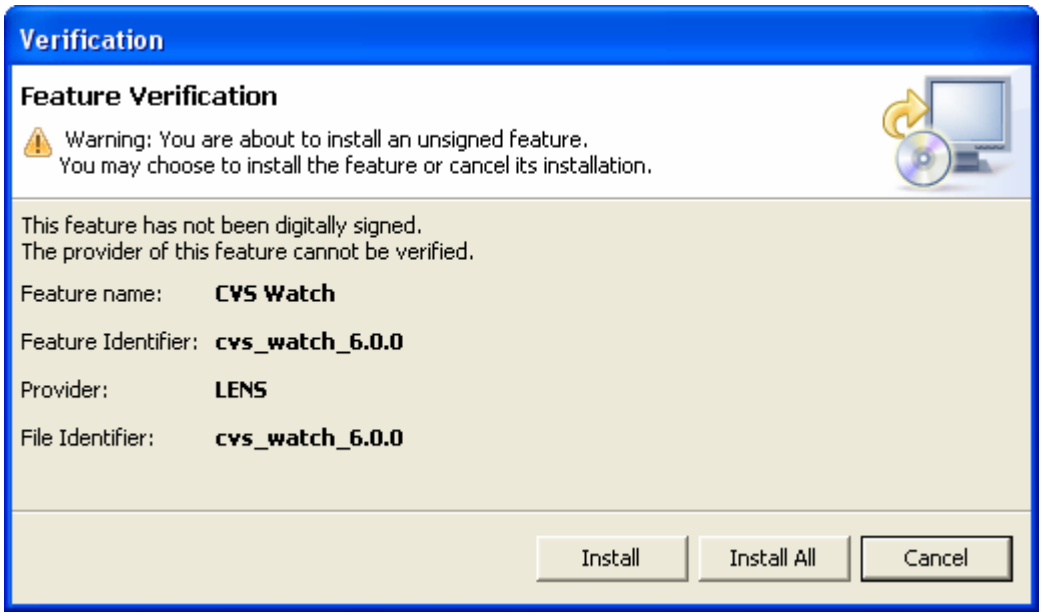

Figura 25 - Confirmação de instalação do módulo *CVS-Watch*

- O módulo será instalado e o usuário será requisitado a reiniciar o Eclipse (Figura 26). Para o bom funcionamento do módulo, é recomendável que se reinicie. Para isso, basta confirmar clicando no botão *Yes*.

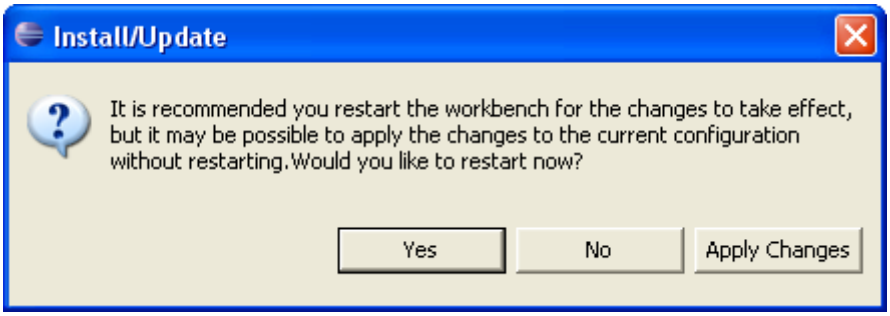

Figura 26 - Confirmação para reiniciar o Eclipse

APÊNDICE B
INSTALAÇÃO DO *WORKRHYTHM* NO *ODYSSEY*

Como dito anteriormente, o *Odyssey* permite a carga dinâmica de módulos de expansão. O ambiente baixa e instala os módulos sem a necessidade de reinicialização. Em seguida, serão listados os passos necessários para carregar dinamicamente o *WorkRhythm* no *Odyssey* (versões 1.x):

- Acionar o menu *Preferences* ▸ *Environment* no ambiente de trabalho do *Odyssey* (Figura 27).

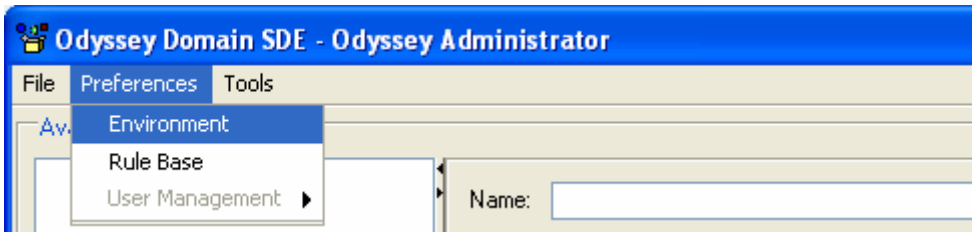

Figura 27 - Menu do *Odyssey* de instalação de novos módulos

- Aparecerá a janela *Model Environment* (Figura 28) com a lista de módulos disponíveis à esquerda. Clicar com o botão direito sobre *gaw-1.1.0.jar* ou versão mais recente e escolher o item *Install Component*.

