MIMIX: SISTEMA DE APOIO À MODELAGEM COOPERATIVA DE SOFTWARE UTILIZANDO FERRAMENTAS CASE HETEROGÊNEAS

Daniele El-Jaick Bentes de Souza

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

Aprovada por:

Profª. Cláudia Maria Lima Werner, D.Sc.

Profª. Marta Lima Queirós Mattoso, D.Sc.

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

Profª. Neide dos Santos, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

JUNHO DE 2004

SOUZA, DANIELE EL-JAICK BENTES DE

MIMIX: Sistema de Apoio à Modelagem Cooperativa de Software Utilizando Ferramentas CASE Heterogêneas [Rio de Janeiro] 2004

IX, 158 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2004)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Trabalho Cooperativo Apoiado por Computador
2. Desenvolvimento de Software
3. Ferramentas CASE
4. Percepção

I. COPPE/UFRJ II. Título (série)

"Jamais considere seus estudos como uma obrigação, mas como uma oportunidade invejável para aprender a conhecer a influência libertadora da beleza do reino do espírito, para seu próprio prazer pessoal e para proveito da comunidade à qual seu futuro trabalho pertencer."

(Albert Einstein)

AGRADECIMENTOS

Com certeza, os conhecimentos adquiridos no mestrado influenciaram profundamente a minha vida. Além do conhecimento técnico, que contribuiu muito para a minha formação profissional, foram marcantes as experiências adquiridas ao longo do curso, que contribuíram significativamente para a minha formação pessoal. Várias pessoas estiveram comigo e me ajudaram durante esses últimos anos.

Queria agradecer à minha **mãe Isis**, que sempre me apoiou em tudo na minha vida. Minha mãe é a mulher mais inteligente e corajosa que eu já conheci. Ela sempre me incentivou a estudar, e conseguiu me mostrar que o estudo pode ser muito prazeroso e não uma obrigação.

Ao meu **marido José Luiz**, pelas idéias, pela parceria, pelo apoio, enfim, por toda a contribuição que deu ao meu trabalho. Ao meu **filho Lucas**, por todo o seu carinho e torcida. Queria agradecer toda a paciência que eles tiveram comigo, mesmo quando nem eu mesma conseguia me suportar. Prometo que agora eu vou ficar mais calminha.

Agradeço às minhas orientadoras, **Profª Marta Mattoso** e **Profª Cláudia Werner**, pelos conhecimentos transmitidos, pelas palavras de incentivo e pela paciência.

Agradeço ao meu co-co-orientador **Mangan**, por sempre ter estado presente ajudando no que fosse preciso. Principalmente, pelas idéias que colaboraram muito para esta dissertação.

Ao **Profº Marcos Borges** e à **Profª Neide dos Santos**, por aceitarem fazer parte dessa banca, mesmo com tantos compromissos.

Aos **amigos da turma de calouros BD 2001**, pelo convívio agradável e divertido durante as diciplinas, pelos trabalhos em equipe, pela amizade e pelo carinho. Um beijo especial para **Vaninha**, **Márcia**, **Manuel**, **Jonice**, **Rafael**, **Rosana** e **Pablo**. Valeu!!!

Enfim, agradeço a todos aqueles que direta, ou indiretamente, contribuíram para a realização deste trabalho, vibraram, torceram, me agüentaram e me apoiaram.

Muitos beijos !!!

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

MIMIX: SISTEMA DE APOIO À MODELAGEM COOPERATIVA DE SOFTWARE UTILIZANDO FERRAMENTAS CASE HETEROGÊNEAS

Daniele El-Jaick Bentes de Souza

Junho/2004

Orientadores: Profª Marta Lima de Queirós Mattoso e
Profª Cláudia Maria Lima Werner

Programa: Engenharia de Sistemas e Computação

O desenvolvimento de software não é uma tarefa simples. O processo de desenvolvimento demanda cooperação intensiva e um grande número de interações entre os membros da equipe de trabalho. Esta dissertação destaca a importância do apoio de ferramentas automatizadas no desenvolvimento cooperativo de software para coordenar as atividades cooperativas, manter o controle do projeto e das informações compartilhadas, com ênfase no suporte à atividade de modelagem de software. As ferramentas *CASE* (*Computer Aided Software Engineering*) tem sido amplamente utilizadas para apoiar essa atividade, mas, em geral, não suportam a cooperação. O apoio à cooperação entre ferramentas CASE diferentes, ainda, é um tópico em aberto. Esta dissertação propõe um sistema, chamado MIMIX, que tem como objetivo fornecer suporte à atividade de modelagem cooperativa de software. O MIMIX disponibiliza um conjunto de serviços para apoiar os membros de uma equipe de desenvolvimento, que trabalham cooperativamente, de forma assíncrona, na construção de modelos UML. Os membros podem estar em estações de trabalho distribuídas geograficamente e podem estar utilizando ferramentas CASE heterogêneas. Essas ferramentas podem, inclusive, funcionar sobre plataformas de hardware e software diferentes. Quando os membros da equipe trabalham distribuídos geograficamente, não têm a consciência do trabalho que está sendo realizado pelos demais membros. O MIMIX se propõe a fornecer informações de percepção do trabalho da equipe e individual de cada membro, entretanto, seu principal objetivo é fornecer informações de percepção sobre o produto que está desenvolvido pela equipe. Para isso, o MIMIX armazena os modelos gerados pela equipe de trabalho e suas versões, permitindo a comparação e a união de versões de um modelo. Um protótipo do MIMIX foi desenvolvido para analisar a viabilidade de implementação da proposta. Nesse protótipo, os serviços do MIMIX são implementados na forma de *serviços WEB*, para alcançar a independência de plataforma pretendida. O MIMIX utiliza o padrão *XMI* (*XML Metadata Interchange*) como formato de intercâmbio de dados. Atualmente, esse padrão tem sido bastante utilizado pelas ferramentas *CASE*, para importação e exportação dos documentos gerados pelas mesmas.

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

MIMIX: SYSTEM FOR COOPERATIVE SOFTWARE MODELING SUPPORT USING HETEROGENEOUS CASE TOOLS

Daniele El-Jaick Bentes de Souza

June/2004

Advisors: Profª Marta Lima de Queirós Mattoso and
Profª Cláudia Maria Lima Werner

Department: Computer Science and Systems Engineering

Software development is not a simple task. The development process demands intensive cooperation and a large number of iterations among team members. This paper highlights the importance of the support given by automated development tools to the cooperative development of software, aiming at cooperative activities coordination as well as project and shared information control, emphasizing software modeling activities support. CASE (Computer Aided Software Engineering) tools have been widely used to assist his activity, but they do not generally support cooperation. Cooperation support between different CASE tools is still an issue. This paper will propose a system called MIMIX, which intends to support cooperative software modeling activities. MIMIX makes available a number of services that help development team members create UML diagrams while working cooperatively and asynchronously with CASE tools. Team members may be in geographically dispersed workstations and use different CASE tools, which may also run in different hardware and software platforms. Geographically dispersed team members may not be aware of the work that is being done by one another. MIMIX will provide awareness information into team and individual member's work. However, its main objective is to provide awareness about the product developed by the team. To this end, MIMIX stores the various versions of the models generated by team workers, allowing comparison and union of a model's versions. A prototype has been developed to analyze the viability of the proposal implementation. In this prototype, MIMIX services are implemented as WEB services in order to reach the intended platform independency. MIMIX uses the XMI (XML Metadata Interchange) standard for its data exchange format. This standard has been being widely used by CASE tools for importing and exporting the documents generated by them.

ÍNDICE

I Introdução

A cooperação entre os membros de uma organização é essencial para qualquer negócio. Isso devido, por exemplo, à necessidade de reduzir custos, acelerar a tomada de decisões, evitar desperdícios e trocar conhecimento. Ou seja, as pessoas precisam se organizar em grupos para compartilhar informações, debater e resolver problemas. Muitas vezes, os membros de um grupo encontram-se dispersos geograficamente e precisam se comunicar de alguma forma, seja através de telefone, e-mail, ou fax (TAYLOR, 2001).

Os sistemas cooperativos são ambientes potencialmente "caóticos". As interações entre seus múltiplos usuários e as aplicações ocorrem de forma pouco previsível e dinâmica. Esse dinamismo presente em sistemas cooperativos se aproxima, intencionalmente, da forma como as pessoas interagem no "mundo físico" (EDWARDS, 1996).

A necessidade de sistemas que suportem formas de cooperação em grupo motivou, na última década, várias pesquisas para o estudo do Trabalho Cooperativo Suportado por Computador (CSCW – *Computer Supported Cooperative Work*). CSCW é o nome dado à área de pesquisa que estuda como grupos e pessoas trabalham, e tenta achar uma maneira de fornecer suporte tecnológico para o processo de trabalho. Seu objetivo é tentar sustentar todas as atividades que as pessoas podem executar usando computadores, independente da natureza do trabalho. Permitindo, assim, que várias pessoas distribuídas geograficamente trabalhem cooperativamente, por exemplo, compartilhando arquivos e figuras, escrevendo documentos ou navegando na Internet (MARIANI e RODDEN, 1991).

O desenvolvimento de software de qualidade não é uma tarefa simples, demanda cooperação intensiva e um grande número de interações entre os membros da equipe de trabalho envolvida no processo. Para coordenar as atividades cooperativas, mantendo o controle do projeto e da informação compartilhada, é necessário o suporte de ferramentas automatizadas apropriadas.

O emprego das ferramentas CASE (*Computer Aided Software Engineering*) acompanhou, de forma geral, o desenvolvimento de software e das metodologias utilizadas

para representar os problemas do mundo real. Contudo, trabalhar no desenvolvimento de software implica na utilização de várias ferramentas CASE diferentes. Isso porque, na prática, uma única ferramenta não consegue fornecer suporte a todas as atividades realizadas durante este processo. Normalmente, uma ferramenta é feita sob medida para realizar determinado tipo de trabalho (VIEIRA et al, 2001).

Atualmente, o processo de desenvolvimento de software se tornou ainda mais complexo. Além do aumento no tamanho dos projetos, as equipes de trabalho são cada vez mais especializadas (heterogêneas) e seus membros encontram-se, normalmente, distribuídos através do tempo (trabalho assíncrono) e do espaço geográfico. Cada membro da equipe pode trabalhar em plataformas de desenvolvimento variadas e com ferramentas diferentes.

I.1 Motivação e Objetivo

As pesquisas relacionadas ao suporte do desenvolvimento cooperativo de software, são impulsionadas pelos estudos nas áreas de CSCW e CASE. A interseção entre essas duas áreas oferece um extenso campo de pesquisa, que visa aplicar as técnicas de CSCW para aprimorar os aspectos de cooperação nas ferramentas CASE.

Atualmente, as ferramentas CASE disponíveis no mercado possuem pouco, ou nenhum suporte à cooperação. O apoio ao desenvolvimento cooperativo de software, onde os membros de uma equipe utilizam ferramentas CASE distintas, ainda é um tópico em aberto. Os estudos nessa área têm se concentrado em estabelecer um formato padrão de intercâmbio de dados entre ferramentas distintas. Entretanto, isso não é suficiente para apoiar o desenvolvimento cooperativo de software.

Durante o desenvolvimento cooperativo, o foco está no produto que está sendo desenvolvido. Esse produto é construído de forma incremental e evolutiva pelos membros da equipe de desenvolvimento. Assim, é preciso que os membros tenham informações sobre o trabalho individual e geral da equipe, para que o produto final atenda os objetivos desejados. Para o sucesso do trabalho realizado, também é preciso que os membros possam

observar, passo a passo, o desenvolvimento do produto. Essas informações são úteis para que seja possível, por exemplo, detectar e corrigir possíveis erros introduzidos durante o desenvolvimento. Muitas vezes, é preciso comparar e/ou fazer a união de trabalhos paralelos de membros distintos para a construção do produto final.

Em ambientes onde a cooperação face-a-face é freqüente são oferecidos, naturalmente, diversos canais de comunicação pelo ambiente físico de trabalho compartilhado. Os membros são capazes de perceber o ritmo de trabalho dos demais, encontrar interesses em comum e realizar tarefas em pequenos grupos, de forma natural e espontânea. Entretanto, quando os membros da equipe de desenvolvimento se encontram distribuídos geograficamente, a comunicação entre eles é prejudicada. Utilizar ferramentas de vídeo conferência ou de correio eletrônico não resolve o problema. Isso porque, essas ferramentas permitem uma comunicação isolada, onde os membros da equipe não possuem visibilidade das informações trocadas pelos demais membros. Além disso, não existe o registro da comunicação realizada entre os membros da equipe. A comunicação realizada entre os membros constitui uma base de informações importante sobre o desenvolvimento do trabalho. Assim, é preciso manter um histórico dessas informações e torná-las disponíveis para a equipe.

Esta dissertação aborda a ligação entre as áreas de CSCW e CASE. O cenário típico aqui tratado é caracterizado pelos membros de uma equipe trabalhando cooperativamente na modelagem de software, distribuídos geograficamente, cada um com sua ferramenta CASE, em horários diferentes, realizando uma mesma tarefa ou tarefas distintas. Os membros da equipe possuem papéis diferentes, que restringem seus acessos às informações do projeto. Para que seja possível compartilhar o conhecimento comum, trocar informações, debater e discutir questões, a fim de resolver possíveis conflitos, é necessário que os membros possam se comunicar. A dinâmica do trabalho realizado deve ser registrada, para que seja possível, por exemplo, desfazer determinado trabalho em caso de erro. Os documentos gerados pela equipe precisam ser armazenados e sua integridade deve ser mantida. Isso porque, os membros da equipe, freqüentemente, precisam consultar, comparar e/ou fazer a união desses documentos, gerando novos documentos. Durante o trabalho de modelagem pode surgir, em qualquer fase do processo, a necessidade de fazer

alterações nos modelos gerados pela equipe de desenvolvimento. Essas alterações precisam ser controladas e os membros da equipe precisam ter conhecimento sobre elas, a fim de não gerar problemas para a equipe, como, por exemplo, precisar refazer parte do trabalho e introduzir possíveis erros nos modelos. Este cenário apresenta vários problemas. Dentre eles, os que estão relacionados com esta dissertação, são:

a) **Formato de Dados:** Embora as ferramentas CASE tenham se desenvolvido significativamente nos últimos tempos, permitir a troca de informações entre ferramentas distintas ainda é um trabalho árduo. Geralmente, ferramentas de fabricantes diferentes possuem interfaces fechadas e formatos de armazenamento de dados proprietários. Por isso, o intercâmbio de informações entre elas depende da utilização de tradutores específicos para cada ferramenta. Ainda que esse formato de armazenamento seja conhecido, seria inviável fazer conversões entre os diversos formatos das ferramentas CASE disponíveis.

b) **Persistência das Informações:** É preciso garantir a persistência e integridade das informações trocadas, dos documentos gerados, e de suas versões. Essa base de dados precisa ser compartilhada pelos membros da equipe de acordo com as permissões de acesso de cada um. Além disso, é preciso disponibilizar mecanismos de busca, comparação e união para as informações armazenadas. Utilizar um Sistema Gerenciador de Banco de Dados (SGBD) tradicional não resolve o problema, pois esses foram projetados para a contemplar o trabalho individual e não cooperativo. O modelo de transações convencional (ACID) dos SGBDs é considerado muito restritivo no contexto de cooperação devido, principalmente, às propriedades de atomicidade e isolamento. Assim, as aplicações cooperativas possuem algumas características particulares, que os SGBDs não conseguem atender, como, por exemplo, suporte a transações longas, compartilhamento dos resultados intermediários das transações enquanto ainda estão em curso, alteração de papéis (controle de acesso) durante o trabalho e detenção de múltiplos papéis pelos membros de uma equipe de

desenvolvimento. Durante o estudo realizado para o desenvolvimento desta dissertação, foi elaborado um trabalho que identificou os problemas de se utilizar um SGBD tradicional para realizar essas tarefas (EL-JAICK et al., 2002). O capítulo II, item II.3.2, apresenta um resumo desse trabalho.

c) **Gerência de Configuração:** Configuração de software é o nome dado a toda informação produzida durante o processo de desenvolvimento de software. Essa informação é constituída por objetos de configuração de software (programas de computador, documentos gerados e dados). É preciso identificar, organizar e controlar as modificações realizadas pelos membros da equipe de desenvolvimento nesses objetos ao longo do processo. A maioria dos sistemas de gerência de configuração[1] utiliza o mecanismo de versionamento para controlar essas modificações, permitindo a comparação e união de versões intermediárias dos membros da equipe. Atualmente, existem vários sistemas de gerência de configuração, que contemplam, em geral, a edição cooperativa de documentos texto e de código fonte de programas. Esses sistemas identificam as diferenças e fazem a união de versões de um documento, baseados na comparação de cadeias de caracteres de arquivos texto. No estudo realizado para a elaboração desta dissertação não foi encontrado um sistema de gerência de configuração, que suporte a edição cooperativa de modelos. Durante a modelagem cooperativa os membros da equipe de desenvolvimento precisam analisar e comparar as várias versões dos modelos criados durante o trabalho. Muitas vezes, pode surgir a necessidade de fazer a união de trabalhos paralelos dos membros da equipe. Entretanto, para que a comparação e a união sejam feitas de forma

[1] A Gerência de Configuração de Software (GCS) pode ser definida como uma disciplina que identifica a configuração de um sistema em pontos discretos no tempo, com o objetivo de controlar sistematicamente as mudanças dessas configurações e manter a integridade e a rastreabilidade do ciclo de vida do sistema (LEON, 2000).

adequada é preciso interpretar e analisar o conteúdo de cada versão, para evitar a comparação de elementos diferentes do modelo, como, por exemplo, comparar uma classe chamada "Aluno", com um atributo de outra classe que contenha o mesmo nome (Aluno), e inferir que se trata do mesmo elemento.

d) **Percepção:** Durante a modelagem cooperativa de software cada membro precisa ter informações sobre o trabalho da equipe e sobre o andamento das atividades individuais, para poder situar seu trabalho no contexto geral do processo. Quando membros distintos da equipe trabalham em um mesmo modelo cada membro precisa, por exemplo, ter informações sobre o andamento do trabalho dos demais, poder comparar sua versão do modelo com uma versão de outro membro e ser avisado das alterações feitas em uma versão do modelo feita por outro membro. A falta dessas informações pode gerar vários problemas, como, por exemplo, inconsistências, redundâncias e contradições no trabalho da equipe. Como, normalmente, os membros trabalham distribuídos geograficamente, a visibilidade dessas informações é prejudicada. Atualmente, as ferramentas CASE disponíveis não fornecem essas informações de percepção. Existem aplicações, chamadas de servidores de notificação, que tem como objetivo intermediar o trabalho dos membros de uma equipe fornecendo informações sobre as modificações realizadas durante o trabalho. Um membro da equipe interage com o servidor de notificação através de uma aplicação cliente, que pode ser qualquer ferramenta utilizada para desenvolvimento, como, por exemplo, uma ferramenta CASE. Esse tipo de aplicação não armazena as versões dos modelos criados, somente informações sobre as alterações realizadas. Outro problema desse tipo de aplicação é que, geralmente, não existe filtro de avisos. Assim, todos os membros recebem todas as notificações de alteração, o que pode acarretar uma sobrecarga de informações para os membros da equipe.

e) **Comunicação:** É necessário disponibilizar mecanismos que permitam a comunicação entre os membros da equipe, inclusive se eles não estiverem conectados em determinado momento (*off-line*). Além disso, é preciso permitir uma forma de comunicação que seja independente da plataforma de software e de hardware utilizada pelos membros da equipe. Utilizar ferramentas para a comunicação como correio eletrônico, vídeo conferência ou salas de bate-papo, não resolve o problema. Esse tipo de comunicação é isolado e não fica registrado como parte do trabalho da equipe. Assim, um membro da equipe não tem visibilidade das mensagens enviadas e recebidas pelos demais membros. No contexto da cooperação, as informações sobre a comunicação realizada entre os membros da equipe são muito importantes para o andamento do trabalho. É preciso disponibilizar mecanismos para a comunicação da equipe, procurando minimizar os problemas gerados pela distância física e mantendo um histórico dessa comunicação. Esse histórico deve fazer parte do registro geral das ações dos membros de uma equipe e deve ficar disponível para ser consultado pelos mesmos.

O suporte adequado à modelagem cooperativa de software, que contemple os problemas relacionados, constitui um amplo campo de pesquisa e a motivação para esta dissertação. A dificuldade de fornecer esse suporte aumenta, quando todos os problemas descritos são tratados em conjunto. Os trabalhos existentes nessa área de pesquisa, em geral, procuram analisar, em profundidade, e solucionar um problema específico.

Esta dissertação entende que os problemas, descritos no item I.1, estão intimamente relacionados, principalmente em se tratando do domínio escolhido. Assim, seu objetivo é tratá-los em conjunto. Devido à complexidade do problema, foi necessário analisar e estabelecer um subconjunto de serviços relacionados com cada um dos problemas, definindo assim uma solução básica, mas abrangente.

Assim, esta dissertação apresenta o sistema MIMIX, que tem como principal objetivo fornecer suporte à modelagem cooperativa de software. O MIMIX apóia o trabalho dos membros de uma equipe de desenvolvimento, que trabalham cooperativamente na

modelagem de um software utilizando ferramentas CASE, baseadas em UML. Eles podem estar em estações de trabalho distribuídas geograficamente e podem estar utilizando ferramentas CASE heterogêneas. Essas ferramentas podem, inclusive, funcionar sobre plataformas de hardware e software diferentes.

O MIMIX disponibiliza um conjunto de serviços que podem ser utilizados pelos membros da equipe para trabalhar cooperativamente, cada um com a sua ferramenta CASE. Estes serviços tratam as questões de persistência, gerência de configuração (versão), comunicação e percepção, com ênfase em fornecer informações sobre o produto no qual uma equipe de desenvolvimento trabalha. Nesse caso, o produto é constituído pelos modelos UML gerados pela equipe. Os serviços contemplam, principalmente, a forma de trabalho assíncrona, mas podem ser estendidos para atender também a forma síncrona.

Os serviços disponibilizados pelo MIMIX têm, portanto, como principal objetivo permitir:

- O controle de acesso dos membros da equipe às informações do projeto;
- Troca de mensagens de texto entre os membros da equipe;
- Fornecer informações de percepção sobre o trabalho individual dos membros e sobre o trabalho coletivo para a equipe. Essas informações se referem, principalmente, à noção de presença dos membros no espaço de trabalho compartilhado, às mensagens trocadas entre os membros e às interações dos membros sobre os modelos compartilhados;
- Garantir a persistência dos modelos, e suas versões, gerados no processo de interação dos membros da equipe, permitindo sua busca e a recuperação;
- Gerenciar as alterações feitas nos modelos criados pela equipe, que podem ocorrer ao longo do processo de desenvolvimento, notificando os membros que têm a necessidade de conhecê-las e

- Permitir a comparação e a união (*merge*) de versões de modelos, inclusive as geradas por ferramentas distintas, com tratamento automático (sem intervenção do usuário) e semi-automático (com intervenção do usuário) de conflitos.

I.2 Proposta do MIMIX

Utilizando o MIMIX, o trabalho de modelagem cooperativa de software pode ser realizado de forma incremental e evolutiva. Os membros da equipe trabalham em paralelo com suas ferramentas CASE, gerando periodicamente documentos que contêm os modelos do sistema que está sendo desenvolvido. Quando um membro da equipe deseja disponibilizar um documento para os demais, utiliza um serviço do MIMIX para publicar o documento. O MIMIX armazena o documento em um repositório centralizado, que fica disponível para os demais membros da equipe. Esses documentos passam por constantes modificações ao longo do trabalho da equipe, onde cada modificação gera uma nova versão do documento. O MIMIX armazena todas as versões do documento geradas, sendo ele o responsável pelo controle dessas versões. O controle de versões é um mecanismo de gerência de configuração, que tem como objetivo: automatizar a busca de documentos, prevenir conflitos entre os membros da equipe, recuperar versões prévias, permitir o desenvolvimento paralelo, etc.

O foco deste trabalho está no produto (modelos) que está sendo desenvolvido de forma cooperativa. No desenvolvimento distribuído, geralmente, as mudanças ocorridas nos documentos armazenados em um repositório não são facilmente percebidas. Assim, o MIMIX disponibiliza um conjunto de informações sobre o produto (percepção do produto) para os membros, que os permita observar as modificações realizadas ao longo do trabalho. O MIMIX possui serviços para a busca, recuperação, comparação e união das versões armazenadas, comparando duas versões, por exemplo, é possível observar a evolução do trabalho e as modificações realizadas no produto que está sendo desenvolvido.

Além disso, o MIMIX mantém um registro das modificações realizadas durante o trabalho, para fornecer informações de percepção do produto. Cada vez que um

membro publica uma nova versão para a equipe, o MIMIX atualiza esse registro com informações sobre a versão, como, por exemplo, identificador da versão, usuário responsável e data da publicação. Assim, os membros podem utilizar o serviço do MIMIX que consulta esse registro para saber, por exemplo, qual versão introduziu um determinado erro, e com essa informação é possível retornar a uma versão anterior. Nesse mecanismo, as informações de percepção do produto são entregues no modo **PUXE**, onde os membros obtêm informações do servidor de percepção através de solicitações explícitas.

Os documentos gerados pelas ferramentas CASE, geralmente, possuem formatos proprietários. O primeiro passo para permitir o compartilhamento de informações entre elas é permitir que uma "entenda" o formato da outra. Não faz parte do escopo desta dissertação tratar as conversões entre os formatos de cada ferramenta CASE, pois seria inviável construir um conversor para cada par de formatos existentes no mercado. Foi necessário, então, buscar uma outra alternativa para resolver esse problema.

Na literatura, são encontrados diversos trabalhos que propõem formatos para intercâmbio de informações entre ferramentas CASE, entre eles, (OMG, 2002) e (SUZUKI e YAMAMOTO, 1998). A adoção de UML e MOF como padrões pela OMG (*Object Management Group*), em 1997, foi um passo importante na tentativa de unificar a tecnologia de modelagem e o armazenamento de metadados (OMG, 2002). Assim, a OMG desenvolveu o padrão XMI (*XML Metadata Interchange),* que está diretamente relacionado com os padrões MOF (OMG), UML (OMG) e XML (W3C). O XMI foi criado para permitir a troca de dados e metadados entre ferramentas em ambientes heterogêneos e distribuídos, garantindo a interoperabilidade e a troca de informações entre elas. O XMI permite representar objetos usando o XML (*Extensible Markup Language*), que é o formato universal para a representação de dados na Web. Ao invés de adotar formas de mapeamentos proprietárias, o XMI propõe uma forma padrão baseada em modelos (GROOSE et al, 2001) e (JIANG e SYSTÄ, 2002). Maiores detalhes sobre esse padrão são encontrados no Anexo 1.

De fato, ferramentas CASE bastante difundidas no mercado, como o Rational ROSE, o System Architect e o WithClass, aderiram ao padrão XMI. Esta dissertação segue esta tendência. O MIMIX, portanto, utiliza o XMI como formato para intercâmbio de

informações e não um formato proprietário. Para que isso seja possível, é pré-requisito para o MIMIX, que as ferramentas CASE possuam o mecanismo de importação e exportação dos modelos gerados em documentos XMI.

A base para a comunicação em aplicações distribuídas é, tipicamente, o mecanismo de RPC (*Remote Procedure Call*). Nesta dissertação, foram analisadas quatro alternativas que se baseiam no mecanismo de RPC, são elas: CORBA, RMI, DCOM e serviços Web. Como permitir a comunicação entre plataformas diferentes é um requisito importante para esta dissertação, a alternativa escolhida foi a de utilizar a tecnologia de serviços Web. No anexo 2, são apresentados maiores detalhes sobre os serviços Web, além de um estudo destas alternativas, destacando as vantagens e desvantagens de cada uma.

As funcionalidades do MIMIX são disponibilizadas na forma de serviços Web, o que aumenta a interoperabilidade e facilita a integração entre as ferramentas envolvidas no trabalho. É possível encapsular as chamadas a esses serviços dentro das ferramentas CASE, ou ainda, utilizar os serviços através de um módulo cliente, independente da linguagem de programação (linguagem neutra) e plataforma (plataforma neutra) em que os serviços foram desenvolvidos. Outra vantagem é poder utilizar os serviços disponibilizados pelo MIMIX de modo independente. Assim, também é possível expandir as funcionalidades do MIMIX, criando novos serviços de acordo com as necessidades da equipe de trabalho.

I.3 Organização dos capítulos

O restante desta dissertação está dividido nos seguintes capítulos:

O **Capítulo 2** apresenta uma revisão e uma análise sobre trabalho cooperativo e sobre o desenvolvimento cooperativo de software, mostrando seus principais conceitos e requisitos. O objetivo desse levantamento é identificar as principais funcionalidades que o MIMIX deve disponibilizar, para apoiar a cooperação durante a modelagem de software, que é uma das atividades do desenvolvimento. Esse capítulo apresenta, também, os principais trabalhos relacionados a esta dissertação.

O **Capítulo 3** apresenta o MIMIX. Inicialmente, é apresentada uma visão geral do MIMIX, caracterizando a solução proposta, seus principais objetivos e restrições. Em seguida, é apresentado um exemplo do funcionamento do MIMIX. Por fim, é descrita sua arquitetura e feita uma análise da solução.

Para analisar a viabilidade de implementação da proposta e das escolhas tecnológicas, foi desenvolvido um protótipo do MIMIX. O **Capítulo 4** descreve a implementação desse protótipo, detalhando os serviços disponibilizados. Para exemplificar a utilização do protótipo, esse capítulo apresenta um estudo de caso de uso, utilizando as ferramentas CASE Rational Rose e Ideogramic UML.

O **Capítulo 5** encerra esta dissertação relacionando as principais contribuições do MIMIX, suas limitações, problemas da proposta e as possíveis melhorias, que podem ser realizadas, em trabalhos futuros.

II Desenvolvimento Cooperativo de Software

O desenvolvimento de software é uma atividade que necessita de intensa cooperação entre as pessoas envolvidas no trabalho. Atualmente, com o aumento da complexidade e do porte dos projetos de software, o número de pessoas envolvidas aumentou proporcionalmente. À medida que o número de pessoas numa equipe de projeto de software aumenta, a produtividade global do grupo pode ser impactada. A solução para esse problema é dividir o projeto em partes, que serão desenvolvidas separadamente, e dividir as pessoas em equipes especializadas, que irão trabalhar em determinada parte do projeto (PRESSMAN, 2002).

Entretanto, as partes de um projeto não são independentes umas das outras. Cada equipe de trabalho precisa ter informações sobre o trabalho das outras equipes para coordenar suas atividades. O problema é que as equipes de desenvolvimento encontram-se, normalmente, distribuídas através do tempo e do espaço geográfico. E, ainda, os membros de uma mesma equipe também podem trabalhar em lugares e horários diferentes (FARSHCHIAN, 2001).

Neste cenário potencialmente caótico, surge a necessidade de soluções que ajudem a organizar e coordenar o trabalho destas equipes, procurando minimizar os problemas de trabalhar sem a proximidade física entre as pessoas. Esta proximidade física, aumenta a freqüência e a qualidade da comunicação entre as pessoas. Uma equipe que trabalha em um mesmo escritório, por exemplo, tem a vantagem de seus membros estarem sempre cientes sobre o estado do produto que está sendo construído. Estas informações podem, inclusive, ser passadas entre os membros através da comunicação informal, como quando as pessoas se encontram no corredor, ou se reúnem para tomar um café (FARSHCHIAN, 2001).

A pesquisa no desenvolvimento cooperativo de software é impulsionada por duas áreas de trabalho (MANGAN, 2003):

- **CSCW:** A sigla CSCW (*Computer Suport Cooperative Work*) se refere à área de pesquisa que estuda como grupos e pessoas trabalham, e tenta achar uma maneira de fornecer suporte tecnológico para o processo de trabalho. A partir das

pesquisas sobre CSCW, novas aplicações foram desenvolvidas para apoiar o trabalho cooperativo, chamadas aplicações cooperativas ou sistemas de *groupware*. Como exemplo, temos: Sistemas de Mensagens, Sistemas de "Bate-papo", Salas de Conferência e Sistemas de Co-autoria e de "Argumentação" (ELLIS e GIBBS, 1989).

- **CASE:** A sigla CASE (*Computer-Aided Software Engineering*) se refere a ferramentas de engenharia de software apoiadas por computador, que assistem toda a atividade associada com o processo de software. Automatizam as atividades de gestão de projetos, gerenciam todos os produtos de trabalho produzidos ao longo do processo e assistem o trabalho de análise, projeto, codificação e teste, ou seja, existe uma ampla variedade de ferramentas CASE. Com o objetivo de apoiar todas as etapas do desenvolvimento de software, surgiu a necessidade de integrar ferramentas CASE em um ambiente único, os ambientes CASE integrados ou I-CASE. Maiores detalhes sobre as ferramentas CASE, são encontrados em Pressman (PRESSMAN, 2002).

A interseção entre essas duas áreas oferece um extenso campo de pesquisa. Uma vez que o desenvolvimento de software é uma atividade cooperativa por natureza, as pesquisas na área de CSCW apresentam diversos trabalhos para aprimorar aspectos de cooperação das ferramentas CASE (MANGAN, 2003).

A área de CSCW vem se desenvolvendo continuamente nos últimos anos, embora o desenvolvimento de aplicações CSCW ainda tenha um custo financeiro muito alto. Entendendo como as pessoas trabalham sozinhas e em grupos, é possível desenvolver novas tecnologias para integração dos ambientes de trabalho (CROW e PARSOWITH, 1997).

II.1 Trabalho Cooperativo

Na literatura, são encontradas várias definições para trabalho cooperativo. Uma das primeiras definições propostas é: "Quando várias pessoas trabalham em conjunto,

lado a lado, de acordo com um plano, em um mesmo processo, ou diferentes, mas conectados, essa forma de trabalho é chamada de cooperação (RITTER, 1994)".

Essa definição não está completa, além de assumir que o trabalho cooperativo é sempre planejado. Uma definição mais genérica e intuitiva é: "Um processo de trabalho envolvendo várias pessoas agindo em conjunto, em um contexto compartilhado, para realizar tarefas com objetivo em comum (RAMAMPIARO e NYGARD, 1999)".

Apesar da falta de uma definição precisa, segundo (RAMAMPIARO e NYGARD, 1999), é consenso que o trabalho cooperativo deva ter pelo menos uma das seguintes características:

a) **Grau de cooperação:** pode variar de acordo com a necessidade da situação, ou seja, pode ser necessária uma cooperação total em alguns casos, ou nenhuma em outros;

b) **Número variável de usuários:** o número de pessoas trabalhando não é fixo, pode ter uma pessoa em determinado momento e várias em outro;

c) **Matriz tempo/espaço:** as pessoas podem trabalhar no mesmo período de tempo (síncrono), ou em períodos de tempo diferentes (assíncrono). Além disso, podem trabalhar no mesmo espaço físico, ou dispersas geograficamente;

d) **Interação variável:** não é possível estimar o tempo de interação necessário previamente. Algumas vezes, um conjunto de ações pode ser especificado antes da tarefa começar, sendo possível estimar a duração aproximada, outras vezes isso não é possível;

e) **Interação dinâmica:** as atividades cooperativas podem, em um curto espaço de tempo, trabalhar com dados estruturados (por exemplo: atividades que envolvem o uso de *workflow*), semi-estruturados (por exemplo: desenvolvimento de software), ou *ad-hoc* (por exemplo: atividades de publicação).

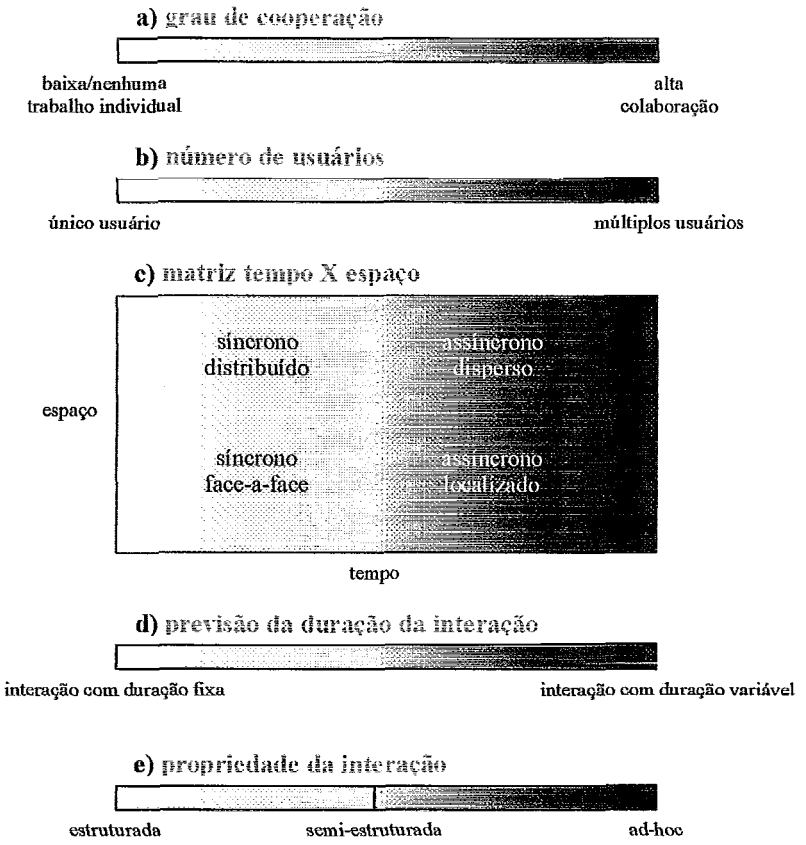


figura 1. Ilustração gráfica das características do trabalho cooperativo (RAMAMPIARO e NYGARD, 1999).

Dentre essas características, a matriz tempo/espaço (figura 1, item c) é utilizada para classificar a natureza das aplicações cooperativas do ponto de vista de sua capacidade de quebrar as fronteiras do tempo e da localização entre indivíduos, e estabelecer comunicação entre eles. Quanto ao espaço, os membros da equipe podem interagir estando no mesmo local (interação localizada) ou geograficamente dispersos (interação distribuída) (VIEIRA, 2003). Quanto ao tempo, a interação entre os membros da equipe pode ocorrer de forma:

- **Síncrona:** Os membros trabalham ao mesmo tempo, no mesmo conjunto de dados. As modificações feitas nos dados compartilhados são imediatamente (em tempo real) vistas pelos outros membros da equipe.

- **Assíncrona:** Os membros não trabalham no mesmo período de tempo, podem inclusive trabalhar desconectados (*off-line*) do resto da equipe. Conseqüentemente, as modificações feitas nos dados compartilhados não são propagadas em tempo real.

