UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Componentes de Percepção para o Ambiente PROSOFT Cooperativo

por

ISABEL DILLMANN NUNES

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Daltro José Nunes
Orientador

Porto Alegre, setembro de 2001.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Nunes, Isabel Dillmann

Componentes de Percepção para o Ambiente PROSOFT Cooperativo / por Isabel Dillmann Nunes. – Porto Alegre: PPGC da UFRGS, 2001.
136 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, RS-BR, 2001. Orientador: Nunes, Daltro José.

1. Trabalho Cooperativo Suportado por Computador. 2. Percepção. 3. Interface Homem-Computador. I. Nunes, Daltro José. II. Título

Dedico este trabalho a minha família:
Meus pais, irmã, cunhado e tios

AGRADECIMENTOS

À Deus, por sua constante presença e força.

Aos meus pais, Décio e Regina, por todo apoio recebido durante esse período em que me encontrei distante e principalmente nos momentos em que o desânimo e cansaço tentaram me achar. Ainda, pelo o que há de vir...

À minha irmã e cunhado, pelos exemplos de dedicação e persistência. E principalmente, à minha irmã, por ter tantas vezes se tornado minha segunda mãe.

Ao meu avô Albino Dillmann, in memoriam, que estará sempre perto de mim...

Ao meu orientador, Daltro José Nunes, pela confiança e dedicação em mim depositada durante toda a jornada e ainda por me permitir roubar um pouco do seu tempo durante o período de férias.

Aos amigos, Rodrigo e Carla, por terem me ajudado durante o período deste trabalho sendo fontes de informações e conhecimento e acima de tudo companheiros.

Aos professores, funcionários e demais integrantes do Programa de Pós-Graduação em Computação da UFRGS que sempre estiverem presentes na minha formação.

Aos meus colegas, Abraham, Eliane, Giovani, Luciana, Márcia, Marco, Ronnie, Silvana, Simone, por estarem juntos nesta jornada e por todo o apoio e amizade recebidos.

Muito Obrigada!

SUMÁRIO

LISTA DE ABREVIATURAS

ADCS	Ambiente de Desenvolvimento Cooperativo de Software
ADS	Ambiente de Desenvolvimento de Software
ATO	Ambiente de Tratamento de Objeto
CSCW	*Computer-Supported Cooperative Work*
GUI	*Graphical User Interface*
HTML	*Hyper Text Markup Language*
IBIS	*Issues-Based Information System*
ICS	**Interface de Comunicação do Sistema**
NSD	*Nassi-Schneidermann Diagram*
PSEE	*Process-Centered Engineering Environments*
QOC	*Question, Option, Criteria*
SADT	*Structured Analysis and Design Technique*
UFRGS	**Universidade Federal do Rio Grande do Sul**
VDM	*Vienna Development Method*
WYSIWID	*What You See Is What I Do*
WYSIWIS	*What You See Is What I See*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A Engenharia de Software tornou-se uma das principais áreas de concentração do mundo da Ciência da Computação, pois ela abrange métodos para construir sistemas, ferramentas que fornecem apoio a construção e os procedimentos a seguir para que os métodos estejam de acordo com as ferramentas usadas [PRE 95].

No desenvolvimento de sistemas cada vez mais complexos vê-se a necessidade da utilização de equipes de pessoas no projeto e implementação. Essas pessoas podem estar geograficamente dispersas e portanto possuem a necessidade de troca de informações para que os sistemas desenvolvidos sejam adequados com o objetivo inicial e de qualidade. Assim, os sistemas cooperativos, chamados de *groupware*, tornaram-se uma importante ferramenta utilizada por esse grupo de profissionais para que as tarefas desenvolvidas em grupo se tornem interativas e eficientes.

A interação entre as pessoas que estão trabalhando cooperativamente deve ser a mais produtiva possível, sendo semelhante ao trabalho em grupo em um único local. Assim, a percepção das atividades que estão sendo realizadas devem estar disponíveis para cada profissional, através da Interface Homem-Computador do sistema *groupware* que estão utilizando.

Este trabalho apresenta uma "biblioteca" de Componentes de Percepção que fornecem as informações necessárias para que as pessoas que estão participando da tarefa cooperativa tenham a percepção das atividades que estão sendo realizadas, como também quem e como as estão fazendo. Esses componentes são uma extensão do Ambiente PROSOFT Cooperativo, fornecendo assim uma especificação formal de forma a garantir completeza, corretude e ausência de ambigüidades que são muito difíceis de se conseguir com uma descrição informal.

Palavras-chave: Engenharia de Software, Trabalho Cooperativo Suportado por Computador, Percepção do Espaço de Trabalho, Interface Homem-Computador, Especificação Formal

TITLE: "AWARENESS'S COMPONENTS AND THE COOPERATIVE PROSOFT ENVIRONMENT"

ABSTRACT

The Software Engineering became one of the main concentration areas in the Computer Science world, since it embraces methods to build systems, tools that provide support to the construction and the procedures to follow so that the methods are consistent with the used tools [PRE 95].

In the development of more and more complex systems one can sees the neecessity of project and implementation teams. People from those teams can be geographically dispersed and therefore there is the necessity of information exchange so that the developed systems can be consistent with the initial goal and quality. In this way, the cooperative systems, called *groupware*, became important tools used by those teams of professionals so that the tasks developed in group become interactive and efficient.

The interaction among people that are cooperatively working must be the most possible productive, being similar to the work developed in group in only one location. Thus, the activities awareness that are being performed must be available for each professional, through the *groupware* Computer Human Interface system that such professionals are using.

This work presents a Awareness Components library that provides the necessary informations to people that are participating in cooperative task in a way they have the activities awareness that are being performed, as well as who and how they are being done. This components are an extension of Cooperative PROSOFT Environment, providing thus a formal specification to guarantee completeness, correctness and absence of ambiguities that are very difficult to get with an informal description.

Keywords: Software Engineering, Computer-Supported Cooperative Work, Worspace Awareness, Computer Human Interface, Formal Specification

1 Introdução

A Engenharia de Software foi primeiramente definida por [NAU 69] como sendo:

> "O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais."

Com o apoio desta definição pode-se dizer que a Engenharia de Software tornou-se uma das principais áreas de concentração do mundo da Ciência da Computação, pois ela abrange métodos para construir sistemas, ferramentas que fornecem apoio a construção e os procedimentos a seguir para que os métodos estejam de acordo com as ferramentas usadas [PRE 95].

O uso de técnicas de especificação formal na Engenharia de Software possibilitou ao desenvolvedor especificar, desenvolver e verificar o sistema em desenvolvimento, proporcionando um meio para que os aspectos de consistência, completude, exatidão e ausência de ambigüidade possam ser avaliados [HAL 90] [PRE 95] [JON 96]. Sendo assim, os métodos formais possibilitam o desenvolvimento de complexos sistemas que atualmente aparecem com maior freqüência.

O grande número de sistemas complexos atualmente causou a necessidade do surgimento de equipes de desenvolvimento. Essas equipes são formadas por pessoas que possuem capacidades diferentes, mas que precisam cooperar para atingir os resultados esperados [BOR 95]. Assim, o desenvolvimento cooperativo de software apoiado por computador – CSCW (*Computer Supported Cooperative Work*) depende da vontade de trabalho dos participantes do grupo e dos recursos fornecidos a eles [BOR 99].

Conforme [BOR 93]:

> "O trabalho da equipe responsável pelo desenvolvimento de software é essencialmente cooperativo, mesmo considerando que os membros do grupo mudem durante as diversas fases do processo."

O trabalho em grupo suportado por computador se torna mais produtivo se os participantes possuem uma visão da tarefa parecida com a que eles possuem se estivessem todos em um mesmo local. Para que isso se torne possível é necessário preocupar-se com a percepção das atividades que estão ocorrendo e quem são as pessoas que estão realizando cada tarefa.

A percepção do grupo em um sistema cooperativo é dada pelos componentes que compõem a interface homem-computador, pois é através dela que as pessoas irão interagir com o sistema e com o restante do grupo [DEW 91]. Essa interação permitirá manter as pessoas informadas e com maior encorajamento de comunicação entre elas, fornecendo

assim uma contextualização das atividades individuais através da compreensão das atividades realizadas pelo grupo [BOR 96].

1.1 Motivação e Objetivos

A maioria dos Ambientes de Desenvolvimento Cooperativo de Software (ADCS) não são tratados formalmente, apresentando apenas uma definição informal dos conceitos usados em relação a comunicação, controle e interação entre os participantes do grupo.

O ADCS PROSOFT Cooperativo [REI 98], é um ambiente especificado formalmente que proporciona a cooperação entre os membros da equipe. Esta especificação proporciona o controle dos usuários, das ferramentas, das estações de trabalho e a integração dos objetos gerados. Porém esse ambiente não se preocupa com a percepção das atividades do grupo, fornecendo uma base para o trabalho cooperativo mas não para a percepção das atividades entre os usuários.

O objetivo deste trabalho é integrar ao PROSOFT Cooperativo conceitos de *Workspace Awareness* – Percepção do Espaço de Trabalho [GUT 96] [GUT 97] para tarefas síncronas, para que os desenvolvedores possam utilizar nas ferramentas que estão sendo desenvolvidas componentes que proporcionam a interação dos futuros usuários com a máquina e entre eles.

Portanto, os componentes especificados neste texto terão como principal função auxiliar ao desenvolvedor na construção de suas ferramentas voltadas para o trabalho em grupo, proporcionando maior rapidez e agilidade. O método utilizado para especificá-los é o PROSOFT-Algébrico [NUN 94], para que possam ser utilizados sem restrições pelos desenvolvedores que utilizam o PROSOFT Cooperativo.

Surge então, uma nova extensão dos sistemas *groupware* desenvolvidos no PROSOFT Cooperativo, sistemas voltados ao trabalho em grupo com componentes de interface homem-computador que fornecem todo o apoio para a percepção das atividades síncronas que estão ocorrendo e para as pessoas que estão realizando essas atividades.

1.2 Apresentação do Texto

O texto será organizado em 7 capítulos e dois anexos. No capítulo 2 será descrito os conceitos e características do Trabalho em Grupo Suportado por Computador, como também as formas de Percepção e os componentes que fornecem essa percepção.

No capítulo 3 apresenta o Ambiente de Desenvolvimento de Software PROSOFT, sua nomenclatura, conceitos e uma breve descrição do PROSOFT Cooperativo (extensão do PROSOFT muito importante para este trabalho).

No capítulo 4 contém a descrição dos componentes desenvolvidos para fornecer a percepção do espaço de trabalho no Ambiente PROSOFT Cooperativo e uma justificativa da forma de criação desses componentes.

O capítulo 5 há uma descrição de como construir os ATOs cooperativos no PROSOFT Cooperativo utilizando os componentes de percepção, a fim de facilitar e melhorar o desempenho dos desenvolvedores.

No capítulo 6 há a especificação formal dos componentes de percepção para interfaces cooperativas para o Ambiente PROSOFT Cooperativo utilizando o paradigma PROSOFT-Algébrico;

No capítulo 7 são apresentadas as conclusões e trabalhos futuros que podem surgir a partir desta dissertação

Finalmente, serão descritos algumas operações internas dos ATOs que compõem os componentes de percepção deste trabalho e as operações que os desenvolvedores podem especificar no momento da construção de suas ferramentas.

2 CSCW e as Formas de Percepção na Interação Humana

O crescente número de sistemas complexos que surgem exige a necessidade de interação entre as pessoas que neles trabalham, para que possam tomar decisões, trocar idéias ou até mesmo jogar [NUS 2000].

CSCW – *Computer Supported Cooperative Work* é a área da Computação que se preocupa com a interação entre um grupo de pessoas que possuem um objetivo em comum. Esta sigla pode ser traduzida como "Trabalho Cooperativo Suportado por Computador" ou ainda "Suporte por Computador ao Trabalho Cooperativo" [BOR 95] [PAL 94].

Para que essa cooperação tenha um entendimento melhor minimizando assim os problemas gerados por ela mesma (por exemplo, forma de comunicação desordenada, como em qualquer trabalho em grupo), há aspectos que devem ser levados em consideração, segundo [ARA 97] [BOR 99], mostrados na Fig. 2.1.

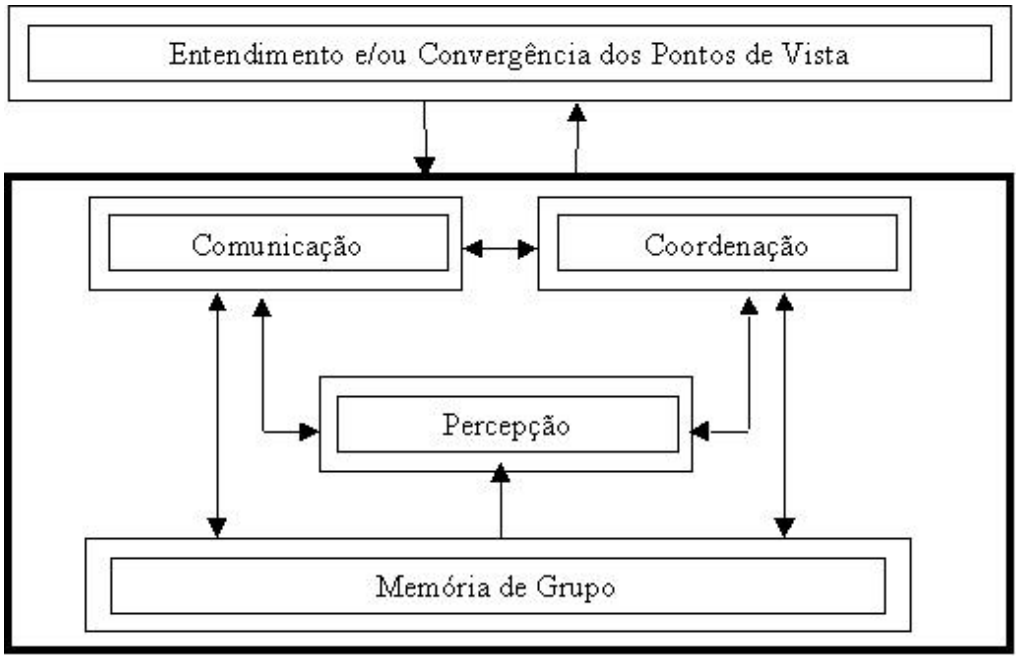


FIGURA 2.1 - Aspectos da Cooperação

O processo de cooperação, ou seja, o entendimento ou convergências de idéias entre as pessoas na realização de uma atividade de trabalho cooperativo, compreende:

- A Comunicação – conhecimento dos canais disponíveis para troca de informações (correio eletrônico, por exemplo), para que as pessoas possam interagir mesmo estando em locais diferentes;

- A Coordenação – controle das atividades do grupo para que a cooperação ocorra de forma ordenada;
- A Memória de Grupo – informações armazenadas que podem ser úteis para futuras tomadas de decisões ou simplesmente para atualização de pessoas que ficaram ausentes do grupo por um determinado tempo;
- A Percepção – conhecimento de cada participante do grupo sobre as atividades que estão sendo realizadas, quem está trabalhando em cada atividade e quais são as futuras tarefas.

Os sistemas que suportam esse conceito são chamados de *Groupware*, os quais atendem as necessidades do trabalho em grupo. Esses sistemas serão melhor detalhados na próxima sessão.

2.1 *Groupware*

Para que o CSCW seja suportado em termos de tecnologia foram criados os *Groupware* [DIX 98], e segundo [ELL 91], *Groupware* pode ser definido como:

> "...sistema baseado em computador que suporta grupos de pessoas engajadas em uma tarefa comum (ou ideal) e que fornece uma interface para um ambiente compartilhado."

Sistemas *groupware* enfrentam grandes problemas no envolvimento cooperativo dos usuários no momento de desenvolverem uma tarefa em comum. Para que esse problema seja amenizado, há requisitos que podem ser seguidos no momento da sua construção [BOR 95] [RED 94]:

- O grau de interação deve ser o maior possível para que o grupo possa atingir o objetivo escolhido, aumentado a produtividade e não causando concorrência entre os membros do grupo;
- Como os sistemas *groupware* podem ser usados por pessoas distribuídas geograficamente, eles devem ser capazes de atender a diferentes culturas, línguas e comportamentos;
- Os sistemas devem disponibilizar todas as informações para a realização das tarefas, como também a possibilidade de mudanças durante o processo de trabalho.

Esses requisitos também são importantes no momento da escolha do *groupware* mais apropriado para cada tarefa e para cada grupo de usuários, auxiliando, desta forma, a distribuição das tarefas e melhorando o andamento do trabalho.

2.1.1 Classificação

As formas de classificar os sistemas de CSCW estão evoluindo, pois cada vez mais sistemas que eram usados somente para trabalho individual estão à disposição para o trabalho em grupo. As classificações mais usuais, de acordo [BOR 95], são:

- Espaço/Tempo – essa classificação leva em consideração as noções de espaço e tempo nas quais as interações são realizadas. Por exemplo, correio eletrônico possibilita uma troca de mensagens em lugares e instantes diferentes. Esta classificação é vista na Tab. 2.1.

TABELA 2.1 - Classificação de *groupware* Espaço/Tempo [BOR 95]

	Mesmo momento	**Momentos diferentes**
Mesmo lugar	Interação face a face	Interação assíncrona
Lugares diferentes	Interação síncrona e distribuída	Interação assíncrona e distríbuída

- Espaço/Tempo considerando a previsibilidade – esta classificação foi proposta por [GRU 94], levando em consideração a previsão de uma tarefa ser realizada em um determinado local ou instante, como mostrado na Tab. 2.2. Esta classificação se torna importante, por exemplo, quando a resposta de uma determinada mensagem enviada por correio eletrônico é esperada dentro de um período de tempo.

TABELA 2.2 - Classificação de *groupware* Espaço/Tempo considerando a previsibilidade [BOR 95]

		Tempo		
		Mesmo	**Diferentes mas previsíveis**	**Diferentes mas imprevisíveis**
Local	**Mesmo**	Auxílio a reuniões	Deslocamento de tarefas	Salas de grupo
	Diferentes mas previsíveis	Tele/Vídeo conferências	Correio Eletrônico	Edição Colaborativa
	Diferentes mas imprevisíveis	Seminários de Interação *multicast*	*Bulletin Boards* eletrônicos	Workflow

- Espaço/Tempo considerando o tamanho do grupo – esta classificação foi proposta por [NUR 91] e considera o tamanho do grupo que está desenvolvendo a atividade, como mostra a Fig. 2.2.