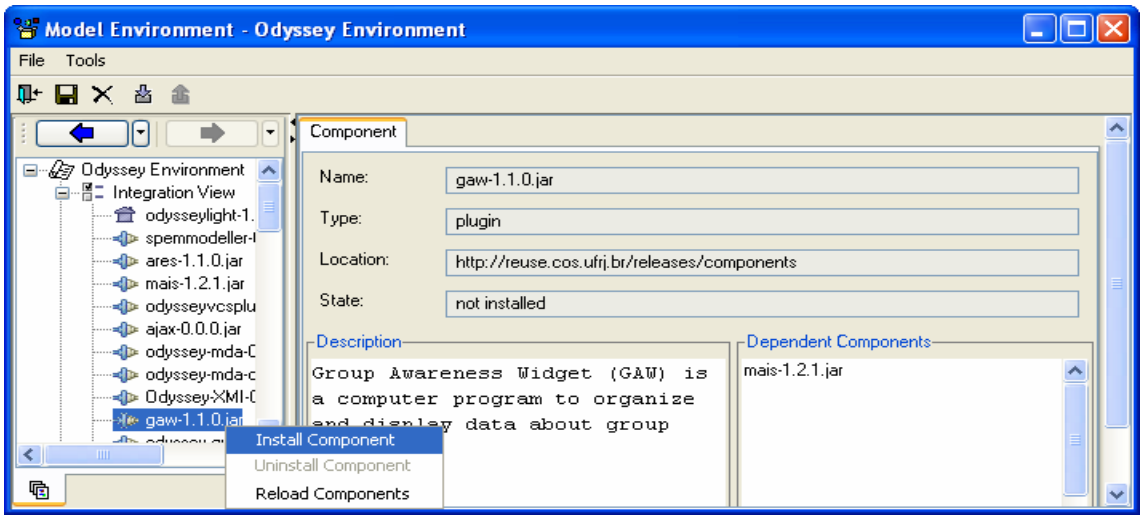

Figura 28 - Janela *Model Environment* com a lista de módulos

- Aparecerá a janela *Component Integration Wizard* (Figura 29) com uma lista que contém, tanto o módulo do *WorkRhythm*, quanto os módulos dos quais este depende, incluindo o MAIS e suas bibliotecas. Clicar em *Next* para prosseguir com a instalação.