Existem ferramentas que suportam os dois tipos de interação, como o Groove (GROOVE, 2001). Outras só contemplam um tipo de interação, como, por exemplo, o NetMeeting (Microsoft), que contempla o trabalho síncrono, e TeamSource (Borland), que contempla o trabalho assíncrono.

II.2 Aspectos de Apoio ao Desenvolvimento Cooperativo de Software

O principal objetivo da engenharia de *software* é o de estabelecer mecanismos para um desenvolvimento de software com qualidade e produtividade. A qualidade do software está intimamente relacionada com a questão de entendimento entre os membros do processo de desenvolvimento, ou ainda, à convergência dos seus pontos de vista. A produtividade, por sua vez, depende de que o **entendimento** seja alcançado de forma organizada, breve e eficiente (ARAUJO et al, 1997).

Os membros de uma equipe de trabalho, geralmente, possuem especializações diferentes. Quando um grupo de pessoas se reúne em equipe com um objetivo em comum, cada um introduz experiências e habilidades individuais, que passam a fazer parte do conhecimento da equipe. Essas diferenças em especialização podem atrapalhar o processo de entendimento entre os membros da equipe, como se estivessem falando idiomas diferentes. Para superar esse problema, a equipe deve construir uma estrutura para compartilhar seus conhecimentos, evitando que o trabalho seja prejudicado por problemas de entendimento (ARAUJO et al, 1997).

Quatro aspectos principais devem ser oferecidos pelas aplicações cooperativas, para suporte eficiente à cooperação durante o desenvolvimento de software, facilitando o entendimento entre os membros da equipe envolvida no trabalho (ARAUJO et al, 1997). Estes aspectos são intimamente dependentes e inter-relacionados uns aos outros. A figura 2,

mostra um diagrama descrevendo o relacionamento entre eles. Neste diagrama, as setas indicam que o suporte a um determinado aspecto pode influenciar ou contribuir para suporte a outros aspectos do esquema.

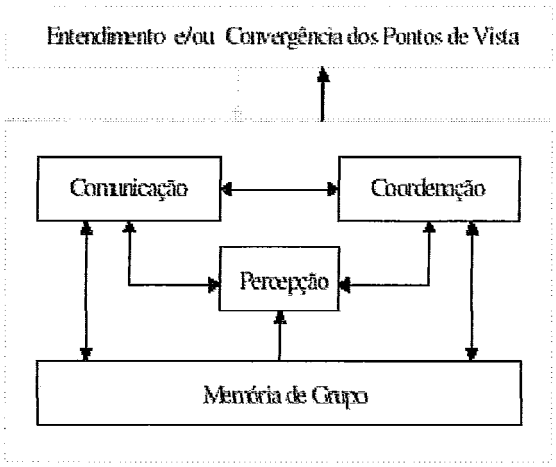


figura 2. Esquema Conceitual dos Aspectos de Suporte por Computador ao Desenvolvimento Cooperativo de Software (ARAUJO et al, 1997).

- **Memória de Grupo:** A memória do grupo é formada pela captura, representação, registro, recuperação e reutilização do conhecimento comum. Ou seja, é o registro de todo o processo de interação do grupo (memória do processo) e dos produtos gerados pelo processo (memória do produto), incluindo a comunicação realizada e os passos desencadeados durante o processo. É sobre os elementos armazenados na memória de grupo, que os usuários, desenvolvedores e coordenadores irão travar suas comunicações, coordenar suas atividades e ter acesso ao conhecimento comum.

- **Percepção:** Fornecer informações de percepção (*awareness*) da existência do grupo, do trabalho que está sendo realizado e de sua evolução. O conceito de percepção pode ser definido como a contextualização das atividades individuais, através da compreensão das atividades realizadas por outras pessoas. Os mecanismos de percepção são as técnicas

empregadas por um sistema para fornecer informações de percepção para os membros da equipe. Através desses mecanismos, um membro do grupo pode ter noção do contexto onde está inserido seu trabalho e o trabalho do grupo. Um exemplo mais comum desses mecanismos é a noção de presença de outros participantes do grupo através da notificação dos membros que estão ativos em determinado momento. O conhecimento da presença de cada membro define canais de comunicação disponíveis para cada interação.

- **Comunicação:** Não se pode esperar cooperação entre as partes sem que haja uma ligação entre elas. Assim, é preciso oferecer e administrar meios de comunicação que permitam a ligação e a troca de informações entre os membros do grupo, como, por exemplo, correio eletrônico e fóruns de discussão. Além desse tipo de comunicação direta, é possível também estabelecer canais de comunicação indiretos de duas formas. A primeira, através da memória de grupo, uma vez que o compartilhamento do conhecimento comum também constitui uma ligação entre os membros do grupo. A segunda, através da percepção, uma vez que, quando um membro passa a ter noção do contexto onde está inserido seu trabalho e o trabalho do grupo, também, é estabelecida uma ligação deste com os demais membros.

- **Coordenação:** Consiste no gerenciamento e acompanhamento do progresso das atividades individuais e coletivas. É preciso, portanto, prover suporte para a execução das tarefas do grupo e das tarefas individuais de cada membro através de mecanismos de definição, visualização e acompanhamento do encaminhamento do processo. A coordenação está relacionada à palavra acompanhamento. Durante o processo, é preciso apoiar atividades como: a especificação da forma de interação entre os membros, definição de regras e limites, controle e acompanhamento da execução de tarefas. A memória de grupo é responsável por manter essas informações disponíveis. Para coordenar essas atividades, é preciso ter a

percepção da existência do grupo e dos membros envolvidos nessas atividades, além de dispor de mecanismos de comunicação com os mesmos.

Na literatura, existem trabalhos com propostas de ambientes que fornecem suporte a todos os aspectos descritos acima, e outros que suportam atividades específicas, como, por exemplo, permitir somente a comunicação entre os membros da equipe de trabalho. O capítulo II, item II.5, apresenta alguns trabalhos, que foram estudados durante a elaboração desta dissertação. Os trabalhos encontrados implementam o suporte aos aspectos descritos de várias formas, como pode ser visto na figura 3.

Aspectos	Abordagens de Suporte
Memória de Grupo	1. Representação de conhecimento através de hipertextos 2. Modelos de argumentação 3. Formalização de conceitos 4. Gerência de configuração 5. Armazenamento em bases de objetos
Percepção	1. Recursos de interface 2. Mecanismos de consulta e navegação 3. Uso de anotações 4. Uso de notificação
Comunicação	1. Flexibilidade de modos de interação 2. Uso de espaços de trabalho compartilhados 3. Comunicação indireta através da memória de grupo 4. Uso de sistemas de mensagens
Coordenação	1. Modelagem de processos 2. Encenação de processos 3. Modelagem de usuários 4. Controle de concomência

figura 3. Resumo das abordagens para suporte ao desenvolvimento cooperativo de software (ARAUJO et al, 1997).

Dentre estes aspectos, os que serão abordados nesta dissertação são o suporte à memória de grupo e suporte à percepção, que serão discutidos a seguir.

II.3 Suporte à Memória do Grupo

Fornecer suporte à memória de grupo significa disponibilizar mecanismos para capturar e registrar o conhecimento comum da equipe de trabalho, que é importante para o desenvolvimento do software que está sendo construído, e torná-lo disponível para os seus membros.

A memória do grupo é constituída pelo **conhecimento informal** da equipe, que é formado pelo registro de idéias, fatos, questões, pontos de vista, conversas, discussões, decisões, ou seja, todo o processo de interação da equipe (memória do processo). E, também, pelo **conhecimento formal**, que é formado pela preservação de documentos nas mais variadas formas, que representam o produto gerado (memória do produto). Estes dois tipos de conhecimento estão intimamente relacionados, pois um se baseia no outro para dar sentido ao processo de desenvolvimento. Por exemplo, é possível entender um determinado estágio em que se encontra o produto recuperando o registro das interações dos membros da equipe (conhecimento informal), que levaram à sua construção.

Existem várias formas de implementar o suporte à memória do grupo, como pode ser visto na figura 3, para esta dissertação os mais relevantes são:

II.3.1 Gerência de Configuração

Toda a informação produzida durante o processo de desenvolvimento de software é chamada de **configuração de software**. Essa informação é constituída por **objetos** (ou itens) **de configuração de software**, que podem ser divididos em três amplas categorias: programas de computador (tanto na forma de código-fonte quanto executável), documentos que descrevem esses programas (voltados tanto para profissionais técnicos quanto para usuários) e dados (contidos nos programas ou externos a eles). À medida que o processo de desenvolvimento progride, a quantidade de objetos de configuração aumenta rapidamente. O problema é que pode ser necessário fazer modificações nesses objetos em qualquer fase do processo de desenvolvimento, além disso, esses objetos são inter-relacionados, modificações em um objeto pode gerar modificações em vários outros.

A gerência de configuração de software é um conjunto de atividades de acompanhamento e controle, que são aplicadas desde o início do processo de desenvolvimento de software, e só termina quando ele é retirado de operação (PRESSMAN, 2002).

Estas modificações são inevitáveis no processo de desenvolvimento de software, e podem gerar problemas para a equipe quando não são analisadas antes de serem feitas, não são registradas antes de serem implementadas, não são relatadas àqueles que têm necessidade de saber delas, ou não são controladas no sentido de melhorar a qualidade e reduzir erros. Ou seja, uma seqüência de modificações fora de controle pode transformar um projeto de software em caos. Como essas modificações podem ocorrer em qualquer época, as atividades de gerência de configuração são desenvolvidas para:

- Identificar os objetos de configuração que podem ser modificados;
- Estabelecer as relações entre eles;
- Definir mecanismos para administrar as diferentes versões dos objetos de configuração;
- Controlar modificações;
- Garantir que as modificações sejam adequadamente implementadas;
- Relatar as modificações aos demais membros da equipe.

A necessidade de fazer modificações no software pode surgir por várias razões, dentre elas, é possível citar:

- Novas condições do negócio ou do mercado ditam modificações nos requisitos do produto ou nas regras do negócio;
- Novas necessidades do cliente exigem modificação dos dados produzidos por sistemas de informação, da funcionalidade incorporada a produtos ou dos serviços realizados por um sistema baseado em computador;

- Reorganização ou crescimento/diminuição dos negócios gera modificações nas prioridades do projeto ou na estrutura da equipe de desenvolvimento;

- Restrições de orçamento ou cronograma causam redefinição do sistema ou produto.

A maioria das abordagens para gerência de configuração do software tende para uma estrutura onde são armazenadas, em um repositório, as modificações feitas em um objeto de configuração, criando várias versões dele. A evolução de um software pode ser rastreada examinando todas essas versões criadas. Como muitos objetos de configuração e suas versões são produzidos ao longo do processo de desenvolvimento, para ser possível controlar e administrar esses objetos, é necessário que cada um seja identificado de modo único. Nesta categoria estão os ambientes EPOS (EPOS, 1989), onde uma versão é constituída pelo resultado de determinadas mudanças que foram aplicadas à base de dados, e o PROSOFT (REIS et al., 1997), onde uma versão é constituída pelo objeto original e pelas modificações ocorridas no mesmo (ARAUJO et al, 1997).

Existe uma certa discordância sobre a granularidade dos objetos (artefatos) armazenados. Em geral, a unidade de trabalho armazenada é o arquivo. O COVEN (CHU-CARROL et al., 2002) é um sistema de gerência de configuração que possibilita operar sobre unidades menores do que um arquivo completo. Os trabalhos baseados no Desenvolvimento Baseado em Componentes (DBC) apontam na direção oposta, trabalhando com unidades maiores do que um arquivo, ou seja, um componente em um de seus diversos formatos (MANGAN, 2003).

Algumas abordagens se preocupam também com o registro dos motivos que levaram às alterações entre as versões, como, por exemplo, o ambiente Quorum (ARAUJO e BORGES, 1995).

II.3.2 Armazenamento das Informações em Bases de Dados

Durante o processo de desenvolvimento cooperativo de software, é preciso garantir a persistência do conhecimento comum da equipe para que ele possa ser compartilhado. Esta base de dados é necessária, para que as informações importantes sobre o trabalho da equipe e os produtos gerados não sejam perdidos, e estejam disponíveis ao longo de todo o processo.

Neste contexto, os Sistemas Gerenciadores de Banco de Dados (SGBDs) têm sido utilizados como repositórios passivos de informação. Isso porque as aplicações cooperativas possuem algumas características particulares que os mecanismos de persistência dos Sistemas de Gerenciamento de Bancos de Dados (SGBDs) tradicionais não conseguem atender (RAMAMPIARO e NYGARD, 1999).

Dentre os requisitos gerados pelas aplicações cooperativas e que geram problemas na utilização dos SGBDs tradicionais, é possível destacar (EL-JAICK et al., 2002):

- **Controle de Acesso:** O controle de permissões de acesso desempenha uma função muito importante em aplicações cooperativas. A maioria dessas aplicações utiliza a definição de papéis (*roles*) para descrever as permissões de acesso dos usuários sobre os objetos. Os papéis podem definir permissões de acesso e os direitos de acesso podem, igualmente, delimitar a natureza de um papel (MARIANI e RODDEN, 1991). O controle de acesso, tradicionalmente, é feito através de um administrador de banco de dados único. Em geral, cada usuário do SGBD é considerado um cliente individual e cada cliente tem permissões garantidas ou negadas para acessar e/ou criar dados. Esse controle de acesso individual é utilizado também nas consultas à base de dados. Com relação a aplicações cooperativas, essa forma de controle de acesso individual não é satisfatória. O mecanismo de controle de acesso nessas aplicações deve ter incorporado o dinamismo associado à atividade de colaboração. No mínimo, deve permitir que os colaboradores possam mudar seus papéis de

forma fácil e sem sobrecarga adicional. Essa troca de papéis é fundamental, pois a limitação de um usuário a um determinado papel pode restringir sua participação no processo cooperativo. Assim, por exemplo, se um usuário está no grupo com o papel apenas de observador e não puder fazer a mudança para um papel que permita que ele exponha sua opinião, a equipe poderá perder uma importante contribuição ao trabalho. Outra funcionalidade que os mecanismos de controle de acesso devem adotar é permitir que os colaboradores assumam múltiplos papéis simultaneamente. Isto é devido à natureza dinâmica das aplicações cooperativas, onde nem sempre o trabalho sai como planejado e algumas tarefas delegadas para um determinado tipo de participante tem que ser desempenhada por um outro (PRAKASH et al., 1999).

- **Controle de Concorrência:** Nas aplicações cooperativas os usuários podem trabalhar simultaneamente, sobre a mesma informação. Por isso, o controle de concorrência em aplicações cooperativas é necessário para resolver conflitos e possibilitar atividades de grupo fortemente acopladas. Seguindo a idéia de que as aplicações cooperativas devem informar e não restringir, os problemas de concorrência devem ser resolvidos através de negociações entre os membros envolvidos (ALTMAN e WEINREICH, 1998). Entretanto, o modelo tradicional de controle de concorrência utilizado nos bancos de dados convencionais, baseado nos conceitos de transação e bloqueio, se preocupa em garantir que os usuários estejam isolados das atividades uns dos outros. Não há mecanismos que permitam que os membros dos grupos percebam o que os outros estão fazendo (percepção) sendo, portanto, muito restritivo e inadequado para as aplicações cooperativas. O bloqueio em bancos de dados tradicionais é, geralmente, atribuído a um usuário. Ou o usuário mantém o bloqueio, ou é informado que o recurso está bloqueado por outro usuário e deve esperar até que o mesmo esteja liberado. Essa forma de bloqueio é contrária aos princípios da cooperação. Em aplicações cooperativas, é importante que se possa conhecer quem detém o bloqueio sobre o recurso, permitir que

usuários possam compartilhar os mesmos recursos, ou que um usuário possa visualizar as operações, em curso, executadas por um outro usuário.

- **Gerenciamento de Versões:** As aplicações cooperativas possuem como característica a necessidade de manter partes alternativas dos trabalhos, para avaliação e comparação, e deve permitir que vários usuários possam trabalhar, simultaneamente, em um mesmo recurso. Neste caso, para evitar interferências de co-autores, o SGBD deve poder gerenciar versões alternativas dos objetos. À medida que for solicitado pelos autores, deve ser possível trocar versões derivadas alternativamente, e combiná-las em uma versão aceita por todos os autores (TESH e WÄSCH, 1995). Uma abordagem para o gerenciamento de versões bastante aceita é o modelo *check-in / check-out* (PINHEIRO, 2000). Nesse modelo, o usuário faz uma cópia do objeto desejado para sua área particular (*check-out*), onde executa suas alterações. A criação de uma nova cópia desse objeto pode ser utilizada, por exemplo, para resolver conflitos entre usuários que desejam trabalhar de forma concorrente, com esse mesmo objeto. Concluída as alterações, a nova versão do objeto é depositada, novamente, no banco de dados (*check-in*). Dessa forma, cada objeto passa a ser uma coleção de versões, com mecanismos de reserva e cópias de versões. A desvantagem desse mecanismo é que ele transfere o problema de tratamento de conflitos no acesso a um mesmo objeto por mais de um usuário, para o problema de fazer a união (*merge*) entre várias versões de um objeto (BREDENFELD, 1995).

- **Suporte à Percepção:** Os mecanismos de percepção são fundamentais para o sucesso de aplicações cooperativas. A integração das informações de percepção nos SGBDs permitirá que cada atualização processada produza uma informação de percepção correspondente, possibilitando que as notificações sejam propagadas aos usuários tão logo ocorram. Cada atualização processada pode produzir um pedaço da informação de percepção e cada tipo de dado pode definir a forma como a informação

será tratada. Assim, é possível suportar diferentes modelos de percepção, como, por exemplo, armazenar os dados de percepção juntamente com os objetos correspondentes, ou propagá-los, imediatamente, para os usuários (PREGUIÇA et al., 2000). Os requisitos de percepção para aplicações cooperativas incluem habilidades para: fornecer relatórios do período de conexão dos usuários, informar o que foi lido por algum participante, detectar e reportar tudo que for novo para alguém do grupo, reportar informações das alterações, produzir notificação automática sobre novos desenvolvimentos de acordo com o perfil do participante (BORGES e PINO, 2000). Além disso, deve ser armazenado o conhecimento de cada membro do grupo sobre seu conteúdo, de forma a prover serviços "customizados" de percepção, filtrando informações que não sejam do interesse do usuário, ou que o mesmo já tenha conhecimento.

- **Replicação dos Dados:** Em aplicações cooperativas, é muito comum ocorrer desconexões dos usuários, que podem acontecer, por exemplo, devido a possíveis falhas ou não disponibilidade da rede. Alguns usuários podem preferir trabalhar desconectados, conectando-se à rede apenas para o recebimento/envio de informações. Desse modo, para que os sistemas possam garantir a alta disponibilidade das informações, é importante que os SGBDs possuam um bom mecanismo de replicação dos dados e atualização concorrente de informações. Os usuários podem manter uma cópia local das informações compartilhadas, fazer atualizações sobre esses dados e, posteriormente, atualizar o servidor (PREGUIÇA et al., 2000).

A maioria das propostas de ambientes cooperativos que armazenam as informações em bases de dados, utiliza bancos de dados orientados a objetos como repositório (ARAUJO et al., 1997). Dentre eles, é possível citar o EPOS (EPOS, 1989) e o SPADE (BARDINELLI et al., 1996).

II.4 Suporte à Percepção

Durante o desenvolvimento cooperativo de software, é necessário que os membros da equipe tenham noção do contexto de suas atividades em relação ao contexto geral do processo, para que possam perceber o andamento das atividades realizadas pelos demais e compreender como os resultados gerados por essas atividades podem ser conjugados com os seus próprios, para mais rapidamente chegarem ao resultado final.

A falta de informações sobre as atividades individuais de cada membro e da equipe pode gerar uma série de problemas para o trabalho, como redundâncias, inconsistências e contradições nas atividades desempenhadas, prejudicando a qualidade e eficiência, e possivelmente impedindo que a equipe atinja seus objetivos. Para evitar esses problemas, é necessário que os membros da equipe tenham acesso a essas informações, ou seja, tenham *percepção* do trabalho que está sendo realizado. Percepção, é portanto, o conhecimento geral sobre as atividades e sobre a equipe. Esse conhecimento é formado por informações sobre o que aconteceu, o que vem acontecendo, o que está se passando agora e o que poderá acontecer dentro das atividades da equipe, e sobre a própria equipe, seus objetivos e sua estrutura (PINHEIRO et al., 2001).

Uma definição para percepção é "uma compreensão das atividades dos outros, que provê o contexto para atividades próprias. Esse contexto é utilizado para garantir que as contribuições individuais sejam relevantes para a atividade do grupo como um todo, e para avaliar as ações individuais em relação aos objetivos e progresso do grupo". As atividades vão desde movimentos físicos da pessoa em uma sala (entrada e saída de uma sala ou prédio) até interações do participante com uma aplicação cooperativa (VIEIRA, 2003).

A percepção é essencial para o fluxo e a naturalidade do trabalho, auxiliando a diminuir as sensações de trabalho impessoal e de distância, comuns nos ambientes virtuais. Através dessas informações, os membros da equipe podem montar o contexto de suas atividades em relação às dos demais. Quando, por exemplo, membros da equipe trabalham em um artefato compartilhado, precisam dessas informações para compreender mudanças ocorridas pela ação de outras pessoas, e os efeitos que essas mudanças tiveram

sobre seu trabalho, para saber como ele deverá agir desse ponto em diante. Além disso, o coordenador do grupo pode utilizar essas informações para ter uma visão geral do andamento dos trabalhos e do estado de cada atividade, para motivar e organizar as pessoas dentro do grupo, encorajando a cooperação.

Durante o desenvolvimento cooperativo de software, os membros de uma equipe podem trabalhar de forma síncrona ou assíncrona. Embora a percepção seja importante nessas duas formas de interação, o tipo de informação de percepção que os membros necessitam nas duas abordagens é bem diferente, em diversos aspectos. Essas diferenças estão relacionadas, por exemplo, a granularidade das informações tratadas, ao fornecimento dessas informações aos usuários finais e a durabilidade das mesmas (PINHEIRO, 2000).

Quando os membros de uma equipe trabalham de forma síncrona, necessitam de informações sobre o que está acontecendo no espaço de trabalho compartilhado no momento atual, com notificação imediata dessas informações. Assim, os membros da equipe podem ter uma mesma visão da área de trabalho. As informações de percepção geradas para apoiar esse tipo de interação estão relacionadas a ações de movimentação dos usuários pelo espaço de trabalho, sua disponibilidade para interação, e ações de baixa granularidade como movimentações de *mouse* e cursor, alteração na posição da barra de rolagem, etc. Esse tipo de informação, geralmente, necessita de baixa ou nenhuma persistência.

Quando os membros trabalham de forma assíncrona, necessitam de informações sobre interações que eles não tenham participado, em um momento posterior à ocorrência das mesmas. Isso porque, nesse tipo de interação os membros trabalham, na maior parte do tempo, desconectados do resto da equipe, conectando-se periodicamente. As informações geradas devem estar em um nível mais macro, com um resumo das últimas ações executadas sobre um determinado artefato, desde a última vez que o membro se conectou. Além disso, essas informações devem responder questões importantes relacionadas ao contexto do trabalho, como, por exemplo, "quais membros estão envolvidos no trabalho", "quem é o responsável por determinada atividade", "qual é o prazo para o término dessa atividade", etc. Esse tipo de informação necessita de alta

persistência, já que periodicamente é preciso reproduzi-las para os membros. Armazenar um histórico dessas informações também é importante para que possam ser consultadas quando, por exemplo, for preciso detectar e corrigir erros.

II.4.1 Mecanismos de Suporte à Percepção

Os mecanismos de suporte à percepção são as técnicas empregadas por um sistema para fornecer informações de percepção à equipe de trabalho. Dentre os problemas para implementar mecanismos de suporte à percepção em aplicações cooperativas, é possível destacar (PINHEIRO, 2000):

- Dificuldade para identificar o tipo e a quantidade de informação necessária para o trabalho que está sendo desenvolvido. Isso porque, a quantidade de informações produzidas durante as interações nestas aplicações tende a crescer progressivamente, ao longo do tempo, podendo ocasionar sobrecarga para os membros da equipe. O excesso de informação é tão prejudicial ao trabalho quanto a sua falta, podendo desviar a atenção dos membros devido à chegada de informações, muitas vezes desnecessárias para o seu trabalho.

- Dificuldade para identificar as informações que podem ser divulgadas, e para quais membros da equipe, respeitando a privacidade do grupo e evitando que informações sigilosas sejam disponibilizadas de forma indevida. Isso porque, os membros de uma equipe de trabalho, geralmente possuem atribuições e responsabilidades diferentes, e assumem uma determinada posição hierárquica dentro da equipe (papéis). A necessidade de percepção de cada um desses membros é muito diferente. Informações importantes para o coordenador da equipe, por exemplo, geralmente não devem ser disponibilizadas para os programadores. Por outro lado, os coordenadores não se interessam por detalhes de programação, que são úteis para os programadores.

Existem vários mecanismos de suporte à percepção propostos na literatura, como pode ser visto na figura 3. Esses mecanismos, em geral, apresentam seis características importantes, cada uma delas identifica aspectos vitais para o fornecimento de percepção dentro de aplicações cooperativas, são elas (PINHEIRO et al., 2001):

- **Quem:** Refere-se a quem está trabalhando no momento. A noção de presença dos outros membros é vital em ambientes síncronos, pois é inviável realizar uma tarefa simultaneamente com os demais membros sem saber quem está conectado. Em ambientes assíncronos, a noção de presença não é essencial, pois a presença de todos não é obrigatória para que o trabalho seja realizado. Nesses ambientes, a noção de presença pode ser vista como uma "oportunidade" de cooperação a ser explorada, pois sabendo quem está conectado em dado momento permite que os membros possam tirar dúvidas, trocar idéias e experiências entre eles, enriquecendo o trabalho da equipe.

- **O Que:** Refere-se a quais informações devem ser fornecidas aos membros da equipe. Essas informações são restringidas pelo papel (hierarquia) que o membro exerce na equipe. Os papéis indicam as responsabilidades e possibilidades dos membros sobre o trabalho. Em ambientes síncronos, é mais importante ter informações sobre os detalhes das atividades que estão sendo realizadas no momento. Em ambientes assíncronos, é mais importante ter uma visão geral e mais ampla dessas atividades.

- **Onde:** Refere-se a onde as informações são geradas e apresentadas. Quando as pessoas trabalham cooperativamente, compartilham um conjunto de objetos dispostos em um espaço de trabalho (*workspace*). A percepção do que está acontecendo no espaço de trabalho compartilhado (*workspace awareness*) é vital em ambientes síncronos, para que os membros do grupo tenham noção de todas as atividades que estão sendo realizadas. Em ambientes assíncronos, não há como garantir a presença de todos os membros em um intervalo de tempo, assim, o foco da percepção são os objetos por eles compartilhados no espaço de trabalho. Nesses

ambientes é através da manipulação dos objetos compartilhados e de seu histórico, que se tem a percepção do que está acontecendo dentro do trabalho da equipe.

- **Quando:** Refere-se a quando ocorrem os eventos geradores das informações de percepção e quando se dá a apresentação dessas informações. As informações de percepção são geradas por eventos que ocorrem durante o trabalho da equipe, e de acordo com seu momento de ocorrência, esses eventos vão ser mais ou menos úteis à percepção. A ocorrência desses eventos pode ser dividida em quatro momentos: **passado**, para eventos que ocorreram há algum tempo e cujos resultados podem não ser mais válidos; **passado contínuo**, para eventos que iniciaram no passado e continuam válidos no presente; **presente**, para eventos que estão ocorrendo no momento; **futuro**, para eventos que ainda poderão ocorrer e que precisam fazer parte da percepção do membro. Alguns desses momentos são mais ou menos importantes de acordo com o ambiente utilizado. Em ambientes síncronos, o mais importante é fornecer informações sobre os eventos que estão ocorrendo no presente, e essas informações devem ser apresentadas para os membros imediatamente (quanto mais cedo melhor). Em ambientes assíncronos, as informações importantes são as referentes a eventos que aconteceram no passado ou no passado contínuo, para que os membros possam encaixar suas próprias atividades nas da equipe. Nesses ambientes, há um intervalo de tempo entre os eventos geradores e os demais membros perceberem. Logo, a informação é apresentada em um momento posterior à ocorrência dos eventos, sendo adequado permitir que o membro decida quando quer recebê-la. Nos dois ambientes, a percepção de eventos futuros é importante para manter os membros atentos aos possíveis rumos do trabalho.

- **Como:** Refere-se a como as informações são apresentadas aos membros, como é sua interface. A interface com o usuário é responsável pelo fornecimento das informações de percepção, devendo apresentá-las de

forma resumida, mas sem perda de informação, a fim de evitar sobrecarga dos membros e permitir uma rápida assimilação. Assim, é preciso projetar uma interface balanceada, com elementos adequados que deverão fazer uma filtragem ou agrupamento das informações, mostrando somente o que for útil e interessante para cada membro. Além disso, a percepção da equipe é afetada se ocorrem diferenças nas visões dos membros do espaço de trabalho e dos objetos compartilhados. Manter uma visão para todos significa que os membros vão ter as mesmas informações, quando a representação diverge, o contexto pode se perder. Essa forma de apresentação ideal varia de acordo com o ambiente utilizado. Assim, o projetista da aplicação cooperativa deve decidir entre beneficiar atividades individuais, ou priorizar a percepção das atividades coletivas. Em ambientes síncronos, é importante a percepção uniforme do espaço de trabalho, onde todos os membros ativos tenham a mesma imagem do espaço de trabalho e seus objetos. Em ambientes assíncronos, é importante manter a liberdade de ação dos membros fornecendo, por exemplo, a possibilidade de ter múltiplas visões do espaço de trabalho. Desta forma, os membros podem escolher a visão mais adequada para sua atividade.

- **Quanto:** Refere-se a qual é a quantidade ideal de informações que deve ser apresentada ao membro, para fornecer percepção sobre o trabalho da equipe e suas atividades. Esta preocupação é uma constante no desenvolvimento de um suporte eficaz à percepção, independente do ambiente utilizado, e afeta todas as questões anteriormente discutidas. O principal foco dessa preocupação é fornecer a quantidade certa de percepção, sem causar a insuficiência de informações, que poderia deixar os membros sem um contexto para suas atividades, e sem causar a sobrecarga de informações, que pode prejudicar os participantes, interrompendo o seu trabalho e desviando sua atenção para informações que não são importantes para eles.

Dada sua importância, o estudo da percepção e seus mecanismos de suporte têm sido alvo de diversas pesquisas na área de CSCW, produzindo uma série de trabalhos publicados ao longo dos anos. Esta dissertação faz referência a vários desses trabalhos, que podem ser consultados para maiores detalhes (FARSHCHIAN, 2001), (GUTWIN e GREENBERG, 1999), (PINHEIRO, 2000) e (VIEIRA, 2003).

II.4.2 Modelos de Percepção

Achar um modelo de percepção adequado ao domínio da aplicação que se deseja apoiar é um grande desafio para a implementação dos mecanismos de percepção. Esse modelo estabelece quais informações devem ser disponibilizadas para quem, de forma que todas as informações disponíveis sejam adequadamente distribuídas, e garantindo que quem precisa delas irá recebê-las.

Na literatura, são encontradas muitas propostas de modelos de percepção. Esses modelos, geralmente, introduzem os conceitos de **espaço compartilhado**, **presença** e **foco**. **Espaço compartilhado** é o lugar onde os objetos compartilhados devem residir, para poderem produzir e consumir informações de percepção. Para que os objetos sejam conhecidos no espaço compartilhado, devem comunicar sua **presença** se registrando com um conjunto de parâmetros pré-determinados. Uma vez presente no espaço compartilhado, os objetos devem enviar informações para atualizar a sua presença regularmente. Um objeto pode manter o **foco** em outro objeto, para receber informações de percepção sobre ele. Esse foco também pode ser atualizado regularmente. Dentre os modelos de percepção encontrados na literatura, é possível citar :

- **Modelo Baseado em Espaço:** As informações de percepção sobre as atividades realizadas em um espaço virtual são propagadas para todos os seus membros. Esses espaços virtuais são delimitados por "paredes" que impedem a propagação das informações para o meio exterior.

- **Modelo Baseado em Atividades:** As informações de percepção são propagadas de acordo com as atividades em que um membro está envolvido. Os membros devem ser contextualizados sobre os objetivos e a

estruturação das atividades que compõe o trabalho da equipe, e devem poder acompanhar o seu andamento. Esse tipo de percepção tem como objetivo evitar redundâncias e falhas na execução das atividades, além de facilitar a coordenação das mesmas.

- **Modelo de Percepção Social:** As informações de percepção social têm como objetivo auxiliar os membros a conhecer a equipe em que está trabalhando, oferecendo dados sobre ele e sobre os demais participantes. O objetivo é facilitar as interações e o entendimento entre os membros, para o bom andamento do trabalho.

O modelo de percepção que será abordado nesta dissertação é o **modelo de percepção do produto** (FARSHCHIAN, 2000) e (FARSHIAN, 2001). Este modelo é descrito a seguir:

II.4.2.1 Modelo de Percepção do Produto

Nesse modelo, o produto é a base para fornecer percepção e coordenar o trabalho no desenvolvimento cooperativo de software distribuído. Esse modelo está baseado na suposição de que os membros da equipe de desenvolvimento necessitam de informações constantes sobre o produto compartilhado e, devido à distribuição geográfica, a visibilidade do que está acontecendo com esse produto é reduzida.

No desenvolvimento distribuído, geralmente, o produto compartilhado é armazenado em um repositório para acesso dos membros da equipe, mas as mudanças ocorridas não são facilmente percebidas. Assim, esse modelo se propõe a fornecer informações aos membros sobre o produto compartilhado, para que eles possam observar as modificações que o produto sofreu pelos demais membros ao longo do trabalho.

O produto que está sendo construído é dividido em várias partes, que são desenvolvidas paralelamente pelas atividades dos membros. Essas partes estão relacionadas e, muitas vezes, um membro precisa de informações sobre as atividades de outro membro para realizar o seu trabalho. Devido a esse relacionamento entre as partes, o modelo de percepção do produto suporta a propagação da percepção, ou seja, informações

sobre alterações feitas em uma parte se propagam para todas as suas partes relacionadas. Esse modelo reduz o risco de problemas de integração das partes do produto e de desperdício de trabalho quando, por exemplo, uma mesma atividade é realizada várias vezes por membros diferentes, pela falta de informação.

As informações de percepção do produto são muito úteis para equipes de desenvolvimento que trabalham cooperativamente com ferramentas CASE, pois nesse caso o produto que está sendo desenvolvido é o foco do trabalho. Essas ferramentas fornecem suporte para manter o relacionamento entre as diversas partes que compõem o produto de forma estruturada, facilitando assim a propagação da percepção.

Dentre os mecanismos de suporte à percepção, utilizados para fornecer informações de percepção do produto, é possível destacar o uso de notificações. O mecanismo de notificação (*watcher*) consiste no uso de notificações automáticas, para permitir que cada membro tenha conhecimento sobre os eventos ocorridos durante as interações da equipe. As dificuldades de implementar esse mecanismo estão relacionadas aos problemas em prover informações de percepção, pois persiste, por exemplo, a dificuldade em identificar que notificações são úteis para determinado membro, quando devem ser enviadas, etc.

O modelo de percepção do produto pode ser combinado com os demais modelos existentes, para fornecer suporte ideal à percepção para o desenvolvimento cooperativo de software, tentando se aproximar da forma como as pessoas trabalhariam se estivessem próximas fisicamente. Esse modelo é proposto pelo Gossip (FARSHCHIAN, 2000) e (FARSHCHIAN, 2001), que é apresentado com maiores detalhes no item II.5.1.

II.5 Trabalhos Propostos na Literatura

Existem dois ramos de pesquisa na área de desenvolvimento cooperativo de software (ARAUJO et al, 1997). O primeiro deles está relacionado à construção de ferramentas isoladas, voltadas para o suporte a aspectos específicos ou a fases isoladas do processo de construção de software. Esse ramo de pesquisa tende a ter uma maior

preocupação com aspectos de interface, comunicação, percepção e coleta de memória de grupo.

O segundo ramo de pesquisa está relacionado com a construção de ambientes cooperativos, que combinam mecanismos de memória do grupo, percepção, comunicação e coordenação. Esses ambientes tem como ênfase, a definição de um núcleo principal para controle das atividades através de abordagens de modelagem de processos, onde ferramentas podem ser acopladas e seu uso gerenciado. Esse ramo de pesquisa se concentra em aspectos de comunicação, registro de memória de grupo e coordenação de tarefas.

Para a elaboração desta dissertação, foram estudados alguns trabalhos propostos para o suporte ao desenvolvimento cooperativo de software, dentre eles é possível destacar: OdysseyShare (ODYSSEY, 2004), (MANGAN et al., 2002) e (WERNER et al, 2003); Gossip (FARSHCHIAN, 2000) e (FARSHCHIAN, 2001); Knight (DAMM et al, 2000) e (HANSEN, 1999). Esses trabalhos são apresentados com maiores detalhes a seguir.

II.5.1 OdysseyShare

O OdysseyShare é um projeto que está em desenvolvimento pela equipe de Reutilização de Software, da COPPE/UFRJ. O OdysseyShare propõe um ambiente cooperativo para o desenvolvimento de software baseado em componentes. O ambiente OdysseyShare possui várias ferramentas para apoiar o desenvolvimento cooperativo de software, como, por exemplo, ferramenta para comunicação (como *chat*), ferramenta para controle de versões geradas e um editor cooperativo de modelos UML.

Assim, faz parte escopo do OdysseyShare apoiar a modelagem cooperativa de software, onde os membros de uma equipe de desenvolvimento trabalham com o editor cooperativo disponível nesse ambiente. Para aumentar a quantidade e a qualidade da informação de percepção disponível para os membros da equipe, o editor cooperativo do OdysseyShare disponibiliza, por exemplo, uma visão de radar (*radar view*). A visão de radar é um serviço de percepção comum em editores gráficos e textuais, que auxilia a determinar a posição corrente de edição em um documento muito grande (figura 4).