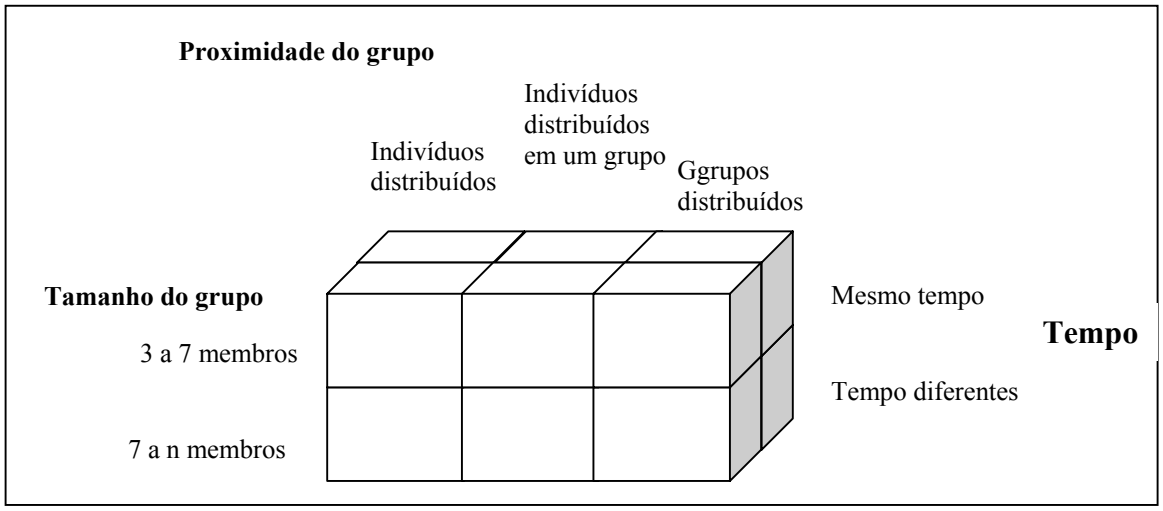


FIGURA 2.2 - Classificação de *groupware* Espaço/Tempo/Tamanho Grupo

- Segundo a funcionalidade das aplicações – esta classificação foi apresentada por [ELL 91], e está construída de acordo com a finalidade de cada sistema:
    - Sistemas de Mensagens – permitem a troca assíncrona de mensagens e arquivos [AND 2001];
    - Editores Multi-usuários – permitem editar documentos em grupo [AND 2001];
    - Suporte à Decisão – permitem que um grupo de pessoas discutem, votem e decidam sobre um determinado assunto [ALV 2000];
    - Salas de Reuniões Eletrônicas – permitem a reunião, local ou remota, como também visualizar e editar documentos simultaneamente [AND 2001];
    - Conferências por Computador – permitem a troca de idéias entre um grupo de pessoas. Podem ser:
        - Tempo Real
        - Teleconferência
        - Desktop
    - Agentes Inteligentes – permitem que não somente pessoas participem de uma atividade cooperativa. Assim, um agente pode ser responsável por um conjunto de tarefas que interagem com os usuários;
    - Sistemas de Coordenação – permitem o gerenciamento de tarefas complexas, juntamente com as informações utilizadas e geradas por ela [BOR 95].

2.1.2 Tipos de *Groupware*

Os vários tipos de *groupware* são usados para tarefas isoladas, por exemplo o correio eletrônico para simples comunicação entre as pessoas que estão cooperando, por isso a necessidade de entender a proposta que cada tipo de sistema cooperativo oferece.

Assim, a seguir, é comentado os *groupware* mas utilizados:

- Agendas de Grupo – são capazes de programar eventos resolvendo os conflitos existentes em relação a disponibilidade de cada membro do grupo. É importante que as agendas possuam calendário, lista telefônica ou *e-mail* das pessoas que participam do grupo e as tarefas a serem agendadas [BOR 99];
- Editores de Co-autoria, Multi-usuários ou Cooperativos – esses sistemas permitem que um grupo de pessoas edite um texto ou gráfico em uma área visualizadas por todos. O editor pode ser síncrono (quando o trabalho é realizado em tempo real possibilitando que visualizem as alterações feitas) ou assíncrono (sendo que um usuário trabalha de cada vez ou existe apenas um editor e o restante do grupo trabalha como revisores, após a edição) [BOR 95] [WOO 97] [ELL 91];
- *Workflow* – possibilitam a seqüência e distribuição de tarefas para os membros do grupo, sendo que quando uma tarefa está completa , automaticamente o *workflow* realiza o roteamento para a próxima tarefa e para a pessoa que irá executá-la [POL 96]. Esses sistemas devem coordenar a distribuição de tarefas, fornecer as informações necessárias para a realização de cada tarefa fazendo com que o usuário se torne o mais participativo possível e padronizar as tarefas para que o trabalho se torne coerente [BOR 99];
- PSEE – *Process-Centered Software Engineering Environments* são ambientes de desenvolvimento de software orientado ao processo que apoiam a definição rigorosa de processos de software com o objetivo de automatizar a gerência do desenvolvimento apoiando o envolvimento cooperativo do grupo [REC 98] [REI 2001].
- Sistemas de Suporte a Reunião – esses sistemas possibilitam a uma reunião não tomar rumos que iriam somente atrasar as tomadas de decisão tornando-as uma perda de tempo. Também se preocupam com o aspecto social, o que cada membro do grupo representa (sua função) ou modelando o comportamento (por exemplo, “o crítico”) [BOR 95]. Dentro dos sistemas de suporte a reuniões, podemos ter:
    - Sistemas de Suporte à Decisão em Grupo – auxiliam na tomada de decisões em reuniões, o que é realizado em três etapas: geração de idéias, organização e votação, evitando assim problemas hierárquicos, discriminatórios, com inibição, perda de informações relevantes ou ainda a presença constante de certos membros do grupo [DAÍ 96] [BOR 95] [WOO 97] [ELL 91]. Esses sistemas baseiam-se em modelos de argumentação, entre os existentes podemos citar o IBIS – *Issue-Based Information System* (para cada questão pode ser dada várias soluções e opiniões contra ou favor para as soluções conseguidas) [YAK 90] [REN 91] e o QOC – *Question, Option, Criteria* (para cada questão há

alternativas de solução, justificativas para elas e argumentos que conduzem a discussão sobre as soluções encontradas) [SHU 94];

- Salas Eletrônicas –apresentam a necessidade de *hardware* (por exemplo estações ligadas em rede) e *software* a fim de apoiar as reuniões. Entre suas vantagens está a geração de memória de grupo aumentando a qualidade de informação em um espaço de tempo menor [BOR 95] [NUR 91];

- Correio Eletrônico – é o *groupware* mais utilizado, pois permite a troca assíncrona de mensagens, trazendo várias funcionalidades, tais como troca de idéias, armazenamento de informações, apoio a decisões, e também acarreta em baixo custo, flexibilidade e portabilidade. Sendo assim, com tantas vantagens, ocorreu uma sobrecarga de troca de mensagens, fazendo com que os sistemas possuem um "filtro" que classifica as mensagens de acordo com seu conteúdo [BOR 99] [BOR 95] [ENS 90] [ELL 91] [DIX 98];
- Sistemas de Conferência – permitem aos participantes trocarem idéias estando dispersos geograficamente ou em um mesmo ambiente, fazendo também com que todos participem igualmente da reunião. Esses sistemas podem ser assíncronos (onde cada participante comparece no momento que lhe for mais oportuno permitindo-lhe um tempo para pensar) ou em Tempo Real [BOR 95] [ELL 91] [MAO 99] [BUX 92].

Alguns destes *groupware* podem ser utilizados durante o Ciclo de Vida de um *software*, fornecendo um auxilio a um grupo de pessoas no desenvolvimento de sistemas cooperativos ou não, como mostra a Tab. 2.3.

TABELA 2.3 - Sistemas *groupware* que apóiam o ciclo de vida de *software* [BOR 95]

Fases do Ciclo de Vida	Groupware
Análise	Sistemas de Apoio a Decisão Editores de Co-autoria Salas Eletrônicas
Projeto	Sistemas de Apoio a Decisão Editores de Co-autoria Salas Eletrônicas
Implementação	Editores de Co-autoria **Sistemas de Conferência**
Verificação	**Sistemas de Apoio a Decisão** **Editores de Co-autoria** **Sistemas de Conferência**
Manutenção	Sistemas de Apoio a Decisão **Sistemas de Conferência**

2.2 Percepção

A Percepção, palavra utilizada pela literatura especializada como tradução do termo *"Awareness"* para Português, visa fornecer informações sobre as atividades dos usuários. Assim, a percepção é o conhecimento dos usuários sobre as atividades que estão ocorrendo, quem está desenvolvendo cada atividade e em qual instante ocorre. Permite portanto, que os usuários tenham conhecimento sobre as ações, sendo estas realizadas conjuntamente com outros humanos ou com a própria máquina [DEW 91] [BOR 95] [ELL 91].

Segundo [DOU 92], *Awareness* é:

> *"an undertanding of the activities of others, wich provides a context for your own activity"*

A interface com o usuário de um sistema *groupware* fornece apenas uma porção do ambiente de trabalho, sendo necessário que ela possibilite que as pessoas conheçam as condições de realização das tarefas da mesma forma que se fosse em um ambiente *face-to-face* [GUT 96].

Para tanto, é necessário levar em consideração certas questões, de acordo com [GUT 97], são elas:

- Especificação do domínio - a informação fornecida sobre um determinado membro do grupo deve estar de acordo com o domínio da aplicação, sendo somente necessário as informações relevantes para a realização de uma determinada tarefa;
- Importância da informação - para certas atividades existem informações que são cruciais, porém há outras informações que são benéficas mas não decisivas. A decisão de quais são essas informações é muito difícil de determinar e os sistemas *groupware* que as disponibilizam são igualmente difíceis de avaliar;
- Condições de mudança - a condição de percepção varia de acordo com o foco estabelecido. Por exemplo, se o foco é uma tarefa individual a condição de percepção é bem menos específica do que em uma tarefa compartilhada;
- Efeitos de perícia - as pessoas se tornam mais familiares com a tarefa e com o grupo de colaboradores se eles são hábeis para inferir sobre as tarefas das outras pessoas, podendo assim antecipar as ações dos outros membros tendo por base as atividades que estão em desenvolvimento no mesmo instante.

2.2.1 Tipos de Percepção

A percepção é baseada no conhecimento que deve ser fornecido para os usuários na realização de suas tarefas, mantendo as pessoas informadas e encorajando a comunicação entre elas. Assim, a possibilidade de rejeição entre o grupo deve diminuir, pois as pessoas sabem quais informações estão sendo disponibilizadas e para quais usuários elas estão sendo direcionadas [SCL 97].

Existe uma divisão da percepção em tipos, de acordo com qual informação é disponibilizada. Segundo [GRE 96] e [GUT 96b], são elas:

- Percepção informal – sobre a localização das pessoas que estão realizando as tarefas (se estão no mesmo ambiente ou geograficamente distantes);
- Percepção da estrutura do grupo – é o conhecimento sobre as funções de cada participante, suas responsabilidades e suas opiniões;
- Percepção social – informações sobre o nível de interesse do grupo na atividade como também sobre o estado emocional de cada pessoa;
- Percepção sobre o *workspace* (espaço de trabalho) – são as informações das atividades que estão sendo realizadas e quem está trabalhando em qual delas. Pode-se dizer também que é o *update* (atualizações e contribuições) das realizações do outros membros do grupo. Isto pode auxiliar as pessoas a moverem-se entre as atividades compartilhadas e individuais, permitindo a antecipação das ações dos outros e reduzindo o esforço necessário para coordenar as tarefas e os recursos disponíveis.

A Tab. 2.4 mostra um conjunto de elementos e questões que as pessoas participantes de uma atividade cooperativa suportada por computador devem levar em consideração, fornecendo um suporte básico sobre os requerimentos de *workspace awareness* segundo [GUT 96] [GUT 96c].

TABELA 2.4 - Elementos e Questões para o *Workspace Awareness* [GUT 96c]

Elemento	Questões Relevantes
Identidade	Quem está participando da atividade?
Localização	Onde os participantes estão?
Nível da atividade	Os participantes estão ativos na área de trabalho? O quão rápido os participantes estão trabalhando?
Ações	O que os participantes estão fazendo? Quais são suas atividades e tarefas correntes?
Intenções	O que os participantes estão indo fazer? Para onde os participantes estão indo?
Mudanças	Quais mudanças os participantes estão fazendo? Onde as mudanças estão sendo feitas?
Objetos	Quais objetos os participantes estão usando?
Espaço	O que os participantes estão vendo?
Habilidades	O que os participantes podem fazer?
Esfera de influência	Onde os participantes podem atuar?
Expectativas	O que os participantes necessitam de mim para continuar suas tarefas?

2.2.2 Componentes de Interface para Percepção do *Workspace*

Existem componentes que são usados para fornecer a percepção para área de trabalho das atividades síncronas que estão sendo executadas. Pode-se citar alguns deles:

- Teleponteiros – cada pessoa do grupo pode possuir seu próprio ponteiro (teleponteiro) [MIT 96], sendo assim o restante do grupo consegue identificar quem está trabalhando e em qual atividade. Eles podem ser identificados por cores diferentes ou por *labels* anexos a eles. Um exemplo está no *GroupWeb* [GRE 97] [GRE 96]. Esta ferramenta é um *browser* que permite membros de um grupo visualizarem partes e navegar em páginas *web* em tempo real, como ilustra a Fig. 2.3;

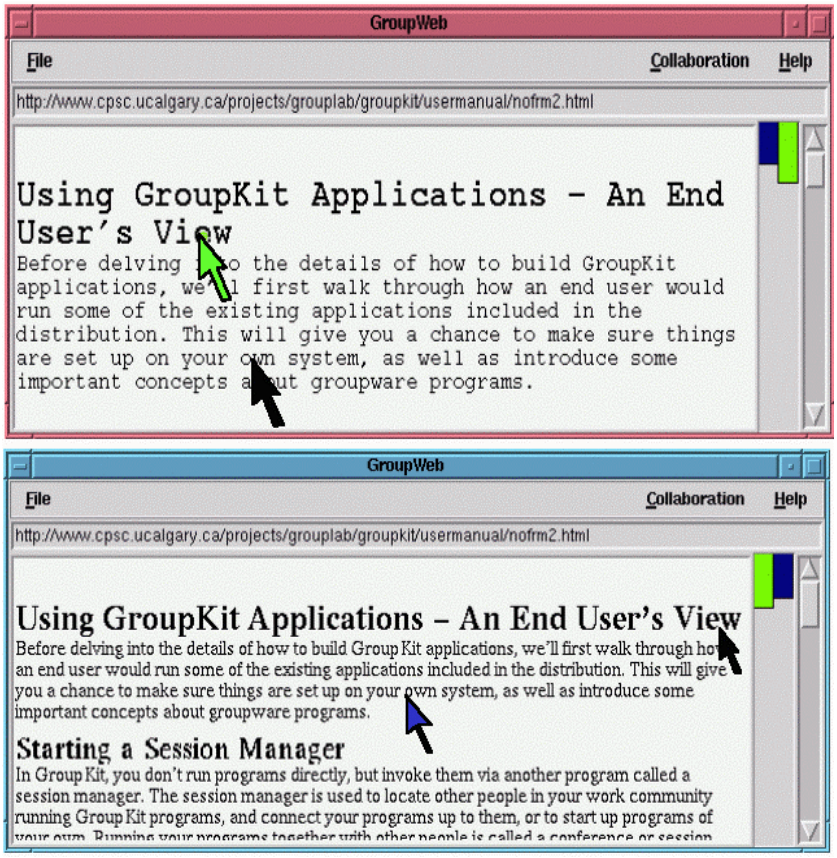


FIGURA 2.3 - Teleponteiros e *Scrollbar* sincronizado

- Múltiplos cursores – igualmente aos teleponteiros, cada indivíduo possuirá o seu próprio cursor. É geralmente utilizado em editores de texto, para localizar a pessoa e o restante do grupo no decorrer da edição [GUT 96c];
- *Radar View* – é uma janela auxiliar que possibilita aos participantes do grupo visualizar as atividades dos outros. Mostra o espaço de trabalho compartilhado em uma versão minimizada, identificando a localização de cada participante [GUT 96c];
- *Fisheye* – é uma forma de visualização onde cada participante pode ver as tarefas de todo o grupo. A área de trabalho do grupo é mostrada diferenciadamente, ou seja, para o

usuário local a sua área de trabalho aparece no tamanho desejado enquanto a área de trabalho do restante do grupo é minimizada e identificada de alguma forma (por exemplo, por cor) [GUT 96c], como ilustra a Fig. 2.4;

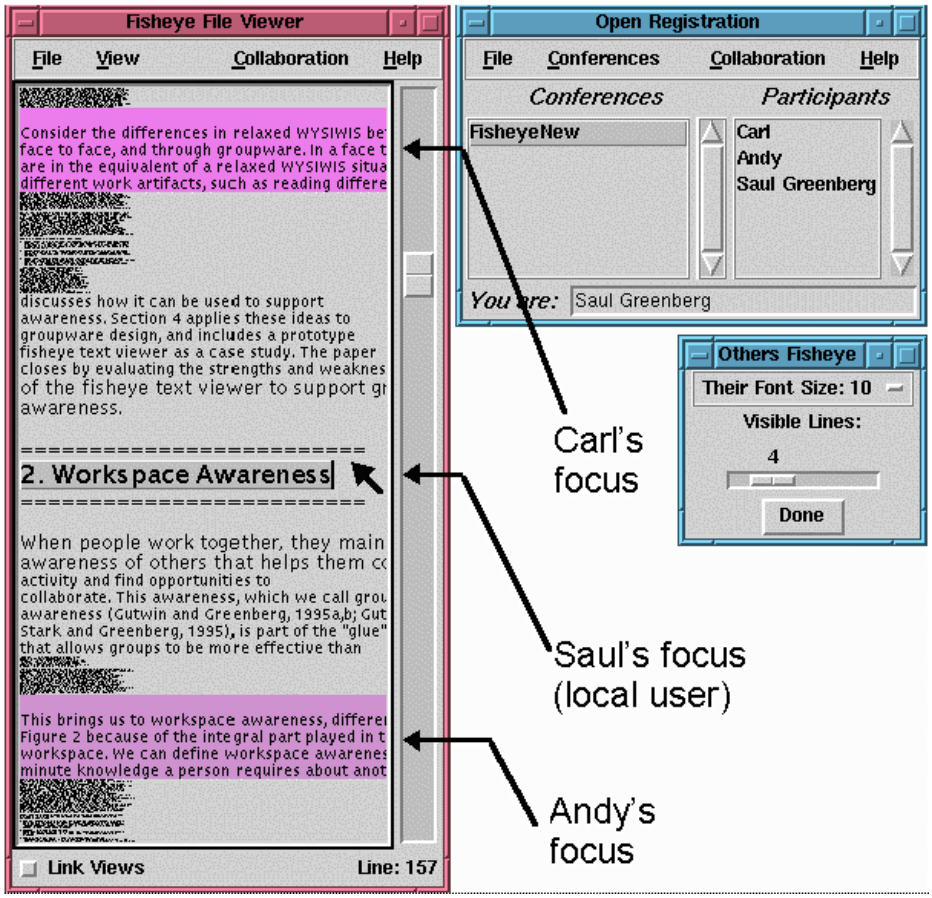


FIGURA 2.4 - Visualizador *Fisheye*

- *Scrollbar* de sincronização – é um *scrollbar* que possui a localização dos outros participantes. Assim, uma pessoa pode estar sincronizada no local onde os outros participantes estão trabalhando [GRE 96] [GRE 97] [MIT 96]. Pode-se ver um exemplo na Fig. 2.3.