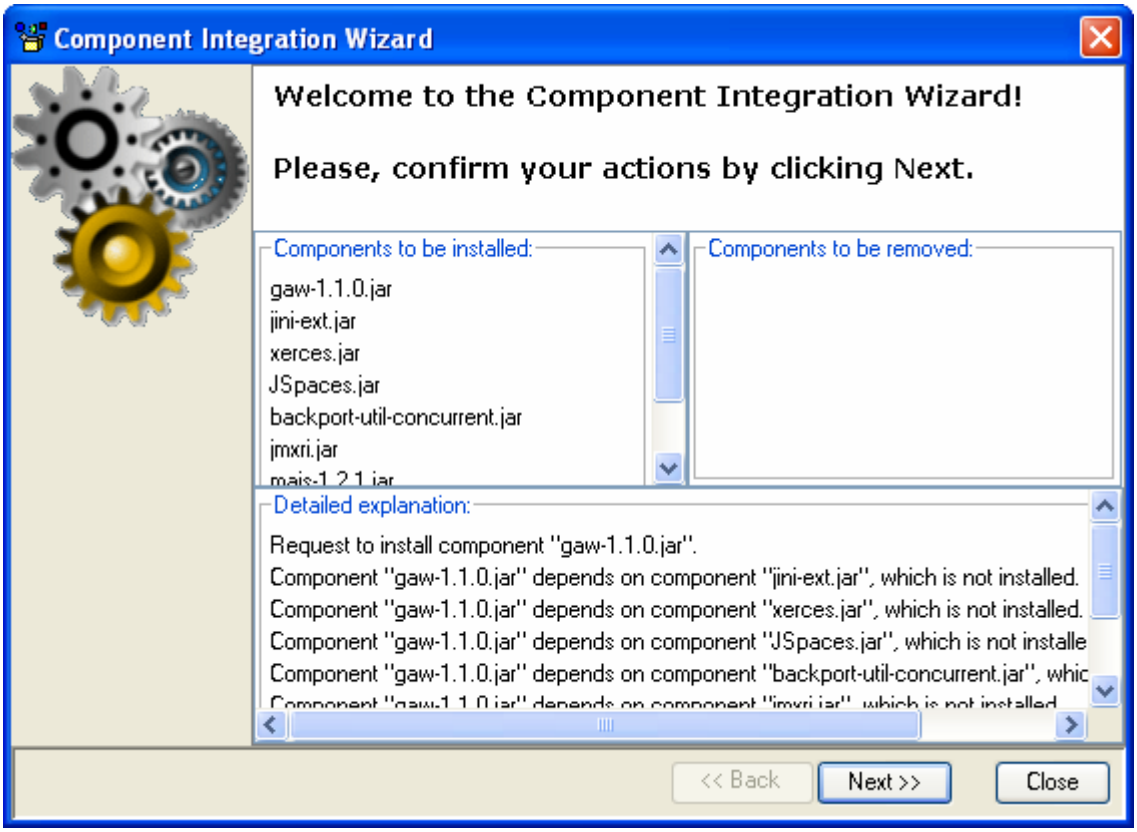

Figura 29 - Janela *Component Integration Wizard* com os módulos a serem instalados

- Os módulos serão, então, baixados e instalado pelo *Odyssey* (Figura 30). Ao final da instalação, clicar em *Finish* para concluir a instalação. Não é necessário reiniciar o ambiente para utilizar os módulos instalados.

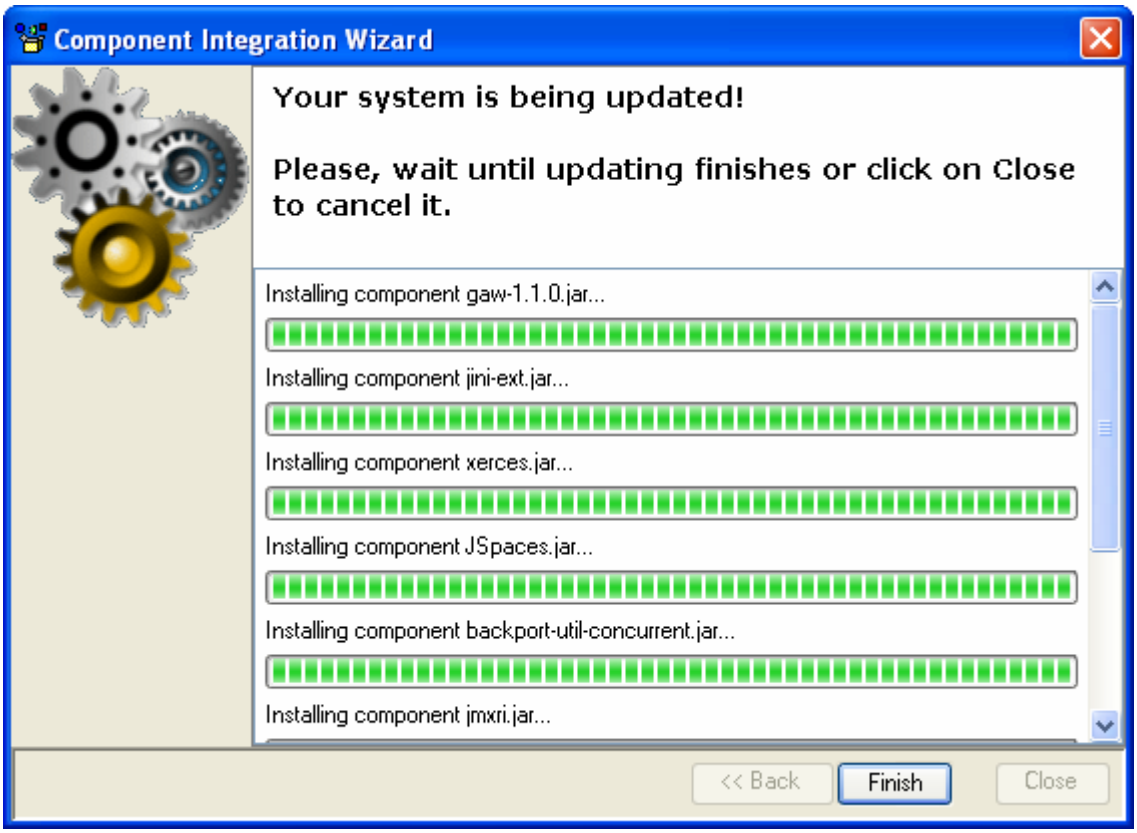

Figura 30 - Instalação dos módulos do *WorkRhythm* e suas dependências