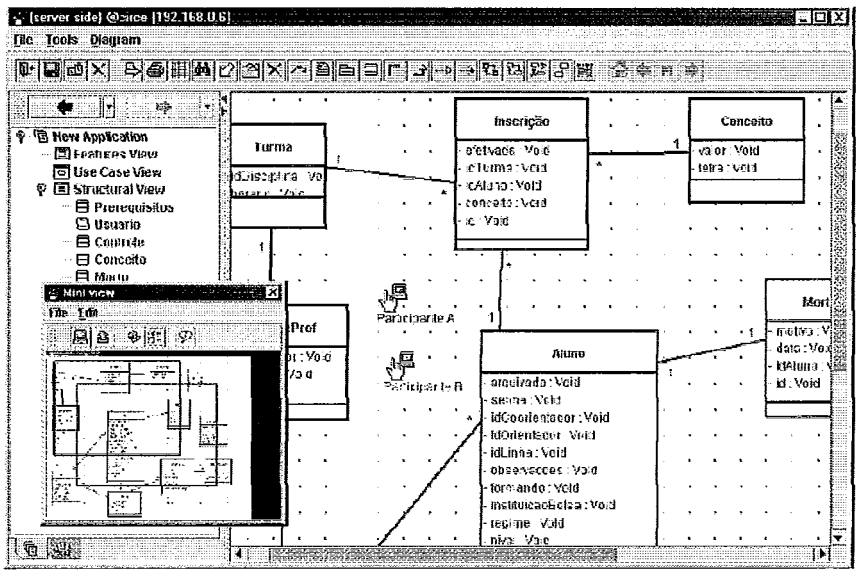


figura 4. Visão de radar do editor cooperativo do OdysseyShare (WERNER et al, 2003).

Dentre as funcionalidades propostas no projeto OdysseyShare, está o mapeamento dos modelos gerados, por essa ferramenta, para documentos XMI (OMG, 2002), para que seja possível trocar modelos com outras ferramentas CASE.

II.5.2 Gossip

O Gossip é um servidor de notificação, que tem como objetivo fornecer informações de percepção do produto para uma equipe que trabalha cooperativamente, e distribuída geograficamente, no desenvolvimento de um determinado software.

A figura 5 apresenta o mecanismo usado pelos servidores de notificação para fornecer percepção. Nessa figura, o usuário da aplicação **cliente A** realiza uma ação. A ação resulta em uma requisição de serviço da **aplicação A** para o servidor de notificação. Se a requisição de serviço é executada, uma confirmação é enviada para a **aplicação A**, e as demais aplicações clientes (**B, C** e **D**) recebem uma notificação da ação que foi realizada. O resultado da ação realizada é mostrado ao usuário da **aplicação A**. As **aplicações B, C** e **D** irão, de alguma forma, apresentar para os seus usuários a ação realizada pela **aplicação A,** como, por exemplo, através de uma mensagem de aviso ou de uma animação visual.

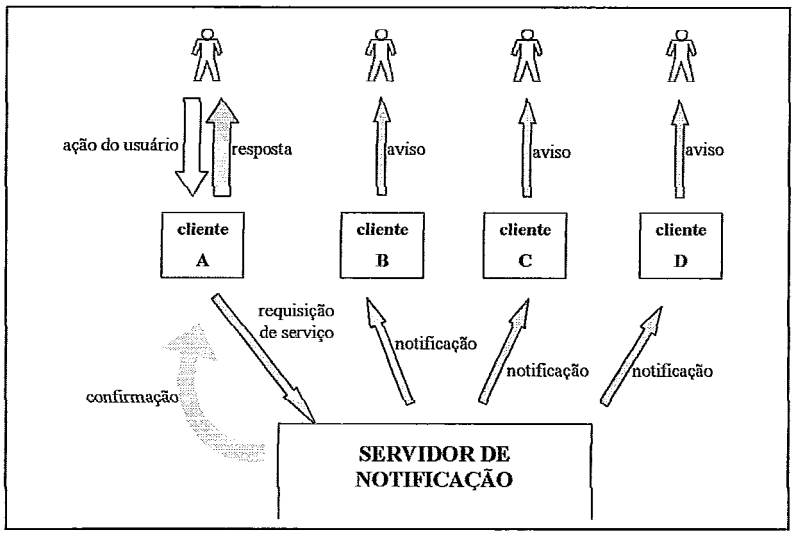


figura 5. Usando um servidor de notificação para fornecer percepção (FARSHCHIAN, 2001).

As aplicações clientes são os sistemas que os usuários utilizam para manipular o produto compartilhado. Utilizando os servidores de notificação, as informações de percepção são fornecidas através de notificações genéricas que podem ser usadas por diversos tipos de aplicações. Uma aplicação cliente pode gerar eventos que são capturados pelo servidor e são distribuídos para as demais aplicações envolvidas no trabalho. O servidor de notificação é responsável também pela escolha de quais clientes devem receber quais informações, baseado no modelo de percepção que está sendo utilizado.

O Gossip utiliza o modo **EMPURRE** (*push*) para a entrega das informações. No modo **EMPURRE**, a entrega das informações do servidor para o cliente é executada por iniciativa do próprio servidor, sem que uma requisição explícita do cliente tenha sido efetuada. A principal dificuldade nesse modo de entrega é decidir quais dados interessam ao cliente e quando o envio deverá ocorrer. Outro problema importante é a possibilidade de sobrecarregar o usuário com informações que não sejam do seu interesse, ou que ele já tenha conhecimento. Esse modo de entrega é mais utilizado em aplicações síncronas, onde o desempenho e o tempo de resposta são críticos.

O Gossip possui um conjunto de serviços para controlar e manter o desenvolvimento do produto compartilhado de forma evolutiva, e para enviar as

notificações sobre as mudanças realizadas ao longo do trabalho. O usuário não interage com o Gossip diretamente. Essa interação é feita através de uma aplicação cliente, que pode ser qualquer ferramenta para desenvolvimento de software, como, por exemplo, uma ferramenta CASE. A aplicação cliente utiliza os serviços do Gossip para registrar o produto compartilhado e suas partes, e para criar as relações de dependência entre elas para a propagação da percepção. O Gossip não armazena o produto compartilhado e suas novas versões com as alterações feitas. Ele cria um identificador único para cada produto registrado, e armazena o mínimo de informações necessárias sobre esse produto (meta informações).

O Gossip suporta operações como criação, alteração, leitura e exclusão das partes do produto compartilhado, e para cada operação ele envia uma notificação diferente. O Gossip fornece também mecanismos de consulta, como, por exemplo, permite que uma aplicação cliente pergunte quais são os produtos e as suas partes que estão registradas no Gossip em determinado momento. Esse tipo de consulta é útil quando um cliente precisa manipular um objeto, e necessita de informações de percepção sobre ele e os relacionamentos existentes entre suas partes. Detalhes do Gossip são encontrados em (FARSHCHIAN, 2000) e (FARSHCHIAN, 2001).

II.5.3 Knight

O Knight é um trabalho acadêmico, que propõe uma ferramenta para apoiar a modelagem cooperativa de software. O Knight tem como objetivo permitir a integração entre ele e as diversas ferramentas CASE existentes no mercado, utilizando o padrão XMI (*XML Metadata Interchange*) como formato para intercâmbio de dados, e a tecnologia COM (*Microsoft's Component Object Model*) para permitir a comunicação entre essas ferramentas. A especificação do Knight propõe duas versões, uma para quadro branco eletrônico (*eletronic whiteboard*) e outra para computadores (*desktop*). A versão para quadro branco eletrônico tem como objetivo permitir que o usuário desenhe informalmente ou produza somente desenhos formais, de acordo com a UML. A versão para computadores propõe uma ferramenta CASE cooperativa, que contemple o trabalho síncrono e assíncrono.

O Knight deu origem a uma ferramenta comercial, chamada Ideogramic UML, que possui uma versão disponível para teste no site da empresa Ideogramic (IDEOGRAMIC, 2000). Na ferramenta Ideogramic UML, os grupos trabalham desenhando seus modelos em um quadro branco (*whiteboard*) eletrônico que é compartilhado. Existe uma versão para computadores (*desktop*), que consiste numa ferramenta CASE, baseada em UML, mas esta permite somente o trabalho individual, não suporta cooperação. Essa ferramenta possui uma visão de radar (*radar view*) para fornecer informações de percepção para os membros da equipe (**figura 6**).

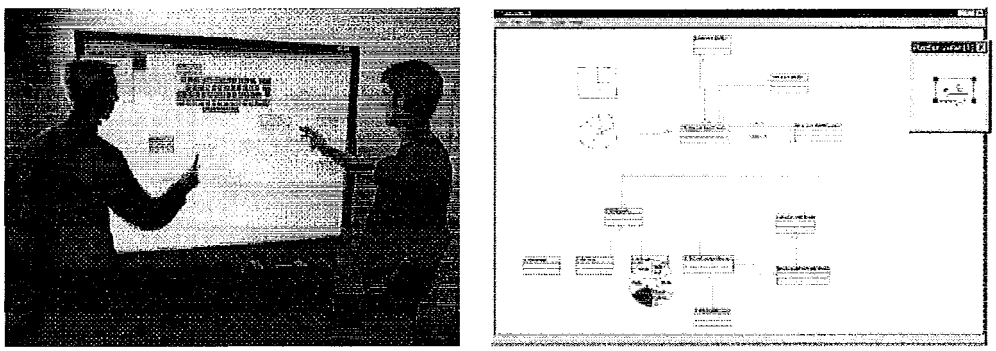


figura 6. Exemplos de utilização da Ideogramic UML (IDEOGRAMIC, 2000).

Os modelos gerados na ferramenta CASE Ideogramic UML são salvos em arquivos XMI, baseados em um DTD, para facilitar o intercâmbio de dados com outras ferramentas, ou seja, ela não utiliza um formato proprietário para o armazenamento de dados.

Detalhes do Knight são encontrados em (DAMM et al, 2000) e (HANSEN, 1999) e detalhes sobre a Ideogramic UML em (IDEOGRAMIC, 2000).

II.6 Considerações Finais

Esse capítulo apresentou um resumo do estudo sobre o desenvolvimento cooperativo de software que foi feito durante a elaboração desta dissertação. Foram exaltados os pontos mais importantes para este trabalho, e mostrados outros trabalhos

encontrados na literatura, que contribuíram para a motivação e para o desenvolvimento da solução aqui proposta. Existem várias propostas na literatura de ferramentas e ambientes para apoio ao desenvolvimento cooperativo de software. Alguns têm como objetivo apoiar tarefas específicas, outros visam apoiar todo o processo de desenvolvimento. Existem ainda propostas que visam utilizar as ferramentas já existentes para o desenvolvimento de software, tornando-as cooperativas. Outras propostas pretendem construir uma ferramenta do início. Embora exista uma diversidade de trabalhos, ainda não existe uma abordagem que suporte eficientemente a cooperação no desenvolvimento de software, principalmente que apóiem atividades de longa duração (ARAUJO et al, 1997).

O OdysseyShare possui um editor de modelos UML e tem como objetivo mapear os modelos gerados em documentos XMI, para ser possível trocar modelos com outras ferramentas CASE. Assim, o membro de uma equipe de desenvolvimento, que trabalhe com o editor do OdysseyShare, poderia utilizar o MIMIX para trabalhar cooperativamente, na modelagem de software, com um outro membro que esteja trabalhando com outra ferramenta CASE e esteja, por exemplo, distante geograficamente. As principais contribuições do Gossip foram o modelo de percepção do produto e o mecanismo de notificação das alterações realizadas no produto compartilhado. O Knight contribuiu com o uso do XMI como formato para intercâmbio de dados. Na documentação do Knight, são apresentados os problemas encontrados na utilização do XMI, que contribuíram significativamente para o uso desse padrão nesta dissertação.

O próximo capítulo apresenta a proposta do MIMIX, descrevendo suas características, os serviços disponibilizados, seu funcionamento, sua arquitetura e uma análise do mesmo em relação aos trabalhos que foram apresentados nesse capítulo.

III O MIMIX

O processo de desenvolvimento de software demanda intensa cooperação entre os membros envolvidos no trabalho. Atualmente, a complexidade e o tamanho dos projetos de software aumentou significativamente. Com isso, o número de pessoas envolvidas no processo também aumentou. Os membros de uma equipe de desenvolvimento possuem, geralmente, especializações diferentes. A diversidade de experiências e as habilidades de cada um formam a base de conhecimento da equipe. Cada membro, muitas vezes, possui uma visão do trabalho diferente dos demais, mas todos possuem um objetivo em comum. Esse objetivo consiste em desenvolver, com sucesso, um determinado software, como, por exemplo, com a qualidade desejada, dentro do prazo determinado e de acordo com o orçamento estipulado. As diferenças podem atrapalhar o processo de entendimento entre os membros, como se estivessem falando idiomas diferentes, prejudicando o desenvolvimento. Geralmente, os membros da equipe trabalham separados fisicamente, o que dificulta ainda mais o entendimento e comunicação entre eles.

É preciso, portanto, fornecer suporte ao desenvolvimento cooperativo de software através de ferramentas automatizadas, que coordenem as atividades, mantendo o controle do projeto e das informações compartilhadas. Na literatura, são encontrados vários trabalhos que tentam aplicar as pesquisas da área de CSCW no suporte ao desenvolvimento cooperativo de software, como pode ser visto em (ARAUJO et al, 1997), (BODKER et al, 1999), (BREDENFELD, 1995), (DIAS e BORGES, 1999), (EPOS, 1989), (FARSHCHIAN, 2000), entre outros. A maioria desses trabalhos, se concentra em fornecer suporte a reuniões, a discussões, a coordenação das atividades da equipe (sistemas de *workflow*) e a codificação cooperativa de software (sistemas de gerência de configuração). Entretanto, poucos trabalhos contemplam o suporte à modelagem de software, com edição cooperativa de modelos UML, por exemplo.

As ferramentas CASE têm sido amplamente utilizadas para apoiar o desenvolvimento de software, mas fornecem pouco ou nenhum suporte à cooperação. Atualmente, a maioria das ferramentas disponíveis no mercado apóia a modelagem de software baseada em UML.

Esta dissertação tem como objetivo fornecer suporte ao cenário de trabalho ilustrado na figura 7.

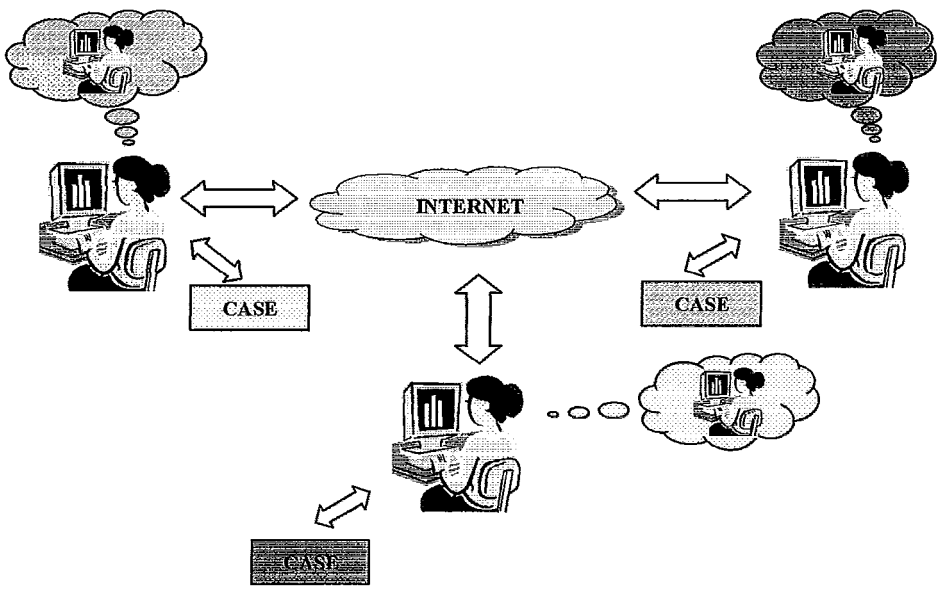


figura 7. Edição cooperativa de modelos UML, utilizando ferramentas CASE.

Neste cenário, vários usuários tentam trabalhar cooperativamente na modelagem de um sistema. Eles se encontram em ambientes de trabalho distintos e distantes fisicamente. Os usuários podem trabalhar em horários diferentes. A comunicação entre eles é feita através da Internet. Cada usuário trabalha com sua ferramenta CASE. Essas ferramentas são diferentes (heterogêneas). Cada ferramenta CASE possui um formato de armazenamento proprietário, assim, uma não entende o formato da outra.

Este cenário apresenta vários problemas. Ainda que a comunicação seja feita por correio eletrônico ou vídeo-conferência, um usuário não consegue ter informações sobre o trabalho que está sendo realizado pelos demais. Podem, assim, trabalhar de forma redundante e gerar conflitos. É preciso enviar periodicamente mensagens para todos os usuários com o trabalho que está sendo feito, o que acarreta uma sobrecarga de informações e de trabalho para os usuários. Neste cenário, não existe uma forma automática de comparar os modelos gerados, nem de fazer a união de trabalhos paralelos desenvolvidos pelos usuários. Cada usuário precisa ter as ferramentas CASE dos demais, para poder visualizar e

comparar o trabalho que foi feito. A união é feita de forma manual pelos usuários. Além disso, não existe registro do que foi realizado pela equipe, o que dificulta a observação da evolução do trabalho e a identificação de possíveis erros introduzidos.

Na literatura, podem ser encontrados trabalhos que apóiam a edição cooperativa de documentos texto, como, por exemplo o Groove (GROOVE, 2001), e de código fonte de programas, como, por exemplo, o Star Team (STAR TEAM, 2003). Em geral, esses trabalhos permitem a união de trabalhos paralelos dos usuários. Entretanto, não foi encontrado nenhum trabalho que apóie a edição cooperativa de modelos UML, permitindo a comparação e a união de versões de modelos gerados pelos membros da equipe. Conforme apresentado no capítulo anterior, o Knight (DAMM et al, 2000) e (HANSEN, 1999) propõe uma arquitetura para a integração de ferramentas CASE e suporte a edição cooperativa de modelos, onde os usuários trabalham com quadros brancos (*whiteboard*). A versão para computadores pessoais (*desktop*) não suporta a cooperação.

Baseado neste cenário, esta dissertação introduz o MIMIX, que tem como objetivo apoiar a dinâmica do trabalho cooperativo descrito anteriormente. As próximas seções descrevem, com detalhes, a solução aqui proposta. Para um melhor entendimento, inicialmente, é caracterizado o cenário apoiado pelo MIMIX, fazendo uma analogia com o cenário apresentado na figura 7.

III.1 Cenário do Trabalho Cooperativo Apoiado pelo MIMIX

O MIMIX deve disponibilizar um conjunto de serviços, que possam ser utilizados pelos membros da equipe para trabalhar cooperativamente. Esses serviços contemplam, principalmente, a forma de trabalho assíncrona. A figura 8 ilustra a mesma dinâmica do trabalho cooperativo apresentado na figura 7, mas com o suporte do MIMIX.

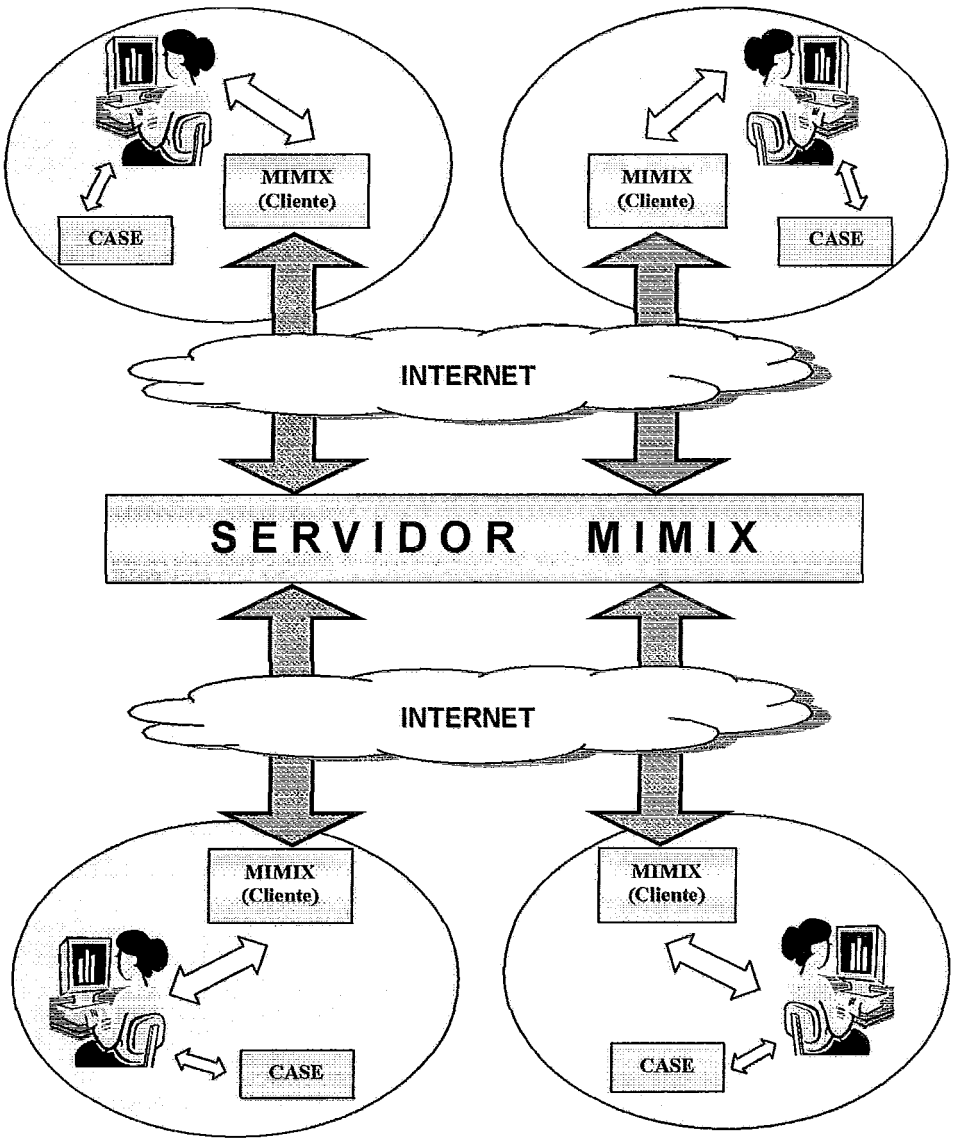


figura 8. Edição cooperativa de modelos UML, utilizando ferramentas CASE e com o suporte do MIMIX.

O MIMIX possui um módulo cliente, que provê uma interface com o usuário, e um módulo servidor, que contém os serviços disponibilizados. Os membros da equipe trabalham com suas ferramentas CASE e interagem com o módulo cliente, que faz as solicitações aos serviços do módulo servidor. As setas escuras indicam a troca de

informações e/ou a transferência de documentos XMI entre os módulos cliente e servidor, que ocorre quando os serviços do MIMIX são solicitados. Os serviços do MIMIX devem poder ser utilizados, independente das plataformas de software e de hardware empregados pelos membros. Além disso, não deve ser obrigatório utilizar o cliente para fazer solicitações aos serviços, deve ser possível fazê-las diretamente das ferramentas CASE utilizadas. O MIMIX deve funcionar como um repositório centralizado, onde residem todas as informações sobre o trabalho que a equipe está realizando. A comunicação entre os membros deve ser feita através do MIMIX.

Para fornecer o suporte necessário ao cenário do trabalho cooperativo descrito, faz parte do escopo do MIMIX, prover serviços para a criação de um espaço compartilhado para os usuários que irão trabalhar nele. O MIMIX deve, também, permitir o gerenciamento dos diferentes papéis dos usuários, restringindo seu acesso aos documentos armazenados, quando necessário.

O MIMIX deve permitir que os membros trabalhem individualmente na edição de seus modelos, publicando seu trabalho para a equipe quando desejarem. Para isso, devem utilizar o serviço de armazenamento (publicação) do MIMIX. A partir desse momento, o documento armazenado no repositório centralizado do MIMIX, deve ficar disponível para os demais membros do espaço de trabalho compartilhado. O documento pode estar protegido, estando disponível somente para a consulta, ou pode estar liberado para ser recuperado e alterado por outros membros da equipe. Cada vez que um membro fizer modificações e salvar um documento que já foi previamente armazenado no MIMIX, uma nova versão desse documento deve ser gerada. O MIMIX deve ser o responsável pelo controle dessas versões, permitindo o armazenamento, consulta, recuperação, comparação e união das mesmas. O MIMIX deve guardar informações – o que, quem e quando – sobre as versões armazenadas. Através dessas informações, é possível fornecer um histórico de evolução do trabalho dos membros da equipe. A criação e o controle de versões deve ser utilizado, principalmente, para tentar evitar conflitos entre o trabalho paralelo dos usuários e gerenciar as modificações realizadas, procurando evitar as redundâncias e o desperdício de trabalho.

Um requisito importante do MIMIX é fornecer informações de percepção do trabalho da equipe e individual de cada membro. O foco do trabalho cooperativo apoiado pelo MIMIX é o produto que está sendo construído. No cenário de trabalho descrito, as mudanças realizadas nos documentos armazenados não são facilmente percebidas. O MIMIX deve ter como foco fornecer informações de percepção do produto compartilhado.

Baseado na descrição do cenário que o MIMIX pretende apoiar (figura 8), é possível citar alguns serviços do MIMIX:

- **Independência:** O MIMIX deve apoiar o trabalho dos membros da equipe e permitir a comunicação entre eles, independente da ferramenta CASE e da plataforma de hardware que estejam sendo utilizadas.

- **Flexibilidade:** Deve ser possível utilizar um subconjunto dos serviços do MIMIX ou todos os serviços disponibilizados. Além disso, deve ser possível adicionar novos serviços de acordo com as necessidades do trabalho desenvolvido.

- **Persistência do produto:** O MIMIX deve armazenar os estados intermediários do produto (versões) que está sendo desenvolvido. Essa persistência é necessária para apoiar as interações assíncronas e fornecer informações de percepção do produto.

- **Percepção do produto:** O MIMIX deve possuir serviços que forneçam informações sobre o produto compartilhado e suas versões. Esses serviços devem permitir, principalmente, a consulta, recuperação, comparação e união de versões.

- **Suporte a diferentes tipos de usuários:** Os membros de uma equipe possuem papéis diferentes no processo de desenvolvimento. Esses papéis restringem seu acesso às informações compartilhadas. O MIMIX deve disponibilizar as informações sobre o produto respeitando a limitação do papel de um usuário.

III.2 Principais Serviços do MIMIX

Considerando o cenário apresentado no item III.1 é possível identificar a necessidade dos seguintes serviços no MIMIX:

III.2.1 Persistência de documentos

Durante o trabalho de modelagem cooperativa é preciso garantir a persistência do conhecimento comum da equipe, para que ele possa ser compartilhado. A persistência é importante para evitar que as informações sobre o trabalho da equipe e sobre os produtos gerados sejam perdidas.

O MIMIX deve utilizar um repositório centralizado para armazenar os documentos gerados pela equipe de trabalho e suas versões. Devido aos problemas de utilizar os SGBDs tradicionais em aplicações cooperativas, apresentados no capítulo II, como, por exemplo, o controle de acesso muito restritivo, a falta de suporte a transações longas, a utilização de bloqueios para controle de concorrência e a falta de percepção, o MIMIX precisa implementar alguns mecanismos de persistência de dados, que são necessários para o suporte da modelagem cooperativa. Dentre esses mecanismos é possível destacar:

III.2.1.1 Controle de Acesso

O MIMIX precisa prover serviços que permitam a criação de um espaço de trabalho compartilhado e de seus usuários, onde cada um possua um papel, que restrinja seu acesso aos documentos armazenados.

O usuário que criou o espaço passa a ser o coordenador, e os papéis dos demais podem ser alterados por ele durante o trabalho. Permitir a troca de papéis é fundamental, pois a limitação de um usuário a um determinado papel pode restringir sua participação no trabalho cooperativo. Um usuário pode também possuir múltiplos papéis, pois devido à dinâmica do trabalho cooperativo, uma tarefa delegada para um determinado tipo de participante pode precisar ser desempenhada por um outro.

III.2.1.2 Gerência de Documentos

O MIMIX precisa prover serviços que permitam armazenar, consultar e recuperar documentos. Os usuários podem trabalhar individualmente, quando desejarem disponibilizar seu trabalho para o resto da equipe, geram um arquivo e utilizam o serviço do MIMIX que o armazena em seu repositório. Esse documento passa a estar disponível para os demais usuários, que tiverem acesso ao espaço de trabalho compartilhado. O serviço de consulta do MIMIX deve listar todos os documentos armazenados no espaço de trabalho, seu autor e a data de publicação. Com essas informações, os usuários podem observar o andamento do trabalho. Através do serviço de recuperação do MIMIX, os usuários podem obter uma cópia de um documento, para continuar o trabalho ou fazer modificações.

III.2.1.3 Controle de Versões

Assume-se que o trabalho cooperativo apoiado pelo MIMIX seja realizado de forma incremental e evolutiva, portanto, surge a necessidade de manter partes alternativas (versões) dos trabalhos, para avaliação e comparação. Os usuários, geralmente, precisam trabalhar simultaneamente em um mesmo modelo. Para resolver os problemas de concorrência e conflitos gerados por esse trabalho simultâneo, o MIMIX utiliza um mecanismo de versionamento.

Cada vez que um usuário armazenar um documento que já foi previamente publicado no MIMIX, ele deve gerar uma nova versão do mesmo. O MIMIX deve ser responsável pelo controle e gerência das versões. Para isso, deve guardar informações – o que, quem e quando – sobre as versões armazenadas. O que, se refere a qual versão, quem, se refere ao autor (membro que a gerou) e quando, se refere à data que foi armazenada. O MIMIX deve permitir a consulta, recuperação, comparação e união das mesmas. Permitir a comparação de versões é importante para que os usuários possam observar a evolução do trabalho e detectar possíveis erros introduzidos.

A união de trabalhos paralelos, também é muito importante para o trabalho. No MIMIX, a união deve poder ser feita de forma automática (sem a intervenção do usuário), ou semi-automática (com intervenção do usuário). Na forma de união automática, o usuário escolhe dois documentos que deseja unir e qual deles tem a

prioridade. Assim, em caso de conflitos, o MIMIX deve dar preferência a este documento. Na forma semi-automática, o usuário pode comparar os documentos e escolher as partes de cada um que quer manter na união. O MIMIX deve resolver problemas sintáticos durante a união, por exemplo, se um usuário exclui duas classes da união e mantém o relacionamento entre elas, o MIMIX deve entender como erro e excluir também o relacionamento.

Não faz parte do escopo do MIMIX, implementar algum tipo de bloqueio de versões. O uso de bloqueios impossibilita o paralelismo de atividades que concorrem pela versão bloqueada. Contudo, se um usuário desejar publicar para a equipe uma versão parcial, que ele ainda não finalizou, deve ser possível armazená-la com uma proteção. Uma versão protegida pode ser consultada ou comparada com outra. Entretanto, não pode ser recuperada pelos usuários, nem ser unida à outra versão. Esse mecanismo é útil, por exemplo, quando um usuário deseja publicar o resultado intermediário de um trabalho longo, para que os demais tenham conhecimento do que ele está fazendo. Quando esse usuário finaliza o trabalho, deve utilizar o serviço do MIMIX que retira a proteção. Esse serviço deve sobrescrever a versão protegida com a nova, que passa a ficar disponível para os demais usuários.

III.2.2 Gerência de Configuração

Configuração de software é o nome dado a toda informação produzida durante o processo de desenvolvimento. Essa informação é constituída por objetos (ou itens) de configuração de software, como, por exemplo, programas de computador, documentos que descrevem esses programas e dados.

Os objetos de configuração de software gerenciados pelo MIMIX são os documentos produzidos pelos membros da equipe. Esses documentos são modificados ao longo do trabalho, para construir o produto final desejado. É preciso, portanto, identificar, organizar e controlar essas modificações. Para isso, o MIMIX deve utilizar o mecanismo de controle de versões. Esse mecanismo é bastante utilizado nos sistemas de gerência de configuração, que tem como objetivo: automatizar a busca de documentos, prevenir conflitos entre os membros da equipe, recuperar versões prévias, permitir o

desenvolvimento paralelo, etc. A utilização do versionamento pelo MIMIX foi apresentado anteriormente no item III.2.1.3.

III.2.3 Suporte à Percepção

O MIMIX se propõe a contemplar a forma de trabalho assíncrona. A falta de informações de percepção nesse caso pode gerar um estado de solidão e inércia para os membros, devido ao desconhecimento do trabalho da equipe. Nesse modo de trabalho, os membros são informados sobre interações que não tenham participado, em um momento posterior à sua ocorrência. O suporte à percepção deve ser derivado de uma interpretação resumida de uma seqüência de eventos, ocorridos em um espaço de tempo, procurando filtrar as informações que não sejam do interesse de um membro, ou que ele já tenha conhecimento.

O modo de entrega das informações de percepção a ser utilizado pelo MIMIX é o **PUXE**. Nesse modo, os membros obtêm as informações através de solicitações explícitas aos serviços de percepção do MIMIX.

Para o trabalho apoiado pelo MIMIX, o mais importante é o produto que está sendo construído, no caso, os modelos UML do sistema que está sendo desenvolvido pela equipe. Por isso, o MIMIX tem como foco fornecer informações de **percepção do produto**. Essas informações são importantes para que os membros possam observar o desenvolvimento do trabalho.

Os serviços de percepção disponibilizados pelo MIMIX, são detalhados no capítulo IV. Dentre eles devem estar incluídos:

- **Armazenamento de versões:** O MIMIX deve armazenar as versões dos documentos criados. Através de um serviço que compare essas versões os usuários podem perceber as modificações que ocorreram no produto. Com essas informações, eles podem orientar os seus trabalhos, discutir pontos de vista divergentes, constatar possíveis erros, etc.

- **Registro por usuário das modificações:** O MIMIX deve armazenar informações, por usuário, das modificações realizadas. Cada vez que um documento (ou versão) é armazenado por um usuário, o MIMIX deve atualizar sua base de dados, com informações sobre: que usuário fez a publicação (quem), a data da publicação (quando), qual foi o documento (o que) e o comentário "versão armazenada". Quando um usuário desejar saber quais são as últimas modificações disponíveis para ele, deve utilizar um serviço específico, que consulte a base de dados. O MIMIX deve enviar as informações solicitadas para o usuário, sem redundâncias, como, por exemplo: enviar informações que o usuário já tenha conhecimento ou que ele mesmo tenha armazenado. Com essas informações, é possível rastrear mudanças nos documentos, identificar que versão introduziu determinado erro e voltar a versão anterior a esta.

- **Registro por usuário de liberação de versão:** O MIMIX também deve armazenar, por usuário, informações sobre a liberação de uma versão protegida. Quando um usuário chamar um serviço que retire a proteção, a versão protegida deve ser sobrescrita com a nova, que passa a ficar disponível para os demais. O MIMIX deve armazenar as informações - que, quando e o que - com o comentário "versão desprotegida". Essa informação é importante para que os usuários tenham o conhecimento de que a versão está disponível para o trabalho.

- **Registro único de documentos recuperados:** O MIMIX deve armazenar informações - quem, quando e o que - dos documentos recuperados pelos usuários. Essas informações devem ficar disponíveis para todos os usuários. Com essa informação é possível fornecer serviços como: "Quem está trabalhando com a versão 2 do documento X"? Esse tipo de informação é importante, para constatar, por exemplo, que um determinado usuário está trabalhando com uma versão muito anterior a atual de um documento, e que provavelmente está ultrapassada.

III.2.4 Comunicação entre os Membros

A comunicação é essencial para o trabalho cooperativo, para que os membros da equipe de trabalho possam trocar conhecimento, discutir idéias, resolver problemas, etc. Quando os membros trabalham dispersos geograficamente, é preciso fornecer mecanismos que permitam a comunicação, como se eles estivessem trabalhando próximos fisicamente.

O MIMIX tem como objetivo fornecer mecanismos para permitir que os membros possam se comunicar e trabalhar cooperativamente, independente da plataforma de hardware em que estejam trabalhando e da ferramenta CASE utilizada. O MIMIX também deve possuir um serviço, que permita o envio de mensagens texto entre os membros da equipe. Além disso, o MIMIX deve manter o registro da comunicação realizada e deve disponibilizar essas informações para os membros da equipe.

III.3 Restrições

Algumas restrições foram impostas ao projeto do MIMIX, para delimitar o escopo da proposta, uma vez que o problema de apoiar a modelagem cooperativa de software é bastante complexo.

As principais restrições estabelecidas foram:

- **Forma de interação:** O MIMIX deve prover serviços que apóiem o trabalho cooperativo assíncrono.

- **Formato de dados:** As ferramentas CASE, geralmente, armazenam seus dados em formatos proprietários. Mesmo que este seja conhecido, não faz parte do escopo do MIMIX tratar as conversões entre os formatos, de todas as ferramentas existentes. Para utilizar o MIMIX no suporte ao trabalho, é necessário que as ferramentas CASE sejam baseadas em UML e possuam o mecanismo de importação/exportação dos seus dados em um formato padrão. O MIMIX deve utilizar como padrão documentos XMI (*XML*

Metadata Interchange). Muitas ferramentas CASE disponíveis no mercado, já importam e exportam seus dados nesse formato.

- **Granularidade da informação:** A unidade de trabalho do MIMIX deve ser um documento XMI completo. Esses arquivos contêm os modelos UML criados pelos usuários nas ferramentas CASE utilizadas no trabalho.

- **Apresentação da informação:** O usuário final do MIMIX não é diretamente um indivíduo, mas uma aplicação sobre a qual o indivíduo atua. O MIMIX deve possuir um módulo cliente, que se preocupe com a forma como as informações são apresentadas para o usuário. Entretanto, deve ser possível construir outras aplicações clientes para interagir com o MIMIX, ou ainda, encapsular as chamadas aos serviços dentro das ferramentas CASE. Nesse caso, a apresentação das informações deve ser tratada pela aplicação que estiver utilizando os serviços.

- **Modo de entrega das informações:** O MIMIX deve disponibilizar as informações de percepção do produto no modo de entrega **PUXE**, onde os usuários fazem solicitações aos serviços para obtê-las.

III.4 Funcionamento do MIMIX

A partir dos serviços identificados nos itens anteriores, é possível detalhar algumas funcionalidades principais do MIMIX. Essas funcionalidades podem ser resumidas através do diagrama de casos de uso do MIMIX (figura 9). O ator, ou seja, a entidade externa que utilizará os serviços do MIMIX, é um membro da equipe. Entretanto, o membro não faz chamadas aos serviços diretamente. As chamadas são feitas através do módulo cliente, ou de uma ferramenta CASE.