2.3 Ferramentas de Desenvolvimento de *Groupware*

Nesta seção serão apresentados alguns dos *toolkits* para o desenvolvimento de *groupware* e será discutido quais componentes para a percepção de grupo são fornecidos por cada um deles.

A maioria dos *groupware*, o que também acontece com os sistemas para mono-usuários atualmente, possuem uma interface gráfica. Para suprir a facilidade de interação que uma interface gráfica proporciona, há necessidade das ferramentas de desenvolvimento de *software* fornecerem aos programadores recursos (componentes) que suportem as atividades encontradas no trabalho em grupo [GRE 99].

Há *toolkits* que não fornecem componentes para a percepção, como é o caso do Rendezvous [PAT 90] [MIL 92], onde seus componentes de interface são construídos a partir das primitivas gráficas.

Em outras ferramentas, como no caso da Visual Obliq [BHA 95], as aplicações *groupware* são implementadas a partir da construção da interface em um *GUI-builder* (ferramenta que constrói GUI - *Graphical UserInterface*, Interface com Usuário Gráfica), permitindo que o programador anexe à interface criada o código do sistema que está sendo desenvolvido. O Visual Obliq fornece *widgets* (componentes) e janelas *pop-up*. Os componentes fornecidos podem ser usados em interfaces cooperativas, tais como *scrollbar*, botões ou caixa de texto, porém não são específicos para a percepção no espaço de trabalho. Sendo assim, o Visual Obliq permite que sejam construídos novos componentes de acordo com a necessidade do desenvolvedor.

Já o *GroupKit* [ROS 92] [ROS 98] é uma *toolkit* para construção de *groupware* desenvolvido na *University of Calgary* que utiliza a linguagem para *scripts* [TCL 2000] [SIL 99] [CET 2000]. Esta ferramenta também é usada para ensinar os conceitos de CSCW. Em relação a percepção do espaço de trabalho o *GroupKit* fornece vários *widgets* (componentes) para a interface do trabalho em grupo. Entre eles podemos encontrar: os teleponteiros, scrollbar sincronizados e *radar view* [GUT 96d]. O *Radar View* é uma miniatura do *workspace* compartilhado. Nele é mostrado os pontos onde os participantes estão trabalhando (através de quadrados coloridos), teleponteiros e também os objetos que estão sendo manipulados, como ilustra a Fig. 2.5.

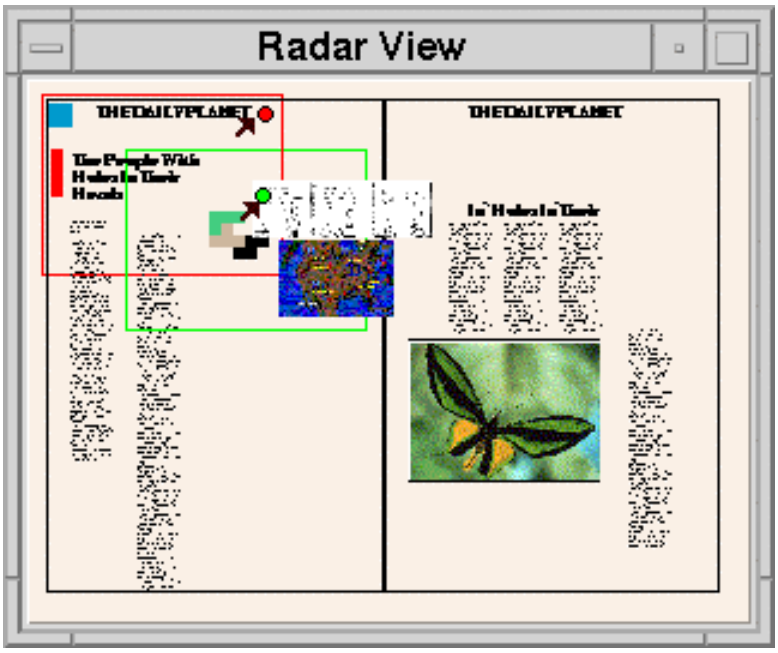


FIGURA 2.5 - *Radar View*

Estas ferramentas são ambientes de desenvolvimento de *groupware* e também possuem as características de cooperação entre os desenvolvedores. Porém, segundo [BIS 95], qualquer aplicação CSCW pode auxiliar no desenvolvimento cooperativo de *software*, embora a existência de uma ferramenta CSCW em um ambiente de desenvolvimento de *software* não o torne, obrigatoriamente, um Ambiente de Desenvolvimento Cooperativo de Software – ADCS.

No próximo capítulo será estudado o ADS PROSOFT, assim como sua extensão o PROSOFT Cooperativo, tornando-se um ADCS.

3 AMBIENTE PROSOFT

O PROSOFT é um ambiente de desenvolvimento de software (ADS), que possui o objetivo principal de auxiliar o desenvolvimento formal de programas. Ele possibilita a definição e desenvolvimento de um conjunto de ferramentas a partir da análise de requisitos até a solução do problema [NUN 94].

.....Este ambiente é um projeto do Instituto de Informática da Universidade Federal do Rio Grande do Sul – UFRGS e está sob coordenação do Prof. Dr. Daltro José Nunes.

Os ADS em geral proporcionam ferramentas aos usuários para facilitar a construção de software fornecendo um ambiente computacional específico e ergonomicamente planejado para seu projeto [PRE 95]. No PROSOFT há a determinação de satisfazer alguns requisitos, tais como [NUN 89]:

- Interface de comunicação homem-máquina única e ergonômica, com o fim de tornar o ambiente padrão e de fácil utilização;
- Integração entre as várias ferramentas do ambiente, tanto em relação as operações como também os objetos. Por exemplo, uma determinada ferramenta utiliza os objetos resultantes de outras ferramentas, como ilustra a Fig. 3.1;
- Possibilitar ao usuário criar suas próprias ferramentas para atender melhor a seus objetivos;

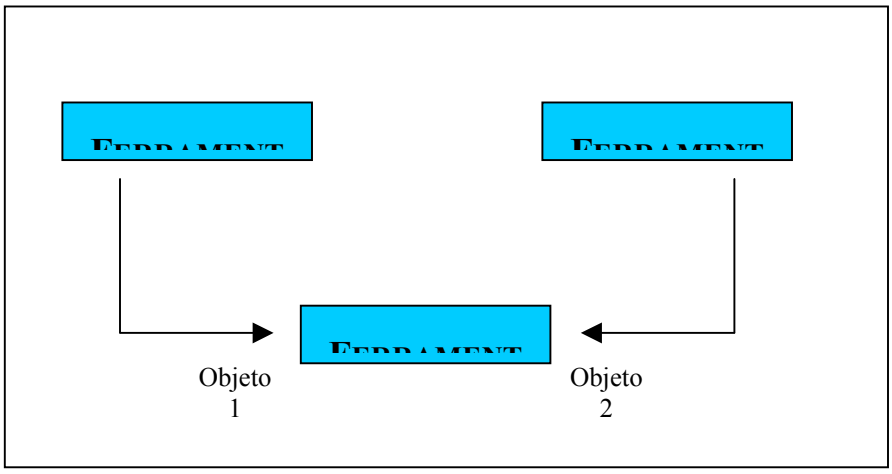


FIGURA 3.1 - Integração entre as Ferramentas no Ambiente PROSOFT

O ambiente PROSOFT é baseado nos seguintes conceitos [NUN 94]:

- A estratégia "data-driven", determina, que para encontrar a solução do problema, a definição das estruturas de dados devem ser definidas antes das operações [NUN 92];
- O conceito de modelos, onde a solução de um problema é o modelo de alguma teoria, por exemplo o VDM [BJO 78];
- O cálculo lambda [SLO 95];
- O conceito de Tipo Abstrato de Dados, onde tipos mais complexos podem ser definidos a partir da instanciação de tipos mais simples [WAT 91];
- O paradigma de Orientação a Objetos, onde a reusabilidade é fortemente aplicada devido aos conceitos de troca de mensagens, de herança e de polimorfismo [COA 90];
- O Método Algébrico [WAT 91]

..... É importante salientar que a especificação algébrica, com assinatura e axiomas [WAT 91], difere do formalismo usado no PROSOFT. Pode-se dizer que [MOR 97]:

- **A assinatura corresponde à instanciação e à interface do ATO e**
- **Os axiomas correspondem às operações (os conceitos de instanciação, interface e operações de um ATO serão vistos na próxima seção).**

..... Essa diferença traz vantagens para o PROSOFT, como a visualização do tipo que está sendo definido através da classe apresentada graficamente e a especificação somente das operações de manipulação dos objetos da classe.

3.1 Estrutura do PROSOFT

..... O desenvolvimento de sistemas no ambiente PROSOFT consiste na construção de um ou mais ATOs – Ambiente de Tratamento de Objetos e na comunicação entre eles, através da ICS – Interface de Comunicação do Sistemas.

.....Os novos ATOs construídos podem usar ATOs já existentes no ambiente, porém como o PROSOFT é um ambiente homogêneo (fechado), ele poderá usar somente esses componentes (ATOs) desenvolvidos no seu paradigma [REI 98].

.....A estrutura do PROSOFT é formada pela integração (comunicação) entre os ATOs através da ICS, como ilustra a Fig. 3.2.

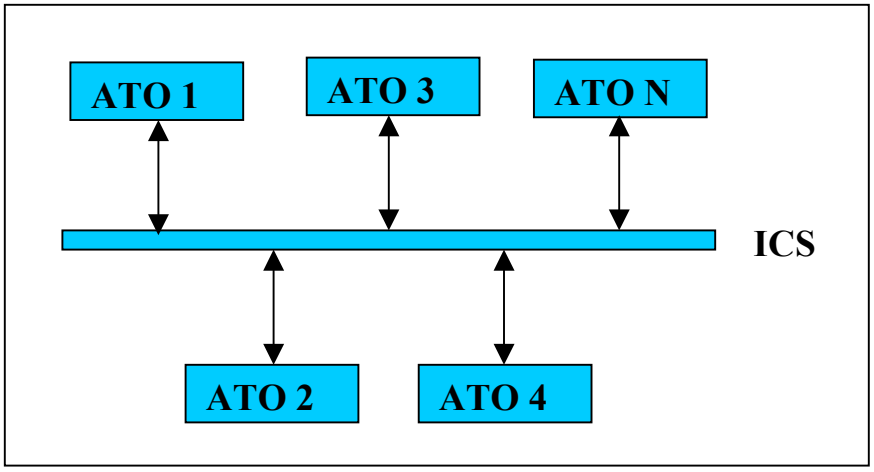


FIGURA 3.2 - Estrutura do Ambiente PROSOFT

3.1.1 Ambiente de Tratamento de Objetos

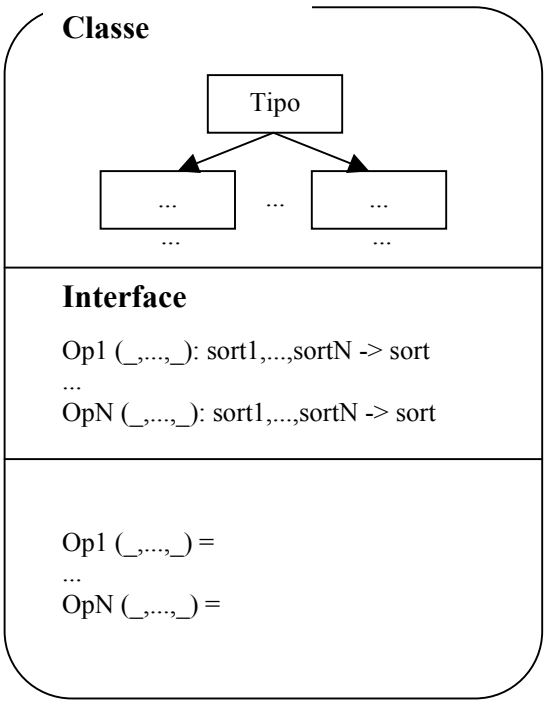
.....ATO – Ambiente de Tratamento de Objeto é o nome dado a qualquer sistema desenvolvido sob o paradigma PROSOFT. Um ATO PROSOFT implementa um tipo abstrato de dados e é composto por uma classe e um conjunto de operações que atuam sobre o objeto dessa classe. É importante ressaltar que o conceito de classe no PROSOFT é uma

estrutura de dados que corresponde aos atributos do objeto ou variáveis de instância nas linguagens orientadas a objeto [KOR 96].

.....A estrutura de um ATO é formada por três partes, segundo [NUN 94]:

- **Classe – é a instanciação, representa graficamente a especificação algébrica dos tipos abstratos de dados;**
- **Interface – especifica as operações sobre a classe em termos de funcionalidade;**
- **Operações – representa a semântica das operações da Interface.**

.....Conforme ilustrado na Fig. 3.3, cada ATO possui um nome. A interface é representada pelos nomes das operações (Op1, Op2, ...) e seus argumentos e as operações por suas especificações (semântica de cada operação).



Operações

FIGURA 3.3 - Estrutura do ATO

.....Todo ATO possui um conjunto de objetos (instâncias da classe do ATO) e esses objetos poderão somente ser manipulados pelo ATO que o originou, pois cada ATO trata apenas termos do *sort* por ele definido. Nos casos em que um ATO necessita manipular um objeto criado por outro ATO ele envia uma mensagem, via ICS, para o ATO criador [MOR 97].

3.1.2 Interface de Comunicação do Sistema

.....A ICS é o meio de comunicação entre os ATOs, sendo responsável pelo fluxo de dados e pelo controle da execução das operações [CAP 92].

.....O formato da ICS é o mostrado na Fig. 3.4 e contém:

- **Nome do ATO – é o nome do ATO da operação a ser utilizada;**
- **Nome da operação – é o nome da operação a ser utilizada;**
- **Seletor – é um objeto do tipo do ATO para qual a mensagem será enviada. Se na operação que está sendo chamada o seletor for um objeto, a ICS compara o seletor (objeto) enviado com o objeto da operação, no caso verdadeiro (se os objetos forem iguais) e os argumentos correspondentes, a operação é então executada. No caso do seletor na operação for uma variável, a operação receberá o objeto enviado, como também os argumentos e executará a operação;**
- **Argumentos – lista de argumentos necessários para utilizar a operação desejada, porém esta lista não precisa ser tratada com operações referentes ao tipo de dados "Lista".**

ICS (nome do ATO, nome da operação, seletor, argumentos)

FIGURA 3.4 - Estrutura da chamada ICS

..... Um exemplo da utilização de uma chamada ICS de um ATO para outro é mostrado na Fig. 3.5. Como, segundo [NUN 94], a pesquisa da operação é feita de cima para baixo dentro do ATO, se o nome do ATO coincidir, o nome da operação for uma operação pertencente ao ATO e o seletor for o mesmo, os argumentos serão ligados aos parâmetros formais e será executada a operação.

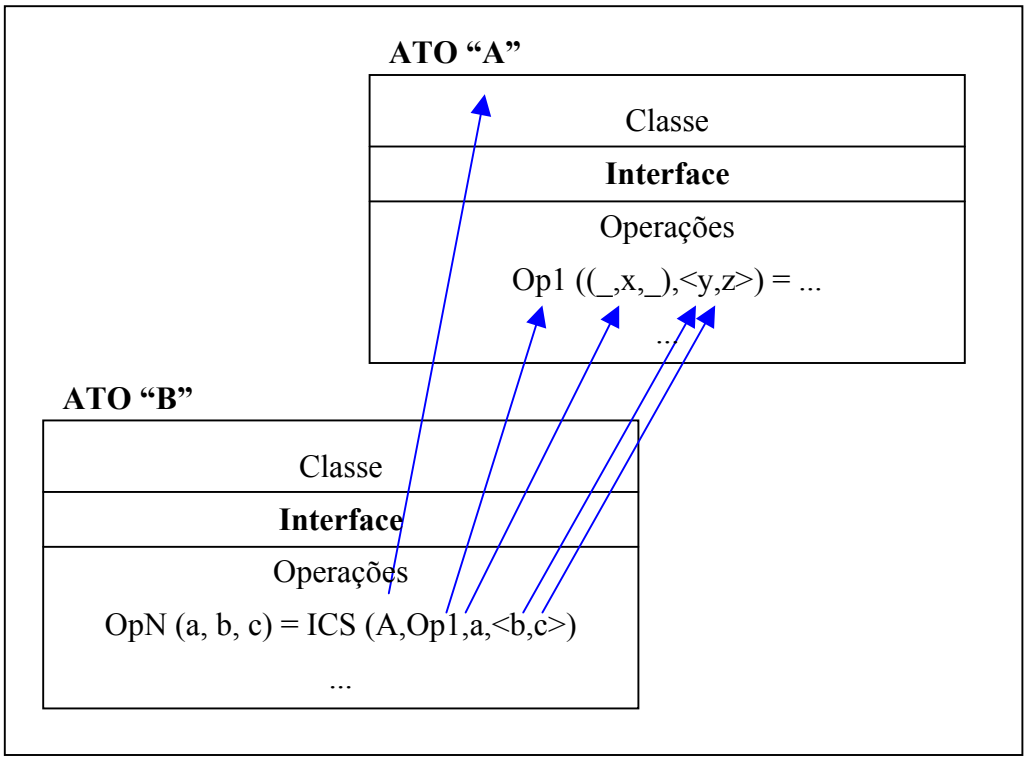


FIGURA 3.5 - Exemplo de uma chamada ICS

A operação chamada pela ICS será executada em função dos argumentos e podendo também utilizar o objeto (seletor) enviado. O objeto que foi enviado só será utilizado no lado direito da operação se na operação contiver uma variável no lugar do seletor e esta variável receber o objeto enviado. Sendo assim, como mostra a Fig. 3.6, dependendo do

seletor (variável ou não) e do predicado (p, p''), a operação executará um tipo de função (f ou f', f'' ou f''').

op (seletor, argumentos) =	**if**	p (seletor) **or** p' (seletor, argumentos)
	then	f (argumentos) **or** f'(argumentos, seletor)
	else	f''(argumentos) **or** f''' (argumentos, seletor)

FIGURA 3.6 - Estrutura de uma operação no PROSOFT

3.2 Tipos de Dados PROSOFT

Os tipos de dados utilizados no PROSOFT são classificados da seguinte forma:

- Primitivos – são os tipos derivados da linguagem hospedeira, o Pascal. São eles: *integer*, *real*, *string*, *boolean*, *char*, *time*, *date*;
- Compostos – são tipos definidos no método algébrico: conjunto, lista, mapeamento, registro e união disjuntiva;
- Definidos pelo Usuário – são tipos construídos a partir de outros tipos, e são representados pelos ATOs desenvolvidos no ambiente.

Com os tipos primitivos são triviais, a seguir serão mostrados os tipos compostos e os tipos desenvolvidos pelo usuário.

3.2.1 Tipos Compostos

Nesta seção serão apresentadas os tipos compostos, juntamente com exemplos de suas aplicações segundo sua representação diagramática, sua operação construtora e as possíveis operações utilizadas em cada tipo [MOR 97] [REI 98].

- Conjunto

Representa uma coleção de objetos, respeitando as características de conjunto, onde a ordem desses objetos não é importante.

Por exemplo, como ilustra a Fig. 3.7, uma Escola é formada por um conjunto (representado por **s**) de Sala. O tipo Sala é uma referência ao ATO Sala.

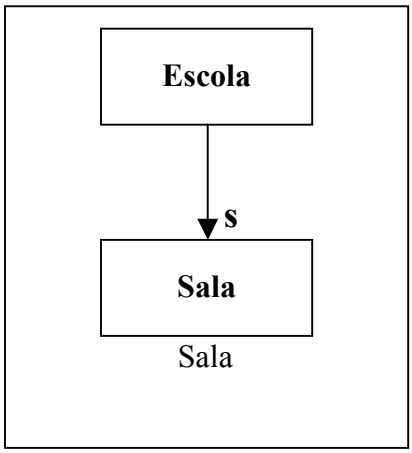


FIGURA 3.7 - Classe Escola

A operação construtora *add* e as outras operações do tipo Conjunto estão presentes na Tab. 3.1, onde contém suas funcionalidades e exemplos de aplicação.

TABELA 3.1 - Operações do tipo composto Conjunto [MOR 97]

Operação	Funcionalidade	Exemplo
Add	Set x Component → Set	add ({x1,x2},x3) = {x1,x2,x3}
Belongs_to (∈)	Component x Set → Boolean	x1 ∈ {x1,x2} = TRUE
Cardinality (car)	Set → Nat	car ({x1,x2,x3}) = 3
Complement (\ )	Set x Set → Set	{x1,x2} \ {x2} = {x1}
Containtion (⊇)	Set x Set → Boolean	{x1,x2,x3} ⊇ {x1,x3} = TRUE
Delete	Set x Component → Set	delete ({x1,x2},x2) = {x1}
Empty-set	→ Set	empty-set = { }
Equal (=)	Set x Set → Boolean	{x1} = {x2} = FALSE
Intersection (∩)	Set x Set → Set	{x1,x2} ∩ {x2} = {x2}
Is_in	Set x Component → Boolean	is_in ({x1,x2},x1) = TRUE
Union (∪)	Set x Set → Set	{x1,x2} ∪ {x3} = {x1,x2,x3}

- Mapeamento

Representa uma função, onde ocorre a relação de um Tipo-Domínio com um Tipo-Imagem.

Por exemplo, como ilustra a Fig. 3.8, um Indivíduo pode ser representado pelo seu Nome e pelo número da sua Carteira de Identidade. Sendo que Nome é do tipo *string* e Carteira de Identidade do tipo *integer*.

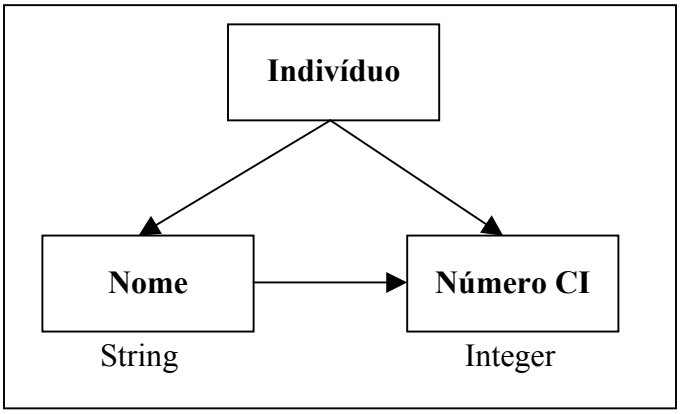


FIGURA 3.8 - Classe Indivíduo

A operação construtora *modify* e as outras operações do tipo Mapeamento estão presentes na Tab. 3.2, onde contém suas funcionalidades e exemplos de aplicação.

TABELA 3.2 - Operações do tipo composto Mapeamento [MOR 97]

Operação	Funcionalidade	Exemplo
Composition (Θ)	Map x Map → Map	[x1 → y1] Θ [x2 → y2] = [x1 → y2]
Domain (dom)	Map → 2^{Set}	dom ([x1 → y1, x2 → y2]) = {x1,x2}
Empty-Mapping	→ Map	empty-mapping = { }
Image_of	Domain, Map → Rng	image_of (x1, [x1 → y1]) = y1
Merge (∪)	Map x Map → Map	[x1 → y1] ∪ [x2 → y2] = [x1 → y1,x2 → y2]
Modify	Domain x Rng x Map → Map	modify (x1, y1, [ ]) = [x1 → y1]
Override (+)	Map x Map → Map	[x1→y1,x2→y2]+[x2→y3]=[x1→y1,x2→y3]
Range (rng)	Map → Set	rng ([x1 → y1,x2 → y2]) = {y1, y2}
Restrict_to (|)	Map x 2^{Set} → Map	[x1 → y1,x2 → y2] | {x1} = [x1 → y1]
Restrict_with (\ )	Map x 2^{Set} → Map	[x1 → y1,x2 → y2] \ {x1} = [x2 → y2]

- Lista

Representa uma seqüência ordenada de zero ou mais componentes.

Por exemplo, como ilustra a Fig. 3.9, uma Sala é formada por uma lista (representada por *) de Aluno. Sendo que Aluno é uma referência ao ATO Aluno.

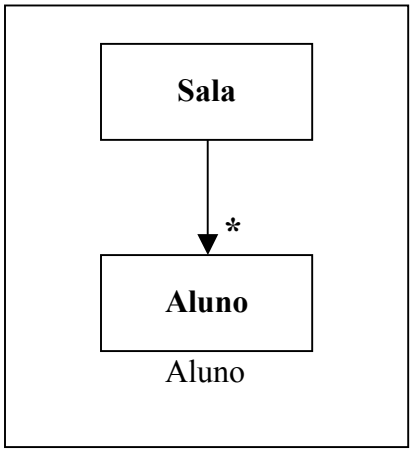


FIGURA 3.9 - Classe Sala

A operação construtora *cons* e as outras operações do tipo Lista estão presentes na Tab. 3.3, onde contém suas funcionalidades e exemplos de aplicação.

TABELA 3.3 - Operações do tipo composto Lista [MOR 97]

Operação	Funcionalidade		Exemplo
Concatenation(^)	List x List	→ List	<x1, x2> ^ <x1> = <x1, x2, x1>
Cons	Component x List	→ List	cons (x1, <>) = <x1>
Elements (elems)	List	→ 2^{Set}	elems (<x1, x2, x1>) = {x1, x2}
Empty-list		→ List	empty-list = < >
Head (hd)	List	→ Component	hd (<x1, x2>) = x1
Index (inds)	List	→ 2^{Nat}	inds (<x1, x2, x1>) = {1, 2, 3}
Last	List	→ Component	last (<x1, x2, x1>) = x3
length (lng)	List	→ Nat	lng (<x1, x2, x1>) = 3
Projection (_,[ ])	List x Nat	→ Component	<x1, x2> [2] = x2
Replace (_,+,[_])	List x Nat x Component	→ List	<x1, x2> + [2, x3] = <x1, x3>
Tail (tl)	List	→ List	tl (<x1, x2>) = x2

- Registro

Define tuplas de tipos heterogêneos, expressando assim um produto cartesiano.

Por exemplo, como ilustra a Fig. 3.10, um Livro possui Nome, Autor, Editora, Ano e Número de Páginas. Sendo que Nome, Autor e Editora são do tipo *string* e Ano e Número de Páginas do tipo *integer*.

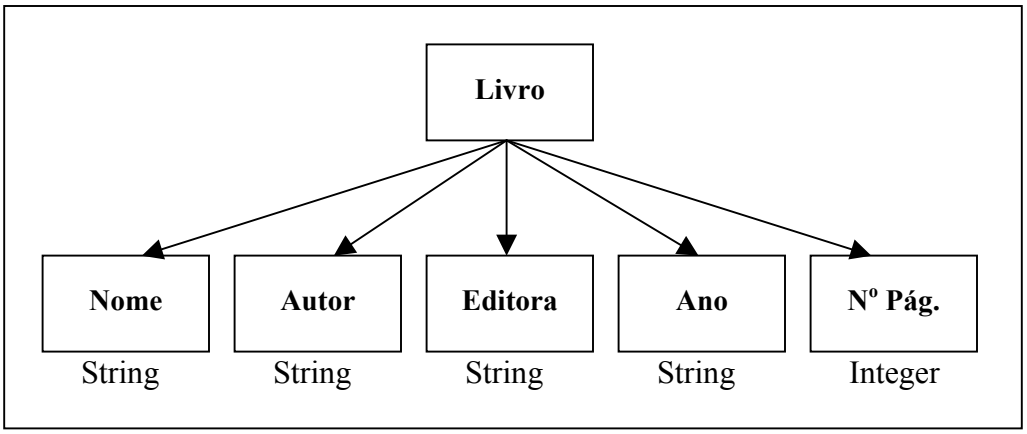


FIGURA 3.10 - Classe Livro

O tipo Registro possui uma operação construtora do tipo (_,_,...,_) e outra operação do tipo observadora, *select* (seleciona um dos campos do registro). Essas operações estão presentes na Tab. 3.4, onde contém suas funcionalidades e exemplos de aplicação. Para facilitar a seleção (e identificação) dos campos dos registros, os tipos dos seus campos são precedidos por uma *tag* (rótulo sublinhado).

TABELA 3.4 - Operações do tipo composto Registro [MOR 97]

Operação e Funcionalidade	Exemplo
(_,_,...,_): C_1, C_2, ..., $C_n \rightarrow$ Record	(Rua rua, Bairro bairro, Cid cidade, Est estado, Cep cep)
Select-_: Tag Record $\rightarrow$ S_1 S_2 ... S_n	select-Bairro (_, Bairro Centro, _,_,_) = Centro

- União Disjuntiva

Define a união de tipos diferentes de elementos, sendo útil quando uma classe de dados necessita expressar valores alternativos.

Por exemplo, como ilustra a Fig. 3.11, um Professor pode ser Titular, Contratato ou Assistente. Cada tipo de Professor é do tipo *string*.

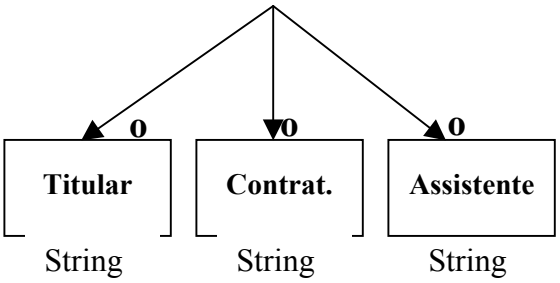


FIGURA 3.11 - Classe Professor

A construtora do tipo União Disjuntiva é o sinal de + e possui a seguinte funcionalidade: + (_,_,...,_): Tag_1 $Sort_1$, Tag_2 $Sort_2$, ..., Tag_N $Sort_N$ → Union

..... Um exemplo seria:

+ (<u>titular</u> Titular, <u>contratado</u> Contratado, <u>assistente</u> Assistente)

3.2.2 Tipos Definidos pelo Usuário

Os tipos definidos pelo usuário são os ATOs existentes no PROSOFT. Sendo que estes ATOs já especificados podem ser usados na criação de novos ATOs.

..... O ATO-ATO [RIB 91], permite incluir um novo ATO no ambiente a partir de sua especificação algébrica, eliminando assim a necessidade de recompilação de todo o ambiente.

..... Um ATO já implementado no PROSOFT que é muito importante é o ATO Diretório. Ele é responsável pelo armazenamento físico dos objetos do ambiente, fornecendo um conjunto de rotinas de tratamento de arquivos [CAP 92].

..... Entre as ferramentas já implementadas, estão as de interface com o usuário:

- **ATO Cenário – permite a definição de uma janela no PROSOFT, sendo a principal responsável pela interação do usuário com o sistema;**

- **ATO Quadro – auxilia na definição de uma quadro (janela de interface) dentro de um cenário (ambiente);**
- **ATO Menu – permite a construção de menus em quadros;**
- **ATO Opção – define as opções de um menu;**
- **ATO Texto – auxilia na edição de textos;**
- **ATO Edgraf – editor que auxilia na manipulação de objetos gráficos;**
- **ATO Figura – permite a definição de objetos gráficos;**
- **ATO Coord – define e localiza uma coordenada de tela.**

.....Outros ATOs foram implementados para auxiliar no desenvolvimento de sistemas, tais como o ATO NSD (para especificação de diagramas Nassi-Schneidermann [NAS 73] [CHA 74]), o ATO SADT [ROD 77] [ROD 85] e o Editor Larch (para especificação em Larch [GUG 85]).

Há também a definição de um ambiente (*shell*) para construção de sistemas especialistas denominado Expert [MOR 97], um gerenciador de processo [REC 98] e um simulador de processos de software [SIL 2001].

3.3 PROSOFT Cooperativo

O PROSOFT Cooperativo [REI 98] é uma parte fundamental para o trabalho presente, por isso a ênfase maior para este tema.

Sua função é fornecer um conjunto de serviços básicos para a gerência de objetos, ferramentas, usuários e estações de desenvolvimento de software, sendo que o controle dos objetos construídos pelos usuários durante o desenvolvimento cooperativo assume um papel principal.

A Fig. 3.12, ilustra o trabalho cooperativo de dois projetos de desenvolvimento de software em andamento. Nos projetos (A e B) as atividades (círculos) são conectadas por arcos em um grafo de dependência. Uma atividade (podendo envolver diversos profissionais) produz um conjunto de objetos de saída a partir de um conjunto de objetos de entrada.

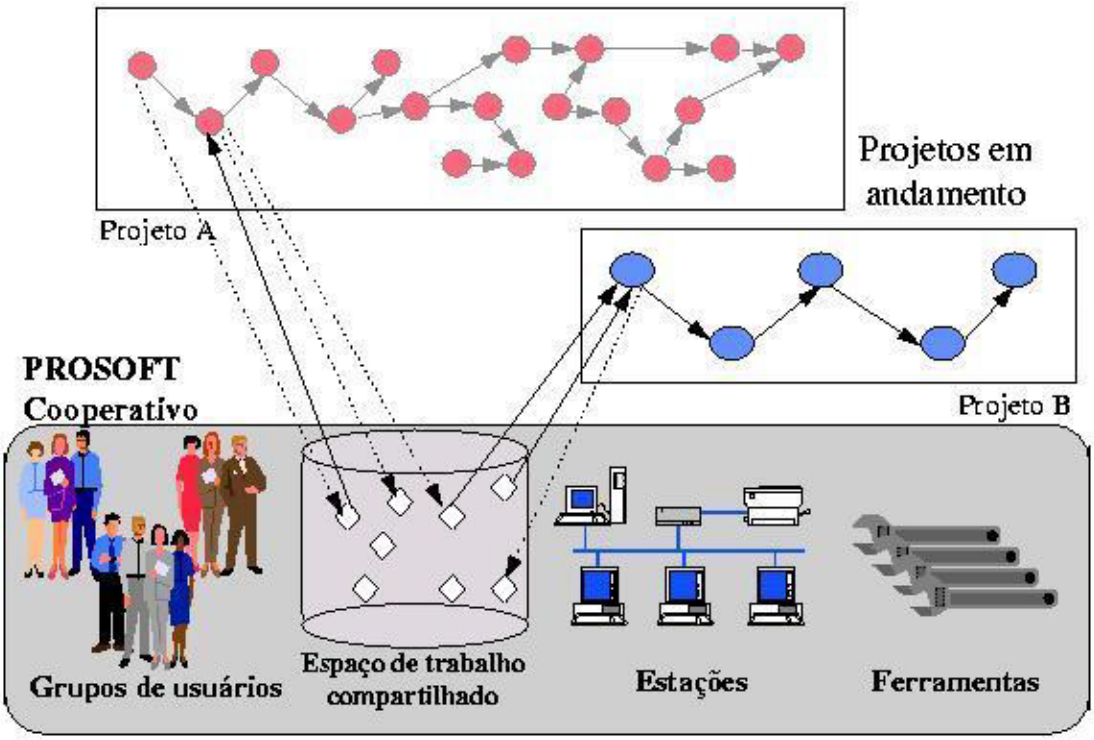


FIGURA 3.12 - Desenvolvimento de Software com o PROSOFT Cooperativo

O PROSOFT Cooperativo tem como características gerais:

- Classificação dos usuários de acordo com as tarefas desempenhadas pelos desenvolvedores durante o ciclo de vida de um software (podem ser super-usuários, gerentes ou usuários comuns);
- Gerência das estações, fornecendo informações sobre sua utilização e quem está em cada estação;
- Integração das ferramentas (ATOs) entre si e com os usuários, sendo que cada ATO deve seguir um "modelo" de especificação para que possa ser considerado cooperativo;
- Controle de acesso e monitoramento dos objetos, sendo que há grupos de usuários que possuem permissões de acesso diferenciadas formando um espaço de trabalho compartilhado;
- Definição de estados paras os objetos (instável, estável ou consolidado);
- Modelo de versões, o qual disponibiliza um conjunto de operações que permite a criação de um árvore de versões derivadas a partir de um objeto consolidado.