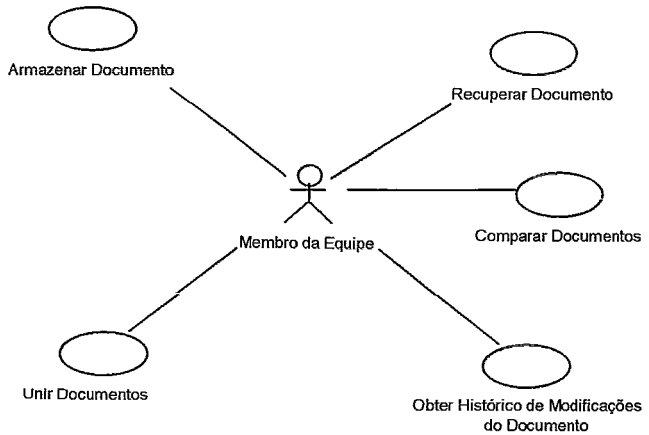


figura 9. Diagrama de casos de uso com as principais funcionalidades do MIMIX.

A figura 10 ilustra um exemplo de funcionamento do MIMIX. Essa figura demonstra uma interação típica entre os membros da equipe, apoiada pelo MIMIX.

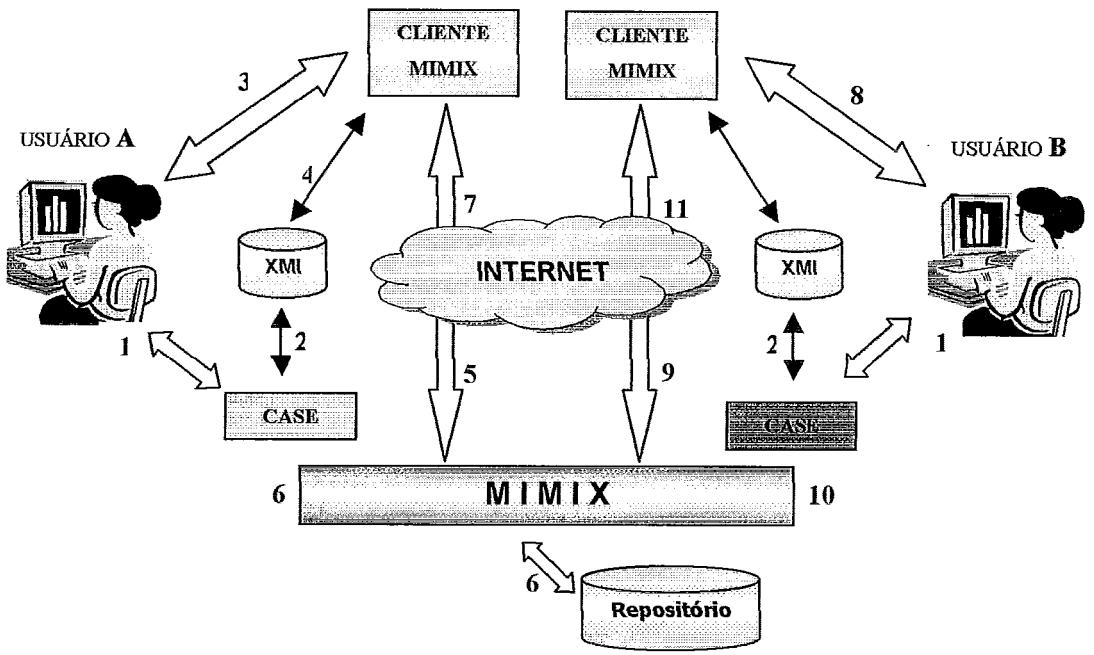


figura 10. Esquema de funcionamento do MIMIX.

1. Cada usuário trabalha com a sua ferramenta CASE.

2. As ferramentas CASE exportam os modelos gerados na forma de documentos XMI. Esses documentos são armazenados no repositório local de cada usuário.

3. Através do módulo cliente do MIMIX, o usuário **A** decide tornar disponível (publicar) uma nova versão de um documento para a equipe.

4. No módulo cliente, o usuário informa o caminho do documento XMI que deseja armazenar o servidor MIMIX, que se encontra em seu repositório local. Utilizando o módulo cliente solicita o serviço de armazenamento do MIMIX.

5. O documento é enviado para o servidor MIMIX.

6. O MIMIX valida o documento XMI, cria a nova versão e a armazena em seu repositório.

7. O MIMIX envia uma mensagem de confirmação para o módulo cliente.

8. O usuário **B** utiliza o módulo cliente para consultar o andamento do trabalho da equipe. Ele chama o serviço que pergunta quais são as modificações relevantes e disponíveis para esse usuário.

9. A solicitação é enviada para o servidor MIMIX.

10. O MIMIX processa a solicitação e recupera as informações para o usuário **B**. Portanto, de acordo com o exemplo da figura 10, essas informações são referentes a publicação feita pelo usuário **A**. Notificando que usuário fez uma nova publicação (quem), em que data (quando), qual foi o documento (o que) e o comentário "versão armazenada".

11. O MIMIX envia essas informações para o módulo cliente, tornando-as disponíveis para o usuário **B**.

III.5 Arquitetura do MIMIX

A arquitetura do MIMIX é dividida em duas camadas: cliente e servidor. O cliente possui interface com o usuário, podendo ser qualquer software que submeta pedidos aos serviços disponibilizados pelo servidor MIMIX. Assim, é possível construir um módulo cliente que utilize esses serviços, ou embutir diretamente nas ferramentas CASE, as solicitações feitas ao servidor. O servidor recebe as solicitações, efetua o processamento e responde os pedidos do cliente. O cliente pode estar em uma máquina, utilizando os serviços do servidor localizado em outra máquina, ou essa divisão pode ser apenas lógica, estando todos fisicamente em um mesmo local. A figura 11 ilustra a arquitetura do MIMIX.

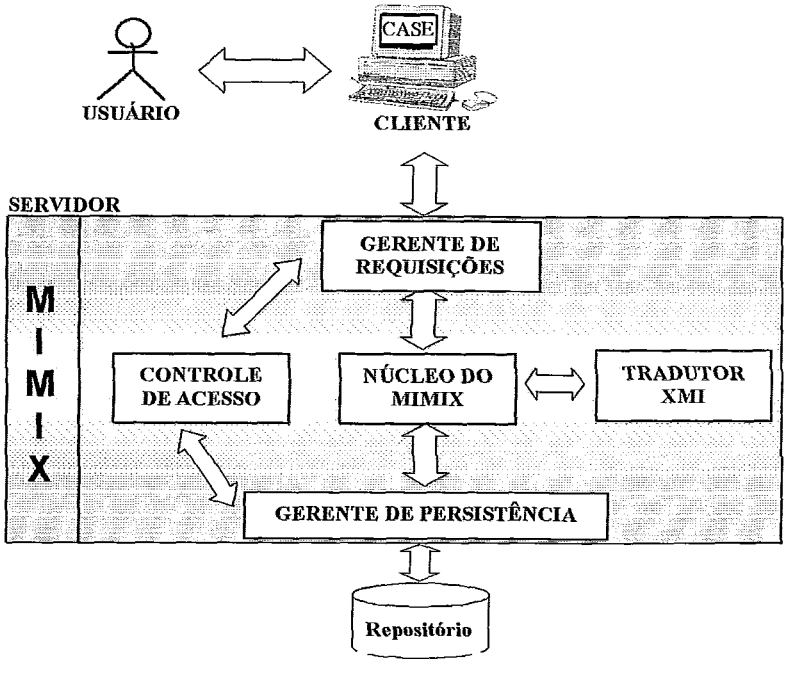


figura 11. Arquitetura do MIMIX

As próximas seções apresentam a definição de cada um dos componentes da arquitetura do servidor MIMIX.

III.5.1 Gerente de Requisições

Para utilizar os serviços do MIMIX, inicialmente, o usuário precisa solicitar o serviço de conexão (*login*). O gerente de requisições cria uma sessão ativa para esse usuário, sendo o responsável pelo controle da mesma.

O gerente de requisições recebe as solicitações do cliente aos serviços do MIMIX. Ele é o responsável pela validação dessas solicitações, verificando, por exemplo, se o serviço existe, se os parâmetros enviados estão corretos e se o usuário que solicitou o serviço tem permissão de acesso (perguntando ao componente de controle de acesso). Uma vez que a solicitação é validada, o gerente de requisições a envia para o núcleo do MIMIX. O gerente de requisições também é responsável pelo envio da resposta ao cliente.

III.5.2 Núcleo do MIMIX

É o responsável pela execução, propriamente dita, dos serviços do MIMIX. Dentre suas funcionalidades, é possível destacar:

- Criação de espaço compartilhado;
- Validação de documentos XMI;
- Criação e gerenciamento de versões de documentos;
- Comparação e união de versões de documentos;
- Solicitação de armazenamento e recuperação de documentos e suas versões.

Para fornecer informações de percepção, o núcleo registra informações sobre todo o processamento realizado. Sendo também responsável pelo registro das "novidades" (modificações realizadas pela equipe que um usuário precisa ter conhecimento) para cada usuário.

III.5.3 Tradutor XMI

O tradutor XMI é utilizado quando são solicitados serviços, que para enviar a resposta, o MIMIX precisa entender o conteúdo dos documentos XMI. Esses serviços estão relacionados com a comparação e união de versões. Se o MIMIX está comparando modelos de classe, por exemplo, ele precisa saber quais são as classes, atributos, métodos, relacionamentos, para informar as diferenças, de forma correta. Na união automática (sem a interferência do usuário) o MIMIX verifica as diferenças (comparação) e sugere como deve ser o documento resultante, por isso precisa conhecer o significado dos elementos. O MIMIX, também, reconhece erros de sintática na união semi-automática (com a interferência do usuário). Isso ocorre quando, por exemplo, um usuário opta por excluir uma das classes de um relacionamento, mas mantém o relacionamento. Nesse caso o MIMIX também exclui esse relacionamento.

Quando são solicitados serviços, como, por exemplo: "Quais são os documentos armazenados no projeto X"? ; "Qual é a última versão do documento A"? Não é necessário traduzir os documentos. O MIMIX recupera apenas as informações solicitadas sobre os documentos, que estão armazenados em seu repositório, e envia a resposta para quem solicitou o serviço. Nesse tipo de serviço não é necessário utilizar o tradutor, já que os documentos armazenados precisam apenas ser repassados, para o quem os solicitou.

III.5.4 Gerente de persistência

É responsável pelo armazenamento e recuperação dos documentos XMI no repositório do MIMIX. Recebe as solicitações do Núcleo do MIMIX.

III.5.5 Controle de Acesso

É o responsável pela criação dos usuários e do controle do seu acesso aos serviços do MIMIX.

III.6 Análise do MIMIX em Relação aos Trabalhos Relacionados

Durante o estudo feito para a elaboração do MIMIX, foram encontrados na literatura trabalhos que propõem a integração de ferramentas CASE distintas em um ambiente único, chamados de Ambientes de Desenvolvimento de Software ou ADS (MIAN et al., 2001), como, por exemplo, o OdysseyShare (ODYSSEY, 2004). Enquanto uma ferramenta CASE se destina a apoiar uma etapa do desenvolvimento de software, um ADS se propõe a disponibilizar ferramentas de apoio ao longo de todo o processo de desenvolvimento, inclusive, para apoiar a atividade de modelagem de software. Esses trabalhos discutem vários níveis de integração: de dados, de apresentação, de controle e de conhecimento (CHEN e NORMAN, 1992) e (MIAN et al., 2001). O MIMIX não segue essa abordagem, não tem como objetivo criar um ambiente único. Seu objetivo é apoiar a cooperação durante uma das atividades do desenvolvimento, a de modelagem de software, e garantir a interoperabilidade entre ferramentas CASE, baseadas em UML. Ambientes de desenvolvimento que forneçam suporte à modelagem cooperativa de software, como o OdysseyShare, podem utilizar o MIMIX para trocar modelos e *trabalhar* cooperativamente com outras ferramentas CASE.

Foram encontrados trabalhos, que não são específicos para ferramentas CASE, e propõem a introdução de uma camada adicional entre a aplicação e o banco de dados para dar suporte explicitamente ao controle de cooperação, fazendo um mapeamento de todas as atividades que ocorrem na aplicação em unidades operacionais que manipulam dados compartilhados, como, por exemplo, o CONCORD (RITTER, 1994). Esses trabalhos se preocupam com a ordem que as tarefas devem ser executadas (*workflow*). Essa também não é a abordagem do MIMIX. No MIMIX, não é tratada a dependência e a ordem de execução entre as tarefas desempenhadas pelos membros durante o desenvolvimento. A principal preocupação no MIMIX, é permitir que as ferramentas CASE possam trocar documentos XMI, que contém os modelos gerados, entre elas. O MIMIX tem como foco, fornecer informações sobre as versões dos modelos, que são resultado do trabalho paralelo dos membros, para a equipe de desenvolvimento. O MIMIX segue a abordagem das ferramentas de gerência de configuração, que utilizam o mecanismo de controle de versões.

Embora, não implemente mecanismos utilizados por essas ferramentas, como, por exemplo, *chek-in/chek-out*, para não restringir o trabalho da equipe de desenvolvimento

O Knight (DAMM et al, 2000) e (HANSEN, 1999) propõe uma arquitetura para a integração de ferramentas CASE e utiliza o XMI como padrão para troca de dados entre as mesmas. Na especificação do Knight foram apresentadas considerações sobre a experiência da utilização do XMI, que contribuíram para o desenvolvimento do MIMIX. O Knight deu origem a ferramenta comercial Ideogramic UML. Enquanto o MIMIX se propõe a apoiar o trabalho distribuído, a Ideogramic UML se concentra em apoiar a cooperação entre indivíduos com a proximidade física. Ela possui uma interface cooperativa, quadro branco eletrônico, onde um grupo pode trabalhar simultaneamente editando os modelos. Nesse caso, não existe tratamento de conflitos com apoio da ferramenta, que são resolvidos entre os membros da equipe. Não existe, também, o registro do trabalho cooperativo realizado pelos membros da equipe. Na Ideogramic, as informações de percepção são obtidas através da observação dos membros, durante desenvolvimento do trabalho. Assim, se um usuário estiver ausente durante uma seção de trabalho, não terá conhecimento do que foi feito. A versão da Ideogramic UML para computadores (*desktop*) é, basicamente, uma ferramenta CASE que não fornece suporte à cooperação.

O Gossip (FARSHCHIAN, 2000) e (FARSHCHIAN, 2001) tem como objetivo apoiar o desenvolvimento cooperativo de software, e possui características interessantes que contribuíram para o desenvolvimento do MIMIX. O Gossip não tem como objetivo permitir a interoperabilidade entre as ferramentas CASE, nem permitir a edição cooperativa de modelos. Os membros da equipe de desenvolvimento trabalham com suas ferramentas CASE, informando para o Gossip sobre as modificações realizadas, que desejam disponibilizar para toda equipe. O Gossip notifica a equipe sobre as modificações realizadas. O Gossip é basicamente um servidor de notificação, que ao contrário do MIMIX, entrega suas informações utilizando o modo **EMPURRE**. Nesse modo, o servidor envia as informações para o cliente por iniciativa própria, podendo gerar sobrecarga de informações para o usuário. O principal objetivo do Gossip é avisar aos membros da equipe que modificações foram feitas no produto compartilhado, ou seja, fornecer percepção do

produto. Entretanto, ele não armazena o produto e suas versões, armazenando somente o mínimo de informações necessárias para avisar que um determinado produto foi alterado. Assim, ele não permite a comparação e união de versões.

A figura 12 apresenta o resumo da avaliação das ferramentas cooperativas estudadas e do MIMIX. Nessa figura, são relacionadas as principais características do MIMIX e quais ferramentas estudadas possuem (SIM) ou não possuem (NÃO) essas características.

	GOSSIP	IDEOGRAMIC UML	MIMIX
Notificação das Alterações	SIM		SIM
Utilização do XMI		SIM	SIM
Suporte à Cooperação	SIM	SIM/NÃO	SIM
Gerência de Versões			SIM
Percepção do Produto	SIM		SIM

figura 12. Avaliação das ferramentas cooperativas e do MIMIX.

Para verificar a viabilidade de implementação da proposta do MIMIX foi desenvolvido um protótipo, que é descrito no próximo capítulo.

IV O Protótipo do MIMIX

O protótipo do MIMIX fornece um conjunto de serviços, que podem ser utilizados pelos membros de uma equipe de desenvolvimento para trabalhar cooperativamente utilizando ferramentas CASE heterogêneas

Para alcançar a independência de plataforma pretendida e facilitar a interoperação entre as ferramentas CASE utilizadas pelos membros, foram estudadas algumas soluções tecnológicas que simplificassem o desenvolvimento e ampliassem a utilização dos serviços do protótipo. A base para a comunicação em aplicações distribuídas é o mecanismo de RPC (*Remote Procedure Call*). Foram analisadas algumas alternativas que implementam o mecanismo de RPC, são elas: CORBA, RMI, DCOM e serviços Web (*Web Services*). O anexo 2 apresenta o estudo dessas alternativas.

A alternativa escolhida foi implementar as funcionalidades do MIMIX na forma de serviços Web. O principal atrativo dos serviços Web é a capacidade de comunicação e utilização de serviços entre plataformas diferentes, aumentando a interoperabilidade e facilitando a integração entre as ferramentas CASE envolvidas no trabalho. Os serviços Web compreendem um conjunto de tecnologias de plataforma-neutra, designadas para facilitar a distribuição de serviços através de *Intranets* ou da *Internet*. A comunicação é feita através do protocolo SOAP (*Simple Object Access Protocol*), que é baseado em XML para a troca de informações. Os serviços Web, geralmente, utilizam o protocolo HTTP para o transporte de mensagens SOAP. Isso porque, sua utilização facilita o transporte de mensagens SOAP entre sistemas, já que os *firewalls* normalmente não bloqueiam o acesso à porta HTTP.

Os serviços podem ser implementados usando qualquer linguagem de programação (*Java*, *C++*, *Visual Basic*,etc.), e em qualquer plataforma. Um cliente de um Serviço Web também pode ser escrito usando qualquer linguagem e em qualquer plataforma. Por exemplo, um cliente escrito em *Delphi* na plataforma *Windows* pode utilizar serviços de um Serviço Web escrito em *Java* na plataforma *Linux*. O cliente não precisa se preocupar em como o serviço foi implementado.

Os serviços são definidos através de uma linguagem baseada em XML, chamada WSDL. Essa linguagem também descreve como eles podem ser invocados, especificando o formato das mensagens esperadas pelos mesmos. O arquivo WSDL descreve como a parte consumidora e a produtora de serviços Web deverão se comunicar, ou seja, pode ser utilizado como uma "receita" para gerar as mensagens SOAP, que irão acessar os serviços. Utilizando o arquivo WSDL gerado pelo MIMIX (anexo 3) é possível escrever um programa que faça chamadas aos seus serviços. Essas chamadas também podem ser encapsuladas diretamente nas ferramentas CASE, não sendo necessário utilizar o módulo cliente. Além disso, é possível utilizar todos os serviços disponibilizados pelo MIMIX, ou apenas parte deles. As funcionalidades do MIMIX podem ser expandidas, criando novos serviços de acordo com as necessidades da equipe de trabalho.

Existem vários trabalhos, como foi mencionado anteriormente no capítulo III, item III.6, que tentam achar uma forma padrão para o intercâmbio de dados entre as ferramentas CASE, permitindo a cooperação entre elas. Isso porque, as ferramentas, geralmente, utilizam um formato próprio para a serialização de dados, de acordo com seu fabricante. Atualmente, as ferramentas CASE mais utilizadas no mercado, incorporaram o mecanismo de intercâmbio de seus documentos no formato XMI (XML Metadata Interchange). Por isso, o MIMIX também utiliza esse padrão como formato para a troca de informações. Assim, as ferramentas CASE que desejarem trabalhar cooperativamente, com o suporte do MIMIX, precisam importar e exportar seus dados no formato XMI.

As próximas seções descrevem a implementação do protótipo do MIMIX, apresentando a infraestrutura utilizada para o desenvolvimento, o módulo cliente e o módulo servidor. A seguir, é apresentado um exemplo prático de utilização do MIMIX. Esse capítulo é finalizado com algumas considerações sobre a implementação.

IV.1 Infraestrutura de Desenvolvimento

A figura 14 ilustra a infraestrutura do ambiente de desenvolvimento do protótipo do MIMIX.

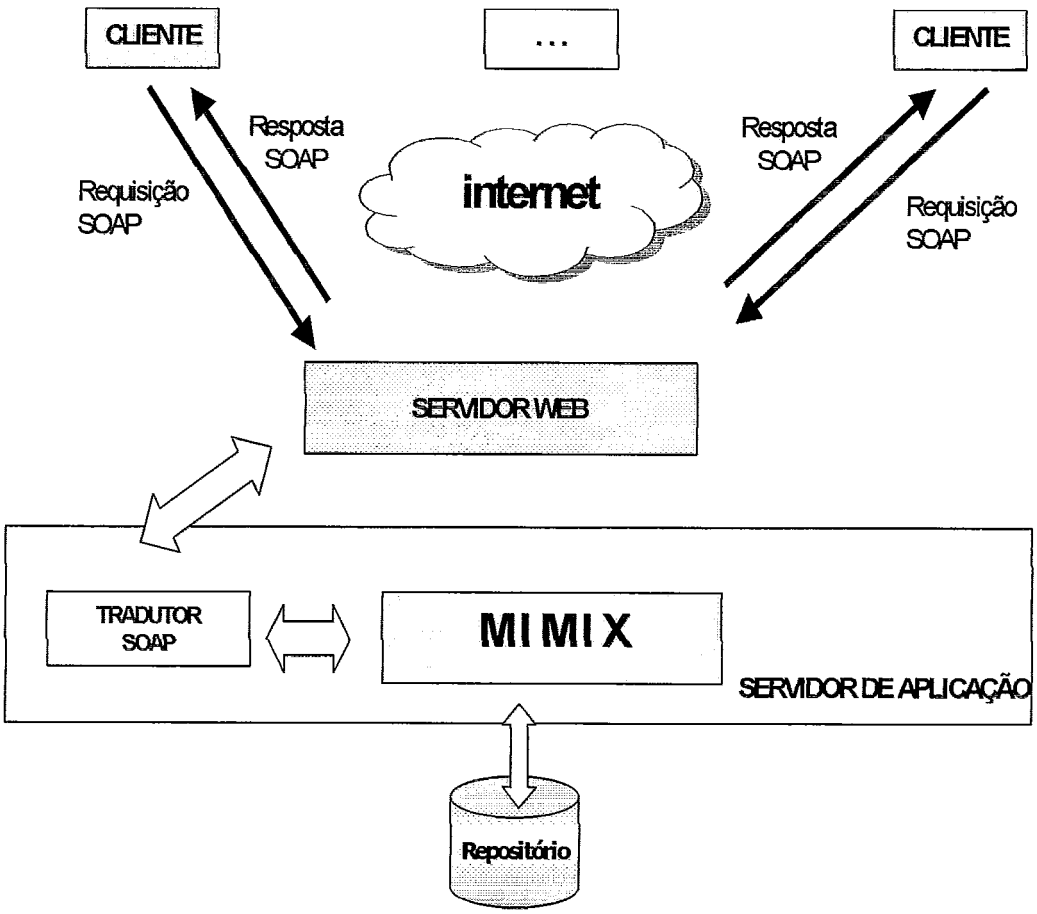


figura 13. Infraestrutura de Desenvolvimento do Protótipo.

O módulo cliente do protótipo foi desenvolvido na linguagem Delphi (Borland), e o módulo servidor em Java. A escolha de linguagens de programação diferentes teve como objetivo mostrar a independência de plataforma, alcançada com a utilização da tecnologia de serviços Web.

Cada usuário interage com o módulo servidor através de um módulo cliente. Os módulos clientes se comunicam com o módulo servidor através da Internet, utilizando o protocolo HTTP, para o transporte das mensagens SOAP de requisição e de resposta. As requisições são as chamadas aos serviços do módulo servidor. O servidor envia mensagens SOAP, contendo as respostas das requisições.

O ambiente que suporta o protótipo do servidor MIMIX é descrito a seguir. Os softwares utilizados e suas documentações podem ser obtidos em (APACHE, 1999):

- **Servidor Web:** Apache, versão para Linux.
- **Servidor de Aplicação:** Tomcat, integrado ao Apache.
- **Interpretador SOAP:** Axis.

IV.2 Módulo Cliente do MIMIX

O módulo cliente possui interface com o usuário, sendo o responsável pelas chamadas aos serviços do MIMIX. Esse módulo foi implementado para testar a utilização dos serviços implementados no módulo servidor. Utilizando o protótipo do módulo cliente, é possível acionar todas as funcionalidades disponíveis no módulo servidor. Esse protótipo está disponível em http://www.deljaick.com.br. A figura 14 ilustra um exemplo de utilização do módulo cliente.

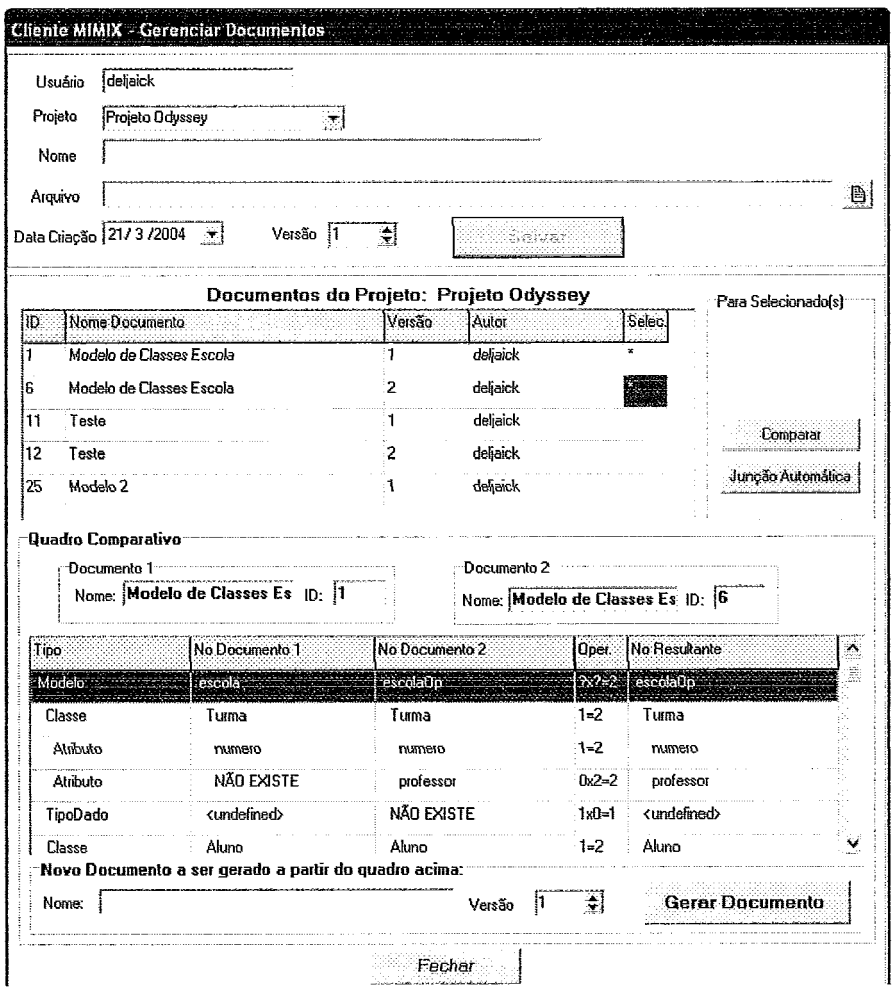


figura 14. Módulo Cliente do MIMIX

O exemplo da figura 14 exibe os documento armazenados no MIMIX, que pertencem a um projeto chamado Odyssey. Duas versões do documento "Modelo de Classes Escola" são selecionadas, para que seja feita a comparação entre elas. Um quadro comparativo (apresentado no final dessa figura) apresenta as diferenças entre os elementos que compõem os documentos.

O descritor dos serviços do MIMIX é um arquivo público, chamado Mimix.wsdl (anexo 3). Com base nesse descritor, é possível desenvolver novos módulos clientes do MIMIX, ou encapsular as chamadas aos serviços diretamente nas ferramentas CASE.

IV.3 Módulo Servidor do MIMIX

O módulo servidor contém os serviços implementados. Os serviços do MIMIX têm como objetivo, principalmente, fornecer informações de percepção do produto que está sendo construído. Esta seção descreve o módulo servidor e os serviços implementados no protótipo. O conjunto de serviços disponibilizados foi especificado após o estudo de trabalhos que propõem ferramentas para apoio à cooperação. Entretanto, de acordo com as necessidades da equipe de desenvolvimento, novos serviços podem ser adicionados ao MIMIX.

IV.3.1 Modelo de Classes

A figura 15 ilustra o modelo de classes do servidor MIMIX. O espaço compartilhado no MIMIX é chamado de projeto. Cada projeto possui um conjunto de usuários, que estão trabalhando cooperativamente, e um conjunto de documentos armazenados no repositório do MIMIX. Cada documento armazenado tem um autor (criador do documento). Os usuários só têm visibilidade dos documentos armazenados nos projetos a que possuem acesso. Um usuário conectado no MIMIX, constitui uma sessão ativa. O MIMIX registra todas as solicitações dos usuários ativos (atividade do grupo) para fornecer as informações de percepção sobre o trabalho de cada membro e da equipe.

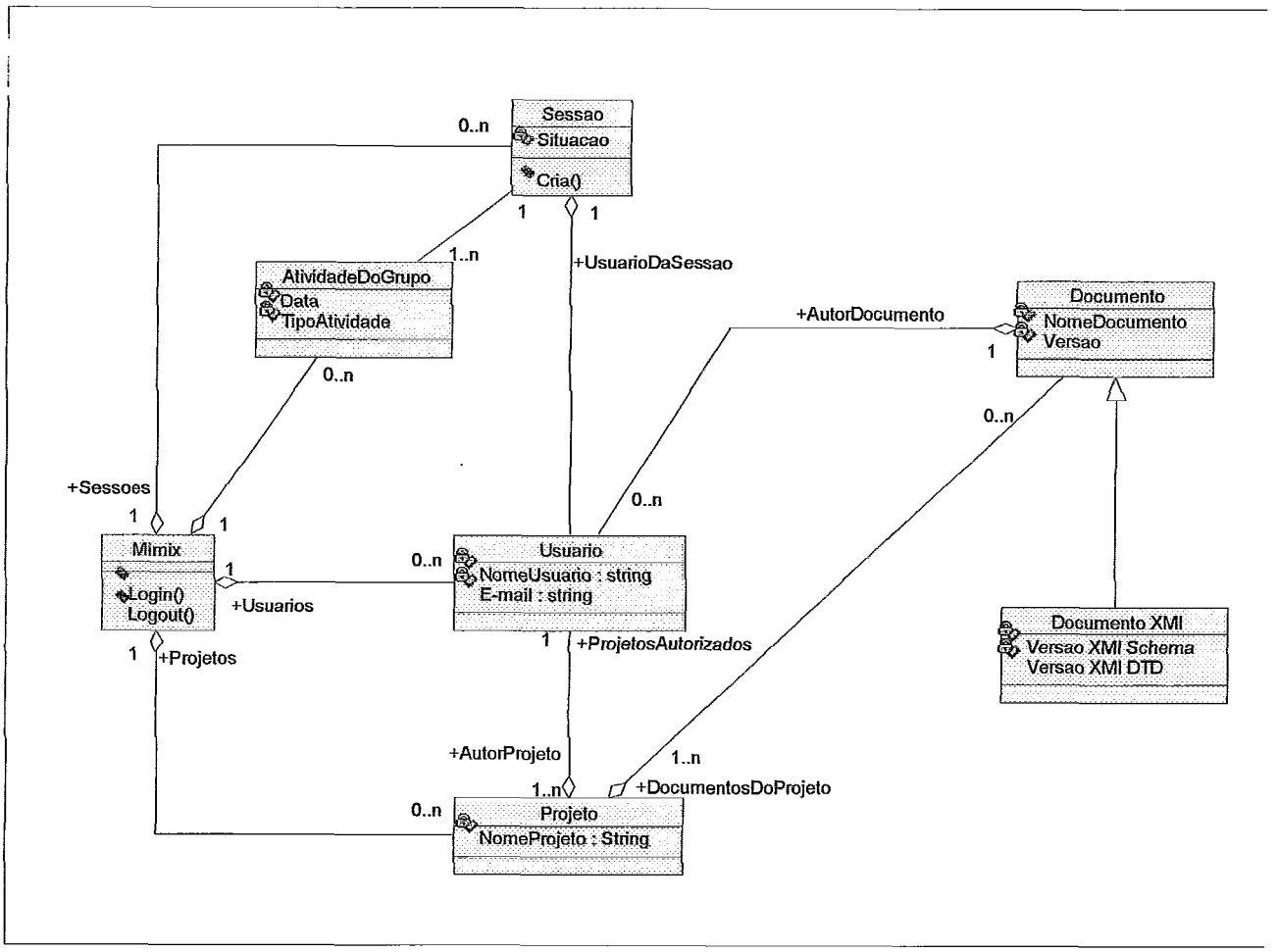


figura 15. Modelo de Classes do Módulo Servidor do MIMIX.

IV.3.2 Serviços do MIMIX

Uma característica importante dos serviços implementados no MIMIX é procurar não introduzir processos restritivos que dificultem o fluxo natural do trabalho cooperativo de um grupo. Para isso, foi realizado um estudo de quais casos podem ser tratados de forma automática e quais necessitam da interferência do usuário. Quanto mais preciso for esse diagnóstico, menos "ruído" o MIMIX introduzirá na comunicação do grupo.

Os serviços do MIMIX podem ser agrupados em três categorias: Controle de Acesso, Percepção e Percepção do Produto. A seguir, é apresentada uma breve descrição dos serviços mais relevantes implementados no protótipo.

- CONTROLE DE ACESSO:

1. **Conectar (*Login*):** Um usuário não conectado têm acesso a alguns serviços do MIMIX, mas esses serviços são basicamente de consulta de dados (somente para leitura) e não englobam os serviços referentes a documentos. Para poder usar os demais serviços, o usuário precisa se conectar, passando para o serviço seu nome e senha. Assim, o MIMIX cria uma nova sessão para esse usuário.

2. **Desconectar (*Logout*):** Quando o usuário se desconecta, sua sessão é encerrada.

3. **Criar Usuário:** Permite que um usuário se cadastre e crie uma senha. Esse usuário novo, a princípio, não tem permissão de acesso a nenhum projeto, mas pode criar um projeto novo e dar permissões para outros usuários (serviço 4), ou através do serviço "Enviar Mensagem Para Usuário" (serviço 11), pode solicitar a um outro usuário permissão para participar de um determinado projeto.

4. **Criar Projeto:** Projeto é o espaço compartilhado onde os usuários trabalham cooperativamente. Para criar um projeto, o usuário precisa estar conectado. Quando um projeto é criado, o MIMIX automaticamente concede permissão de acesso ao usuário que o criou. Esse usuário tem visibilidade total do projeto. É chamado de "autor do projeto".

5. **Associar Usuário ao Projeto:** Consiste em dar permissão a um determinado usuário para participar de um projeto. Assim que um projeto é criado, somente o autor do projeto pode conceder permissão de acesso a outros usuários. Os tipos de permissões são: **livre**, acesso a todos os dados

e documentos do projeto; **consulta**, só permite a consulta aos dados e documentos do projeto; **parcial**, permite a consulta aos dados e recuperação de documentos do projeto, mas não permite que o usuário armazene nenhum documento ou versão. O autor do projeto, ou um usuário com permissão livre, podem alterar as permissões dos demais usuários durante o trabalho, quando for necessário. Um usuário pode participar de projetos diferentes, possuindo permissões de acesso distintas em cada um.

6. **Desassociar Usuário do Projeto:** Consiste em retirar a permissão de um usuário para participar de um projeto. Somente o autor do projeto, ou um usuário com permissão livre, podem retirar as permissões de outros usuários.

- PERCEPÇÃO:

7. **Obtém Usuários:** Retorna a coleção de todos os usuários cadastrados no MIMIX.

8. **Obtém Usuário:** Retorna todos os dados cadastrais de um usuário específico.

9. **Obtém Usuários Conectados:** Retorna a coleção de todos os usuários que estão conectados no MIMIX, no instante em que o serviço for solicitado.

10. **Obtém Usuários do Projeto:** Retorna a coleção de todos os usuários de um projeto específico.

11. **Envia Mensagem Para Usuário:** Envia uma mensagem para um usuário. A mensagem pode ser enviada inclusive para usuários que não estejam conectados no momento.

12. **Obtém Mensagens:** Retorna uma coleção de mensagens endereçadas ao usuário que solicitou o serviço.

13. **Obtém Mensagem:** Retorna o texto de uma mensagem específica.

14. **Obtém Projetos:** Retorna as informações de todos os projetos cadastrados no MIMIX. Esse serviço pode ser utilizado mesmo por usuários desconectados, ou sem permissão aos projetos, somente para consulta.

15. **Obtém Projetos do Usuário:** Retorna as informações de todos os projetos que um determinado usuário tem acesso, e o tipo de permissão que ele tem em cada um. Somente para consulta. Pode ser utilizado por usuários desconectados ou sem permissão aos projetos.

- PERCEPÇÃO DO PRODUTO:

16. **Criar Documento:** O usuário precisa estar conectado para usar os serviços relacionados à percepção do produto. Esse serviço permite a criação de um documento em um projeto. O documento pode ser armazenado com uma proteção. Essa proteção não permite que ele seja recuperado por outros usuários, nem que seja feita sua união com outros documentos. Entretanto, é possível fazer a comparação de um documento protegido com outros documentos. Esse serviço verifica se o usuário tem permissão de acesso ao projeto no qual deseja criar um documento. Se esse documento já tiver sido armazenado anteriormente no MIMIX, o serviço cria uma nova versão dele.

17. **Retirar Proteção de Documento:** Esse serviço retira a proteção de um documento armazenado no MIMIX. Ao solicitar esse serviço, o usuário deve enviar para o MIMIX uma nova versão liberada desse documento. A versão protegida é sobrescrita com a nova versão, ficando disponível para os usuários do projeto.

18. **Obtém Documento:** Retorna o documento solicitado. Esse serviço verifica se o usuário tem permissão de acesso ao projeto ao qual esse documento pertence.

19. **Obtém Documentos do Usuário:** Retorna uma coleção contendo informações sobre todos os documentos que um usuário é autor.

20. **Obtém Documentos do Projeto:** Retorna uma coleção contendo informações sobre todos os documentos de um determinado projeto. Esse serviço verifica se o usuário tem permissão de acesso ao projeto.

21. **Validar Documento:** Esse serviço valida um documento, verificando se ele é um documento XMI válido e se está de acordo com o DTD XMI no qual se baseia.

22. **Excluir Documento:** Exclui um documento. Esse serviço verifica se o documento existe e se o usuário, que quer excluí-lo, tem permissão para fazê-lo.

23. **Obtém Usuários que Recuperaram um Documento:** Retorna uma coleção de usuários que solicitaram um determinado documento (serviço 17) armazenado no MIMIX. Utilizando esse serviço, é possível, por exemplo, saber ser um usuário está trabalhando com uma versão inadequada do documento.

24. **Obtém Usuários com Versão Desatualizada:** Retorna uma coleção de usuários, que ainda não solicitaram a última versão um determinado documento. Utilizando esse serviço, é possível, por exemplo, saber quais usuários estão trabalhando com uma versão ultrapassada de um documento.

25. **Obtém Modificações no Projeto:** Retorna uma coleção contendo informações sobre as ações dos usuários em um projeto, a partir de uma determinada data.

26. **Obtém Modificações no Projeto por Usuário:** Retorna para um usuário, uma coleção contendo informações sobre as ações dos demais membros de um projeto, desde a última vez que ele solicitou esse serviço.

27. **Comparar Documentos:** Esse serviço compara dois documentos. Embora o documento XMI seja um documento texto, o MIMIX não utiliza uma ferramenta de comparação de texto. Isso porque, a estrutura do documento XMI é diferente da estrutura de um documento texto. Além disso, a ordem que os elementos dos modelos são mapeados para um documento XMI não é padronizada. Utilizar uma ferramenta de comparação de texto, que verifica a posição do caractere no documento, poderia introduzir erros na comparação. Um exemplo, seria identificar uma classe e um método que possuem o mesmo nome e inferir que se trata do mesmo elemento. Portanto, o MIMIX precisa conhecer a semântica dos elementos comparados. Para isso, ele interpreta cada um dos documentos XMI, procura as igualdades e as diferenças entre eles, e retorna o resultado dessa análise. O algoritmo de comparação utilizado pelo MIMIX é descrito a seguir. No exemplo, são comparados os documentos A e B.

   1° Passo: Interpreta o documento A e monta uma árvore onde cada nó representa um elemento do modelo contido em A (figura 16).

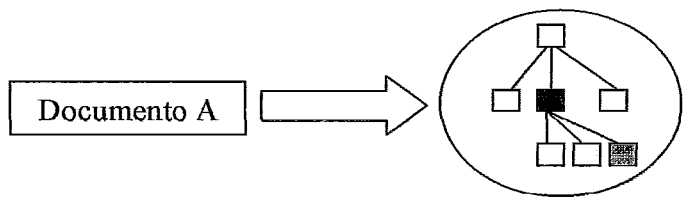


figura 16. Árvore do documento A gerada pelo MIMIX.

   2° Passo: Interpreta o documento B e monta uma árvore onde cada nó representa um elemento do modelo contido em B (figura 17).

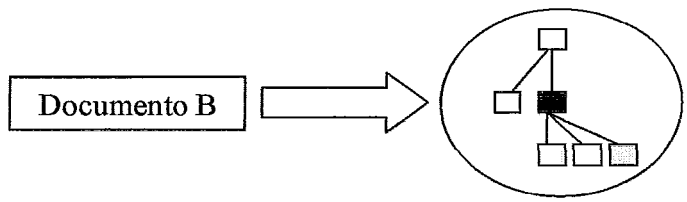


figura 17. Árvore do documento B gerada pelo MIMIX.

3º Passo: As duas árvores são então analisadas, gerando um quadro comparativo (figura 18).

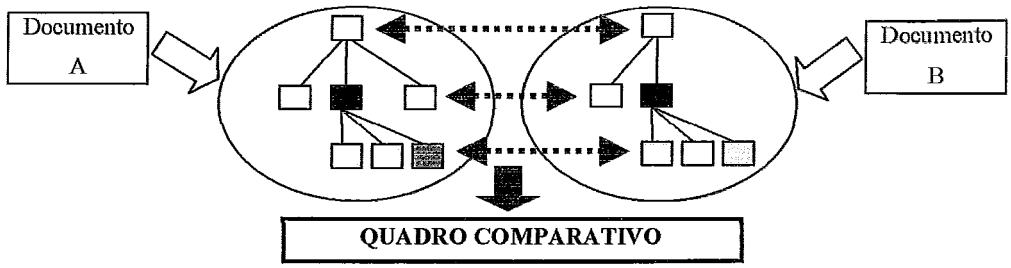
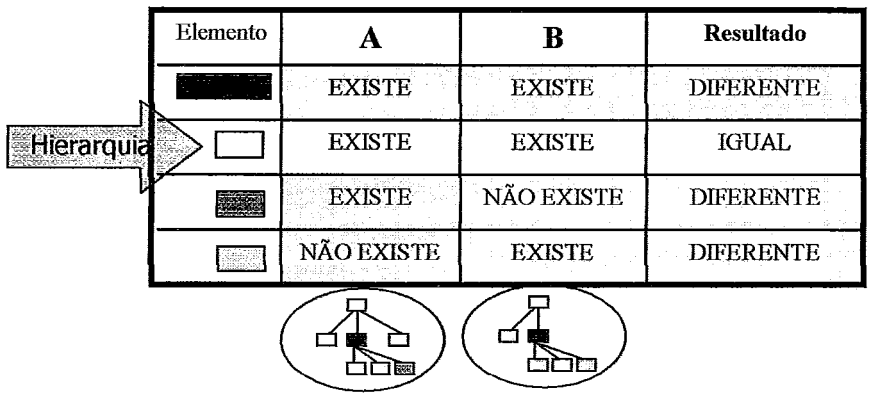


figura 18. Análise das árvores dos documentos A e B.

4º Passo: O quadro comparativo apresenta as igualdades e desigualdades entre os dois documentos, respeitando a hierarquia. Assim, se o elemento azul for uma classe, os elementos abaixo (amarelo, marrom e laranja) são atributos e/ou métodos dessa classe (figura 19).



Elemento	A	B	Resultado
	EXISTE	EXISTE	DIFERENTE
	EXISTE	EXISTE	IGUAL
	EXISTE	NÃO EXISTE	DIFERENTE
	NÃO EXISTE	EXISTE	DIFERENTE

figura 19. O quadro comparativo apresenta as igualdades e desigualdades entre os dois documentos.

28. **União Automática de Documentos:** O usuário, ao solicitar o serviço, informa qual documento deve ter prioridade em caso de conflito. A primeira etapa desse serviço compara dois documentos, mas não apresenta o resultado da comparação para o usuário. A partir da análise do resultado da comparação, cria um novo documento baseado em regras, que resolvem as diferenças entre os dois documentos. Afigura 20 ilustra as regras implementadas no protótipo.

ELEMENTO DO DOCUMENTO XMI	DOCUMENTO 1 (com prioridade)	DOCUMENTO 2	RESULTADO
A	EXISTE	EXISTE	A (documento 1)
B	EXISTE	NÃO EXISTE	B (documento 1)
C	NÃO EXISTE	EXISTE	C (documento 2)

figura 20. Regras para a união de documentos.

29. **União Semi-Automática de Documentos:** A primeira etapa desse serviço compara dois documentos, como na união automática, mas apresenta o resultado da comparação para o usuário. Apresenta, também, sugestões para resolver os conflitos encontrados. O usuário pode aceitar as sugestões ou alterá-las se quiser. As sugestões apresentadas por esse serviço são baseadas nas mesmas regras da união automática (figura 20), procurando unir os elementos dos dois documentos. O algoritmo de união utilizado pelo MIMIX é descrito a seguir.

    1° Passo: A partir do quadro gerado na comparação, o MIMIX apresenta sugestões para a união, o usuário pode aceitar ou fazer modificações, escolhendo os elementos que irão compor o documento resultante. O usuário pode, também, excluir um elemento. O usuário solicita novamente o serviço, enviando o quadro comparativo para o MIMIX (figura 21).

Element	A	B	Resultado
	EXISTE	EXISTE	A
	EXISTE	EXISTE	A
	EXISTE	NÃO EXISTE	A
	NÃO EXISTE	EXISTE	B

figura 21. Quadro comparativo com sugestões para a união.

2° Passo: O MIMIX verifica as escolhas feitas pelo usuário, procurando possíveis inconsistências. O MIMIX trata inconsistências sintáticas do modelo, por exemplo, quando o usuário escolhe manter os atributos de uma classe, mas exclui a classe. Se não tiver nenhuma, ele armazena o resultado da união criando uma nova versão do documento, que passa a ficar disponível para os membros da equipe. Se encontrar alguma inconsistência o MIMIX retorna quadro comparativo para o usuário, apontando a inconsistência encontrada, para que ele a resolva.

IV.4 Exemplo de Utilização do MIMIX

Esta seção apresenta, passo a passo, um exemplo de utilização do protótipo. Esse exemplo ilustra as interações realizadas entre usuários que trabalham cooperativamente em um modelo de classes, utilizando as ferramentas CASE: Rose Enterprise Edition (RATIONAL, 2004) e Ideogramic UML (IDEOGRAMIC, 2000). Essas ferramentas possuem o mecanismo de importação e exportação de documentos XMI. Os modelos gerados por essas ferramentas são mapeados para documentos XMI, de acordo com a especificação do XMI 1.0, UML 1.3 e UMLX13.DTD.

O exemplo descreve cenários da modelagem cooperativa, que são apoiados pelo MIMIX. Ao longo da descrição do exemplo, são apresentadas algumas ilustrações da dinâmica do trabalho da equipe e são feitas referência a figuras, que exibem as telas do módulo cliente com o resultado de cada etapa do trabalho realizado utilizando o protótipo do MIMIX. Essas figuras são apresentadas após a descrição do exemplo de utilização do MIMIX.

IV.4.1 Ilustrações de Cenários da Modelagem Cooperativa

Este exemplo, ilustra o trabalho de 2 membros da equipe, cujos nomes fictícios são: **Usuário 1** e **Usuário 2**. O primeiro utiliza a ferramenta Rational Rose e o segundo utiliza o Ideogramic. Eles encontram-se geograficamente dispersos, têm acesso à Internet e possuem o módulo Cliente do MIMIX instalado em suas estações de trabalho. A presença deste módulo é necessária para que os membros possam interagir com os serviços do MIMIX.

O exemplo se inicia com o **Usuário 1** criando um modelo de Classes no Rational Rose (figura 30). Nesta etapa, esse usuário pode fazer várias alterações no modelo, sem que o resto da equipe tome conhecimento do que está sendo feito. Isto representa um cenário típico do trabalho cooperativo assíncrono, que é contemplado pelo MIMIX (figura 22).

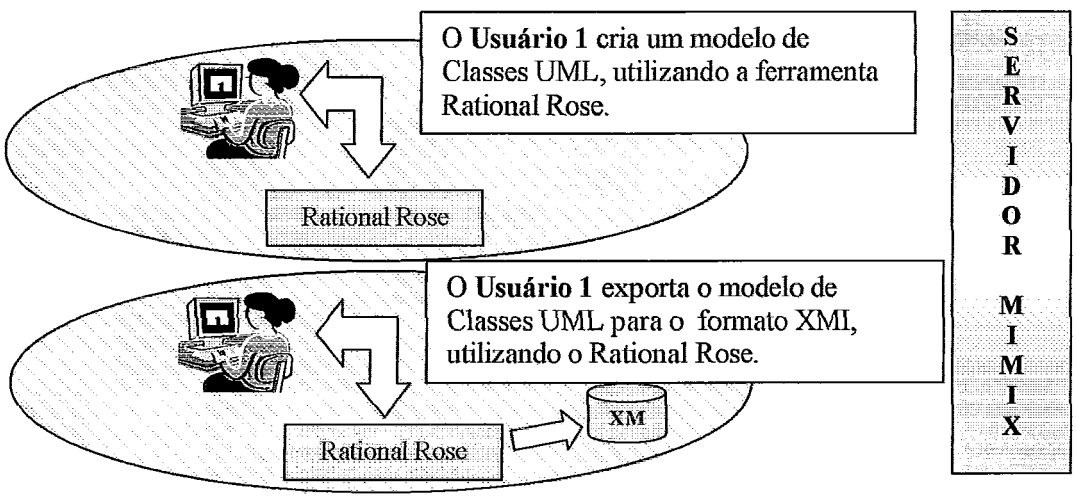


figura 22. Criação de modelo no Rose e exportação para o formato XMI.

Quando este usuário quiser disponibilizar seu modelo para a equipe, utiliza o módulo Cliente (tela da figura 30) para armazená-lo no MIMIX. Para isto, ele cria um espaço compartilhado de trabalho (projeto), chamado **Projeto Teste**, onde residirá seu modelo. Como autor do espaço de trabalho, o **Usuário 1** pode dar permissão para que outros membros da equipe trabalhem nele. O MIMIX contempla diversos níveis de acesso dentro de um espaço de trabalho (figura 23).

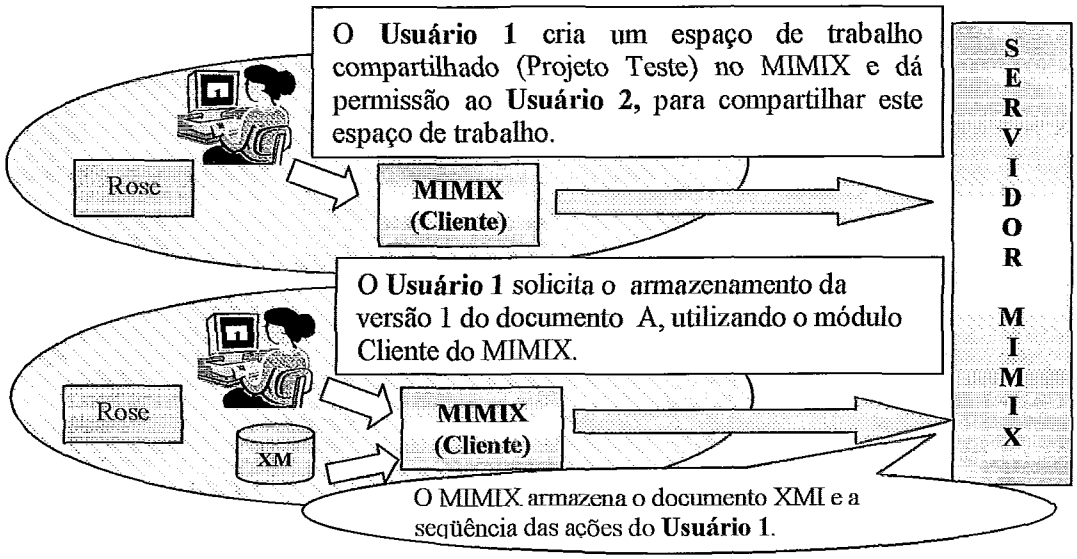


figura 23. Armazenamento do modelo criado pelo **Usuário 1** no MIMIX.

Em seguida, o **Usuário 2**, utilizando seu módulo Cliente (tela da figura 31), solicita a relação dos documentos (modelos), referentes ao **Projeto Teste**, armazenados no MIMIX. Assim, ele verifica que o **Usuário 1** armazenou a primeira versão de seu modelo no espaço compartilhado de trabalho. É interessante observar que não há, necessariamente, um sincronismo de tempo entre a ação dos membros da equipe, ou seja, o **Usuário 2** pode fazer esta verificação minutos, horas ou dias após o armazenamendo efetuado pelo **Usuário 1**. Após constatar a existencia do modelo armazenado pelo **Usuário 1**, o **Usuário 2**, solicita a recuperação do documento, através do módulo Cliente (tela da figura 32). Uma vez recuperado, este documento pode ser alterado através de sua ferramenta CASE, que neste exemplo, é o Ideogramic UML (tela da figura 33). Novamente, várias interações podem ocorrer (e tipicamente ocorrem) antes que este membro decida divulgar seu trabalho para a equipe (figura 24).

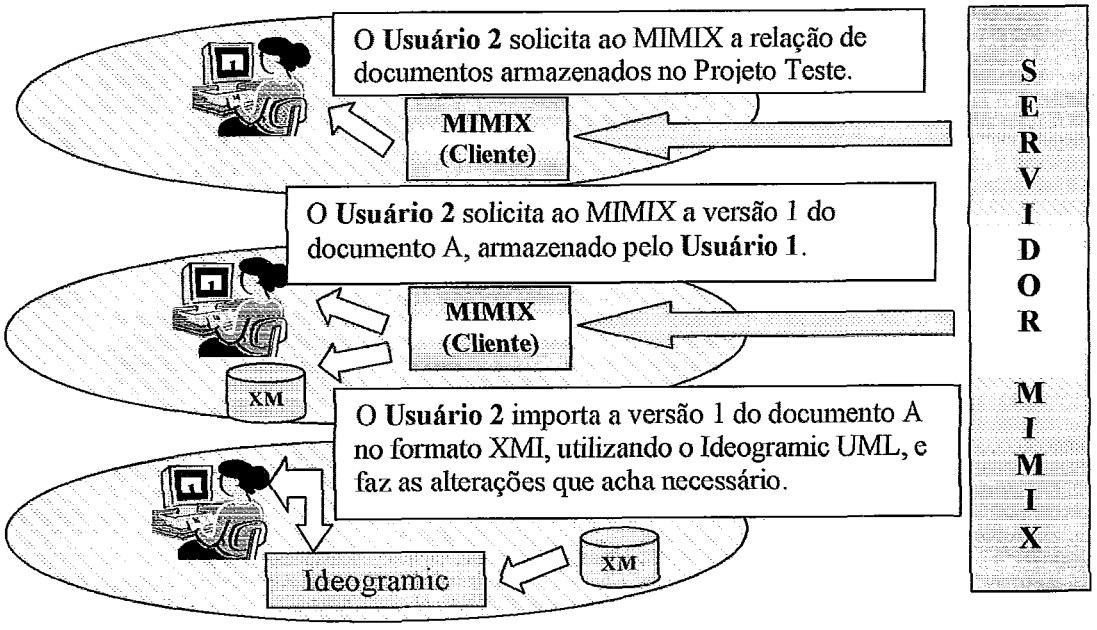


figura 24. Busca e recuperação de documentos do Projeto Teste pelo **Usuário 2**.

O **Usuário 1**, verifica quais os usuários que recuperaram a primeira versão de seu modelo (tela da **figura 34**). Este é um serviço de percepção importante, pois fornece resposta à pergunta: Quem esta trabalhando com um determinado documento? Este tipo de serviço não foi encontrado em outras ferramentas analisadas que, em geral, registram apenas quando há uma operação de armazenamento. O **Usuário 1** fica ciente que o **Usuário 2** recuperou seu modelo e envia uma mensagem para ele (tela da **figura 35**) para fornecer algumas informções sobre o modelo (figura 25).

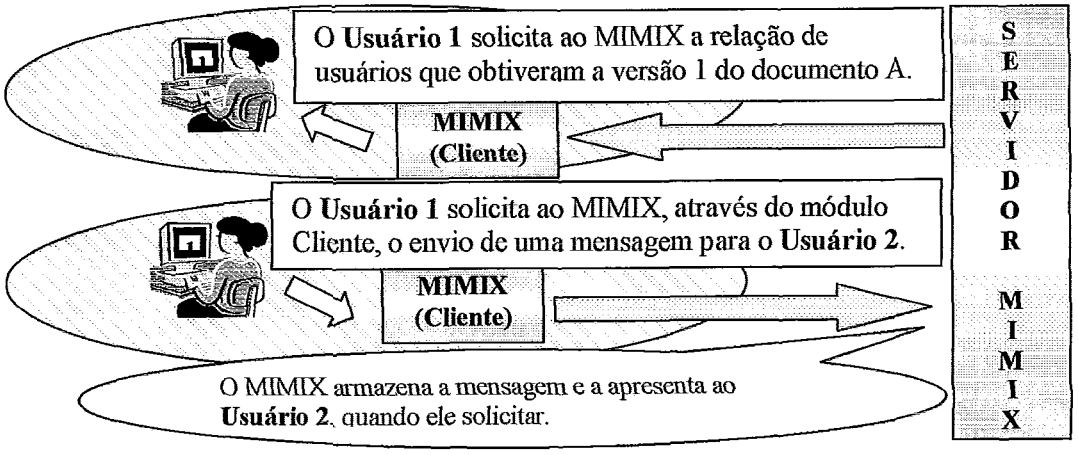


figura 25. O **Usuário 1** consulta quem está trabalhando na versão que ele armazenou.

O **Usuario 2**, após fazer as alterações que achou convenientes na versão 1 do modelo, solicita o seu armazenamento (tela da **figura 36**). O MIMIX cria a versão 2 desse modelo. O **Usuário 2** solicita, também, a recuperação das mensagens a ele endereçadas (figura 26).

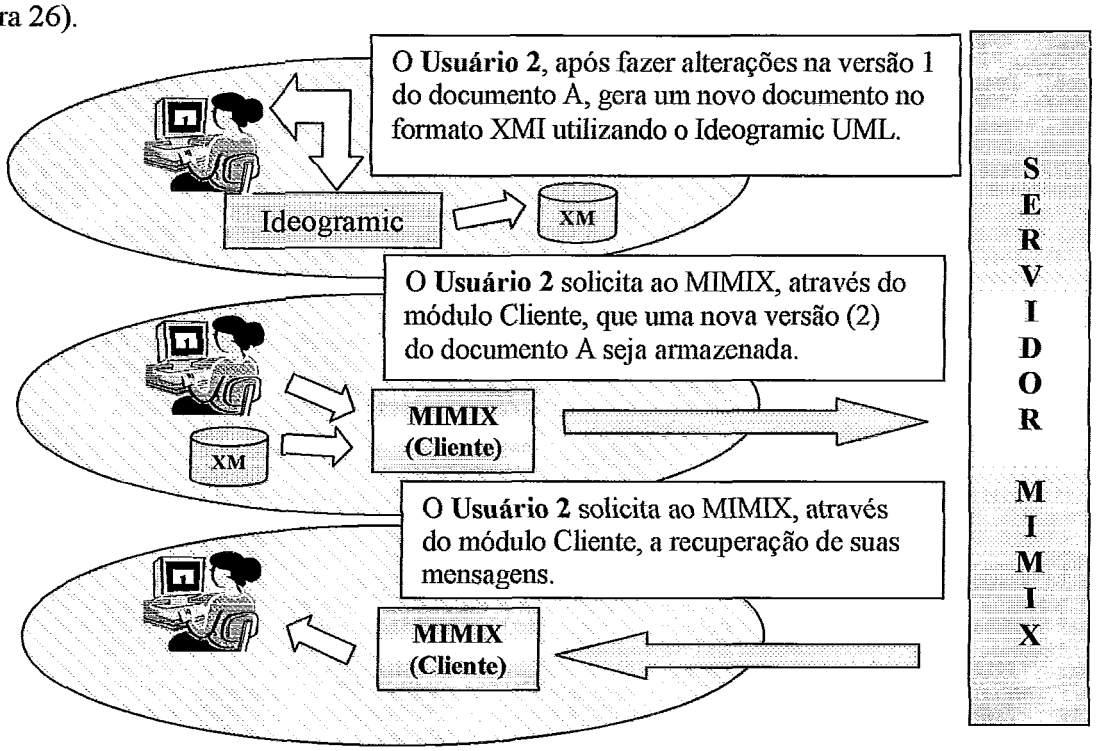


figura 26. O **Usuário 2** gera uma nova versão do documento A e solicita seu armazenamento no MIMIX.

O **Usuário 1** verifica que existem duas versões de seu modelo armazenadas no MIMIX. A primeira ele mesmo criou e a segunda foi armazenada pelo **Usuário 2**. Ele solicita a comparação, e depois a união automática das duas versões. Para fazer a união, ele escolhe qual versão deve ter prioridade sobre a outra (tela da figura 37). O MIMIX então gera (e armazena) a terceira versão do modelo, contendo elementos da versão 1 e da versão 2. Os mecanismos de união e de comparação do MIMIX permitem que os membros da equipe verifiquem as diferenças existentes entre os documentos envolvidos. No processo de união, os membros podem ainda definir quais elementos devem ser mantidos ou alterados no documento resultante (figura 27).

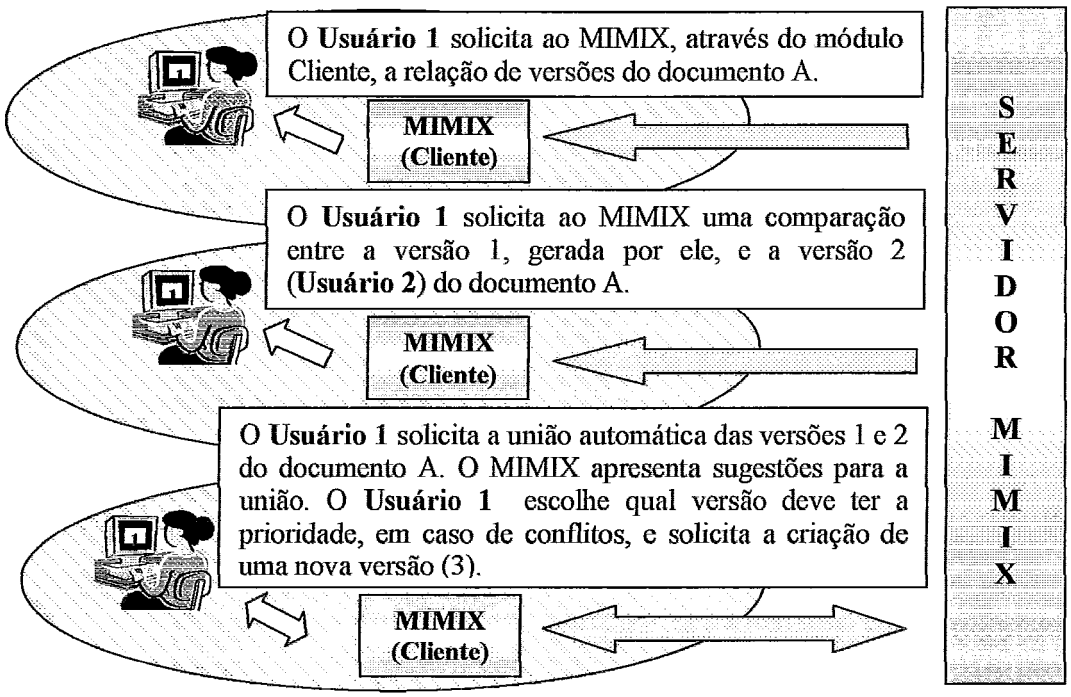


figura 27. O **Usuário 1** solicita a comparação e a união das versões 1 e 2 do documento A, gerando a versão 3.

Por fim, o **Usuário 2** consulta os documentos armazenados no Projeto teste (tela da **figura 38**). Ele verifica que o **Usuário 1** criou uma nova versão do documento A, utilizando uma versão de sua autoria figura 28

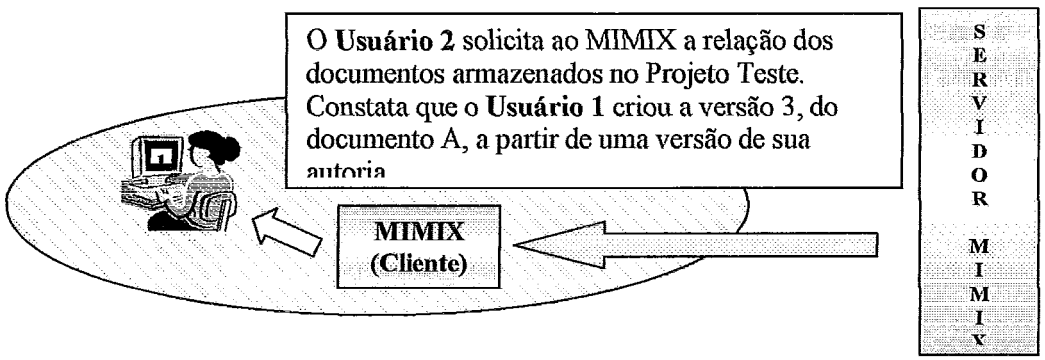


figura 28. O **Usuário 2** constata que foi criada uma nova versão do documento A.

Como pôde ser visto no exemplo, durante a modelagem cooperativa de software ocorrem diversas interações entre os membros da equipe de desenvolvimento, gerando várias versões intermediárias, que ficam disponíveis para a equipe através do

MIMIX. Para que os membros da equipe possam observar o andamento do trabalho e ter consciência das ações realizadas pelos demais membros, é muito importante que eles possam analisar e comparar os estados intermediários dos modelos gerados. Da mesma forma, manter um histórico da sequência das ações dos membros, como, por exemplo, a geração de novas versões, consultas e/ou recuperação de documentos e troca de mensagens entre os membros da equipe, constitui uma base de informações importantes para uma equipe que trabalha cooperativamente.

IV.4.2 Telas do Protótipo do Cliente

O exemplo de utilização do MIMIX faz referência as telas do módulo cliente, que ilustram os passos descritos. A seguir, são apresentadas essas telas.

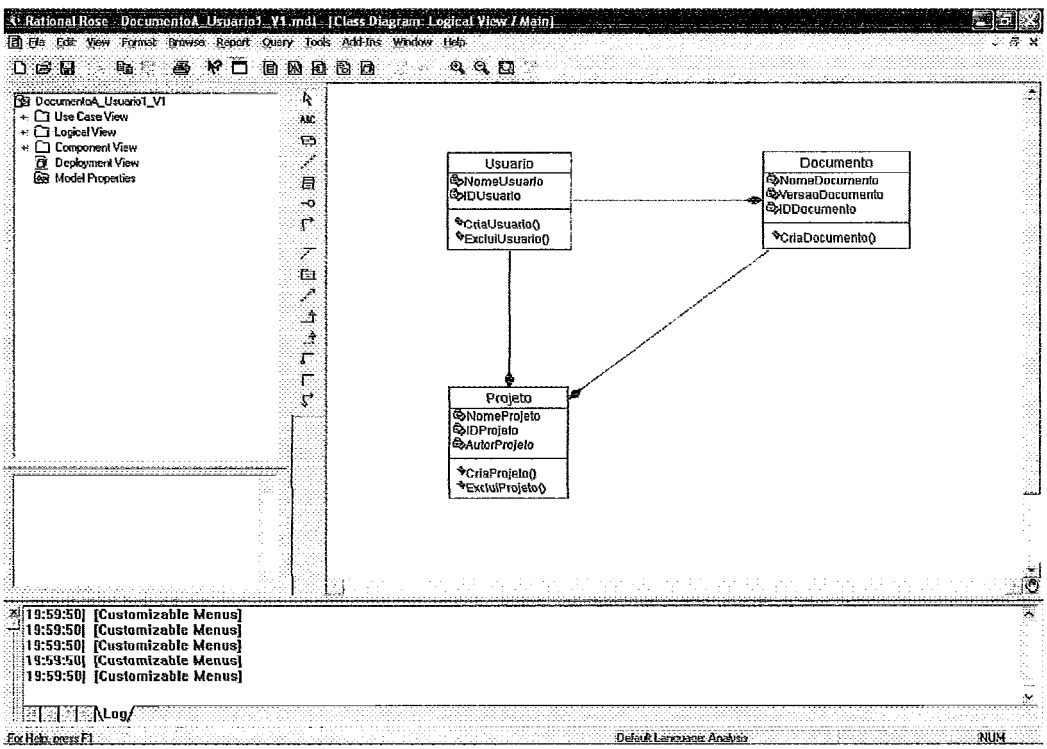


figura 29. O **Usuário** 1 cria modelo de classes no Rose.

figura 30. Tela Principal do Módulo Cliente

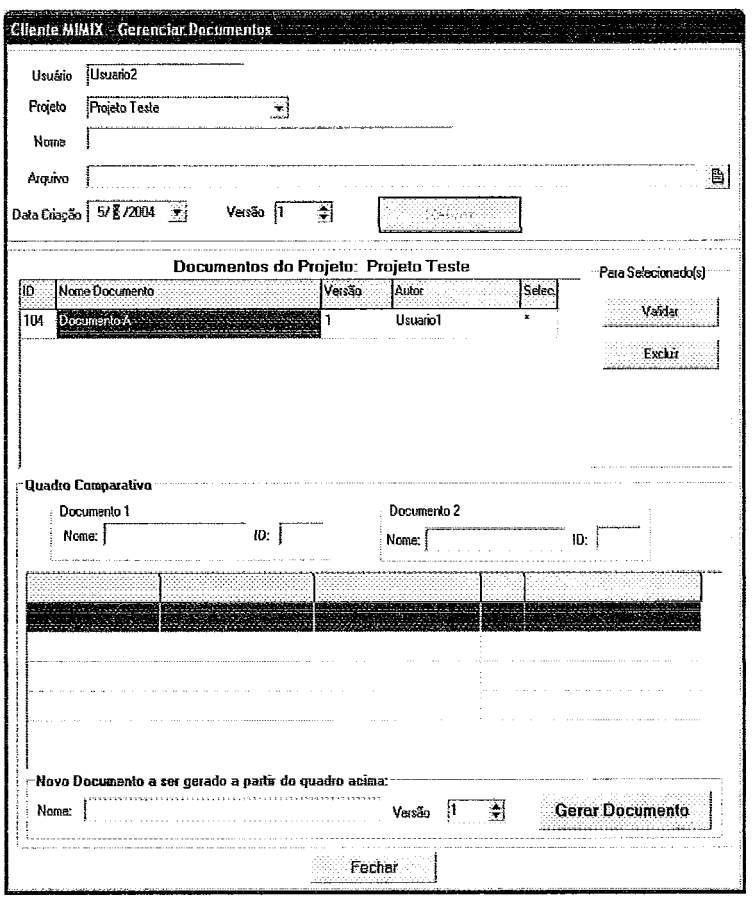


figura 31. O Usuário 2 solicita a relação dos documentos armazenados no Projeto Teste.

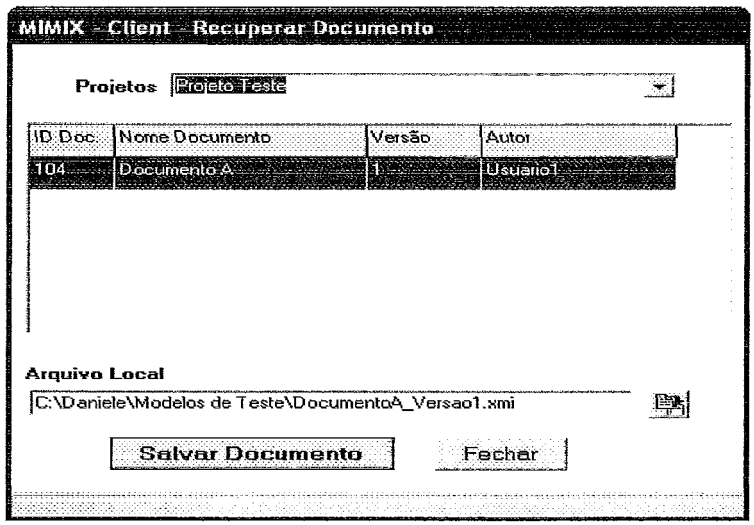


figura 32. O Usuário 2 solicita a versão 1, do documento A, cujo autor é o Usuário 1.

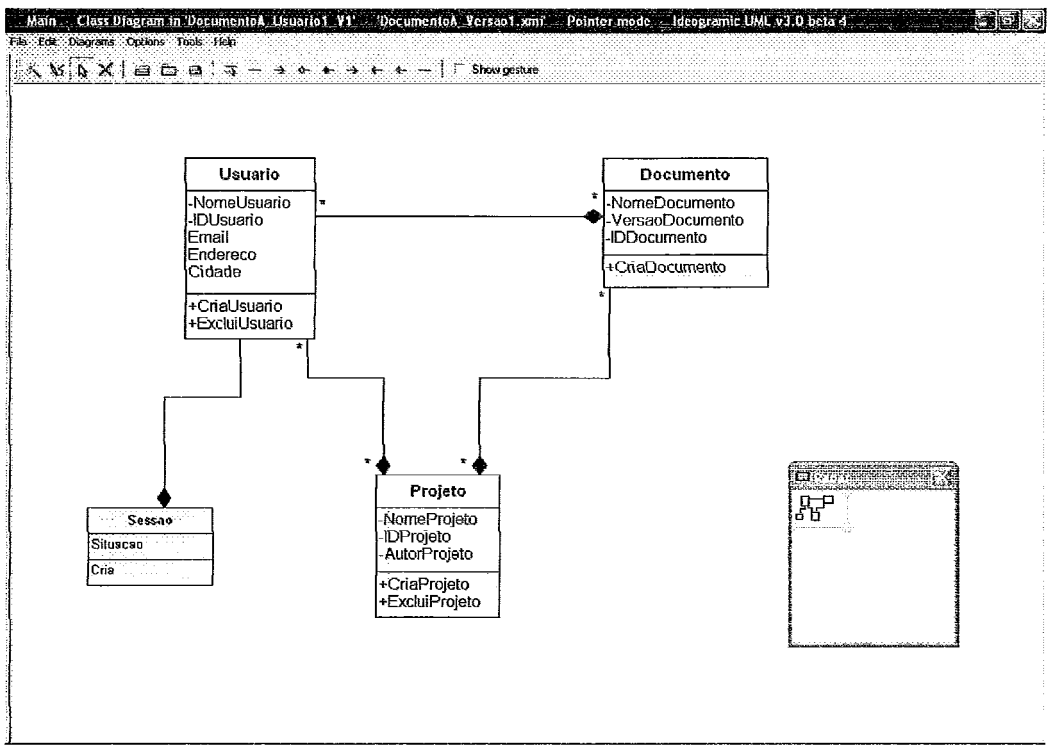


figura 33. O Usuário 2 importa a versão 1, do documento A, no Ideogramic e faz alterações.

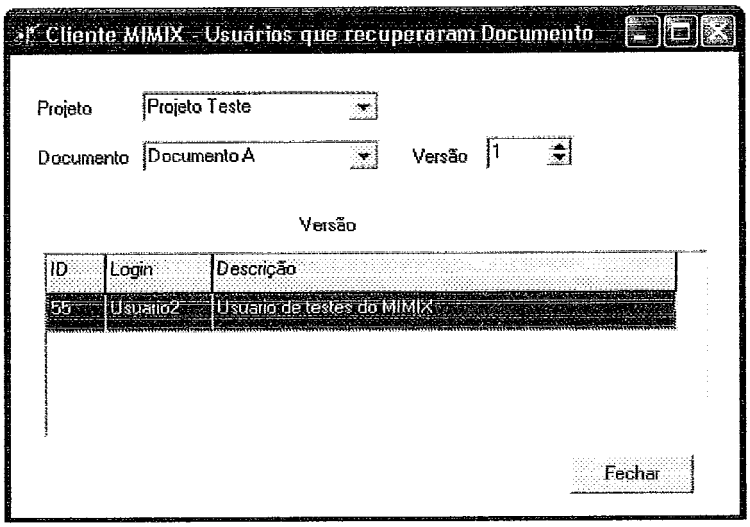


figura 34. O **Usuário 1** solicita a relação dos usuários que obtiveram a versão 1, do documento A.