Esses conceitos são importantes pois a próxima seção conterá um metodologia sobre a construção de novos ATOs cooperativos para o ambiente PROSOFT (no qual foi baseado este trabalho), como também o funcionamento e comportamento dos objetos, usuários e estações que fazem parte do processo de cooperação.

4 COMPONENTES DE PERCEPÇÃO PARA INTERFACES COOPERATIVAS PROPOSTOS PARA O AMBIENTE PROSOFT COOPERATIVO

Este capítulo apresenta um modelo de percepção para interface cooperativa especificado para o PROSOFT Cooperativo [REI 98], fornecendo assim componentes que complementam o projeto de um sistema desenvolvido neste ambiente.

O ambiente PROSOFT Cooperativo foi escolhido para este trabalho por apresentar um modelo para o trabalho em grupo, porém não se preocupando com a percepção dos membros do grupo durante a realização das atividades. Os componentes aqui propostos, específicos para atividades síncronas, fornecem essa percepção da interação do grupo fazendo com que o sistema projetado se torne mais completo e interativo.

4.1 Componentes de Percepção propostos

Os componentes aqui apresentados são utilizados para fornecer o *workspace awareness* para os participantes do trabalho cooperativo, assim como para serem utilizados na construção de novas ferramentas para o Ambiente PROSOFT. Para ilustrar alguns desses componentes, são construídas telas de exemplo a partir do Editor de Texto Microsoft Word.

Esses componentes foram determinados a partir da pesquisa sobre o tema de Percepção para os sistemas cooperativos, seguindo o paradigma WYSIWIS – *What You See Is What I See*. Viu-se portanto, a necessidade de especificar componentes que já estão disponíveis em alguns *toolkits* (tal como, múltiplos cursores, teleponteiros e barra de rolagem de sincronização, papel para os usuários), como também sistemas que eram citados como exemplos de *groupware* e que fornecem uma percepção do ambiente de trabalho satisfatória (tal como Ambiente de Percepção, Bloco de Notas e Janelas Auxiliares).

São eles:

- Múltiplos Cursores;
- Teleponteiros;
- Ambiente de Percepção;
- Bloco de Notas;
- Papéis para os Usuários;

- Barra de Rolagem de Sincronização;
- Janelas Auxiliares.

A maioria dos sistemas cooperativos utiliza mais de um componente para fornecer a percepção, por isso é fornecido um recurso que controla as preferências dos usuários. Desta forma, é possível fornecer, por exemplo, uma única cor para um determinado usuário em qualquer componente que compõe a ferramenta construída.

4.1.1 Preferências dos Usuários

Este recurso possibilita a utilização de vários componentes em uma ferramenta que está sendo construída fornecendo um controle sobre as preferências dos usuários (cor, label, ícone). Assim, os usuários participantes do grupo terão as mesmas características em qualquer componente.

Para tanto, é necessário a utilização deste recurso sempre que um desenvolvedor colocar pelo menos um dos componentes aqui propostos. No momento que o usuário "cadastrar" suas preferências, elas serão únicas e nenhum outro usuário poderá ter preferências iguais. Isto permite que um label seja característico de um determinado usuário e portanto reconhecido por todo o grupo.

A representação dos sistemas é conseguida através da função chamada Exibe. Essa função não foi especificada neste trabalho por se tratar de uma operação pertencente ao PROSOFT Java e por ser especifica de cada sistema. Assim, quando um desenvolvedor utilizar um dos componentes aqui propostos, a função Exibe deste componente será especificada juntamente com o sistema que está sendo desenvolvido.

4.1.2 Múltiplos Cursores

Os múltiplos cursores são usados principalmente em editores de textos. São cursores que indicam a posição, o movimento e as atividades que cada participantes está desenvolvendo. Sendo assim, cada usuário que participa da tarefa cooperativa possui seu próprio cursor, como ilustra a Fig. 4.1.

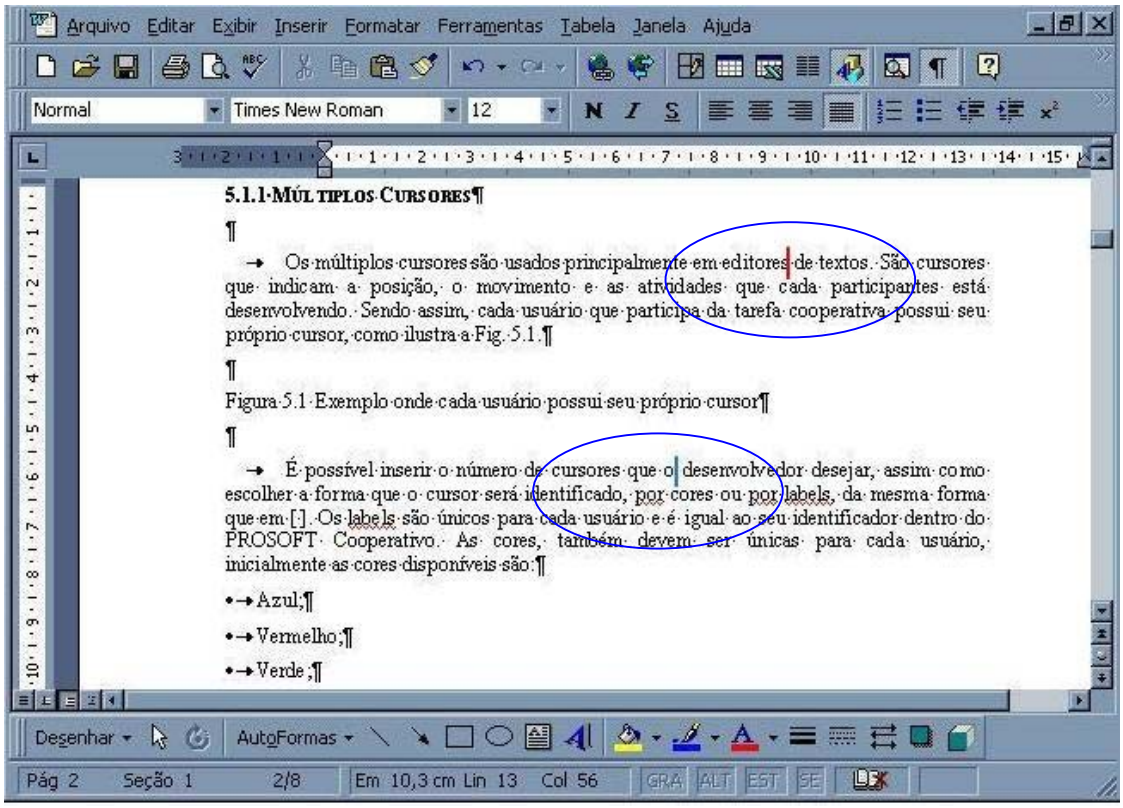


FIGURA 4.1 - Exemplo de Múltiplos Cursores

É possível inserir o número de cursores que o desenvolvedor desejar, assim como escolher a forma que o cursor será identificado, por cores ou por labels, da mesma forma que em [MIT 96] [GUT 96c]. Os labels são únicos para cada usuário e é igual ao seu identificador dentro do PROSOFT Cooperativo.

A movimentação do cursor é possível através da operação *movimenta_cursor*, assim como conhecer onde o cursor está localizado, através da operação *posição_cursor*.

4.1.3 Teleponteiros

Os teleponteiros [GUT 98] [NUS 2000b] são usados para "chamar a atenção" dos participantes do grupo para a movimentação de cada usuário. São ponteiros individuais que podem ser, igualmente aos múltiplos cursores, identificados por labels únicos (identificadores dos usuários no PROSOFT Cooperativo) ou por cores.

Assim como os múltiplos cursores, é possível movimentar o teleponteiro através da operação *movimenta_teleponteiro* e verificar sua posição através da operação *posição_teleponteiro*.

Os teleponteiros possuem a capacidade de modificar seu formato de acordo com a tarefa que está executando, conforme visto em [GRE 92]. Por exemplo, quando o usuário

deve esperar por um determinado tempo enquanto a máquina está executando uma atividade o teleponteiro pode assumir o formato de uma ampulheta.

É possível também determinar qual atividade que o sistema deve realizar através dos *clicks* feitos pelo *mouse* e pela posição que ele está localizado na interface. Para tanto existem as operações *características_click* (fornece o botão que foi clicado e a quantidade de *clicks*) e *posição_teleponteiro*.

A Fig. 4.2 ilustra uma atividade em uma seção cooperativa mostrando os teleponteiros de cada usuário movimentando-se durante a execução de uma tarefa.

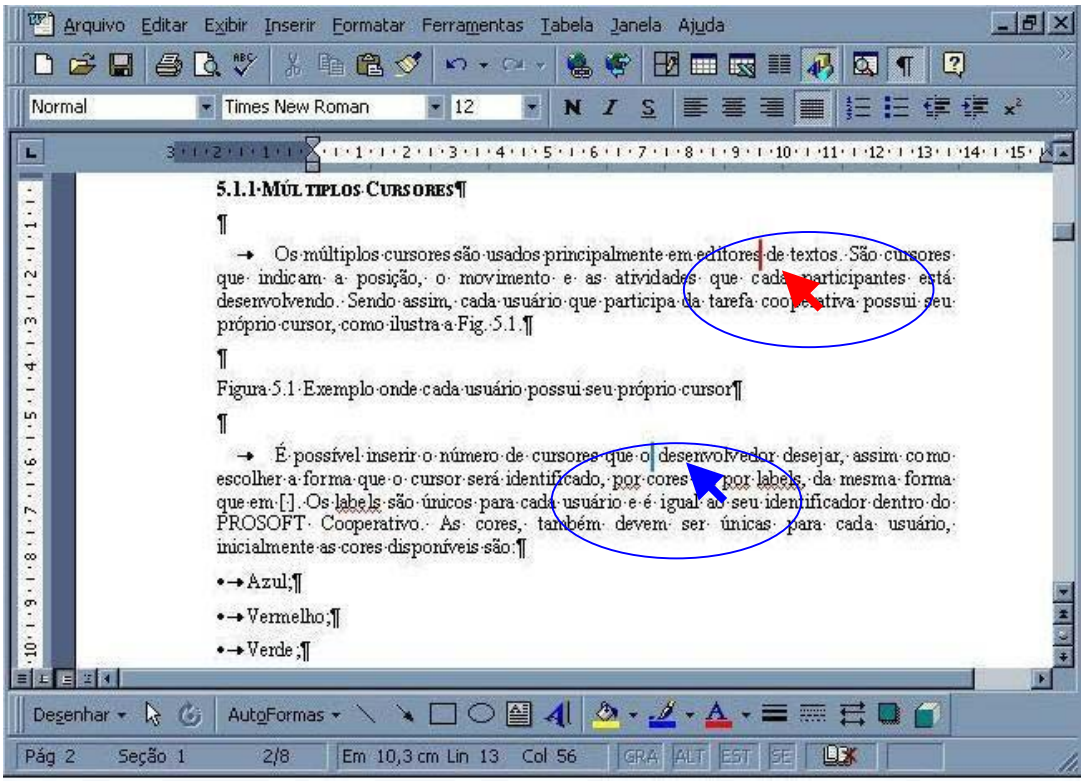


FIGURA 4.2 - Exemplo de Teleponteiros

4.1.4 Ambiente de Percepção

O Ambiente de Percepção permite às pessoas o conhecimento de quem está disponível para o trabalho em grupo [DOU 92b] [GAJ 95]. É possível saber se uma determinada pessoa está pronta para participar de uma reunião ou se ela está ausente temporariamente [KAR 97]. Podemos citar como exemplos os ambientes *EasyMeeting* [KAR 97], *TeleFreek* [COC 93] e IRIS *Group Editor Environment* [KOC 97] [SCL 97].

Quando é criado o Ambiente de Percepção, através da operação *cria*, não há nenhum ambiente interno formado. Os ambientes são criados a medida que os usuários participantes necessitam de uma determinada situação.

Para o PROSOFT Cooperativo, esta ferramenta apresenta três ambientes internos possíveis:

- Reunião – as pessoas se posicionam neste ambiente quando estão prontas para uma reunião previamente marcada para um determinado local e horário. Assim, é possível saber quando todos os participantes da reunião estão disponíveis e a partir desse ponto começar a reunião, sem que seja necessário estarem em um mesmo local a espera do restante do grupo;
- Social – este ambiente é usado quando uma determinada pessoa está a disposição para qualquer eventual compromisso. Geralmente, as pessoas que se encontram nesse quadro, podem estar trabalhando porém disponíveis para uma determinada conversa ou até mesmo para uma confraternização;
- Conversa Particular – é usado quando duas pessoas estão ocupadas e não podem ser incomodadas. Mesmo o restante do grupo sabendo que estas pessoas estão presentes (ou trabalhando em locais distantes), não poderão ser incomodadas até voltarem para a área do Ambiente Social ou Reunião.

Cada usuário participante desse ambiente possui uma forma de identificação. Essa identificação é feita através de um ícone representativo de cada um, escolhido pelo próprio usuário.

A movimentação do usuário dentro dos três ambientes é através da inserção do ícone representativo (para entrar em um ambiente), retirada do ícone (para sair de um ambiente) ou substituição do ícone por outro representando uma saída temporária. Ou ainda, a colocação de labels significativos referente a cada uma das três situações possíveis que o usuário pode assumir.

4.1.5 Bloco de Notas

O Bloco de Notas, semelhante a um *chat* [MIT 96] [ARA 97], tem finalidade de facilitar a discussão dos participantes de uma sessão cooperativa. Poderá ser usado, por exemplo, durante uma edição cooperativa de um texto, em que os membros do grupo necessitam formalizar uma determinada parte do texto para ser escrito no trabalho que estão construindo.

Cada usuário será identificado por seu identificador único conhecido em todo o PROSOFT Cooperativo e poderá enviar mensagens desde que esteja cadastrado nos Preferências dos Usuários.

4.1.6 Papel para os Usuários

São papéis que os usuários podem assumir durante o desenvolvimento de uma tarefa cooperativa [BOR 95] [ARA 97].

Cada papel possuirá um ícone representativo e cada usuário terá seu ícone aparecendo no decorrer do trabalho. Sendo possível também, a critério do desenvolvedor, que dependendo da área do trabalho um usuário poderá ter diferentes papéis em diferentes espaços.

Também é possível, através da operação *altera_papel_usuário* modificar o papel de um determinado usuário no decorrer da atividade cooperativa.

4.1.7 Barra de Rolagem de Sincronização

A Barra de Rolagem de Sincronização, tal como ilustra a Fig. 4.3, permite que um usuário consiga localizar a outra pessoa que está trabalhando cooperativamente com ele e assim posicionar sua barra de rolagem na posição ideal para visualizar a tarefa que estão realizando. Dessa forma é possível perceber o que as outras pessoas do grupo estão realizando e onde estão trabalhando.

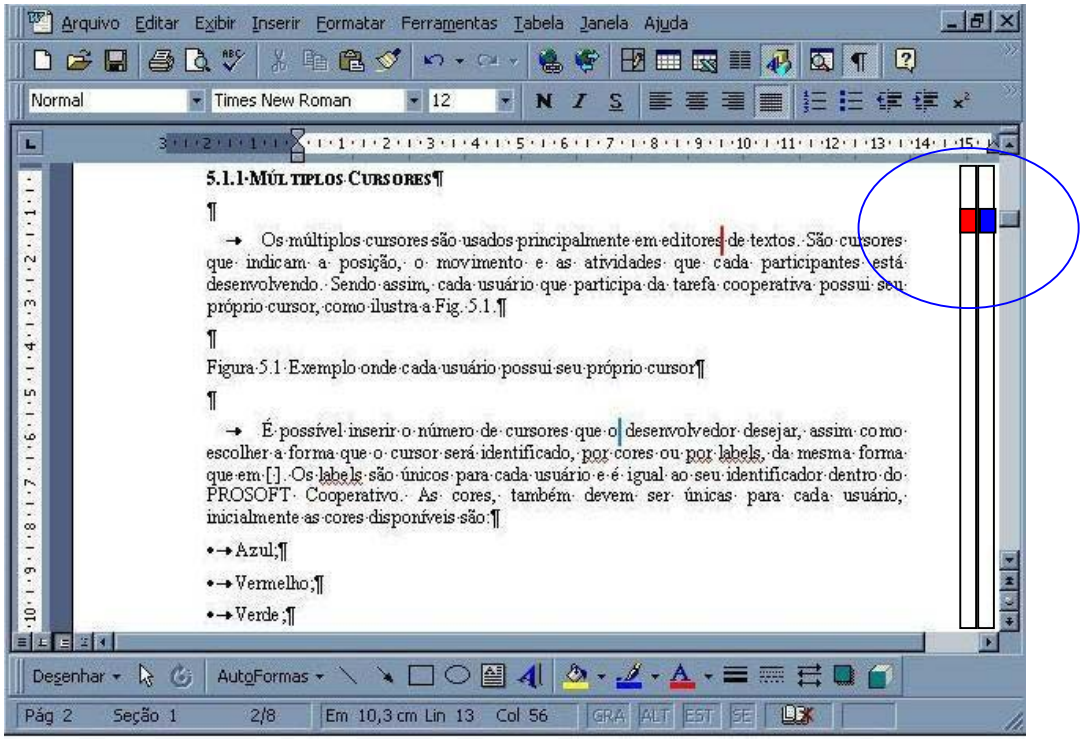


FIGURA 4.3 - Exemplo de Barra de Rolagem de Sincronização

A identificação de cada usuário é feita através da cor escolhida por ele. Essa cor é a mesma disponibilizada para os múltiplos cursores e teleponteiros, permitindo assim que haja uma consistência entre os três componentes quando utilizados na mesma ferramenta [MIT 96] [NUS 2000b].