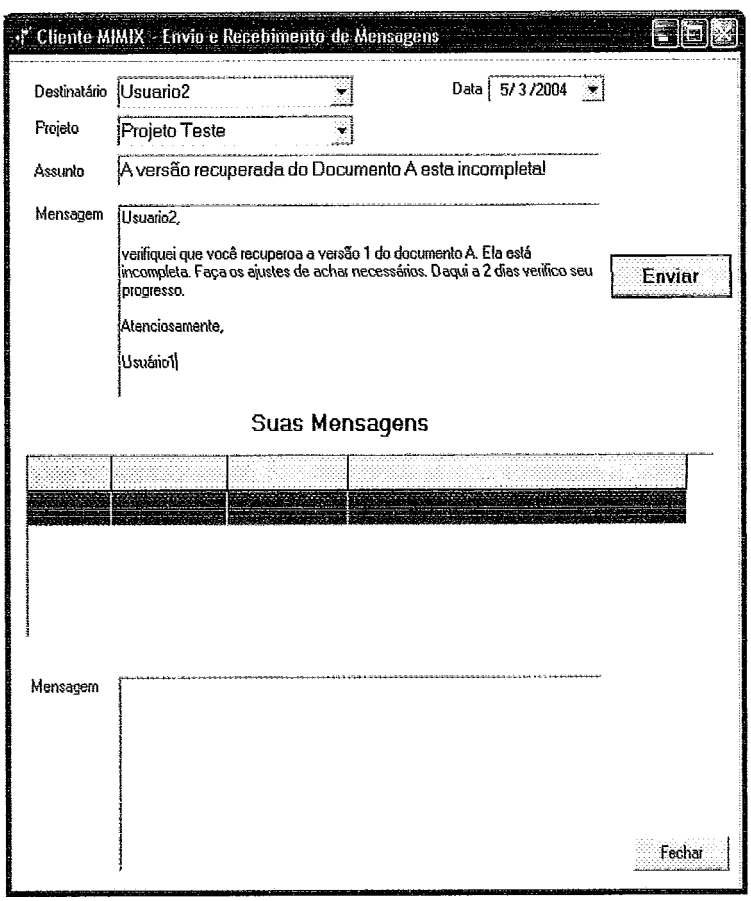


figura 35. O **Usuário 1** envia mensagem para o **Usuário 2**.

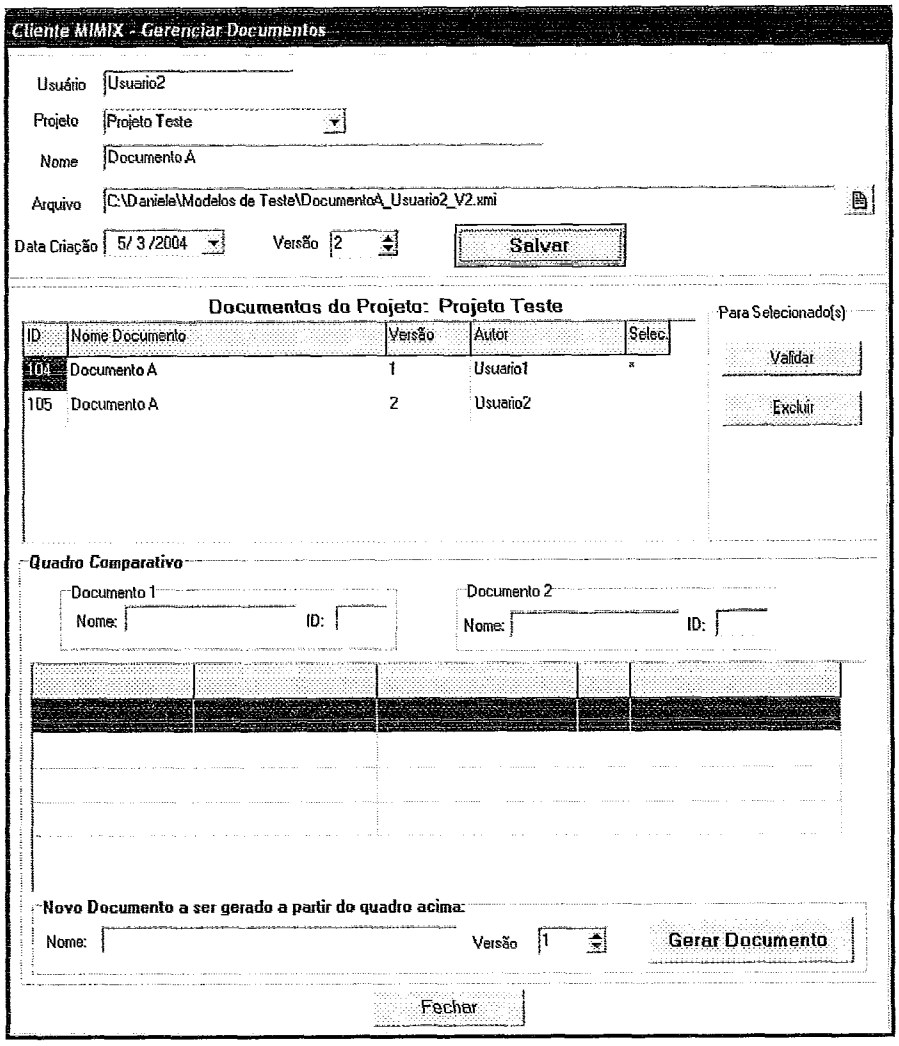


figura 36. O Usuário 2 armazena uma nova versão do documento A (versão 2). Na tela é possível observar, que esse o **Usuário 1** já tinha armazenado anteriormente uma versão desse documento (versão 1).

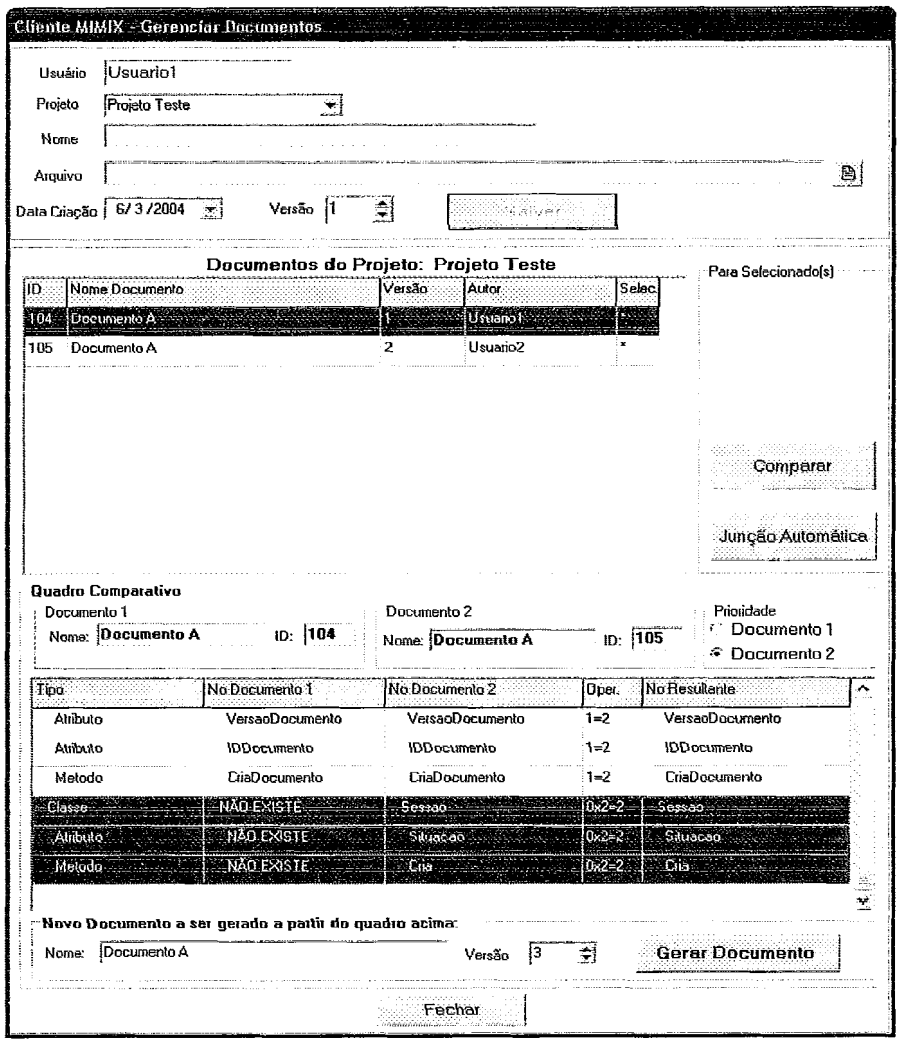


figura 37. O **Usuário 1** solicita a comparação entre as versões 1 e 2, do documento A. O MIMIX exibe as diferenças entre as versões, e fornece sugestões para a união. O **Usuário 1** solicita a união automática das versões 1 e 2. Escolhe a prioridade da versão 2. Assim, é criada a versão 3, do documento A.

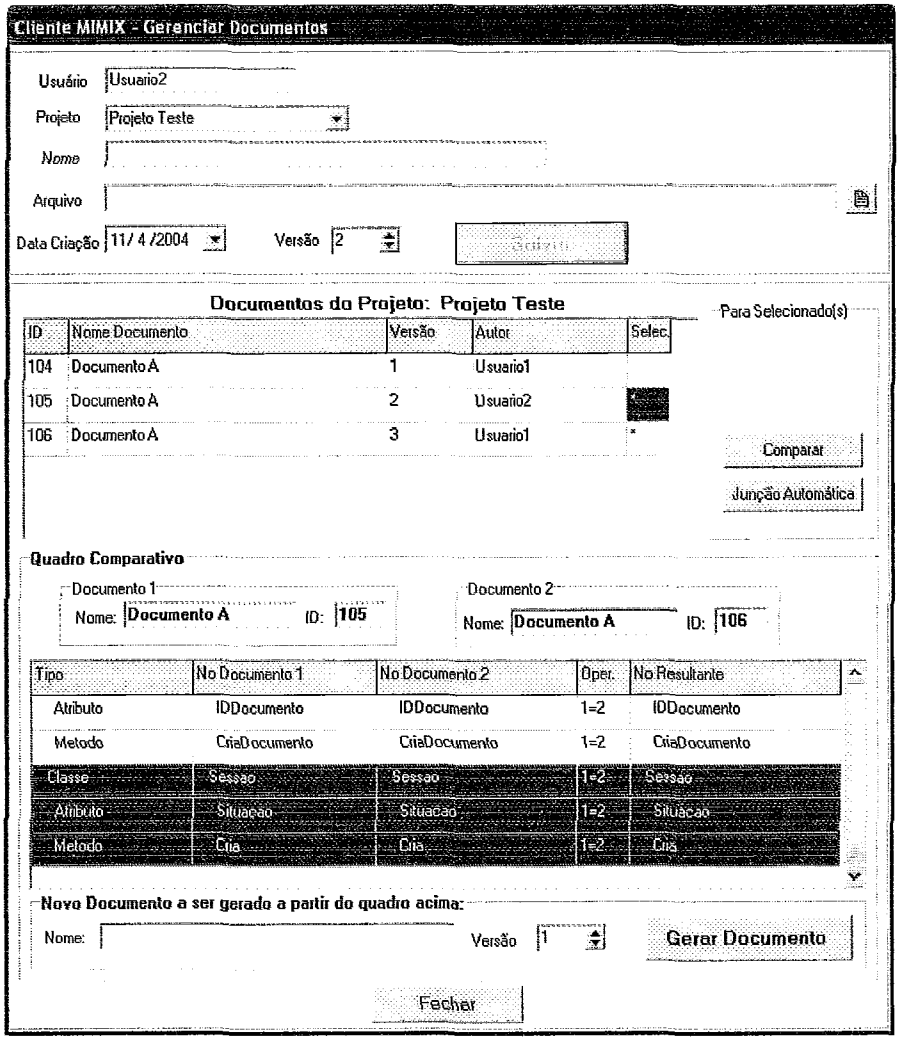


figura 38. O **Usuário 2** solicita os documentos armazenados no Projeto Teste, e verifica que uma nova versão foi criada. Ele solicita a comparação entre as versões 2, que foi a última armazenada por ele, e a versão 3, que é a nova versão, para observar as diferenças entre elas.

IV.5 Considerações Sobre a Utilização do MIMIX

Disponibilizando as funcionalidades do MIMIX na forma de serviços Web foi possível alcançar a independência de plataforma pretendida. Isso pode ser observado com a utilização do protótipo, já que os módulos cliente e servidor foram construídos em plataformas heterogêneas sem que nenhum problema de interação tenha sido observado.

O conjunto de serviços implementados no MIMIX foi definido após o estudo de vários trabalhos que propõem o apoio à cooperação, em várias atividades. Como as equipes de desenvolvimento, geralmente, possuem necessidades diferentes, é possível tirar proveito da tecnologia de serviços Web para desenvolver novos serviços para o MIMIX. Além da possibilidade de expansão, a arquitetura do MIMIX permite que os serviços sejam utilizados de formas peculiares, por cada módulo cliente desenvolvido. Assim, a partir do mesmo conjunto de serviços, é possível construir módulos clientes com funcionalidades diferentes, para equipes com necessidades específicas.

Utilizando o MIMIX, é possível criar um espaço de trabalho compartilhado virtual, que disponibiliza funcionalidades, para reduzir a sensação de isolamento dos membros de uma equipe. Os serviços de percepção do MIMIX fornecem aos membros de uma equipe informações importantes sobre o andamento dos trabalhos individuais e do grupo. Essas informações são essenciais para que o objetivo comum da equipe seja alcançado, que é construir o produto desejado. Os serviços relacionados à percepção do produto fornecem informações sobre o estado do produto que está sendo construído, como, por exemplo, as modificações que foram feitas ao longo do trabalho, as diferenças existentes entre versões de um documento e o tempo gasto para a geração de uma determinada versão. Essas informações permitem que os membros possam acompanhar o trabalho realizado, evitando conflitos, redundâncias e possíveis erros.

O Anexo 3 contém outras telas do módulo cliente do MIMIX, que ilustram o exemplo de utilização apresentado nesta seção.

V Conclusão

Nesta dissertação, foi destacada a importância do apoio de ferramentas automatizadas ao desenvolvimento cooperativo de software, com ênfase no suporte à atividade de modelagem de software. Nessa atividade, a principal preocupação da equipe de desenvolvimento é o produto que está sendo desenvolvido. Esse produto é construído de forma incremental e evolutiva pelos membros da equipe. Assim, é muito importante que os membros tenham informações sobre o trabalho individual e geral da equipe, para que o produto final atenda os objetivos desejados. Para o sucesso do trabalho realizado, também é preciso que os membros possam observar, passo a passo, o desenvolvimento do produto (percepção do produto).

As ferramentas CASE, disponíveis no mercado possuem pouco ou nenhum suporte à cooperação. O apoio à cooperação entre ferramentas CASE diferentes, ainda é um tópico em aberto. Não foi encontrado na literatura nenhum trabalho, que forneça informações sobre as versões intermediárias dos modelos gerados durante a modelagem cooperativa, nem que permita a sua comparação e união.

Devido à grande diversidade de ferramentas CASE disponíveis para o apoio a modelagem de software, o objetivo desta dissertação consistiu em permitir que essas ferramentas sejam aproveitadas e reutilizadas, de modo cooperativo. Esse suporte deve ser independente da plataforma de desenvolvimento e da ferramenta CASE utilizada, permitindo a comunicação e troca de informações entre elas. Além disso, fez parte do objetivo, fornecer informações sobre os estados intermediários (versões) do produto para os membros de uma equipe durante o trabalho de desenvolvimento, possibilitando a comparação e união de versões de um produto.

Para alcançar esses objetivos, alguns desafios tecnológicos tiveram que ser transpostos. De início, existe o problema da interoperação entre ferramentas CASE. Embora existam algumas tentativas de padronizar o formato de serialização de dados, para permitir o intercâmbio entre ferramentas CASE, ainda não existe um consenso sobre isso. As ferramentas, em geral, utilizam formatos diferentes e proprietários. Mesmo que esses

formatos sejam conhecidos, seria inviável fazer conversores para todas as ferramentas existentes. Uma vez que exista um consenso sobre o formato de dados, para que possam trabalhar cooperativamente e trocar informações, as ferramentas CASE precisam interagir.

Portanto, foi preciso:

1. Buscar uma forma de permitir a interoperação entre plataformas de software e hardware diferentes.

2. Disponibilizar informações sobre o trabalho da equipe e sobre as versões do produto para os membros da equipe ao longo do trabalho desenvolvido.

3. Garantir a persistência e a integridade dessas informações. A utilização de um SGBD tradicional não resolve o problema, pois esses não reconhecem o uso cooperativo.

Assim, foi projetado um sistema de apoio à modelagem cooperativa de software, denominado MIMIX. Esse sistema fornece suporte à cooperação entre membros de uma equipe de desenvolvimento, que trabalham com ferramentas CASE heterogêneas, distribuídos geograficamente.

Para viabilizar o item 1, o MIMIX utiliza o XMI como formato para intercâmbio de dados (modelos) entre as ferramentas CASE envolvidas na modelagem cooperativa. O XMI é um padrão recente, da OMG, bastante utilizado pelas ferramentas CASE, baseadas em UML. Para que seja possível trocar informações, as ferramentas precisam ser capazes de salvar e recuperar seus dados no formato XMI.

Novamente, para viabilizar o item 1, o MIMIX implementa suas funcionalidades na forma de serviços Web, provendo a independência de plataforma pretendida. Isso porque, os serviços Web compreendem um conjunto de tecnologias de plataforma-neutra, como o HTTP e o XML, designadas para facilitar a distribuição de serviços através de Intranets ou da Internet, permitindo a comunicação e utilização desses serviços entre plataformas diferentes. A tecnologia de serviços Web é voltada para a interoperabilidade entre componentes, no caso, as ferramentas CASE.

Para viabilizar os itens 2 e 3, o MIMIX armazena, em um repositório centralizado, informações sobre as interações e as ações realizadas pelos membros da equipe. Os membros podem consultar essas informações, referentes ao espaço de trabalho compartilhado a que possuem acesso, para poder observar o andamento do trabalho da equipe. O MIMIX também armazena os estados intermediários dos modelos gerados pelos membros da equipe, criando versões desses modelos. Os modelos e suas versões são armazenados na forma de documentos XMI. Os membros podem consultar, comparar ou fazer a união de versões de um modelo. Assim, podem obter informações sobre o trabalho realizado pelos demais membros.

Como resultado do estudo realizado para a elaboração desta dissertação, foi desenvolvido um protótipo do MIMIX, para verificar a viabilidade de implementação da proposta. Um exemplo da utilização desse protótipo foi apresentado, no qual usuários trabalhavam cooperativamente utilizando as ferramentas CASE Rational Rose e Ideogramic UML, que possuem o mecanismo de importação e exportação de documentos XMI.

Uma contribuição importante do MIMIX foi implementar um mecanismo de gerência de versões específico para a atividade de modelagem cooperativa de software. Assim, o MIMIX consegue fornecer informações sobre os estados intermediários dos modelos gerados (versões de modelos), para uma equipe de desenvolvimento. Essas informações são chamadas de percepção do produto, e permitem que as mudanças feitas pelos membros da equipe sejam rastreadas, durante todo o desenvolvimento. O MIMIX permite a comparação de versões de um modelo, para consultar a evolução do trabalho, e a união de trabalhos paralelos dos membros. O mecanismo de união de versões empregado é flexível, permitindo que os membros interfiram diretamente no resultado final. Essa funcionalidade não foi encontrada em nenhum trabalho disponível na literatura. Assim, é possível detectar e corrigir possíveis erros introduzidos no trabalho, colaborando para o seu sucesso e para que seja atingido o objetivo da equipe.

V.1 Limitações e Trabalhos Futuros

Esta dissertação apresenta algumas limitações, descritas a seguir, que podem ser contornadas em trabalhos futuros:

a) O XMI ainda não está estável. Periodicamente a OMG disponibiliza novas versões. Os DTDs XMI definem como mapear os modelos UML, em documentos XMI. Quando a UML e o XMI sofrem modificações, os DTDs também mudam. As ferramentas CASE mapeiam seus modelos de acordo com um DTD. Assim, com essas mudanças, ferramentas distintas podem acabar utilizando versões de DTDs diferentes, prejudicando a troca de informações entre elas. Muitas vezes, é preciso fazer a conversão de formatos de DTDs diferentes. Esta dissertação se baseia no: XMI, versões 1.0, 1.1 e 1.2; na UML 1.3; DTDs, UMLX13.DTD e UMLX13-11.DTD. Atualmente, essas versões são as utilizadas pelas ferramentas CASE. Entretanto, o XMI tem sido amplamente utilizado pelas ferramentas CASE. Várias ferramentas CASE comerciais e acadêmicas já incorporaram o mecanismo de importação e exportação de documentos XMI. Isso demonstra a boa aceitação desse padrão.

b) O XMI é um padrão recente, ele ainda não atende perfeitamente às necessidades das diversas ferramentas do mercado. Por exemplo, diversas informações, que não estão padronizadas pela OMG e pela UML, são armazenadas no documento XMI na forma de atributos de extensão, como, por exemplo, informações sobre a visualização (representação gráfica) dos modelos, que não estão padronizados no metamodelo UML. Cada fabricante cria seus próprios atributos criando dificuldades quando uma aplicação tenta decodificar um documento criado por outra, podendo ocorrer perda de informação durante o processo de importação e exportação para documentos XMI. Na maioria dos casos, esta perda ocorre, pois no processo de importação de documentos XMI, as ferramentas descartam informações que não são reconhecidas ou

relevantes para ela. Como conseqüência, quando o documento XMI é novamente exportado, as informações do documento original não estão mais presentes, dificultando a nova importação pela ferramenta que originou o documento. Espera-se, que esse problema seja solucionado na nova versão da UML (2.0), que está em fase de finalização.

c) A granularidade das informações tratadas por ferramentas, como o MIMIX, é uma questão em aberto. Devido ao contexto de trabalho apoiado pelo MIMIX, a unidade de trabalho é um documento contendo um modelo UML inteiro. Entretanto, serviços como os de comparação e de união de documentos, exigiram a decomposição destes documentos para ser possível identificar, por exemplo, as diferenças entre elementos neles contidos. Uma continuidade natural do MIMIX seria na direção de aprofundar o estudo da utilização de outras granularidades.

d) Questões de desempenho não foram analisadas no protótipo. Embora essa questão não seja crítica, se tratando do suporte ao trabalho assíncrono, se refere a um tópico importante para viabilizar a utilização do MIMIX em ambientes de produção.

e) Questões relacionadas ao tratamento de falhas e à segurança, também precisam ser tratadas, para viabilizar a utilização do MIMIX. Essas questões podem ser beneficiadas com a evolução da tecnologia de serviços Web.

VI REFERÊNCIAS BIBLIOGRÁFICAS

ABERER, K., KLINGEMANN, J., TESCH, T., et al., 1996, "Transaction Models Supporting Cooperative Work: The TransCoop Experiences". In: *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications* (CODAS'96), pp. 467-476, Kyoto, Japan.

ALONSO, G., AGRAWAL, D., EL ABBADI, A., et al., 1996, "Advanced Transaction Models in Workflow Contexts". In: *Proceedings 12th International Conference on Data Engineering*, pp.574-583, New Orleans, Louisiana, USA.

ALTMAN, J., POMBERGER, G., 1999, "Cooperative Software Development: Concepts, Model and Tools". In: *Proceedings of the 30th Conference on Technology of Object-Oriented Languages and Systems (TOOLS-30)*, pp.194-277, Santa Barbara, CA, USA.

ALTMAN, J., WEINREICH, R., 1998, "An Environment for Cooperative Software Development Realization and Implications". In: *Proceedings of 31th Hawaii International Conference on Systems Sciences, Collaboration Systems and Technology Track (HICSS-31)*, volume: I, pp.27-37, Kohala Coast, HI, USA.

APACHE, 1999. Apache Foundation Software. In: http://www.apache.org. Acessado em: 04/2004.

ARAUJO, R., BORGES, M., 1995, "Quorum-W: A Group Decision Support Tool for the Internet Environment". In: *Proceedings of the International Workshop on Groupware (CRIWG'95)*, pp.39-52, Lisboa, Portugal.

ARAUJO, R., DIAS, M., BORGES, M., 1997, "A Framework for the Classification of Computer Supported Collaborative Design Approaches". In: *Proceedings of the III CYTED-RITOS International Workshop on Groupware (CRIWG'97)*, pp.91-100, Madrid, Spain.

ARGOUML, 1996, ArgoUML Home Page. In: http://argouml.tigris.org. Acessado em: 03/2004.

ARMSTRONG, E., BODOFF, S., CARSON, D., et al., 2002, "The Java Web Services Tutorial". *SUN Tutorial*. In: http://www.sun.com. Acessado em: 03/2004.

BARDINELLI, S., NITTO, E., FUGGETTA, A., et al., 1996, "Supporting Cooperation in the SPADE Environment". *IEEE Transactions on Software Engineering*, pp.841-865.

BARISH, G., 2002, *Building Scalable an High-Performance Java Web Applications Using J2EE Technology*, Press: Addison-Wesley, ISBN: 0-201-72956-3.

BODKER, M. KYNG, M. SCHMIDT K., 1999, "NESSIE: An Awareness Environment for Cooperative Settings". In: *Proceedings of the 6th European Conference on Computer Supported Cooperative Work (ECSCW'99)*, pp.391-41 0, Copenhagen, Denmark.

BORGES, M.R.S., PINO, J. A., 2000, "Requirements for Shared Memory in CSCW Applications". In: *Proceedings of the 10th Workshop on Information Technologies and Systems*, pp. 211-216, Australia.

BORLAND, 2002, "Developer's Guide Web Services Kit for Java". In: http://www.borland.com, 2002. Acessado em: 03/2004.

BREDENFELD, A., 1995, "Cooperative Concurrency Control for Design Environments". *In: Proceedings of the European Design Automation Conference with EURO-VHDL (EURODAC'95)*, pp. 308-313, Brighton, Great Britain.

CANTÚ, M., 2002, *Dominando o Delphi 6, A Bíblia,* Editora: Makron Books, ISBN: 85-346-1408-3.

CATTELL, R., 1991, *Object Data Management: Database Systems for Engineering and Object-Oriented Applications*, Press: Addison-Wesley, ISBN: 0201530929.

CHEN, M., NORMAN, R., 1992, "A Framework for Integrated CASE". *IEEE Software Magazine*, volume: 9, number: 2, pp.18-22.

CHU-CARROLL, M., WRIGHT, J., SHIELDS, D., 2002, "Supporting Aggregation in Fine Grained Software Configuration Management". *ACM SIGSOFT Software Engineering Notes*, volume: 27, issue: 6, session: 6, pp.99-108, ACM Press.

CONRADI, R., WESTFECHTEL, B., 1998, "Version Models for Software Configuration Management". *ACM Computing Surveys (CSUR)*, volume: 30, issue: 2, pp.232-282.

COX, D., GREENBERG, S., 2000, "Supporting Collaborative Interpretation in Distributed Groupware". In: *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, pp.289-298, Philadelphia, Pennsylvania, USA.

CROW, D., PARSOWITH, S., WISE, G., 1997, "The Evolution of CSCW". *ACM SIGHI Bulletin*, volume: 29, issue: 2.

DAMM, C., HANSEN, K., THOMSEN, M., et al., 2000, "Tool Integration: Experiences and Issues in Using XMI and Component Technology". In: *Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS 00)*, pp.94-106, France.

DIAS, M., BORGES, M., 1999, "Development of Groupware Systems with the COPSE Infraestruture". In: *Proceedings of the International Workshop on Groupware*, pp.278-285, Cancun, México.

EDWARDS, W., 1996, "Policies and Roles in Collaborative Applications". In: *Proceedings of the Computer Support Cooperative Work (CSCW'96)*, pp.11-20, Boston, MA, USA.

EL-JAICK, D., VIEIRA, V., WERNER, C., MATTOSO, M., 2002, "Suporte de Banco de Dados Convencionais a Aplicações Cooperativas". *Publicações Técnicas do Projeto OdysseyShare*, COPPE/UFRJ, Rio de Janeiro, Brasil, 1/2002.

ELLIS, L., GIBBS, S. J., 1989, "Concurrency Control in Groupware Systems". In: *Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data*, pp. 399-407, Portland, Oregon, USA.

ELLIS, L., GIBBS, S. J., REIN, G. L., 1991, "Groupware: Some Issues and Experiences". *Communications of the ACM*, volume: 34, number: 1, pp. 38-58.

EPOS, 1989, EPOS group home page. In: http://www.idt.unit.no/~epos. Acessado em: 03/2004.

ESTUBLIER, J., LEBLANG, D., CLEMM, G., et al., 2002, "Impact of the Research Community on the Field of Software Configuration Management". In: *Proceedings of the10th ACM SIGFOFT Symposium of Foundations of Software Engineering*, volume: 27, issue: 5, pp.31-39, Charleston, USA.

FARSHCHIAN, B., 2000, "Gossip: An Awareness Engine for Increasing Product Awareness in Distributed Development Projects". In: *Proceedings of the 12th International Conference on Advance Information Systems Engineering (CAiSE'2000)*, pp.264-278, Stockholm, Sweden.

FARSHCHIAN, B., 2001, "Integrating Geographically Distributed Development Teams Through Increased Product Awareness". *Information Systems Journal*, volume: 26, number: 3, pp.123-141.

GREENBERG, S., 1989, "The 1988 Conference on Computer-Supported Cooperative Work: Trip Report". *ACM SIGCHI Bulletin*, volume: 21, issue: 1.

GREENBERG, S., 2002, "Real Time Distributed Collaboration". *Encyclopedia of Distributed Computing*. In: http://www.cpsc.ucalgary.ca. Acessado em: 03/2004.

GREENBERG, S., MARWOOD, D., 1994, "Real Time Groupware as Distributed System: Concurrency Control and its Effect on the Interface". In: *Proceedings of Computer Supported Cooperative Work (CSCW'94)*, pp.207-212, Chapel Hill, USA.

GROOVE, 2001, "Groove Product Backgrounder". Groove Technical Report. Disponível em: http://www.groove.net. Acessado em: 03/2004.

GROSE, T., DONEY, G., BRODSKY, S., 2001, *Mastering XMI: Java Programming with XMI, XML and UML*, Press: OMG, ISBN: 0-471-38429-1.

GUNZER, H., 2002, "Introduction To Web Services". *Borland White Paper*. In: http://www.borland.com/webservices/white_papers. Acessado em: 03/2004.

GUTWIN, C., GREENBERG, S., 1999, "The Effects of Workspace Awareness Support on the Usability of Real-Time Distributed Groupware". *ACM Transactions On CHI*, v.6, n.3, pp.243-281.

HANSEN, K., 1999, "The Knight Project: Supporting Collaboration in Object-Oriented Analysis and Design". *Workshop on the Integrating Human Factors into USE CASES and OO Methods (ECOOP'99)*, pp.245-289, Lisboa, Portugal.

HANSEN, K., DAMM, C., 2002, "Instant Collaboration: Using Context-Aware Instant Messaging for Session Management in Distributed Collaboration Tools". In: *Proceedings of NordiCHI'2002*, pp.279-282, Aarhus, Denmark.

IDEOGRAMIC, 2000, Ideogramic Home Page. In: http://www.ideogramic.com. Acessado em: 03/2004.

JIANG, J., SYSTÄ, T., 2002, "Report: UML Model Exchange using XMI". *Technical Report of the Finland Tampere University of Technology*, Finland.

KOTLARSKY, J., KUMAR, K., HILLEGERSBERG, J., 2002, "Coordination and Collaboration for Globally Distributed Teams: The Case of Component-Based / Object-Oriented Software Development". In: *Proceedings of the International Workshop on Global Software development*, pp.23-28, Orlando, USA.

KRUEGER, C., 1992, "Software Reuse". *ACM Computing Surveys*, volume: 24, number: 2, pp.131-183.

LEON, A., 2000, *A Guide to Software Configuration Management*, Press: Artech House, ISBN: 1580530729.

LI, Q., XU, H., LIU, T., 2002, "Coordinating Transaction Model in CDBMS". In: *Proceedings of the 7th International Conference on Computer Supported Cooperative Work in Design*, pp.421-425, Rio de Janeiro, Brasil.

MANGAN, M., 2003, *Uma Arquitetura Transparente para um Ambiente Colaborativo de Desenvolvimento de Software*. Exame de Qualificação de D.Sc COPPE/UFRJ, Rio de Janeiro.

MANGAN, M., WERNER, C., BORGES, M., 2002, "Componentes para Colaboração Síncrona em um Ambiente de Reutilização de Software". Em: Anais do *XVI Simpósio Brasileiro de Engenharia de Software (SBES'02)*, pp. 372-377, Gramado, RS, Brasil.

MARIANI, J., RODDEN, T., 1991, "The Impact of CSCW on Database Technology". In: *Proceedings of IFIP International Workshop on CSCW*, pp.146-161, Berlin, Germany.

MAURER, F., MARTEL, S., 2002, "Process Support for Distributed Extreme Programming Teams". In: *Proceedings of* the *International Workshop on Global Software Development (ICSE'02)*, pp.39-43, Orlando, USA.

MIAN, P., NATALI, A., FALBO, R., 2001, "Ambientes de Desenvolvimento de Software e o Projeto ADS". *Revista Engenharia, Ciência e Tecnologia,* volume: 04, pp.3-10, Vitória, ES, Brasil.

MOLLI, P., SKAF-MOLLI, H., OSTER, G., et al., 2002, "SAMS: Synchronous, Asynchronous, Multi- Synchronous Environments". In: *Proceedings of the 7th International Conference on CSCW in Design (CSCWD'02),* pp.80-84, Rio de Janeiro, Brasil.

MOULTIS, N., KIRK, C., 2000, *XML Black Book,* Press: Makron Books, ISBN: 85-346-1262-5.

NARAYANASWAMY, K., GOLDMAN, N., 1992, "Lazy Consistency: A Basis for Cooperative Software Development". In: *Proceedings of the Computer Supported Cooperative Work (CSCW'92)*, pp. 257-264, Toronto, Canada.

ODYSSEY, 2004, Página do Projeto Odyssey. Em: http://www.cos.ufrj.br/~odyssey. Acessado em: 04/2004.

OHST, D., WELLE, M., KELTER, U., 2003, "Differences Between Versions of UML Diagrams". In: *Proceedings of the 9th European Software Engineering Conference and ACM SIGSOFT International Symposium of Foundations of Software Engineering,* volume: 28, issue: 5, pp.227-236, Helsinki, Finland.

OMG, 2002, Especificação do XMI 1.2. In: http://*www.omg.org*. Acessado em: 03/2004.

OMG, 2003, Especificação do UML 1.5. In: http://www.omg.org/uml. Acessado em: 03/2004.

OMG, 2003, Especificação do XMI 2.0. In: http://*www.omg.org*. Acessado em: 03/2004.

PINHEIRO, M. K., 2000, *Mecanismo de Suporte à Percepção em Ambientes Cooperativos*, Dissertação de M.Sc., PPGC/UFRGS, Porto Alegre, RS, Brasil.

PINHEIRO, M., LIMA, J., BORGES, M., 2001, "Awareness in Groupware Systems". In: *Proceedings of the International Database Engineering and Applications Symposium* (IDEAS'01), pp.323-335, San Jose, Costa Rica.

PINHEIRO, M., LIMA, J., BORGES, M., 2002, "A Framework for Awareness Support in Groupware Systems". In: *Proceedings of the 7th International Conference on Computer Supported Cooperative Work in Design (CDCWD'02)*, pp.13-18, Rio de Janeiro, Brasil.

PRAKASH, A., KNISTER, M., 1992, "Undoing Actions in Collaborative Work". In: *Proceedings of the Computer Supported Cooperative Work (CSCW'92)*, pp.273-280, Toronto, Canada.

PRAKASH, A., SHIM, H.S., LEE, J.H., 1999, "Data Management Issues and Tradeoffs in CSCW Systems". *IEEE Transactions on Knowledge and Data Engineering*, volume: 11, number: 1, pp. 213-227.

PREGUIÇA, N., MARTING, J.L., DOMINGOS, H., DUARTE, S., 2000, "Data Management Support for Asynchronous Groupware". In: *Proceedings of the 2000 ACM Conference on Computer-Supported Cooperative Work*, pp.69-78, Philadelphia, USA.

PRESSMAN, R., 2002, *Engenharia de Software*, Editora: Mc Graw Hill, 5ª edição, ISBN: 85-86804-25-8.

RAMAMPIARO, H., NYGARD, M., 1999, "Cooperative Database System: A Constructive Review of Cooperative Transactions Models". In: *Proceedings of the International Symposium on Database Applications in Non-Traditional Environments (DANTE 99)*, pp.315-324, Kyoto, Japan.

RATIONAL, 2004, Rational Home Page. In: http://*www.rational.com.* Acessado em: 03/2004.

REIS, R., LIMA, C., NUNES, D., 1997, "Cooperative Software Development and the Prosoft Environment". In: *Proceedings of The Twelfth International Symposium on Computer and Information Sciences (ISCIS XII)*, pp.494-501, Antalya, Turkey.

RITTER, N., 1994, "Supporting Cooperative Work by Conventional Database Transactions". In: *Proceedings of 7th International Conference On Systems Research, Informatics and Cybernetics, Advances in Database and Expert Systems*, pp.69-75, Baden-Baden, Germany.

ROBBINS, J., REDMILES, D., 2000, "Cognitive Support, UML Adhrence, and XMI Interchange in Argo/UML". In: *Proceedings of the Information & Software Technology Conference*, volume: 42, number: 2, pp.79-89.

ROY, J.; RAMANUJAN, A., 2001, "Understanding Web Services". *IEEE IT Professional Magazine*, volume: 3, number: 6, pp.69-73.

RUMBAUGH, J., JACOBSON, I., BOOCH, G., 1998, *The Unified Modeling Language Reference Manual,* Press: Addison-Wesley, ISBN: 020130998X.

SARMA, A., NOROOZI, Z., HOEK, A., 2003, "Palantír: raising awareness among configuration management workspaces". In: *Proceedings of the 25th International Conference on Software Engineering,* pp.444-454, Portland, USA.

SCHWARTZ, D., 1997, "Database Support for Conflict Detection in a Computer-Supported Cooperative Work Environment". In: *Proceedings of International Database Engineering and Applications Symposium (IDEAS'97)*, pp.240-249, Montreal, Canada

SOUZA, C., BASAVESWARA, S., REDMILES, D., 2002, "Supporting Global Software Development with Event Notification Servers". In: *Proceedings of the International Workshop on Global Software development*, pp.9-13, Orlando, USA.

SOUZA, C., REDMILES, D., DOURISH, P., 2003, "Breaking the code, moving between private and public work in collaborative software development". In: *Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work*, pp.105-114 Sanibel Island, USA.

STAR TEAM, 2003. Star Team, Borland. In: http://www.borland.com/starteam. Acessado em 04/2004.

SUN, C., CHEN, D., 2000, "A Multi-Version Approach to Conflict Resolution in Distributed Groupware Systems". In: *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS'2000)*, pp.316-325, Taipei, Republic of China.

SUZUKI, J., YAMAMOTO, Y., 1998, "Making UML Models Exchangeable Over the Internet With XML: UXF approach". *International Workshop (UML'98)*, pp.78-91, Mulhouse, France.

TAYLOR, J., 2001, "Can We Work Together"? *Computer Society Magazine*, volume: 34, number: 1, pp.4-6.

TELLIOGLU, H., 1996, "Configuration management in Collaborative Writing". In: *Proceedings of the Americas Conference on Information Systems (AIS)*, pp.594-598, Phoenix, USA.

TESCH, T., WÄSCH, J., 1995, "Transaction Support for Cooperative Hypermedia Document Authoring – A Study on Requirements". In: *Proceedings of the 8th Database Research Group Workshop on Database Issues and Infrastructure in Cooperative Information Systems*, pp.31-42, Trondheim, Norway.

TOLEDO, M., 1998, "A Framework Supporting Integrated Transaction and session Management for Cooperative Applications". *Relatório Técnico* número: TR-IC-98-18, UNICAMP, SP, Brasil.

TOMMARELLO, J., DEEK, F., 2002, "Collaborative Software Development: A Discussion of Problem Solving Models and Groupware Technologies". In: *Proceedings of the 35th Hawaii Internatonal Conference on System Sciences*, pp.41-51, Big Island, Hawaii.

TREVOR, J. KOCH, T. WOETZEL G., 1997, "MetaWeb: Bringing Synchronous Groupware to the World Wide Web". In: *Proceedings of the 6th European Conference on Computer Supported Cooperative Work (ECSCW'97)*, pp.65-80, Lancaster, UK.

UDDI, 2002, Especificação UDDI. In: http://www.uddi.org. Acessado em: 03/2004.

VAUGHAN-NICHOLS, S.J., 2002, "Web Services: Beyond the Hype". *Computer Magazine*, volume: 35, number: 2, pp.18-21.

VIEIRA, A., DIAS, D., CERQUEIRA, E., 2001, "Análise Comparativa de Ferramentas de Modelagem". *Relatório Técnico do Departamento de Ciência da Computação do IME-RJ*, Rio de Janeiro, Brasil.

VIEIRA, V., 2003, *Ariane: Um Mecanismo de Apoio à Percepção em Bases de Dados Compartilhadas*, Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

W3C, 2001, Especificação WSDL. In: http://www.w3c.org/TR/wsdl. Acessado em: 03/2004.

W3C, 2003, Especificação SOAP, In: http://www.w3c.org/TR/soap. Acessado em: 03/2004.

WEAR, A., GONG, Y., CHANG, K., 1995, "Database Management for Multimedia Distributed Collaborative Writing". In: *Proceedings of 33rd ACM Southeast Conference*, pp. 42-51, Clemson, USA.

WEINREICH, R., ALTMANN, J., 1997, "An Object-oriented Infraestructure for a Cooperative Software Development Environment". In: *Proceedings of the 5th International Sysmposium on Applied Corporate Computing (ISACC'97)*, Monterrey, Mexico.

WERNER, C., BRAGA, R., 2000, "Desenvolvimento Baseado em Componentes". Em: *Anais do XIV Simpósio Brasileiro de Engenharia de Software (SBES'00)*, pp.297-329, João Pessoa, Brasil.

WERNER, C., BRAGA, R., MATTOSO, M., et al., 2000, "Infra-estrutura Odyssey: Estágio Atual". *XIV Simpósio Brasileiro de Engenharia de Software (SBES'00)*, Caderno de Ferramentas, Seção: 3, João Pessoa, Brasil.

WERNER, C., MANGAN, M., MURTA, L., et al., 2002, "OdysseyShare: Um ambiente para o Desenvolvimento Cooperativo de Componentes". Em: Anais do *XVI Simpósio Brasileiro de Engenharia de Software (SBES'02)*, Ferramentas, pp. 444-449, Gramado, RS, Brasil.

WERNER, C., MANGAN, M., MURTA, L., et al., 2003, "OdysseyShare: an Environment for Collaborative Component-Based Development". *Information Reuse and Integration (IRI)*, Las Vegas, Nevada, USA.

WERNER, C., MATTOSO, M., BRAGA, R., et al., 1999, "Odyssey: Infra-Estrutura de Reutilização Baseada em Modelos de Domínio". Em: Anais do *XIII Simpósio Brasileiro de Engenharia de Software (SBES'99), Caderno de Ferramentas,* pp. 17-20, Florianópolis, Brasil.

ZHANG, A., NODINE, M., BHARAT, B., et al., 1994, "Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems". *SIGMOD Conference,* pp.67-78, Minneapolis, USA.

ZHANG, Y., KAMBAYASHI, Y., YANG, Y., SUN, C., 1998, "A Novel Timestamp Ordering Approach for Co-existing Traditional and Cooperative Transaction Processing". In: *Proceedings of the 3rd IFCIS Conference on Cooperative Information Systems (CoopIS'98),* pp.132-147, New York, USA.

ANEXO 1: XMI (*XML Metadata Interchange*)

INTRODUÇÃO

O XMI (XML Metadata Interchange) é um padrão usado para a troca de dados e metadados entre ferramentas em ambientes heterogêneos e distribuídos, garantindo interoperabilidade e consistência dessas informações. Em um cenário típico, as equipes de trabalho se encontrariam divididas em vários grupos, trabalhando em lugares geograficamente dispersos e utilizando ferramentas diferentes de vários fabricantes (JIANG e SYSTÄ, 2002).

A primeira versão do XMI foi adotada pelo OMG em Fevereiro de 1999, e a versão mais recente é a XMI 2.0 adotada em Maio de 2003.

O padrão XMI está diretamente relacionado com os padrões: MOF (OMG), UML (OMG) e XML (W3C). Para usar XMI, os dados devem estar mapeados em objetos definidos de acordo com a especificação MOF (Meta Object Facility). A UML foi adotado pelo OMG porque é um padrão bastante difundido para definir objetos, podendo ser usado para definir objetos que precisam ser compartilhados entre aplicações. Assim, como é possível mapear os modelos UML utilizando o MOF, e devido a maior "popularidade" da UML, esta dissertação faz referência a modelos UML, ao invés de modelos MOF. Combinando a UML com uma forma padrão de representar dados, é possível compartilhar os objetos usando efetivamente padrões e não tecnologias proprietárias. O XMI utiliza a UML para definir a estrutura dos objetos e das classes, e documentos XML para salvar e recuperar objetos (GROOSE et al., 2001) e (JIANG e SYSTÄ, 2002).

Embora não seja preciso ser um especialista em XML e UML para entender o padrão XMI, se faz necessário ter pelo menos um conhecimento básico sobre esses padrões. Não é objetivo dessa dissertação introduzir esses padrões. Um resumo prático é encontrado em (GROOSE et al., 2001). Maiores detalhes sobre XML são encontrados em (MOULTIS, 2000) e sobre UML em (RUMBAUGH et al, 1998).

XMI e XML

O XML é um padrão para representar dados na forma de documentos (arquivos) XML, que podem ser acessados por várias aplicações diferentes da que originou esses dados. Os documentos XML definem elementos e atributos, mas não definem objetos. O conteúdo dos documentos XML é definido pelos XML *Schemas* e pelos DTDs (*Document Type Definitions*). Um analisador (*parser)* XML pode determinar se um documento XML está de acordo com um XML Schema ou DTD através de um processo chamado validação.

O foco de estudo desta dissertação está relacionado ao trabalho cooperativo entre ferramentas CASE baseadas no paradigma da orientação a objetos. Quando ferramentas CASE distintas trabalham cooperativamente compartilhando dados entre elas, compartilham na verdade objetos. Por isso, o XMI é necessário, já que o XML não é orientado a objetos. O XMI define uma forma padrão de mapeamento entre documentos XML e objetos. Uma ferramenta que utiliza o XMI pode compartilhar objetos com outras ferramentas que também utilizem o XMI, mas para isso precisa ser capaz de salvar e recuperar os objetos compartilhados no formato XMI.

A figura 39 ilustra os dois passos envolvidos para representar objetos em XML.

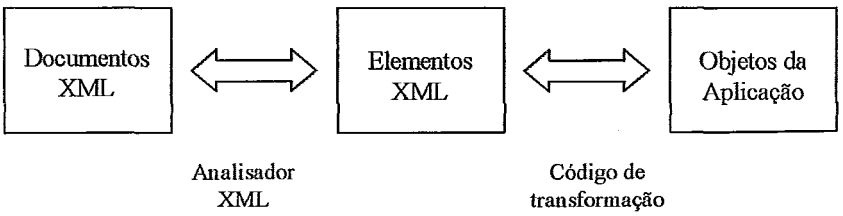


figura 39. Transformando objetos em elementos XML (GROOSE et al., 2001).

Depois que o analisador (*parser*) lê os elementos XML do documento XML, esses elementos precisam ser transformados em objetos. Na direção oposta, é preciso transformar os objetos em elementos XML, que são salvos pelo analisador XML em

documentos XML. Sem uma forma automática de fazer isso, é preciso escrever esse código de transformação, e cada vez que a aplicação mudar, ou a representação XML dos objetos mudar, é necessário atualizar esse código. Além disso, se cada aplicação implementar essa transformação de forma diferente, compartilhar objetos entre elas será muito difícil. Como o XMI define uma forma padrão de mapeamento entre o XML e os objetos, não é preciso definir uma forma proprietária para implementar a transformação entre objetos e XML e vice-versa.

A figura 40 mostra como um analisador XMI faz essa transformação, criando objetos a partir de um documento XMI e salvando os objetos em documentos XMI. Essa transformação é feita de forma automática e transparente para o usuário. Todo documento XMI é um documento XML, assim, os objetos são restaurados a partir de um documento XML e são convertidos para XML. Contudo, esses documentos XML estão de acordo com o padrão XMI.

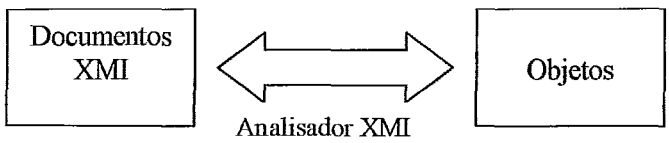


figura 40. Transformação usando o XMI (GROOSE et al., 2001).

O XMI não é uma extensão do XML, é construído sobre o XML. O XML é construído sobre o padrão unicode para representar dados. Da mesma forma, o XMI é construído sobre o XML para representar objetos (figura 41).

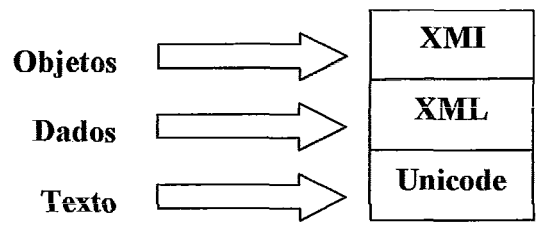


figura 41. O XMI é construído sobre o XML (GROOSE et al., 2001).

XMI e UML

O XMI utiliza a UML para definir a estrutura dos objetos. A versão do XMI 2.0 especifica como criar XML Schemas de modelos, e as versões anteriores especificam como criar DTDs de modelos. Além disso, o XMI 2.0 especifica como fazer engenharia reversa de documentos XML, XML Schemas e DTDs, para modelos UML (GROOSE et al., 2001).

A figura 42 ilustra um modelo UML e como ele é representado em XMI. O XMI especifica como criar um XMI Schema a partir do modelo.

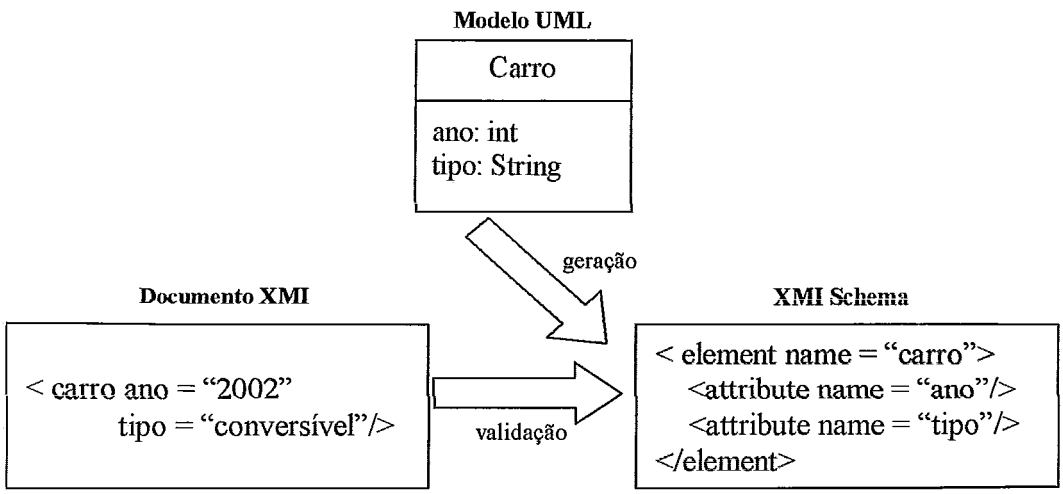


figura 42. Relacionamento entre UML, XMI e XMI Schema (GROOSE et al., 2001).

VANTAGENS X DESVANTAGENS DO XMI

Dentre as vantagens, ainda não citadas, podem ser destacadas (JIANG e SYSTÄ, 2002):

- O XMI pode enviar apenas as diferenças entre documentos, assim, o documento completo só precisa ser enviado uma única vez. O modelo novo pode ser obtido adicionando as diferenças ao modelo original.

- Um documento XMI pode fazer referência a um elemento XMI de outro documento.

- O XMI possui mecanismos para extensão, assim, ferramentas diferentes podem modificar o modelo livremente, mas esse mecanismo também pode ser uma desvantagem, como pode ser visto a seguir na lista das desvantagens do XMI.

- O XML possui a capacidade de enviar somente uma parte de um modelo.

Dentre as desvantagens, ainda não citadas, podem ser destacadas (JIANG e SYSTÄ, 2002):

- O XMI especifica como representar um modelo UML, mas não guarda informações sobre como apresentá-lo graficamente. Para armazenar essas informações, os fabricantes das ferramentas normalmente estendem o XMI. Muitas vezes a forma de fazer isso de uma ferramenta não é entendida por outra ferramenta. Isso porque não existe um padrão para representar essas extensões, o que acarreta perda de informação. Essas questões já estão sendo estudadas pela OMG.

- O XMI não é ideal para acessar informações de forma instantânea. Isso porque os documentos XMI tendem a ser muito grandes, principalmente devido ao grande uso de *tags* para representar elementos diferentes. Analisar um documento XMI e carregar um modelo para uma ferramenta CASE, pode ser uma tarefa muito lenta. O tamanho do documento XMI pode variar muito dependendo da ferramenta utilizada, mesmo representando um único modelo. Uma solução possível para otimizar a tarefa de acessar informações de um documento XMI seria primeiro analisar o documento XMI e armazenar as informações extraídas em um repositório, para depois fazer as consultas desejadas.

UTILIZAÇÃO DO XMI EM FERRAMENTAS CASE

As ferramentas CASE estão incorporando o mecanismo de importação e exportação dos seus modelos em documentos XMI, para permitir o intercâmbio de dados com outras ferramentas CASE. Como o XMI ainda não está padronizado e periodicamente surgem novas versões de XMI DTDs ou XMI Schemas, ferramentas CASE distintas podem utilizar versões diferentes para representar seus modelos em documentos XMI. Dependendo das ferramentas CASE utilizadas, é preciso fazer conversão do formato do documento, procurando minimizar a perda de informação que pode ocorrer durante essa conversão, para que essas ferramentas possam trocar dados entre elas.

A OMG está trabalhando na convergência destes padrões para um que atenda a maior parte das necessidades das diversas ferramentas CASE. Este padrão provavelmente deverá mapear modelos UML através de esquemas XMI (XMI Schemas) e não, como ainda a maioria das ferramentas CASE fazem, através de DTDs.

A seguir, é apresentado um exemplo, utilizando o Rational Rose[2], (RATIONAL, 2004), que mapeia seus modelos de acordo com o XMI 1.1 e com o UMLX13-11.dtd, e o Ideogramic (IDEOGRAMIC, 2001), que mapeia seus modelos de acordo com o XMI 1.0 e com o UMLX13.dtd. Algumas das diferenças na sintaxe desses DTDs estão destacadas em negrito nas figuras 19 e 20.

[2] O Rose mapeia seus modelos de acordo com a versão XMI 1.0, que utiliza o UMLX13.dtd, ou com a versão XMI 1.1, que utiliza o UMLX13-11.dtd.

```
<?xml version = '1.0' encoding='ISO-8859-1'?>
<!DOCTYPE XMI SYSTEM 'UMLX131.dtd'>
<XMI xmi.version = '1.1' xmlns:UML='href://org.omg/UML/1.3' timestamp = 'Sat Apr 05 13:46:25 2003' >
<XMI.header>
<XMI.documentation>
</XMI.documentation>
<XMI.metamodel xmi.name = 'UML' xmi.version='1.3'/>
</XMI.header>
<XMI.content>
<!-- ========== escola    [Model] ==========>
<UML:Model xmi.id = 'G.0'
name = 'escola' visibility = 'public' isSpecification = 'false'
isRoot = 'false' isLeaf = 'false' isAbstract = 'false' >
<UML:Namespace.ownedElement>
<!-- ========== escola::Turma    [Class] ==========>
<UML:Class xmi.id = 'S.094.1346.05.1'
name = 'Turma' visibility = 'public' isSpecification = 'false'
.....
```

figura 43. Exemplo de arquivo XMI exportado pelo Rose (GROOSE et al., 2001).

```
<?xml version="1.0" encoding="UTF    -8" ?>
<!DOCTYPE XMI SYSTEM   "UMLX13.DTD"
<!ELEMENT DiagramExtensions (RealDiagramType|ActivationView|ZoomAndPan|Freehan        dDrawing|NameFigurePosition)*>
.....
<XMI xmi.version="1.0">
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Ideogramic UML</XMI.exporter>
      <XMI.exporterVersion>2.3.3</XMI.exporter    Version>
    </XMI.documentation>
    <XMI.metamodel xmi.name="UML" xmi.version="1.3"/>
  </XMI.header>
  <XMI.content>
    <Model_Management.Model   xmi.id="idmodel0  -3E81E0BD">
      <Foundation.Core.ModelElement.name>Untitled</Foundat      ion.Core.ModelElement.name>
      <Foundation.Core.ModelElement.visibility xmi.value="public"/>
      <Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
      <Foundation.Core.Namespace.ownedElement>
        <Founda tion.Core.Class  xmi.id="idclass0  -3E81E0BD">
          <Foundation.Core.ModelElement.name>Aluno</Foundation.Core.ModelElement.name>
          <Foundation.Core.ModelElement.visibility xmi.value="public"/>
          <Foundation.C   ore.ModelElement.isSpecification xmi.value="false"/>
  .....
```

figura 44. Exemplo de arquivo XMI exportado pelo Ideogramic (GROOSE et al., 2001).

ANEXO 2: Serviços WEB (*WEB Services*)

INTRODUÇÃO

Uma definição para Serviço Web é: "Um módulo de software que contém uma única tarefa ou um conjunto de tarefas, que podem ser localizadas e invocadas através de uma rede, incluindo e especialmente a Internet (BORLAND, 2002)". Atualmente, se tem falado muito sobre serviços Web e, constantemente, novas definições surgem para o termo, embora todas sejam similares (GUNZER, 2002).

Este anexo tem como objetivo apresentar algumas definições, encontradas na literatura, para serviços Web, suas principais características e as tecnologias envolvidas. Antes disso, será apresentada uma introdução sobre a história da programação distribuída, para um melhor entendimento dos serviços Web, e mostrar que se trata de um novo passo nesse contexto.

PROGRAMAÇÃO DISTRIBUÍDA

Com o surgimento das redes de computadores, nasceu o paradigma da programação distribuída. Primeiro, as aplicações foram separadas entre servidores (aplicações que disponibilizam algum serviço) e clientes (aplicações que usam esses serviços). Essa arquitetura de distribuição (duas camadas) foi a predominante nos anos 90. Um dos seus atrativos é o aumento da confiabilidade (a falha de uma máquina não necessariamente inviabiliza a operação do sistema como um todo) e redução de custo (máquinas mais simples podem executar os serviços isoladamente, ao invés de ter uma grande máquina fazendo todos os serviços).

As aplicações cliente/servidor são programas executados em máquinas distintas, e que trocam informações através de uma rede de computadores. Para que os serviços possam ser solicitados, a aplicação cliente deve conhecer quem fornece o serviço (o endereço da aplicação servidora) e qual o protocolo pré-estabelecido para realizar a solicitação.

Para aumentar a confiabilidade e a escalabilidade, uma terceira camada foi introduzida, separando a aplicação em três camadas: camada de apresentação, uma camada do meio - contendo a lógica do negócio, e uma terceira camada - de armazenamento.

A base para a comunicação entre as partes distribuídas de uma aplicação é o mecanismo de Chamada a Procedimentos Remotos (*Remote Procedure Call - RPC*). Os ambientes de desenvolvimento que suportam o RPC "escondem" do programador boa parte dos detalhes envolvidos no uso das facilidades de comunicação providas pela rede de computadores. O RPC tem por objetivo possibilitar que procedimentos que se encontram em outras máquinas possam ser chamados da mesma forma que procedimentos encontrados na máquina que originou a chamada ao procedimento. Ambientes de desenvolvimento que suportam RPC existem para diversas plataformas computacionais.

Para poupar os desenvolvedores das tarefas de "baixo nível" , como conversão de tipos de dados entre máquinas diferentes, foi introduzida uma nova camada chamada *middleware*. Consiste em uma camada de software que permite ligar componentes de diferentes aplicações. Também é chamada de *plumbing (*encanamento*)*, pois liga dois lados de uma aplicação e permite a passagem de informação entre eles. Esta ligação é efetuada dinamicamente sem intervenção do programador (GUNZER, 2002).

Entre os vários *middleware* existentes, os três mais populares são baseados no modelo de programação orientada a objetos, são eles (GUNZER, 2002):

- **CORBA** (*Common Object Request Broker Architecture*) - CORBA é uma solução baseada em padrões abertos para computação distribuída, permite que partes de um programa, especificamente objetos, se comuniquem entre si independentemente da linguagem ou sistema operacional em que foram desenvolvidos. O CORBA possui alto desempenho em aplicações distribuídas, contudo escrever uma aplicação distribuída com CORBA é uma tarefa complexa, por exemplo, o mapeamento dos objetos CORBA, definidos pela linguagem de definição de interface (IDL - *Interface Definition Language),* para as linguagens de programação é restrito às linguagens suportadas. A comunicação entre objetos é feita através dos

ORBs (*Object Request Brokers), que* atuam como um *middleware* entre clientes e servidores. Para que a comunicação seja feita, é preciso que nos dois lados da comunicação estejam ORBs compatíveis. O CORBA possui um grande conjunto de serviços que podem ser chamados para atender vários tipos de aplicação. Isso torna a especificação muito complexa e extensa. Por isso, normalmente é implementado um conjunto pequeno de serviços.

- **RMI** (*Remote Method Invocation*) - Permite a criação de aplicações distribuídas Java-to-Java, tendo como base um sistema de objetos distribuídos. Nesse sistema, os nós servidores exportam referências de objetos, chamados objetos remotos. Um cliente que obtenha uma dessas referências pode fazer chamadas aos objetos Java remotos no servidor. Os parâmetros do método são empacotados e enviados para o objeto no servidor que, depois de desempacotá-los, executa o método. O valor de retorno é novamente empacotado e enviado para o cliente. O protocolo de comunicação utilizado é o JRMP (*Java Remote Method Protocol).* Dentre suas vantagens, é possível citar: nenhuma linguagem abstrata, como a IDL, é utilizada para descrever os objetos remotos; o RMI tem suporte ao coletor de lixo (*Garbage Collection)* distribuído. Por outro lado, para ser usado, é preciso ter aplicações Java em ambos os lados da comunicação. A simplicidade do uso é alcançada mantendo o *middleware* o mais simples possível. Além disso, não fornece serviços como os do CORBA. Implementar os serviços necessários fica por conta do desenvolvedor.

- **DCOM** (*Distributed Object Model*) - O DCOM é uma extensão do COM (*Component Object Model),* para suportar componentes num ambiente distribuído. Foi desenvolvido pela Microsoft e, desde 1996, foi integrado no sistema operacional *Windows NT.* O COM define como componentes e seus clientes interagem, sendo que a comunicação é feita sem a necessidade de um componente intermediário. O COM provê esta comunicação de uma forma completamente transparente. O COM fornece

serviços orientados a objetos para clientes e componentes, usa RPC para a comunicação (o protocolo usado é o *ORPC - Object Remote Procedure Call*) e provê mecanismos de segurança para gerar pacotes de rede que se ajustam com o protocolo padrão DCOM. Como o RMI, o DCOM tem suporte ao coletor de lixo de objetos remotos distribuídos. Ao contrário do CORBA, que possui implementações para vários sistemas operacionais, o DCOM é usado geralmente para Windows. Contudo, existem algumas implementações para outras plataformas, como para Unix.

Baseado no que foi dito, é possível fazer as seguintes considerações sobre os três *middlewares:*

- Eles trabalham de forma similar. As diferenças estão nas características que fornecem suporte e no nível de complexidade.
- Devido aos diferentes protocolos utilizados torna-se inviável, por exemplo, fazer chamadas a um servidor DCOM de um cliente RMI.
- São tipicamente usados em *Intranets.* Todos eles precisam usar um "túnel" HTTP, para acessar um servidor atrás de um *firewall.*

QUE É UM SERVIÇO WEB?

Um Serviço Web pode ser: "Um conjunto de protocolos e padrões, que permitem a comunicação entre aplicações através de uma rede, geralmente da Internet. Essa comunicação baseada em padrões permite que as aplicações descrevam o que fazem, e que possam assim chamar e utilizar os serviços de outra aplicação (VAUGHAN-NICHOLS, 2002)".

As definições encontradas na literatura são similares. Na essência, serviços Web são aplicações que podem ser publicadas, localizadas e invocadas através da Internet. Na Internet, existem ambientes heterogêneos nos dois lados da comunicação e não é possível saber previamente qual tipo de *middleware* cada lado utiliza. Essa é uma das

principais razões para a não qualificação de *middlewares* para serviços Web. Assim, é necessária uma nova abordagem para serviços Web (GUNZER, 2002).

Trata-se de uma tecnologia emergente que tem o potencial de mudar o modo como a Internet funciona para as empresas. Navegar em páginas *Web* para fazer seus pedidos é ótimo para indivíduos (aplicativos conhecidos como B2C ou *business-to-consumer*, empresa ao consumidor), mas não para empresas (aplicativos B2B ou *business-to-business*, empresa a empresa). Se uma pessoa deseja comprar um livro, ir até um sítio especializado e fazer o pedido é, provavelmente, muito prático. Mas se uma livraria deseja fazer centenas de pedidos por dia, essa está longe de ser uma boa estratégia, principalmente se tiver um programa que ajude a controlar as vendas e determinar novos pedidos. Pegar a saída desse programa e fornecê-la como entrada de dados para outro aplicativo não é viável.

A noção de serviços Web serve para resolver esse problema: o programa usado para controlar vendas pode criar, automaticamente, um pedido e enviá-lo para um Serviço Web, que pode retornar imediatamente as informações sobre o pedido. O próximo passo poderia ser solicitar um número de controle para o envio. Nesse ponto, o programa poderia usar outro Serviço Web para controlar o envio até seu destino assim, seria possível dizer aos clientes quanto tempo eles terão que esperar. Os serviços Web servem para o intercâmbio de computadores tanto quanto a *Web* e o *e-mail* permitem às pessoas interagirem (CANTÚ, 2002).

Os serviços Web compreendem um conjunto de tecnologias de *plataforma-neutra*, designadas para facilitar a distribuição de serviços através de Intranets ou da Internet. A capacidade de comunicação e utilização de serviços entre plataformas diferentes é o principal atrativo dos serviços Web, já que há muito tempo a interoperabilidade tem sido o principal objetivo da comunidade que estuda sistemas distribuídos (VAUGHAN-NICHOLS, 2002).

Um Serviço Web deve possuir as seguintes características:

- Ser hábil para mostrar e descrever a si mesmo para outras aplicações, permitindo que estas entendam o que seus serviços fazem.
- Poder ser facilmente localizado por outras aplicações.
- Poder ser usado pela aplicação que o invocou usando o protocolo de rede.

Um Serviço Web é um componente remoto que contém funcionalidades (serviços) que podem ser invocados usando os protocolos da Internet, principalmente HTTP, (BARISH, 2002).

Os serviços Web dependem da habilidade das partes para se comunicar, mesmo que usem sistemas de informação diferentes. Toda a troca de dados necessária entre os dois lados da comunicação é feita através de documentos XML.

Os serviços podem ser implementados usando qualquer linguagem de programação (Java, C++, Visual Basic,...), e em qualquer plataforma. Um cliente de um Serviço Web também pode ser escrito usando qualquer linguagem e em qualquer plataforma. Por exemplo, um cliente escrito em Delphi na plataforma Windows pode utilizar serviços de um Serviço Web escrito em Java na plataforma Linux. O cliente não precisa se preocupar em como o serviço foi implementado. Ou seja, os serviços Web dependem da habilidade das partes para se comunicar mesmo que usem plataformas diferentes.

Assim, é possível pensar em um Serviço Web como funcionalidades acessíveis aos clientes, isto é (BARISH, 2002):

- **Linguagem Neutra:** a linguagem usada para implementar o cliente não tem que ser igual à usada para implementar o serviço
- **Plataforma Neutra:** o protocolo HTTP é o "igualador" das plataformas.

- **Tecnologia-Objeto Neutra:** o cliente não precisa se preocupar como os objetos são gerenciados ou distribuídos.

Como o acesso aos serviços Web é feito através da Internet, possuem automaticamente os seguintes benefícios (BARISH, 2002):

- A comunicação é feita via HTTP (porta 80), por isso, a integração entre as aplicações não implica em mudanças nas suas configurações de rede, como de *firewalls* e *proxies* especificamente. Problemas com *firewall* não deverão ocorrer, pois normalmente não bloqueiam a porta HTTP. Mas é importante lembrar que pela definição os serviços Web não usam somente HTTP, o uso de smtp ou ftp também pode ser considerado.

- Autenticação HTTP automática, já que essa é uma característica do protocolo.

- Conexão persistente HTTP, na versão 1.1, também é uma característica do protocolo.

ARQUITETURA

Sua arquitetura permite o desenvolvimento de serviços Web que encapsulam todos os níveis de funcionalidade necessários. Ou seja, um Serviço Web pode ser bem simples ou muito complexo. A arquitetura também permite que vários serviços Web sejam combinados para criar uma nova funcionalidade.

Sua arquitetura é composta por três entidades (figura 45):

- **Fornecedor** (*provider*)**:** Quem cria e publica o Serviço Web para o "mundo externo", registrando o serviço no *Broker*.

- **Intermediário** (*broker*)**:** Mantém o registro da publicação do serviço.

- **Requisitor** (*requestor*)**:** É a aplicação cliente que acha e consome o serviço.

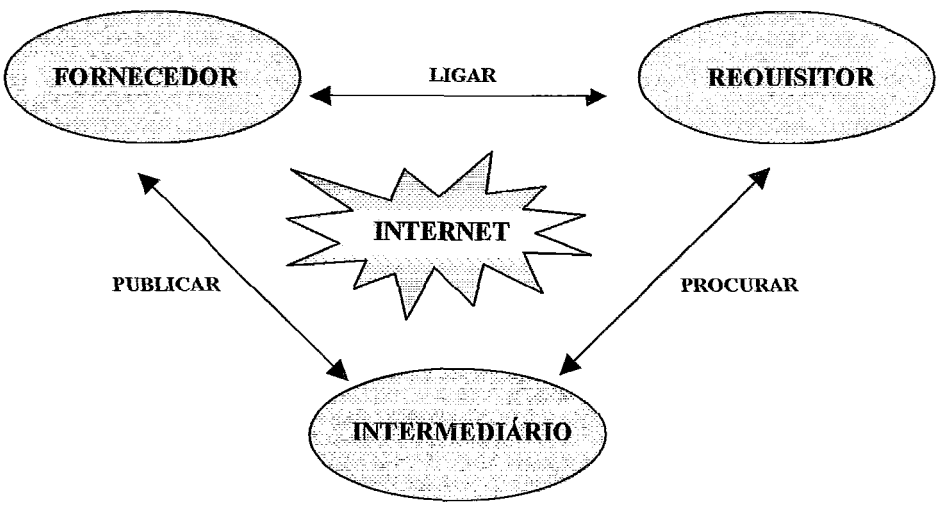


figura 45. Arquitetura dos serviços Web (BORLAND, 2002).

O **intermediário** fornece uma maneira para o **fornecedor** e o **requisitor** se comunicar. O **requisitor** também pode ser cliente de outro Serviço Web.

As três entidades interagem entre si através das operações **publicar** (*publish*), **achar** (*find*) e **ligar** (*bind*). O **fornecedor** informa ao **intermediário** a existência de um Serviço Web, usando a interface de publicação do **intermediário (publicar)**, para tornar o serviço disponível aos clientes. A informação publicada descreve o serviço e especifica o local onde se encontra. O **requisitor** consulta o **intermediário** para localizar **(procurar)** um Serviço Web publicado. Com essa informação, o **requisitor** faz uma ligação **(ligar)** da sua aplicação com o **fornecedor** do serviço para poder utilizá-lo.

TECNOLOGIAS ENVOLVIDAS

Serviços Web são essencialmente baseados em três tecnologias (ROY e RAMANUJAN, 2001):

- **WSDL** (*Web Services Description Language*)

Linguagem baseada em XML usada para definir os serviços Web e descrever como podem ser invocados. O documento WSDL tem um papel semelhante ao arquivo IDL (*Interface Definition Language*) do CORBA. São outro tipo de documento XML, e fornecem a definição de metadados de uma requisição SOAP. Um arquivo nesse formato é publicado para definir um serviço, sendo possível escrever um programa para chamar esse serviço (CANTÚ, 2002).

Enquanto o SOAP especifica a comunicação entre a aplicação cliente e o fornecedor do serviço, WSDL descreve o serviço disponível. Pode ser utilizado como uma "receita" para gerar as mensagens SOAP para acessar o serviço. Sua especificação é encontrada em (W3C, 2001).

- **UDDI** (*Universal Description, Discovery and Integration*)

Uma vez que um Serviço Web é definido usando WSDL, é preciso publicar de alguma forma sua existência para o "resto do mundo".

O UDDI é um Serviço Web que funciona como um registro para outros serviços Web, servindo como uma base de dados de serviços disponíveis. Através dele é possível registrar (publicar) e procurar, sem qualquer custo, por serviços Web disponíveis. Desta forma, as empresas aumentam a visibilidade de seus serviços, pois UDDI pode ser acessado tanto pelo navegador (*browser*) como se fosse qualquer outro sítio de busca, como também de dentro de qualquer aplicação (BARISH, 2002).

Utilizando as tecnologias e protocolos do UDDI, é possível publicar os serviços disponíveis no UDDI *Registry* (registro de negócio). O registro de negócio UDDI é

um serviço, descrito em um documento XML, logicamente centralizado e fisicamente distribuído, com múltiplos nós raiz, que replicam os dados de um para o outro. Cada serviço disponível só precisa ser publicado em um único registro. Uma vez que o serviço é publicado em um registro UDDI, os dados são automaticamente replicados para outros nós raiz UDDI, e se tornam disponíveis para os clientes.

O modelo de informação usado pelos registros UDDI é definido em um XML Schema. O padrão XML Schema foi escolhido por fornecer suporte a tipos de dados complexos, e pela habilidade de descrever e validar a informação, baseado nos modelos de informação representados em esquemas.

UDDI é desenvolvido e mantido por várias empresas, incluindo, Microsoft, Oracle e IBM. Sua especificação é encontrada em (UDDI, 2002).

- **SOAP** (*Simple Object Access Protocol*)

Uma vez que os serviços Web são utilizados em ambientes heterogêneos, o protocolo usado para transferência de dados entre esses ambientes deve ser independente de qualquer plataforma. SOAP é um protocolo com essa característica (GUNZER, 2002).

O SOAP foi desenvolvido originalmente pela DevelopMentor (empresa de treinamento) e pela Microsoft para superar as deficiências do uso de DCOM, dentro de servidores *Web*. Submetido ao W3C para padronização, vem sendo adotado por várias empresas.

A especificação do SOAP descreve como chamar um Serviço Web e processar sua resposta, ou seja, define uma maneira simples de "empacotar" a informação para ser trocada entre sistemas. O SOAP vem incorporado no protocolo HTTP padrão, de modo que um servidor *Web* pode manipular os pedidos SOAP e os pacotes de dados relacionados podem passar por *firewalls*. O SOAP define uma notação baseada em XML para solicitar a execução de um método por um objeto no servidor (*RPC*), passando parâmetros para ele, e uma notação para definir o formato de uma resposta. (CANTÚ, 2002).

Resumindo, o SOAP é um protocolo aberto que define uma maneira uniforme de realizar RPCs, usando HTTP como protocolo de comunicação e XML como formato para serialização de dados.

O SOAP é um protocolo estritamente textual. Para garantir a uniformização da interpretação dos parâmetros, dos nomes dos métodos e dos valores de retorno, o protocolo SOAP utiliza um vocabulário XML para descrever suas mensagens.

Uma mensagem SOAP contém a *tag* **ENVELOPE** como raiz dos elementos. Essa *tag* possui (ROY e RAMANUJAN, 2001):

- **Cabeçalho:** É opcional. Quando existe, contém informações importantes como as relacionadas a segurança, autenticação e codificação de dados, itinerário das mensagens ou ainda informações sobre como um recipiente de mensagens SOAP deve processar uma mensagem.