4.1.8 Janelas Auxiliares

As janelas auxiliares podem ser utilizadas juntamente com a ferramenta que o grupo está trabalhando, sendo assim uma ferramenta de auxílio para a percepção que não promove o acumulo de componentes na área que o usuário está trabalhando, como por exemplo os teleponteiros e os múltiplos cursores.

Uma janela auxiliar poderá assumir um desses três tipos:

- WYSIWID – *What You See Is What I Do* – nesse tipo de janela é possível visualizar as atividades que os outros participantes estão realizando. Ela mostra a área do objeto que um outro usuário está trabalhando durante o trabalho cooperativo [GUT 96b] [GUT 96c];
- *Radar View* – nesse tipo de janela é possível visualizar o objeto que as pessoas do grupo estão trabalhando miniaturizado, mostrando ainda o local e qual dos usuários estão trabalhando. Essa identificação poderá ser feita por um retângulo em torno da área de trabalho e identificando também os teleponteiros de cada usuário [GUT 96d] [GUT 98] [NUS 2000b];
- *Miniature View* – parecido com o tipo de janela *Radar View*, porém não informa quem está trabalhando. A área de trabalho é visualizada em um formato miniaturizado, porém não é possível verificar qual dos participantes está trabalhando em uma determinada posição [GUT 96b].

Cada usuário que participa da sessão cooperativa poderá escolher o tipo de janela que mais se adapte as suas necessidades, sendo possível alterar o tipo posteriormente através da operação *altera_tipo_janela*.

4.2 Utilização dos ATOs de Percepção no PROSOFT Cooperativo

Os ATOs de Percepção são ATOs cooperativos, criados para auxiliar no desenvolvimento de ferramentas cooperativas no Ambiente PROSOFT Cooperativo. Após estudos mais avançados com o Ambiente PROSOFT e com as características de CSCW [BOR 95] [KYN 95] [GUT 97] [NUS 2000b], viu-se a necessidade de modificar o formato inicial do trabalho.

Primeiramente o estudo foi voltado para a criação de um ferramenta que auxiliasse os desenvolvedores a criar seu sistema cooperativo. Esta ferramenta disponibilizaria os componentes necessários para percepção em uma interface cooperativa de acordo com as características do sistema em criação e com as necessidades que o desenvolvedor desejasse, como ilustra a Fig. 4.4. Assim seria fornecido um conjunto de componentes que o projetista poderia utilizar ou não sem que ele conhecesse profundamente esses componentes.

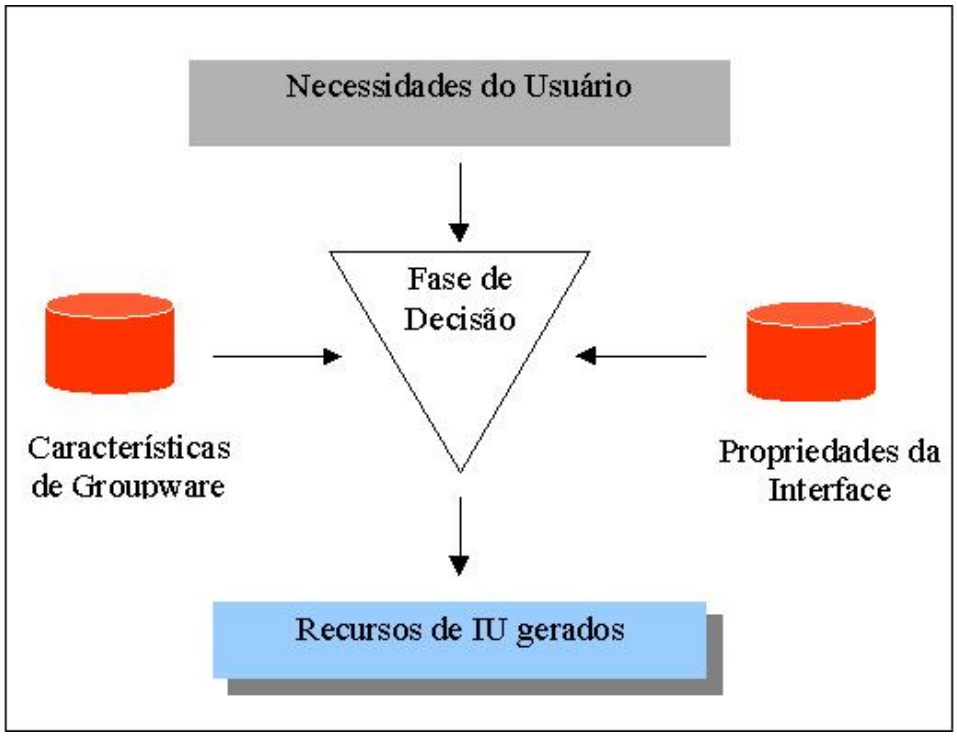


FIGURA 4.4 - Ferramenta que seria desenvolvida para o Ambiente PROSOFT

Porém, com conhecimentos mais aprofundados em CSCW e *groupware* foi detectado que um desenvolvedor quando chega ao ponto em que projeta um sistema para o trabalho em grupo também irá preocupar-se com a percepção das atividades que irão ocorrer durante o processo. Sendo assim, viu-se que essa forma de auxilio seria muito interessante para um inexperiente projetista porém mais trabalhoso e demorado para um projetista com uma certa experiência e com conhecimentos sobre *awareness*.

Para reforçar o abandono da idéia inicial, veio a forma de utilização dos objetos fornecidos pelos ATOs construídos no Ambiente PROSOFT. Quando um projetista cria seus ATOs ele coloca o ATO principal da ferramenta de percepção para conseguir utilizar o objeto por ele gerado, como ilustra o exemplo da Fig. 4.5, onde o desenvolvedor "transforma" a ferramenta de Diagramas NSD [NAS 73] [CHA 74] em uma ferramenta cooperativa. Desta forma estaria, juntamente com os componentes necessários, outros componentes que não seriam utilizados, tendo assim o objeto vários campos em branco. E também criaria uma dificuldade grande em fazer as chamadas (ICS) do objeto gerado pela ferramenta desenvolvida e pela ferramenta de percepção.

FIGURA 4.5 - Exemplo de uso da Ferramenta para
Percepção para Interfaces Cooeprativas no
Ambiente PROSOFT Cooperativo

Com a criação de uma "biblioteca" de componentes, é possível criar ferramentas no Ambiente PROSOFT com maior agilidade. Assim, o projetista, já conhecedor desses componentes, pode facilmente adequar os recursos necessários a sua ferramenta. Os componentes seriam colocados no lugar necessário e seriam facilmente ligados (através da ICS) com a ferramenta em desenvolvimento.

5 Criação de ATOs no PROSOFT Cooperativo utilizando os Componentes de Percepção

O PROSOFT Cooperativo foi desenvolvido por Rodrigo Q. Reis na sua dissertação de Mestrado [REI 98] na Universidade Federal do Rio Grande do Sul - UFRGS. Para a realização do trabalho presente, foi necessário a utilização de toda a base oferecida pelo PROSOFT Cooperativo, com isso tornou-se útil a realização de uma metodologia de trabalho.

A partir desta metodologia é possível criar ATOs cooperativos utilizando os componentes de percepção com maior facilidade e rapidez e compreender seu funcionamento dentro do ambiente, como também os objetos gerados por eles.

Primeiramente será visto a criação do Ambiente Cooperativo dentro do PROSOFT, sendo assim iniciado o processo de cooperação. Neste tópico também será mostrado como é realizado o controle de usuários e das estações de trabalho. Após a criação do ambiente, é possível criar os ATOs cooperativos (mostrado no próximo tópico), com suas particularidades, necessidades e utilizando os componentes de percepção. E finalmente, o comportamento dos objetos compartilhados gerados a partir dos ATOs cooperativos.

5.1 Criação do Ambiente Cooperativo no PROSOFT

A criação do ambiente cooperativo envolve a criação de um super-usuário. Este super-usuário é que terá todo acesso às informações do sistema e ainda pode incluir e excluir usuários, grupos e estações do ambiente.

Para que seja concretizada a criação do ambiente cooperativo, caso ainda não exista uma instância do PROSOFT Cooperativo, é necessário fazer uma chamada para a operação *cria* do ATO PROSOFT Cooperativo, passando como parâmetros o identificador do usuário (neste caso, este usuário será o super-usuário), o nome do usuário e sua senha. Neste momento a operação *cria* irá fazer uma chamada (através da ICS) às operações que criam os componentes necessários para o ambiente cooperativo. São eles: ATO Usuários Cooperativos, ATO Estações, ATO Espaço de Trabalho Compartilhado e o ATO Ferramentas, como ilustra a Fig. 5.1.

.....Sendo assim, o super-usuário poderá iniciar o processo de cooperação, inserindo novos usuários e grupos de usuários, controlando as estações envolvidas no trabalho, gerando novas ferramentas ou ainda compartilhando os objetos gerados a partir de ATOs cooperativos.

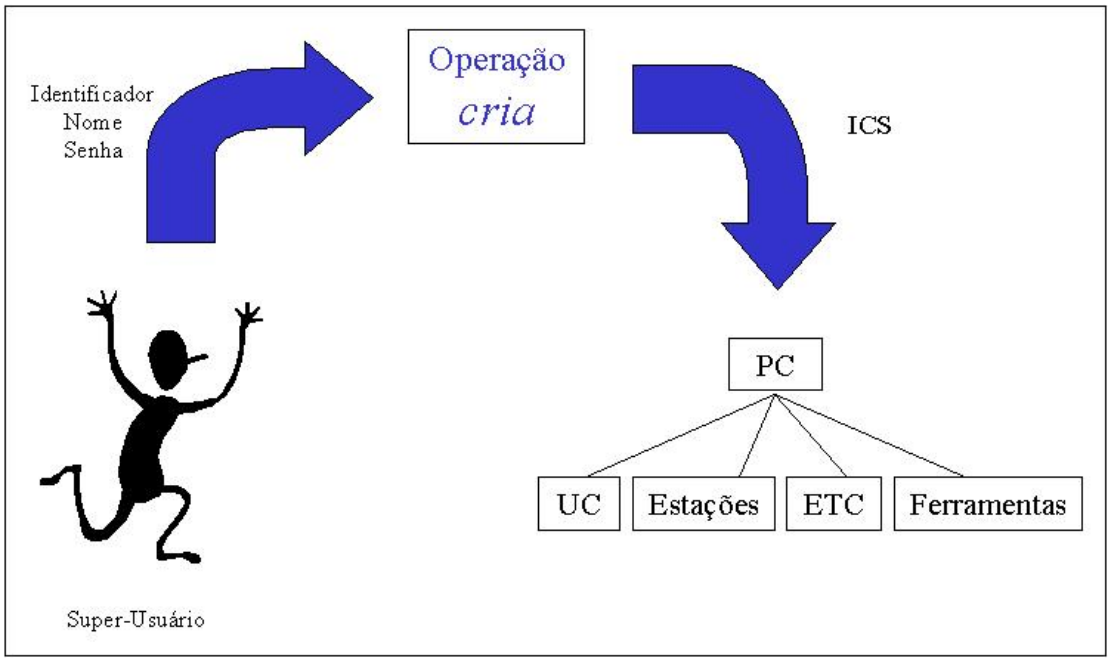


FIGURA 5.1 - Criação do PROSOFT Cooperativo

5.1.1 Gerência de Usuários

Como dito anteriormente um super-usuário é definido na criação do ambiente cooperativo, sendo que este usuário possui acesso irrestrito a todas as informações do ambiente.

Porém, cada usuário inserido no ambiente (com a operação *inclui_usuário*) terá uma função de acordo com as tarefas que ele irá desempenhar. Os tipos de usuários são:

- Super-usuários – podem realizar qualquer atividade, pois estão no nível mais alto do ambiente;
- Gerentes – para cada grupo do ambiente há um usuário que é promovido a gerente, sendo que ele pode incluir ou excluir um usuário do grupo, criar novos gerentes e controlar os usuários restantes do grupo;
- Usuários comuns – são usuários que não possuem nenhum dos dois títulos acima.

Cada usuário possuirá um identificador único e uma senha, sendo que esta senha só poderá ser alterada pelo super-usuário, e também poderá participar de mais de um grupo ou ainda não pertencer a nenhum.

Para os usuários definidos no ambiente, está disponível um correio eletrônico. Sendo assim, cada usuário possui uma caixa postal com as mensagens recebidas e enviadas.

5.1.2 Gerência das Estações de Trabalho

As estações de trabalho no ambiente do PROSOFT Cooperativo podem estar em uso ou ociosas. Cada estação terá um nome único e estão disponíveis informações sobre sua utilização, ou seja, quem utilizou uma certa estação em determinado instante.

As operações realizadas sobre as estações estará disponível somente para os super-usuários. Estas operações são, por exemplo:

- Criação de uma estação, através da operação *inclui_estação*;
- Exclusão de uma estação, através da operação *exclui_estação*.

Para utilização das estações estão disponíveis operações de *login* e *logout* dos usuários, como também operações que possibilitam uma auditoria sobre todas as atividades realizadas nas estações.

5.2 Criação de ATOs no Ambiente PROSOFT Cooperativo

Os ATOs cooperativos criados no ambiente devem ser conhecidos por todos, para que possa ser controlado o acesso a estas ferramentas.

Para que o ATO seja usado de forma cooperativa ele deve ser cadastrado no PROSOFT Cooperativo. Isso ocorre da seguinte forma:

1. Criação de um ATO cooperativo através da operação *inclui_ato* (operação que inclui um novo ATO no ambiente), ficando este ATO armazenado em um "banco de ATOs";
2. Para cada operação inserida no ATO deve-se utilizar a operação *inclui_operação_ato*.

Outro fator importante é a forma de definir as operações dos ATOs Cooperativos. Deve-se ter dois cuidados:

- A cada operação definida deve-se ter como último argumento o identificador do usuário, para que possa ter o controle sobre quem realizou cada atividade;
- As operações devem ser classificadas, segundo [MEN 89], em modificadoras ou observadoras. Essa classificação se faz necessário para que possa ocorrer a sincronização das operações modificadoras e permitir o *undo* das modificações realizadas pelo usuário. Essas operações são descritas da seguinte forma:

- Modificadoras – operações que alteram um objeto retornando um novo objeto a partir do objeto passado como argumento;
- Observadoras – operações que extraem informações de objetos que foram passados como parâmetros.

Com essas informações é possível criar novos ATOs (ferramentas) cooperativos. Assim, o PROSOFT Cooperativo intermedia toda a interação do usuário com o ambiente e suas ferramentas, como ilustra a Fig. 5.2.

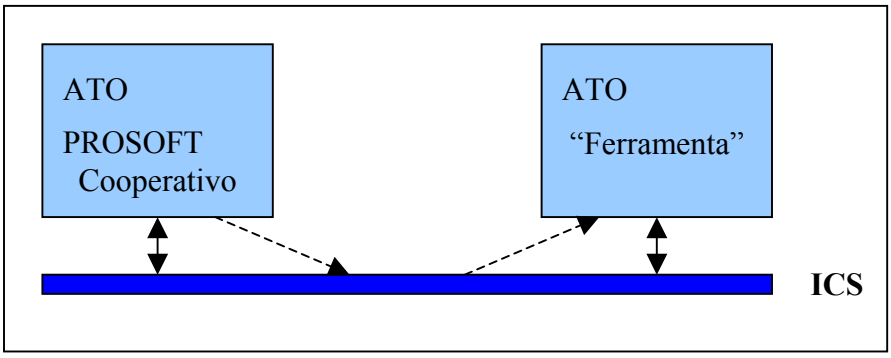


FIGURA 5.2 - PROSOFT Cooperativo intermedia a interação do usuário com os ATOs

5.2.1 Utilização dos Componentes de Percepção

No momento da construção dos ATOs que compõem uma ferramenta, o desenvolvedor já é conhecedor dos Componentes de Percepção que necessita utilizar. Desta forma, pode facilmente inserir no lugar adequado os recursos para percepção e ligá-los através de uma chamada ICS com a ferramenta em desenvolvimento.

Para tanto, é necessário que o desenvolvedor utilize essa metodologia, composta pelos seguintes passos, ilustrado na Fig. 5.3:

1º Passo – Construir a ferramenta sem a utilização das características de percepção das atividades cooperativas.

2º Passo –Após a construção da ferramenta, associar os componentes necessários e o ATO Preferências dos Usuários. Desta forma, para qualquer componente utilizado a representação de um determinado usuário será a mesma e única.

3º Passo – Utilizar as operações que estão especificadas em cada ATO dos Componentes de Percepção e do ATO Preferências dos Usuários.

4º Passo – Especificar as operações que são fornecidas a seguir para que ocorra a coerência entre os Componentes de Percepção e as preferências de cada usuário. Estas operações são apresentadas no ANEXO 2.

5º Passo – Definir a operação Exibe, a qual é responsável pela visualização do sistema utilizando as informações disponíveis pelo ATO.

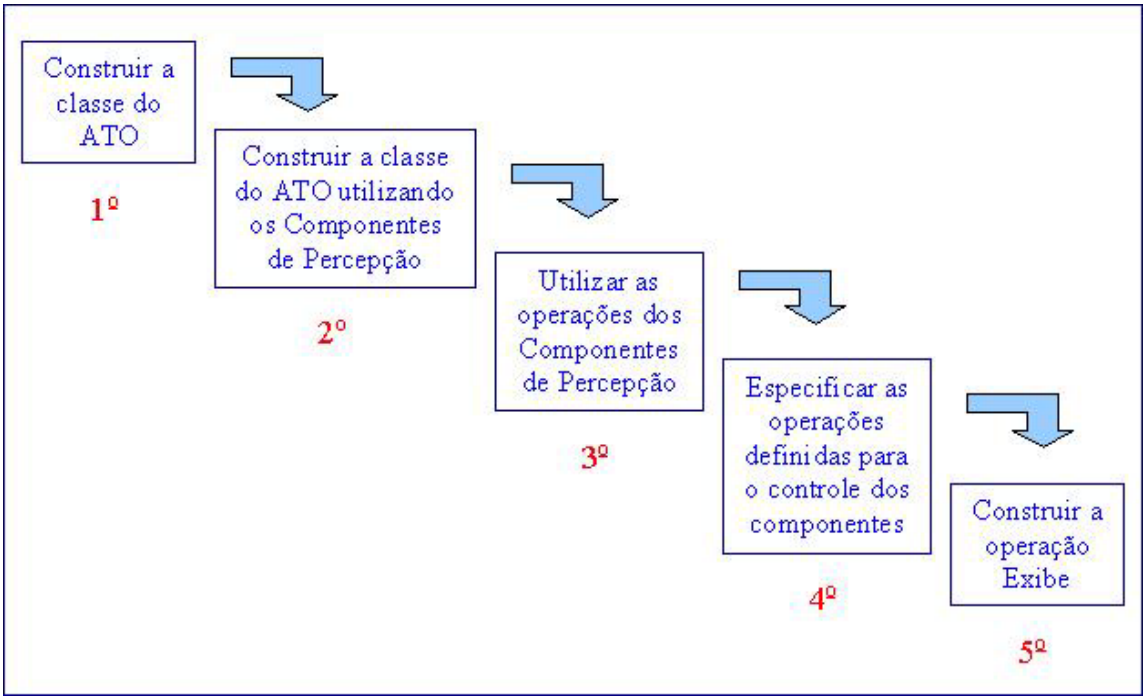


FIGURA 5.3 - Passos para construção de um ATO Cooperativo utilizando Componentes de Percepção

5.3 Comportamento dos Objetos no Espaço de Trabalho Compartilhado

Todo objeto do ambiente possui um grupo de usuários que podem ter acesso a ele formando assim um Espaço de Trabalho Compartilhado. Através da operação *op_obj_coop* é que o usuário atua sobre o objeto, sendo que antes é necessário verificar se esse usuário possui permissão de acesso.