- **Corpo:** É obrigatório. Contém a mensagem XML que vai ser transmitida.

SOAP é um protocolo relativamente fácil de usar. Um dos principais objetivos no projeto do SOAP é que a especificação seja a mais concisa possível, para que desta forma o protocolo seja facilmente entendido e utilizado. Sua especificação SOAP é encontrada em (W3C, 2003).

RELAÇÃO ENTRE AS TECNOLOGIAS

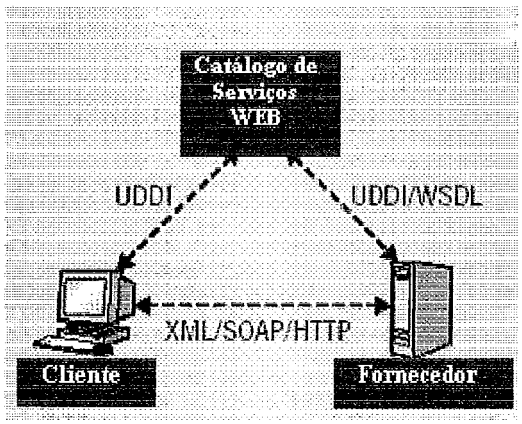


figura 46. Tecnologias Envolvidas (VAUGHAN-NICHOLS, 2002).

O **fornecedor** usa o protocolo UDDI para enviar informações ao diretório de **serviços Web**, onde se encontram as descrições de serviços Web disponíveis, publicando seus serviços. A informação enviada é codificada em WSDL. O **cliente** envia um pedido de serviço ao diretório de **serviços Web**. O diretório manda uma resposta ao **cliente** com os serviços disponíveis, essa comunicação também é feita utilizando o protocolo UDDI. O **cliente** e o **fornecedor** interagem via HTTP, SOAP e XML (VAUGHAN-NICHOLS, 2002).

Os *middlewares* apresentados anteriormente (CORBA, RMI e DCOM) utilizam, normalmente, algum tipo de protocolo binário para comunicação. Os serviços Web utilizam XML "no topo" do HTTP.

CONSIDERAÇÕES FINAIS SOBRE SERVIÇOS WEB

Desde seu começo, a Web tem sido uma ampla rede de dados distribuídos. Com a tecnologia de serviços Web, espera-se que no futuro a Web seja também uma rede de serviços distribuídos. A introdução de serviços Web no mercado tornará o

desenvolvimento de aplicações integradas através da Web uma realidade através da adoção do protocolo SOAP.

Um dos principais fatores para a forte aceitação da tecnologia de serviços Web pelo mercado é o fato de ser baseada em protocolos abertos, e que utilizam tecnologias amplamente estabelecidas na Internet, como HTTP e XML. Os padrões XML e de serviços Web oferecem um alto grau de interoperabilidade. Assim, se torna muito mais fácil construir sistemas que precisam ser cada vez mais integrados, em um mundo que se torna cada vez mais heterogêneo.

Questões relativas à programação de aplicações distribuídas, como desempenho, escalabilidade, tolerância à falhas e segurança, também precisam ser tratadas quando se fala em serviços Web. Estas questões ainda precisam ser amadurecidas na tecnologia de serviços Web. Os serviços Web ainda têm um longo caminho para percorrer antes que os programadores consigam usá-los para criar uma aplicação de negócio robusta. Mas, os primeiros passos já foram dados.

ANEXO 3: WSDL dos Serviços do MIMIX

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions
    targetNamespace="http://www.interbrainwebservices.com.br/teste/
    Mimix" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://www.interbrainwebservices.com.br/teste/Mimix
    -impl"
    xmlns:intf="http://www.interbrainwebservices.com.br/teste/Mimix"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <wsdl:types>
    - <schema
        targetNamespace="http://www.interbrainwebservices.com.b
        r/teste/Mimix"
        xmlns="http://www.w3.org/2001/XMLSchema">
        <import
          namespace="http://schemas.xmlsoap.org/soap/encodi
          ng/" />
      - <complexType name="ArrayOf_xsd_string">
        - <complexContent>
          - <restriction base="soapenc:Array">
              <attribute ref="soapenc:arrayType"
                wsdl:arrayType="xsd:string[]" />
            </restriction>
          </complexContent>
        </complexType>
        <element name="ArrayOf_xsd_string" nillable="true"
          type="intf:ArrayOf_xsd_string" />
      - <complexType name="ProjetoMimix">
        - <sequence>
```

```
        <element name="dataInicioProjeto" nillable="true"
          type="xsd:string" />
        <element name="dataPrevistaTerminoProjeto"
          nillable="true" type="xsd:string" />
        <element name="descricaoProjeto" nillable="true"
          type="xsd:string" />
        <element name="idProjeto" type="xsd:int" />
        <element name="nomeProjeto" nillable="true"
          type="xsd:string" />
      </sequence>
    </complexType>
  - <complexType name="ArrayOfProjetoMimix">
    - <complexContent>
      - <restriction base="soapenc:Array">
          <attribute ref="soapenc:arrayType"
            wsdl:arrayType="intf:ProjetoMimix[]" />
        </restriction>
      </complexContent>
    </complexType>
    <element name="ArrayOfProjetoMimix" nillable="true"
      type="intf:ArrayOfProjetoMimix" />
  - <complexType name="DocumentoMimix">
    - <sequence>
        <element name="enderecoDocumento"
          nillable="true" type="xsd:string" />
        <element name="idAutorDocumento"
          type="xsd:int" />
        <element name="idDocumento" type="xsd:int" />
        <element name="idProjeto" type="xsd:int" />
        <element name="nomeDocumento" nillable="true"
          type="xsd:string" />
        <element name="versaoDocumento" type="xsd:int"
          />
      </sequence>
```

```
  </complexType>
- <complexType name="ArrayOfDocumentoMimix">
  - <complexContent>
    - <restriction base="soapenc:Array">
        <attribute ref="soapenc:arrayType"
          wsdl:arrayType="intf:DocumentoMimix[]" />
      </restriction>
    </complexContent>
  </complexType>
  <element name="ArrayOfDocumentoMimix" nillable="true"
    type="intf:ArrayOfDocumentoMimix" />
- <complexType name="UsuarioMimix">
  - <sequence>
      <element name="descricaoUsuario" nillable="true"
        type="xsd:string" />
      <element name="idUsuario" type="xsd:int" />
      <element name="nomeUsuario" nillable="true"
        type="xsd:string" />
      <element name="senhaUsuario" nillable="true"
        type="xsd:string" />
    </sequence>
  </complexType>
- <complexType name="ArrayOfUsuarioMimix">
  - <complexContent>
    - <restriction base="soapenc:Array">
        <attribute ref="soapenc:arrayType"
          wsdl:arrayType="intf:UsuarioMimix[]" />
      </restriction>
    </complexContent>
  </complexType>
  <element name="ArrayOfUsuarioMimix" nillable="true"
    type="intf:ArrayOfUsuarioMimix" />
  <element name="UsuarioMimix" nillable="true"
    type="intf:UsuarioMimix" />
```

```
- <complexType name="ElementoComparacaoMimix">
  - <sequence>
      <element name="nivel" type="xsd:int" />
      <element name="nomeElementoDocumento1"
        nillable="true" type="xsd:string" />
      <element name="nomeElementoDocumento2"
        nillable="true" type="xsd:string" />
      <element name="nomeElementoResultante"
        nillable="true" type="xsd:string" />
      <element name="operacao" nillable="true"
        type="xsd:string" />
      <element name="tipoElemento" nillable="true"
        type="xsd:string" />
    </sequence>
  </complexType>
- <complexType
    name="ArrayOfElementoComparacaoMimix">
  - <complexContent>
    - <restriction base="soapenc:Array">
        <attribute ref="soapenc:arrayType"
          wsdl:arrayType="intf:ElementoComparacao
          Mimix[]" />
      </restriction>
    </complexContent>
  </complexType>
  <element name="ArrayOfElementoComparacaoMimix"
    nillable="true"
    type="intf:ArrayOfElementoComparacaoMimix" />
</schema>
</wsdl:types>
- <wsdl:message name="getDocumentosDoProjetoResponse">
  <wsdl:part name="getDocumentosDoProjetoReturn"
    type="intf:ArrayOfDocumentoMimix" />
</wsdl:message>
```

```
  <wsdl:message name="getVersaoMimixRequest" />
- <wsdl:message name="LoginRequest">
    <wsdl:part name="usuario" type="xsd:string" />
    <wsdl:part name="senha" type="xsd:string" />
  </wsdl:message>
- <wsdl:message name="getDocumentosDoProjetoRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
    <wsdl:part name="idProjeto" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="NovoProjetoRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
    <wsdl:part name="nomeNovoProjeto" type="xsd:string" />
    <wsdl:part name="descricaoNovoProjeto" type="xsd:string" />
    <wsdl:part name="dataInicioNovoProjeto" type="xsd:string" />
    <wsdl:part name="dataTerminoNovoProjeto" type="xsd:string"
      />
  </wsdl:message>
- <wsdl:message name="getUsuariosDoProjetoRequest">
    <wsdl:part name="idProjeto" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="getDocumentosResponse">
    <wsdl:part name="getDocumentosReturn"
      type="intf:ArrayOfDocumentoMimix" />
  </wsdl:message>
- <wsdl:message name="getProjetosResponse">
    <wsdl:part name="getProjetosReturn"
      type="intf:ArrayOfProjetoMimix" />
  </wsdl:message>
- <wsdl:message name="getDocumentoResponse">
    <wsdl:part name="getDocumentoReturn"
      type="intf:ArrayOf_xsd_string" />
  </wsdl:message>
- <wsdl:message name="JuncaoAssistidaDocumentosResponse">
```

```
    <wsdl:part name="JuncaoAssistidaDocumentosReturn"
      type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="getProjetosDoUsuarioRequest">
    <wsdl:part name="idUsuario" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="AssociaUsuarioProjetoRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
    <wsdl:part name="idUsuario" type="xsd:int" />
    <wsdl:part name="idProjeto" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="LoginResponse">
    <wsdl:part name="LoginReturn" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="getUsuariosOnLineRequest" />
  <wsdl:message name="getUsuariosRequest" />
- <wsdl:message name="getProjetosDoUsuarioResponse">
    <wsdl:part name="getProjetosDoUsuarioReturn"
      type="intf:ArrayOfProjetoMimix" />
  </wsdl:message>
- <wsdl:message name="getUsuarioResponse">
    <wsdl:part name="getUsuarioReturn" type="intf:UsuarioMimix"
      />
  </wsdl:message>
- <wsdl:message name="NovoUsuarioRequest">
    <wsdl:part name="nomeNovoUsuario" type="xsd:string" />
    <wsdl:part name="senhaNovoUsuario" type="xsd:string" />
    <wsdl:part name="descricaoNovoUsuario" type="xsd:string" />
  </wsdl:message>
- <wsdl:message name="LogoutResponse">
    <wsdl:part name="LogoutReturn" type="xsd:boolean" />
  </wsdl:message>
- <wsdl:message name="DesassociaUsuarioProjetoRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
```

```
    <wsdl:part name="idUsuario" type="xsd:int" />
    <wsdl:part name="idProjeto" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="NovoProjetoResponse">
    <wsdl:part name="NovoProjetoReturn" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="NovoDocumentoRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
    <wsdl:part name="novoDocumento"
      type="intf:ArrayOf_xsd_string" />
    <wsdl:part name="nomeNovoDocumento" type="xsd:string" />
    <wsdl:part name="idProjetoNovoDocumento" type="xsd:int" />
    <wsdl:part name="versaoNovoDocumento" type="xsd:int" />
    <wsdl:part name="dataCriacaoNovoDocumento"
      type="xsd:string" />
  </wsdl:message>
- <wsdl:message name="getUsuariosResponse">
    <wsdl:part name="getUsuariosReturn"
      type="intf:ArrayOfUsuarioMimix" />
  </wsdl:message>
- <wsdl:message name="getDocumentoRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
    <wsdl:part name="idDocumento" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="JuncaoAutomaticaDocumentosRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
    <wsdl:part name="idDocumento1" type="xsd:int" />
    <wsdl:part name="idDocumento2" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="getProjetosDoUsuarioDaSessaoResponse">
    <wsdl:part name="getProjetosDoUsuarioDaSessaoReturn"
      type="intf:ArrayOfProjetoMimix" />
  </wsdl:message>
- <wsdl:message name="ComparaDocumentosRequest">
```

```
    <wsdl:part name="idSessao" type="xsd:int" />
    <wsdl:part name="idDocumento1" type="xsd:int" />
    <wsdl:part name="idDocumento2" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="getUsuariosDoProjetoResponse">
    <wsdl:part name="getUsuariosDoProjetoReturn"
      type="intf:ArrayOfUsuarioMimix" />
  </wsdl:message>
- <wsdl:message name="ExcluiDocumentoResponse">
    <wsdl:part name="ExcluiDocumentoReturn" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="AssociaUsuarioProjetoResponse">
    <wsdl:part name="AssociaUsuarioProjetoReturn"
      type="xsd:boolean" />
  </wsdl:message>
- <wsdl:message name="getDocumentosRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="getUsuarioRequest">
    <wsdl:part name="idUsuario" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="getProjetosRequest" />
- <wsdl:message name="ValidaDocumentoResponse">
    <wsdl:part name="ValidaDocumentoReturn" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="DesassociaUsuarioProjetoResponse">
    <wsdl:part name="DesassociaUsuarioProjetoReturn"
      type="xsd:boolean" />
  </wsdl:message>
- <wsdl:message name="NovoDocumentoResponse">
    <wsdl:part name="NovoDocumentoReturn" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="getUsuariosOnLineResponse">
```

```
    <wsdl:part name="getUsuariosOnLineReturn"
      type="intf:ArrayOfUsuarioMimix" />
  </wsdl:message>
- <wsdl:message name="NovoUsuarioResponse">
    <wsdl:part name="NovoUsuarioReturn" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="LogoutRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="ExcluiDocumentoRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
    <wsdl:part name="idDocumento" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="getProjetosDoUsuarioDaSessaoRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="ComparaDocumentosResponse">
    <wsdl:part name="ComparaDocumentosReturn"
      type="intf:ArrayOfElementoComparacaoMimix" />
  </wsdl:message>
- <wsdl:message name="JuncaoAutomaticaDocumentosResponse">
    <wsdl:part name="JuncaoAutomaticaDocumentosReturn"
      type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="JuncaoAssistidaDocumentosRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
    <wsdl:part name="quadroString"
      type="intf:ArrayOf_xsd_string" />
  </wsdl:message>
- <wsdl:message name="getVersaoMimixResponse">
    <wsdl:part name="getVersaoMimixReturn" type="xsd:string" />
  </wsdl:message>
- <wsdl:message name="ValidaDocumentoRequest">
    <wsdl:part name="idSessao" type="xsd:int" />
```

```
    <wsdl:part name="idDocumento" type="xsd:int" />
  </wsdl:message>
- <wsdl:portType name="Mimix">
  - <wsdl:operation name="Login" parameterOrder="usuario senha">
      <wsdl:input message="intf:LoginRequest"
        name="LoginRequest" />
      <wsdl:output message="intf:LoginResponse"
        name="LoginResponse" />
    </wsdl:operation>
  - <wsdl:operation name="getDocumento"
      parameterOrder="idSessao idDocumento">
      <wsdl:input message="intf:getDocumentoRequest"
        name="getDocumentoRequest" />
      <wsdl:output message="intf:getDocumentoResponse"
        name="getDocumentoResponse" />
    </wsdl:operation>
  - <wsdl:operation name="getProjetosDoUsuario"
      parameterOrder="idUsuario">
      <wsdl:input message="intf:getProjetosDoUsuarioRequest"
        name="getProjetosDoUsuarioRequest" />
      <wsdl:output
        message="intf:getProjetosDoUsuarioResponse"
        name="getProjetosDoUsuarioResponse" />
    </wsdl:operation>
  - <wsdl:operation name="getDocumentosDoProjeto"
      parameterOrder="idSessao idProjeto">
      <wsdl:input
        message="intf:getDocumentosDoProjetoRequest"
        name="getDocumentosDoProjetoRequest" />
      <wsdl:output
        message="intf:getDocumentosDoProjetoResponse"
        name="getDocumentosDoProjetoResponse" />
    </wsdl:operation>
  - <wsdl:operation name="getVersaoMimix">
```

```
      <wsdl:input message="intf:getVersaoMimixRequest"
         name="getVersaoMimixRequest" />
      <wsdl:output message="intf:getVersaoMimixResponse"
         name="getVersaoMimixResponse" />
   </wsdl:operation>
-  <wsdl:operation name="NovoUsuario"
      parameterOrder="nomeNovoUsuario senhaNovoUsuario
      descricaoNovoUsuario">
      <wsdl:input message="intf:NovoUsuarioRequest"
         name="NovoUsuarioRequest" />
      <wsdl:output message="intf:NovoUsuarioResponse"
         name="NovoUsuarioResponse" />
   </wsdl:operation>
-  <wsdl:operation name="getUsuariosOnLine">
      <wsdl:input message="intf:getUsuariosOnLineRequest"
         name="getUsuariosOnLineRequest" />
      <wsdl:output message="intf:getUsuariosOnLineResponse"
         name="getUsuariosOnLineResponse" />
   </wsdl:operation>
-  <wsdl:operation name="getUsuarios">
      <wsdl:input message="intf:getUsuariosRequest"
         name="getUsuariosRequest" />
      <wsdl:output message="intf:getUsuariosResponse"
         name="getUsuariosResponse" />
   </wsdl:operation>
-  <wsdl:operation name="getUsuario"
      parameterOrder="idUsuario">
      <wsdl:input message="intf:getUsuarioRequest"
         name="getUsuarioRequest" />
      <wsdl:output message="intf:getUsuarioResponse"
         name="getUsuarioResponse" />
   </wsdl:operation>
-  <wsdl:operation name="Logout" parameterOrder="idSessao">
```

```
    <wsdl:input message="intf:LogoutRequest"
      name="LogoutRequest" />
    <wsdl:output message="intf:LogoutResponse"
      name="LogoutResponse" />
  </wsdl:operation>
- <wsdl:operation name="getProjetosDoUsuarioDaSessao"
    parameterOrder="idSessao">
    <wsdl:input
      message="intf:getProjetosDoUsuarioDaSessaoRequest"
      name="getProjetosDoUsuarioDaSessaoRequest" />
    <wsdl:output
      message="intf:getProjetosDoUsuarioDaSessaoResponse
      " name="getProjetosDoUsuarioDaSessaoResponse" />
  </wsdl:operation>
- <wsdl:operation name="NovoProjeto" parameterOrder="idSessao
    nomeNovoProjeto descricaoNovoProjeto
    dataInicioNovoProjeto dataTerminoNovoProjeto">
    <wsdl:input message="intf:NovoProjetoRequest"
      name="NovoProjetoRequest" />
    <wsdl:output message="intf:NovoProjetoResponse"
      name="NovoProjetoResponse" />
  </wsdl:operation>
- <wsdl:operation name="getProjetos">
    <wsdl:input message="intf:getProjetosRequest"
      name="getProjetosRequest" />
    <wsdl:output message="intf:getProjetosResponse"
      name="getProjetosResponse" />
  </wsdl:operation>
- <wsdl:operation name="AssociaUsuarioProjeto"
    parameterOrder="idSessao idUsuario idProjeto">
    <wsdl:input message="intf:AssociaUsuarioProjetoRequest"
      name="AssociaUsuarioProjetoRequest" />
    <wsdl:output
      message="intf:AssociaUsuarioProjetoResponse"
      name="AssociaUsuarioProjetoResponse" />
```

```
  </wsdl:operation>
- <wsdl:operation name="DesassociaUsuarioProjeto"
      parameterOrder="idSessao idUsuario idProjeto">
      <wsdl:input
         message="intf:DesassociaUsuarioProjetoRequest"
         name="DesassociaUsuarioProjetoRequest" />
      <wsdl:output
         message="intf:DesassociaUsuarioProjetoResponse"
         name="DesassociaUsuarioProjetoResponse" />
  </wsdl:operation>
- <wsdl:operation name="getUsuariosDoProjeto"
      parameterOrder="idProjeto">
      <wsdl:input message="intf:getUsuariosDoProjetoRequest"
         name="getUsuariosDoProjetoRequest" />
      <wsdl:output
         message="intf:getUsuariosDoProjetoResponse"
         name="getUsuariosDoProjetoResponse" />
  </wsdl:operation>
- <wsdl:operation name="NovoDocumento"
      parameterOrder="idSessao novoDocumento
      nomeNovoDocumento idProjetoNovoDocumento
      versaoNovoDocumento dataCriacaoNovoDocumento">
      <wsdl:input message="intf:NovoDocumentoRequest"
         name="NovoDocumentoRequest" />
      <wsdl:output message="intf:NovoDocumentoResponse"
         name="NovoDocumentoResponse" />
  </wsdl:operation>
- <wsdl:operation name="ValidaDocumento"
      parameterOrder="idSessao idDocumento">
      <wsdl:input message="intf:ValidaDocumentoRequest"
         name="ValidaDocumentoRequest" />
      <wsdl:output message="intf:ValidaDocumentoResponse"
         name="ValidaDocumentoResponse" />
  </wsdl:operation>
```

```
- <wsdl:operation name="JuncaoAutomaticaDocumentos"
      parameterOrder="idSessao idDocumento1 idDocumento2">
      <wsdl:input
         message="intf:JuncaoAutomaticaDocumentosRequest"
         name="JuncaoAutomaticaDocumentosRequest" />
      <wsdl:output
         message="intf:JuncaoAutomaticaDocumentosResponse"
         name="JuncaoAutomaticaDocumentosResponse" />
   </wsdl:operation>
- <wsdl:operation name="JuncaoAssistidaDocumentos"
      parameterOrder="idSessao quadroString">
      <wsdl:input
         message="intf:JuncaoAssistidaDocumentosRequest"
         name="JuncaoAssistidaDocumentosRequest" />
      <wsdl:output
         message="intf:JuncaoAssistidaDocumentosResponse"
         name="JuncaoAssistidaDocumentosResponse" />
   </wsdl:operation>
- <wsdl:operation name="ComparaDocumentos"
      parameterOrder="idSessao idDocumento1 idDocumento2">
      <wsdl:input message="intf:ComparaDocumentosRequest"
         name="ComparaDocumentosRequest" />
      <wsdl:output message="intf:ComparaDocumentosResponse"
         name="ComparaDocumentosResponse" />
   </wsdl:operation>
- <wsdl:operation name="ExcluiDocumento"
      parameterOrder="idSessao idDocumento">
      <wsdl:input message="intf:ExcluiDocumentoRequest"
         name="ExcluiDocumentoRequest" />
      <wsdl:output message="intf:ExcluiDocumentoResponse"
         name="ExcluiDocumentoResponse" />
   </wsdl:operation>
- <wsdl:operation name="getDocumentos"
      parameterOrder="idSessao">
```

```
        <wsdl:input message="intf:getDocumentosRequest"
          name="getDocumentosRequest" />
        <wsdl:output message="intf:getDocumentosResponse"
          name="getDocumentosResponse" />
      </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="MimixSoapBinding" type="intf:Mimix">
    <wsdlsoap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http" />
  - <wsdl:operation name="Login">
      <wsdlsoap:operation soapAction="" />
    - <wsdl:input name="LoginRequest">
        <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/
          encoding/"
          namespace="http://www.interbrainwebservices.co
          m.br/teste/Mimix" use="encoded" />
      </wsdl:input>
    - <wsdl:output name="LoginResponse">
        <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/
          encoding/"
          namespace="http://www.interbrainwebservices.co
          m.br/teste/Mimix" use="encoded" />
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="getDocumento">
      <wsdlsoap:operation soapAction="" />
    - <wsdl:input name="getDocumentoRequest">
        <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/
          encoding/"
          namespace="http://www.interbrainwebservices.co
          m.br/teste/Mimix" use="encoded" />
      </wsdl:input>
```

<wsdl:output name="**getDocumentoResponse**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="**getProjetosDoUsuario**">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="**getProjetosDoUsuarioRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
<wsdl:output name="**getProjetosDoUsuarioResponse**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="**getDocumentosDoProjeto**">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="**getDocumentosDoProjetoRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
<wsdl:output name="**getDocumentosDoProjetoResponse**">

```
        <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/"
            namespace="http://www.interbrainwebservices.co
            m.br/teste/Mimix" use="encoded" />
        </wsdl:output>
    </wsdl:operation>
- <wsdl:operation name="getVersaoMimix">
        <wsdlsoap:operation soapAction="" />
    - <wsdl:input name="getVersaoMimixRequest">
        <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/"
            namespace="http://www.interbrainwebservices.co
            m.br/teste/Mimix" use="encoded" />
        </wsdl:input>
    - <wsdl:output name="getVersaoMimixResponse">
        <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/"
            namespace="http://www.interbrainwebservices.co
            m.br/teste/Mimix" use="encoded" />
        </wsdl:output>
    </wsdl:operation>
- <wsdl:operation name="NovoUsuario">
        <wsdlsoap:operation soapAction="" />
    - <wsdl:input name="NovoUsuarioRequest">
        <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/"
            namespace="http://www.interbrainwebservices.co
            m.br/teste/Mimix" use="encoded" />
        </wsdl:input>
    - <wsdl:output name="NovoUsuarioResponse">
```

```
        <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/"
            namespace="http://www.interbrainwebservices.co
            m.br/teste/Mimix" use="encoded" />
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="getUsuariosOnLine">
      <wsdlsoap:operation soapAction="" />
    - <wsdl:input name="getUsuariosOnLineRequest">
        <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/"
            namespace="http://www.interbrainwebservices.co
            m.br/teste/Mimix" use="encoded" />
      </wsdl:input>
    - <wsdl:output name="getUsuariosOnLineResponse">
        <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/"
            namespace="http://www.interbrainwebservices.co
            m.br/teste/Mimix" use="encoded" />
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="getUsuarios">
      <wsdlsoap:operation soapAction="" />
    - <wsdl:input name="getUsuariosRequest">
        <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/"
            namespace="http://www.interbrainwebservices.co
            m.br/teste/Mimix" use="encoded" />
      </wsdl:input>
    - <wsdl:output name="getUsuariosResponse">
```

<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
- <wsdl:operation name="**getUsuario**">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="**getUsuarioRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
- <wsdl:output name="**getUsuarioResponse**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
- <wsdl:operation name="**Logout**">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="**LogoutRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
- <wsdl:output name="**LogoutResponse**">

<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
- <wsdl:operation name="**getProjetosDoUsuarioDaSessao**">
<wsdlsoap:operation soapAction="" />
- <wsdl:input
name="**getProjetosDoUsuarioDaSessaoRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
- <wsdl:output
name="**getProjetosDoUsuarioDaSessaoResponse**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
- <wsdl:operation name="**NovoProjeto**">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="**NovoProjetoRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>

<wsdl:output name="**NovoProjetoResponse**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="**getProjetos**">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="**getProjetosRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
<wsdl:output name="**getProjetosResponse**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="**AssociaUsuarioProjeto**">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="**AssociaUsuarioProjetoRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
<wsdl:output name="**AssociaUsuarioProjetoResponse**">

<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
- <wsdl:operation name="**DesassociaUsuarioProjeto**">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="**DesassociaUsuarioProjetoRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
- <wsdl:output name="**DesassociaUsuarioProjetoResponse**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
- <wsdl:operation name="**getUsuariosDoProjeto**">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="**getUsuariosDoProjetoRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
- <wsdl:output name="**getUsuariosDoProjetoResponse**">

<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
- <wsdl:operation name="**NovoDocumento**">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="**NovoDocumentoRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
- <wsdl:output name="**NovoDocumentoResponse**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:output>
</wsdl:operation>
- <wsdl:operation name="**ValidaDocumento**">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="**ValidaDocumentoRequest**">
<wsdlsoap:body
encodingStyle="**http://schemas.xmlsoap.org/soap/encoding/**"
namespace="**http://www.interbrainwebservices.com.br/teste/Mimix**" use="**encoded**" />
</wsdl:input>
- <wsdl:output name="**ValidaDocumentoResponse**">

```
        <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/
          encoding/"
          namespace="http://www.interbrainwebservices.co
          m.br/teste/Mimix" use="encoded" />
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="JuncaoAutomaticaDocumentos">
      <wsdlsoap:operation soapAction="" />
    - <wsdl:input
        name="JuncaoAutomaticaDocumentosRequest">
        <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/
          encoding/"
          namespace="http://www.interbrainwebservices.co
          m.br/teste/Mimix" use="encoded" />
      </wsdl:input>
    - <wsdl:output
        name="JuncaoAutomaticaDocumentosResponse">
        <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/
          encoding/"
          namespace="http://www.interbrainwebservices.co
          m.br/teste/Mimix" use="encoded" />
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="JuncaoAssistidaDocumentos">
      <wsdlsoap:operation soapAction="" />
    - <wsdl:input name="JuncaoAssistidaDocumentosRequest">
        <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/
          encoding/"
          namespace="http://www.interbrainwebservices.co
          m.br/teste/Mimix" use="encoded" />
      </wsdl:input>
```

```
- <wsdl:output
      name="JuncaoAssistidaDocumentosResponse">
      <wsdlsoap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/
        encoding/"
        namespace="http://www.interbrainwebservices.co
        m.br/teste/Mimix" use="encoded" />
    </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="ComparaDocumentos">
    <wsdlsoap:operation soapAction="" />
  - <wsdl:input name="ComparaDocumentosRequest">
      <wsdlsoap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/
        encoding/"
        namespace="http://www.interbrainwebservices.co
        m.br/teste/Mimix" use="encoded" />
    </wsdl:input>
  - <wsdl:output name="ComparaDocumentosResponse">
      <wsdlsoap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/
        encoding/"
        namespace="http://www.interbrainwebservices.co
        m.br/teste/Mimix" use="encoded" />
    </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="ExcluiDocumento">
    <wsdlsoap:operation soapAction="" />
  - <wsdl:input name="ExcluiDocumentoRequest">
      <wsdlsoap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/
        encoding/"
        namespace="http://www.interbrainwebservices.co
        m.br/teste/Mimix" use="encoded" />
    </wsdl:input>
```

```
    - <wsdl:output name="ExcluiDocumentoResponse">
        <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/
          encoding/"
          namespace="http://www.interbrainwebservices.co
          m.br/teste/Mimix" use="encoded" />
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="getDocumentos">
      <wsdlsoap:operation soapAction="" />
    - <wsdl:input name="getDocumentosRequest">
        <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/
          encoding/"
          namespace="http://www.interbrainwebservices.co
          m.br/teste/Mimix" use="encoded" />
      </wsdl:input>
    - <wsdl:output name="getDocumentosResponse">
        <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/
          encoding/"
          namespace="http://www.interbrainwebservices.co
          m.br/teste/Mimix" use="encoded" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="MimixService">
  - <wsdl:port binding="intf:MimixSoapBinding" name="Mimix">
      <wsdlsoap:address
        location="http://www.interbrainwebservices.com.br/te
        ste/servlet/AxisServlet/Mimix" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

ANEXO 4: Telas do Módulo Cliente Referentes ao Exemplo de Utilização do MIMIX.

figura 47. Usuário 1 se conecta no MIMIX.

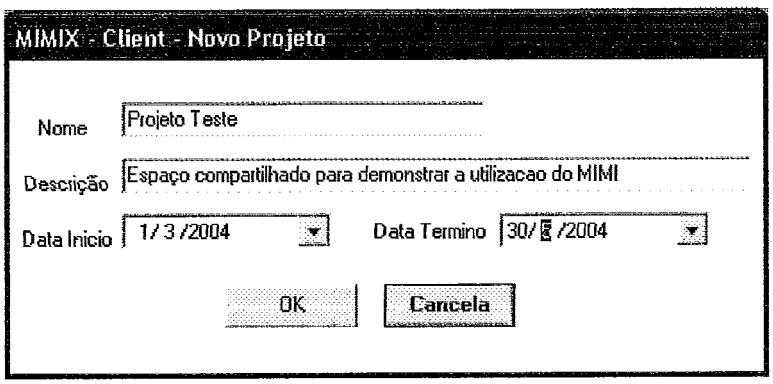


figura 48. O Usuário 1 cria o espaço de trabalho compartilhado, chamado Projeto Teste.

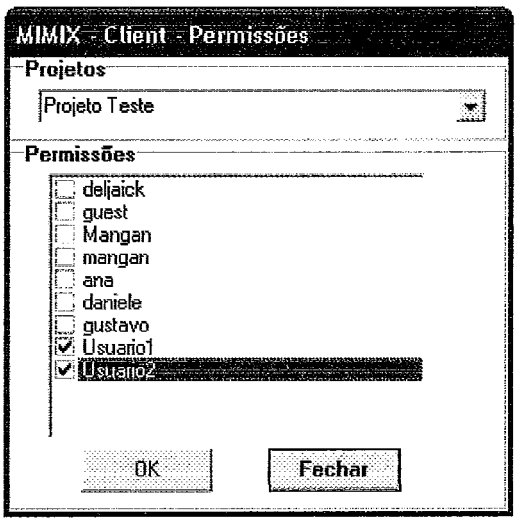


figura 49. O **Usuário 1** fornece permissão de acesso ao Projeto Teste para o **Usuário 2**.

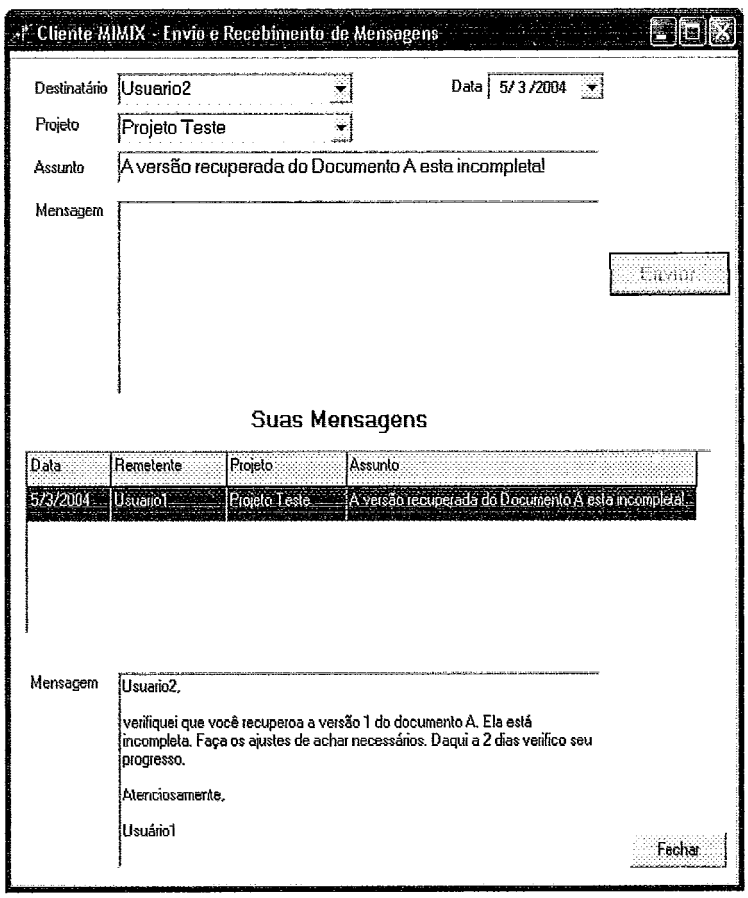


figura 50. O **Usuário 2** solicita as mensagens direcionadas a ele.

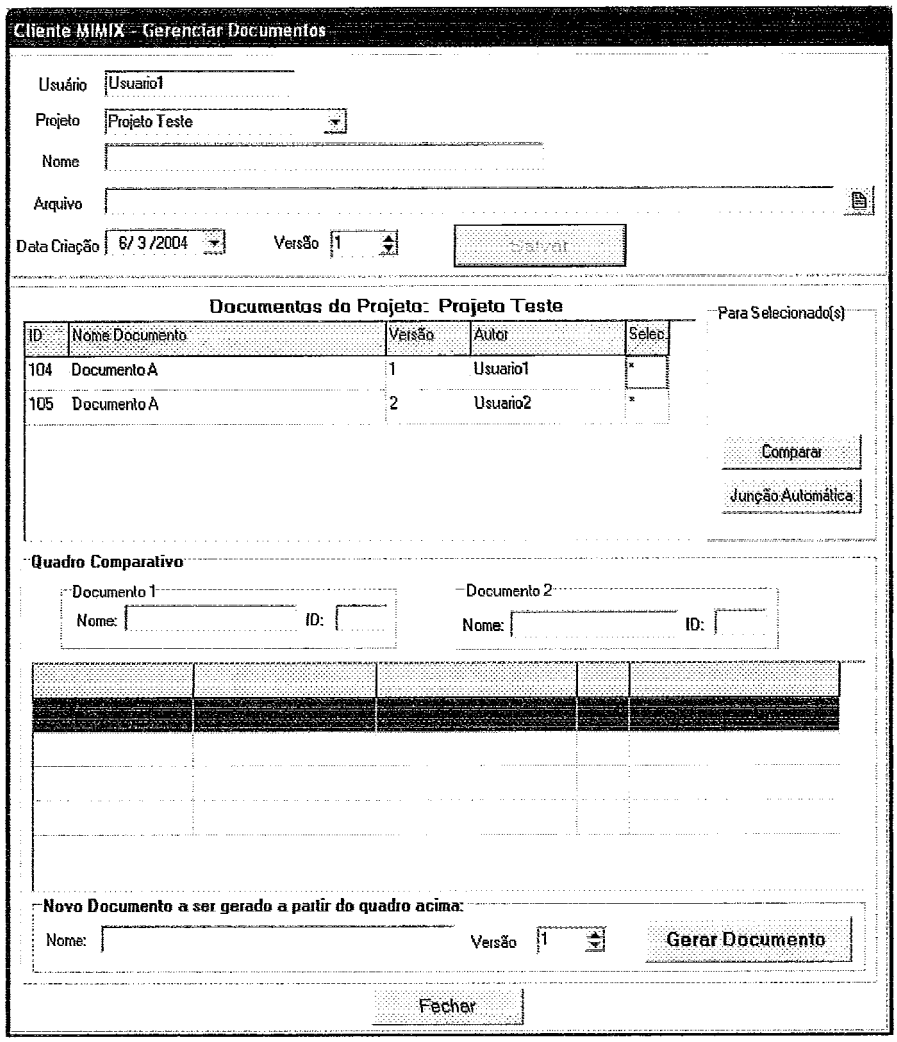


figura 51. O Usuário 1 solicita as versões armazenadas do documento A.