A permissão de acesso é dada de acordo com a categoria a qual o usuário pertence. São elas:

- Criador – é o usuário que cria o objeto e é este usuário o único que pode manipular o objeto enquanto ele permanece no estado instável (como será viso mais adiante);
- Editor – são os usuários que podem modificar um objeto;
- Gerente – possui o controle sobre o objeto, sendo que ele pode alterar o estado do objeto, desfazer alterações feitas sobre o objeto e alterar permissões de acesso. O gerente também atua como moderador da seção cooperativa;

- Visualizador – este usuário tem permissão apenas para visualizar o trabalho, sendo assim somente poderá utilizar as operações observadoras sobre o objeto.

Com relação aos tipos de usuários, as permissões de acesso possuem maior prioridade, sendo que um super-usuário possui a capacidade de gerência sobre qualquer objeto.

O objeto pode evoluir com o tempo, como ilustra a Fig. 5.4, sendo assim ele pode estar em um dos seguintes estados:

- Instável – o objeto só pode ser acessado pelo seu criador e pelo super-usuário;
- Estável – o objeto está pronto para ser compartilhado. Neste momento o criador do objeto possui a função de gerente, sendo então permitido que ele, juntamente com o super-usuário, poça dar permissão de acesso aos usuários e tornar outros usuários também gerentes;
- Consolidado – é quando o objeto não pode mais receber modificações, sendo então disponibilizados para todos os usuários do ambiente. Qualquer gerente do objeto pode trocar seu estado para consolidado.

De acordo com o PROSOFT Distribuído [SCU 94], onde o PROSOFT Cooperativo é baseado, o objeto está localizado no servidor que o disponibiliza para todos os usuários (de acordo com os tipos de usuários e permissões de acesso). Quando um usuário realiza uma operação sobre o objeto, esta operação é enviada para o servidor (onde está localizado o objeto) via ICS e é o servidor que atua sobre o objeto repassando o resultado, também via ICS, para todas as estações e usuários que estão participando do trabalho cooperativo, até que o objeto seja considerado consolidado.

A partir de um objeto consolidado é possível gerar versões, através da operação *cria_versão_obj_coop*. No PROSOFT Cooperativo, o modelo de versões é inspirado no modelo descrito em [GOL 96], permitindo assim versões de objetos no contexto de banco de dados orientados a objetos.

No PROSOFT Cooperativo, diferentemente de [GOL 96], não é possível gerar versões a partir de dois ou mais objetos, sendo então que uma versão possui somente um ancestral.

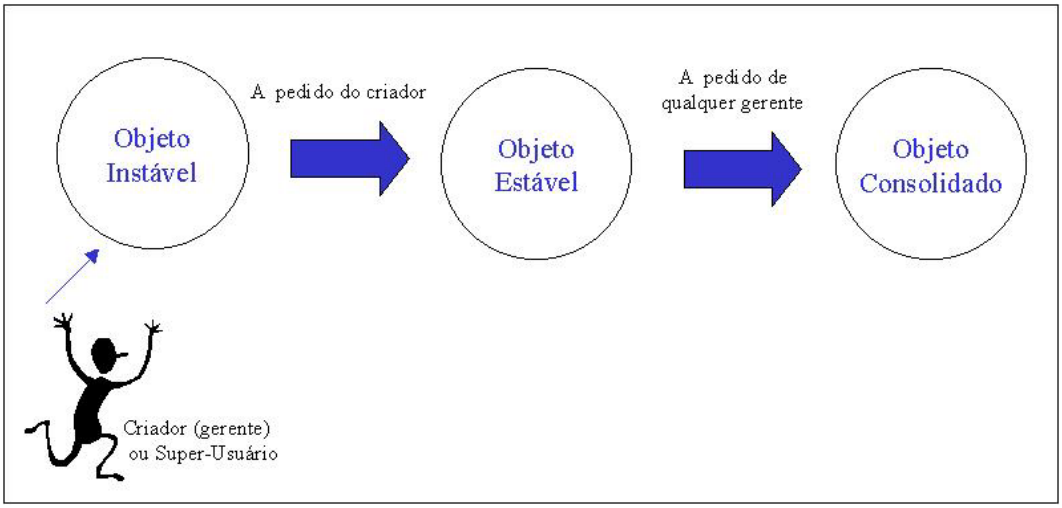


FIGURA 5.4 - Evolução de um objeto dentro do Espaço de Trabalho Compartilhado

5.4 Exemplo de um ATO criado no PROSOFT Cooperativo utilizando Componentes de Percepção

Este trabalho está voltado para os Componentes de Percepção utilizados no desenvolvimento de sistemas cooperativos no PROSOFT Cooperativo. Portanto, o exemplo aqui ilustrado considera que o ambiente já está formado e pronto para a utilização, iniciando então com a construção de um ATO Cooperativo e seguindo os passos mostrados na seção 5.2.1.

Para a exemplificação irá ser construído um *Browser* Cooperativo, o qual foi baseado em [GRE 96] [RAP 99]. Este *browser* permite a um grupo de pessoas compartilhar visualmente e navegar conjuntamente através de páginas *web*. Assim, um grupo de usuários pode visualizar a mesma página HTML onde as janelas dos seus *browser* se tornam um espaço compartilhado.

Este *browser* possui como Componentes de Percepção:

- Barra de Rolagem de Sincronização – permitindo aos usuários se localizarem na página onde as outras pessoas estão visualizando;
- Teleponteiros – permitindo que os usuários chamem a atenção para uma determinada parte do texto e
- Bloco de Notas – permitindo que os participantes possuam um forma de comunicação.

A construção de um sistema *groupware* no Ambiente PROSOFT Cooperativo utilizando os Componentes de Percepção segue 4 passos, por isso, o desenvolvimento do *Browser* Cooperativo aqui proposto segue da seguinte forma:

1º Passo – construção da classe do ATO

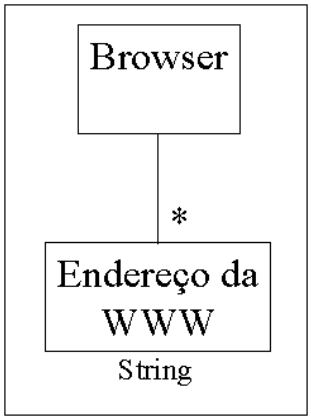


Figura 5.5 - Instanciação do ATO *Browser*

2º Passo – construção da classe do ATO Cooperativo utilizando os Componentes de Percepção

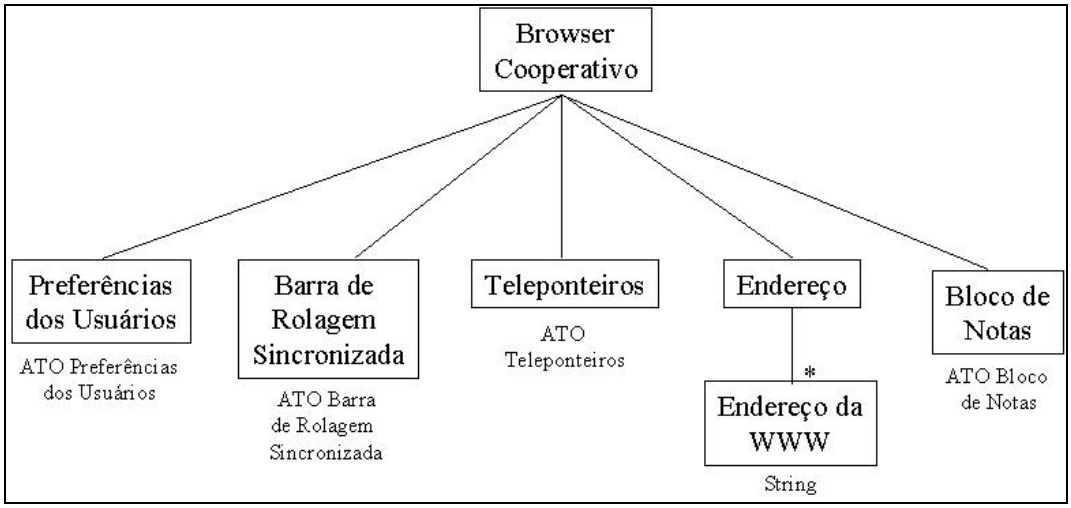


FIGURA 5.6 - Instanciação do ATO *Browser* Cooperativo

3º Passo – uso das operações dos ATOs de Percepção utilizados

As operações que estão disponíveis para serem utilizadas dos Componentes de Percepção são as seguintes, mostradas na Tab. 5.1:

TABELA 5.1 - Operações disponíveis para os Componentes de Percepção utilizados no *Browser* Cooperativo

Componentes	**Operações**
Preferências dos	*Existe_usuário, representação_usuário, ícone_usuário,*

Usuários	*existe_ícone, existe_label, existe_cor, inclui_usuário_label, inclui_usuário_cor, exclui_usuário, altera_cor, altera_label, altera_ícone*
Barra de Rolagem de Sincronização	*Existe_usuário, posição_barra_usuário, inclui_usuário, exclui_usuário, movimenta_barra*
Teleponteiros	*Posição_teleponteiro, existe_operação, formato_operação, características_click, existe_formato, existe_usuário, existe_click, movimenta_teleponteiro, define_formato_operação, altera_formato_operação, inclui_click, exclui_click, inclui_usuário, exclui_usuário*
Bloco de Notas	*Usuário_enviou_mensagem, envia_mensagem*

4º Passo – especificar as operações definidas para o controle dos componentes (ANEXO 2)

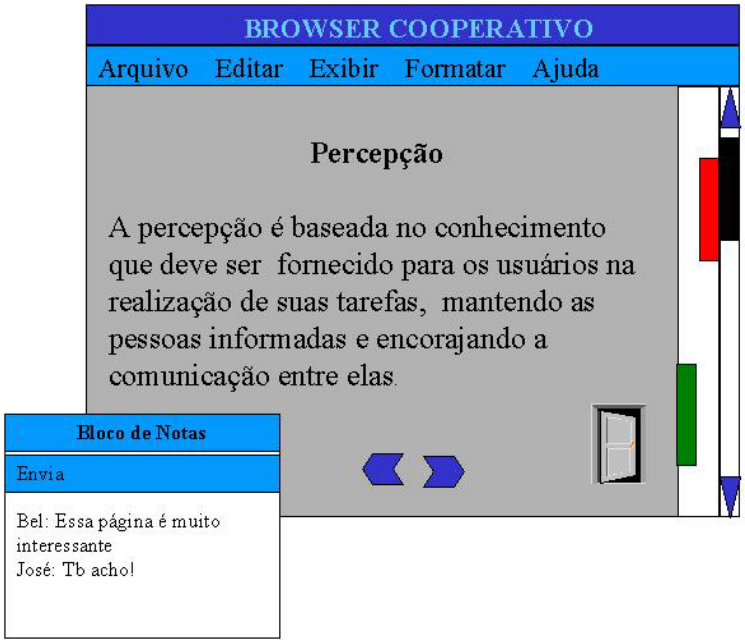
As operações definidas para a interação dos Componentes de Percepção utilizados no *Browser* Cooperativo estão descritas no ANEXO 2 deste trabalho. Porém, para exemplificação, a operação "*inclui_usuário*" está especificada como segue:

inclui_usuário_cor ((Preferências-Usuários pu, Barra-Rol-Sinc brs, Teleponteiros tp, _, _),
id-usuário, cor, ícone, posição_barra, posição_tp) =
if **ICS** (PU, *existe_usuário*, pu, <id-usuário>)
then **ICS** (BRS, *inclui_usuário*, brs, <id-usuário, posição-barra>) **and**
ICS (TP, *inclui_usuário*, tp, <id-usuário, posição-tp>)
else **if** **not ICS** (PU, *existe_cor*, pu, <cor>) **and**
not ICS (PU, *existe_ícone*, pu, <ícone>) **and**
not ICS (TP, *existe_formato*, tp, <ícone>)
then **ICS** (PU, *inclui_usuário_cor*, pu, <id-usuário, cor, ícone>) **and**
ICS (BRS, *inclui_usuário*, brs, <id-usuário, posição-barra>) **and**
ICS (TP, *inclui_usuário*, tp, <id-usuário, posição-tp>)
else (Preferências-Usuários pu, Barra-Rol-Sinc brs, Teleponteiros tp, _, _)

5º Passo – construir a operação Exibe

A operação Exibe, que fica a cargo do desenvolvedor, é responsável pela visualização do sistema que está sendo criado. Esta operação utiliza as informações disponíveis no ATO ou ATOs que formam o sistema.

Para o exemplo do *Browser* Cooperativo, pode-se visualizar o resultado da operação Exibe como mostra a Fig. 5.7



FIGURS 5.7 - *Browser* Cooperativo

Este capítulo apresenta a especificação formal dos componentes de percepção propostos no Capítulo 4 para o Ambiente PROSOFT Cooperativo.

Os componentes de percepção aqui apresentados são formados por 7 ATOs, correspondentes a cada componente, e pelo ATO Preferências dos Usuários que fornece o controle das preferências que os usuários podem assumir durante o decorrer do trabalho que estiverem realizando. Esses ATOs são apresentados na Fig. 6.1.

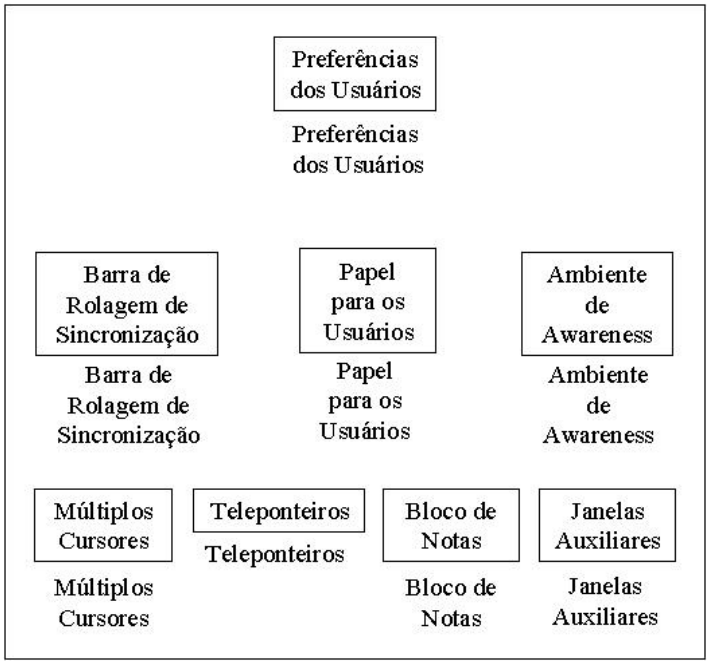


FIGURA 6.1 - Componentes de Percepção

A notação usada nesta especificação segue as características adotas por [MOR 97] [REI 98]: quando uma operação do mesmo ATO é chamada por outra operação, sem o uso de ICS, esta chamada é apresentada em itálico e as palavras reservadas (**and**, **or**, **not**, **if**, **then** e **else**) são apresentadas em negrito.

As operações principais serão vistas nas sessões a seguir, sendo que as operações auxiliares (chamadas Operações Internas) estão no Anexo 1. As operações internas são operações apenas acessíveis às operações aqui presentes, não sendo possível um usuário utilizá-las.

6.1 ATO Preferências dos Usuários

Este ATO é responsável pelo gerenciamento das preferências dos usuários que estão participando de uma atividade cooperativa. Sempre que uma ferramenta construída no PROSOFT Cooperativo utilizar pelo menos um dos componentes de percepção aqui apresentados, se faz necessário a utilização desse ATO. Assim é possível conhecer as preferências de cada usuário e controlar o uso de cores, labels e ícones, pois cada usuário possui uma representação única.

Também como os usuários, esse ATO permite o controle de ícones utilizados nos componentes Ambiente de Percepção, Papel para os Usuários e Teleponteiros onde também são usados para representação de outras funções. Por exemplo, no ATO Teleponteiros as operações realizadas pelo ponteiro podem fazer com que ele assuma um formato diferente, em um operação em que o usuário apaga uma figura o teleponteiro pode assumir o formato de uma borracha.

A Fig. 6.2 apresenta a instanciação do ATO Preferências dos Usuários.

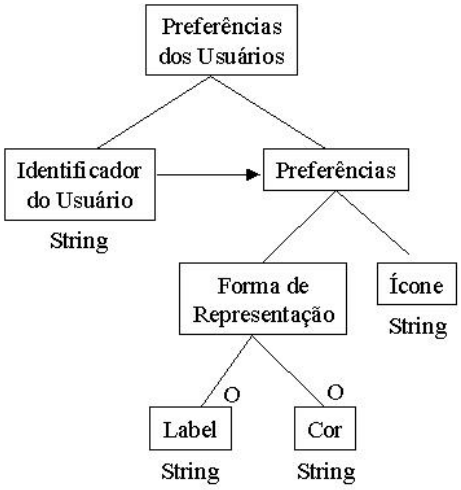


FIGURA 6.2 - Instanciação do ATO Preferências dos Usuários

6.1.1 Operação de criação

A operação de criação não necessita de argumentos pois a medida que os usuários são inseridos na ferramenta que utilizar este ATO, eles serão cadastrados com suas preferências. Sendo assim, é criado um mapeamento vazio para futuras inserções de informações.

Interface:

cria () : → PU

Operação

cria = (Preferências-Usuários empty-mapping)

6.1.2 Operações Observadoras

As operações observadoras pertencentes a essa classe são:

- *existe_usuário* – operação que verifica se um determinado usuário (seu identificador) está inserido no ambiente de percepção;
- *representação_usuário* – esta operação retorna a forma de representação usada por um determinado usuário;
- *ícone_usuário* – retorna o ícone pelo qual um determinado usuário é representado;
- *existe_ícone* – verifica se um determinado ícone já pertence a algum usuário;
- *existe_label* – verifica se um determinado label já pertence a algum usuário;
- *existe_cor* – verifica se uma determinada cor já pertence a algum usuário.

Interface

existe_usuário (_,_): PU, STRING → BOOL
representação_usuário (_,_): PU, STRING → STRING
ícone_usuário (_,_): PU, STRING → STRING
existe_ícone (_,_):PU, STRING → BOOL
existe_label (_,_):PU, STRING → BOOL
existe_cor (_,_):PU, STRING → BOOL

Operações

Variáveis formais: id-usuário: STRING
pu: PU

existe_usuário ((Preferências-Usuários empty-mapping), _) =
FALSE

existe_usuário (pu, id-usuário) =
if id-usuário ∈ dom (pu)
then TRUE

else FALSE

representação_usuário (pu, id-usuário) =
if *existe_usuário* (pu, id-usuário)
then *representação_usuário_aux* (pu, id-usuário)
else "Não_existe_usuário"

ícone_usuário (pu, id-usuário) =
if existe_usuário (pu, id-usuário)
then ícone_usuário_aux (pu, id-usuário)
else "Não_existe_usuário"

existe_ícone (pu, ícone) =
if existe_ícone_aux (pu, ícone)
then TRUE
else FALSE

existe_label (pu, label) =
if existe_label_aux (pu, label)
then TRUE
else FALSE

existe_cor (pu, cor) =
if existe_cor_aux (pu, cor)
then TRUE
else FALSE

6.1.3 Operações Modificadoras

As operações modificadoras que pertencem a essa classe são:

- *inclui_usuário_label* – esta operação inclui um usuário e sua preferência de representação sendo por label;
- *inclui_usuário_cor* – esta operação inclui um usuário e sua preferência de representação sendo por cor;
- *exclui_usuário* – esta operação exclui um usuário e suas preferências;
- *altera_cor* – altera a cor de representação de um usuário;
- *altera_label* – altera o label de representação de um usuário;
- *altera ícone* – altera o ícone representativo de um usuário.

Interface

inclui_usuário_label (_,_,_,_): PU, STRING, STRING, STRING → PU

inclui_usuário_cor (_,_,_,_): PU, STRING, STRING, STRING → PU
exclui_usuário (_,_,_,_): PU, STRING → PU
altera_cor (_,_,_):PU, STRING, STRING → PU
altera_label (_,_,_):PU, STRING, STRING → PU
altera_ícone (_,_,_):PU, STRING, STRING → PU

Operações

Variáveis formais: ..... id-usuário, label, ícone, cor: STRING
novo-label, novo-ícone, nova-cor: STRING
pu: PU

inclui_usuário_label ((Preferências-Usuário empty-mapping), id-usuário, label, ícone) =
inclui_usuário_label_aux ((Preferências-Usuário empty-mapping), id-usuário, label, ícone)

inclui_usuário_label (pu, id-usuário, label, ícone) =
if **not** *existe_usuário* (pu, id-usuário) **and**
not *existe_ícone* (pu, ícone) **and**
not *existe_label* (pu, label)
then *inclui_usuário_label_aux* (pu, id-usuário, label, ícone)
else pu

inclui_usuário_cor ((Preferências-Usuário empty-mapping), id-usuário, cor, ícone) =
inclui_usuário_cor_aux ((Preferências-Usuário empty-mapping), id-usuário, cor, ícone)

inclui_usuário_cor (pu, id-usuário, cor, ícone) =
if **not** *existe_usuário* (pu, id-usuário) **and**
not *existe_ícone* (pu, ícone) **and**
not *existe_cor* (pu, cor)
then *inclui_usuário_cor_aux* (pu, id-usuário, cor, ícone)
else pu

exclui_usuário ((Preferências-Usuário empty-mapping), _) =
(Preferências-Usuário empty-mapping)

exclui_usuário (pu, id-usuário) =
if *existe_usuário* (pu, id-usuário)
then *exclui_usuário_aux* (pu, id-usuário)
else pu

altera_cor (pu, id-usuário, nova-cor) =
if *existe_usuário* (pu, id-usuário) **and**

```
            not existe_cor (pu, nova-cor)
        then  altera_cor_aux (pu, id-usuário, nova-cor)
        else  pu
altera_label (pu, id-usuário, novo-label) =
        if    existe_usuário (pu, id-usuário) and
              not existe_label (pu, novo-label)
        then  altera_label_aux (pu, id-usuário, novo-label)
        else  pu

altera_ícone (pu, id-usuário, novo-ícone) =
        if    existe_usuário (pu, id-usuário) and
              not existe_ícone (pu, novo-ícone)
        then  altera_ícone_aux (pu, id-usuário, novo-ícone)
        else  pu
```

6.2 ATO Múltiplos Cursores

Este ATO permite a criação de cursores individuais para cada usuário participante de uma atividade cooperativa. Assim é possível visualizar as tarefas que os outros usuários estão realizando a partir da localização dos seus cursores.

A instanciação do ATO Múltiplos Cursores é vista na Fig. 6.3.

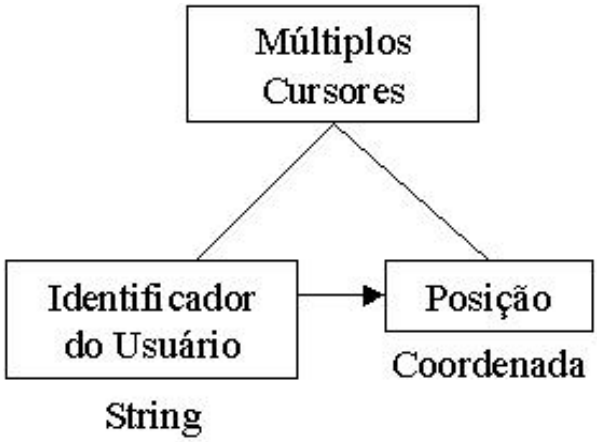


FIGURA 6.3 - Instanciação do ATO Múltiplos Cursores

6.2.1 Operação de criação

Esta operação permite a criação de um conjunto de cursores e suas posições, formando assim um mapeamento.

Interface

cria (): → MC

Operação

cria = (Múltiplos-Cursores empty-mapping)

6.2.2 Operações Observadoras

As operações observadoras definidas para esta classe são:

- *posição_cursor* – retorna a posição atual do cursor de um determinado usuário;
- *existe_usuário* – esta operação verifica se um determinado usuário possui um múltiplo cursor.

Interface

posição_cursor (_,_):MC, STRING → COORDENADA
existe_usuário (_,_): MC, STRING → BOOL

Operações

Variáveis formais: ..... múltiplos-cursores: MAPEAMENTO
id-usuário: STRING

posiçao_cursor ((Múltiplos-Cursores múltiplos-cursores), id-usuário) =
if *existe_usuário* ((Múltiplos-Cursores múltiplos-cursores), id-usuário)
then *posição_cursor_aux* ((Múltiplos-Cursores múltiplos-cursores), id-usuário)
else *retorna_posição_nula*

existe_usuário ((Múltiplos-Cursores empty-mapping), _) =
FALSE

existe_usuário ((Múltiplos-Cursores múltiplos-cursores), id-usuário) =
if *existe_usuário_aux* (múltiplos-cursores, id-usuário)
then TRUE
else FALSE

6.2.3 Operações Modificadoras

As operações definidas para esta classe são:

- *movimenta_cursor* – movimenta o cursor de um determinado usuário de uma coordenada para outra;
- *inclui_usuário* – inclui um usuário e a posição do seu cursor;
- *exclui_usuário* – exclui um usuário e o seu cursor.

Interface

movimenta_cursor (_,_,_): MC, STRING, COORDENADA → MC
inclui_usuário (_,_,_): MC, STRING, COORDENADA → MC
exclui_usuário (_,_,_): MC, STRING → MC

Operações

Variáveis formais: ..... id-usuário: STRING
nova-posição, posição: COORDENADA
múltiplos-cursores: MAPEAMENTO

movimenta_cursor ((Múltiplos-Cursores múltiplos-cursores), id-usuário, nova-posição) =
if *existe_usuário_aux* (múltiplos-cursores, id-usuário)
then *movimenta_cursor_aux* ((Múltiplos-Cursores múltiplos-cursores), id-usuário, nova-posição)
else (Múltiplos-Cursores múltiplos-cursores)

inclui_usuário ((Múltiplos-Cursores empty-mapping), id-usuário, posição) =
(Múltiplos-Cursores *inclui_usuário_aux* (id-usuário, posição, empty-mapping))

inclui_usuário ((Múltiplos-Cursores múltiplos-cursores), id-usuário, posição) =
if **not** *existe_usuário* ((Múltiplos-Cursores múltiplos-cursores), id-usuário)
then (Múltiplos-Cursores *inclui_usuário_aux* (id-usuário, posição, múltiplos-cursores))
else (Múltiplos-Cursores múltiplos-cursores)

exclui_usuário ((Múltiplos-Cursores empty-mapping), id-usuário) =

(Múltiplos-Cursores empty-mapping)

exclui_usuário ((Múltiplos-Cursores múltiplos-cursores), id-usuário) =
if *existe_usuário* ((Múltiplos-Cursores múltiplos-cursores), id-usuário)
then (Múltiplos-Cursores *exclui_usuário_aux* (id-usuário, múltiplos-cursores))
else (Múltiplos-Cursores múltiplos-cursores)

6.3 ATO Teleponteiros

O ATO Teleponteiros permite a criação de teleponteiros para os usuários que estão participando de uma atividade cooperativa. Assim como os múltiplos cursores, servem para a melhor localização e visualização da tarefa que estão realizando em relação ao grupo.

A instanciação deste ATO é visto na Fig. 6.4.

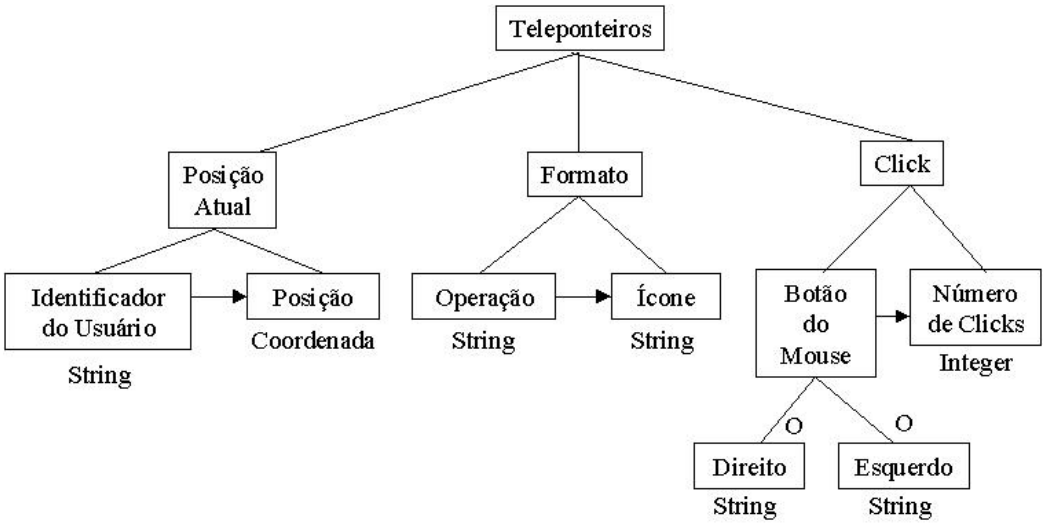


FIGURA 6.4 - Instanciação do ATO Teleponteiros

6.3.1 Operação de criação

Esta operação permite a criação de um conjunto de teleponteiros, onde cada usuário possui um único teleponteiro. Também é permitido que para cada operação que o teleponteiro realize, ele obtenha um formato diferente. Além disso, através da formatação dos clicks do mouse é possível verificar a posição que o teleponteiro está e portanto saber se está em cima de um determinado botão.

Interface

cria (): → TP

Operação

cria = (Posição-Atual empty-mapping, Formato empty-mapping, Click empty-mapping)

6.3.2 Operações Observadoras

As operações observadoras definidas para esta classe são:

- *posição_teleponteiro* – retorna a posição atual do teleponteiro de um determinado usuário;
- *existe_operação* – verifica se os teleponteiros possuem uma representação para uma determinada operação;
- *formato_operação* – esta operação retorna a forma que o teleponteiro assume quando realiza uma determinada operação;
- *características_click* – retorna as características atuais do click;
- *existe_formato* – verifica se já existe um formato para alguma operação;
- *existe_usuário* – verifica se um determinado usuário possui um teleponteiro;
- *existe_click* – esta operação verifica se um determinado formato de click já existe.

Interface

posição_teleponteiro (_,_): TP, STRING → COORDENADA
existe_operação (_,_): TP, STRING → BOOL
formato_operação (_,_): TP, STRING → STRING
caracterísitcas_click (_):TP, STRING → MAPEAMENTO
existe_formato (_,_): TP, STRING → BOOL
existe_usuário (_,_): TP, STRING → BOOL
existe_click (_,_,_):TP, STRING, INTEGER → BOOL

Operações

Variáveis formais: ..... posição, formato: MAPEAMENTO
id-usuário, operação, ícone, botão: STRING
número: INTEGER

posiçao_teleponteiro ((Posição-Atual posição, _, _), id-usuário) =

if *existe_usuário* ((Posição-Atual posição, _, _), id-usuário)
then *posição_teleponteiro_aux* ((Posição-Atual posição, _, _), id-usuário)
else *retorna_posição_nula_tele*

existe_operação ((_, Formato empty-mapping, _), _, _) =
FALSE

existe_operação ((_, Formato formato, _), operação) =
if *existe_operação_aux* (formato, operação)
then TRUE
else FALSE
formato_operação ((_, Formato empty-mapping, _), _) =
"Não_existe_operação"

formato_operação ((_, Formato formato, _), operação) =
if *existe_operação* ((_, Formato formato, _), operação)
then *formato_operação_aux* (formato, operação)
else "Não_existe_operação_ou_usuário"

características_click ((_, _, Click empty-mapping)) =
empty-mapping

características_click ((_, _, Click click)) =
click

existe_formato ((_, Formato formato, _), ícone) =
if ícone ∈ rng (formato)
then TRUE
else FALSE

existe_usuário ((Posição-Atual posição, _, _), id-usuário) =
if id-usuário ∈ dom (posição)
then TRUE
else FALSE

existe_click ((_, _, Click empty-mapping), _, _) =
FALSE

existe_click ((_, _, Click click), botão, número) =
if *existe_click_aux* (click, botão, número)
then TRUE
else FALSE

6.3.3 Operações Modificadoras

As operações modificadoras definidas para esta classe são:

- *movimenta_teleponteiro* – movimenta o teleponteiro de um determinado usuário de uma posição para outra;
- *define_formato_operação* – define qual a representação (ícone) que o teleponteiro terá quando for realizar uma determinada operação;
- *altera_formato_operação* – altera a forma de representação (ícone) que o teleponteiro terá quando for realizar uma determinada operação;
- *inclui_click* – inclui como um formato de click, ou seja, qual o botão do mouse e quantos clicks foram efetuados;
- *exclui_click* – exclui um formato de click definido;
- *inclui_usuário* – inclui um usuário e a posição do seu teleponteiro;
- *exclui_usuário* – exclui um usuário e o seu teleponteiro.

Interface

movimenta_teleponteiro (_,_,_):TP, STRING, COORDENADA → TP
define_formato_operação (_,_,_):TP, STRING, STRING → TP
altera_formato_operação (_,_,_):TP, STRING, STRING → TP
inclui_click (_,_,_):TP, STRING, INTEGER → TP
exclui_ click (_,_,_):TP, STRING, INTEGER → TP
inclui_teleponteiro (_,_,_):TP, STRING, COORDENADA → TP
exclui_teleponteiro (_,_):TP, STRING → TP

Operações

Variáveis formais: ..... id-usuário, operação, ícone, novo-ícone, botão: STRING
nova-posição, posição-usuário: COORDENADA
posição, formato, click: MAPEAMENTO
número: INTEGER

movimenta_teleponteiro ((Posição-Atual posição,_,_), id-usuário, nova-posição) =
if *existe_usuário* ((Posição-Atual posição, _, _), id-usuário)
then *movimenta_teleponteiro_aux* ((Posição-Atual posição,_,_), id-usuário, nova-posição)
else (Posição-Atual posição,_,_)

define_formato_operação ((_, Formato empty-mapping, _), operação, ícone) =
(_, Formato *define_formato_operação_aux* (operação, ícone, empty-mapping),_)

define_formato_operação ((_, Formato formato, _), operação, ícone) =
if **not** *existe_formato* ((_, Formato formato, _), ícone) **and**

not *existe_operação* ((_, Formato formato, _), operação)
then (_, Formato *define_formato_operação_aux* (operação, ícone, formato), _)
else (_, Formato formato, _)

altera_formato_operação ((_, Formato formato, _), operação, novo-ícone) =
if **not** *existe_formato* (formato, ícone) **and**
existe_operação ((_, Formato formato, _), operação)
then (_, Formato *altera_operação_formato_aux* (formato, operação, novo-ícone),_)
else (_, Formato formato, _)

inclui_click ((_, _, Click empty-mapping), botão, número) =
if (botão = "Esquerdo" **or** botão = "Direito") **and**
(número = 1 **or** número = 2)
then (_, _, Click modify (botão, número, empty-mapping))
else (_, _, Click empty-mapping)

inclui_click ((_, _, Click click), botão, número) =
if **not** *existe_click* (click, botão, número)
then (_, _, Click modify (botão, número, click))
else (_, _, Click click)

exclui_click ((_, _, Click empty-mapping), botão, número) =
(_, _, Click empty-mapping)

exclui_click ((_, _, Click click), botão, número) =
if *existe_click* (click, botão, número)
then (_, _, Click *exclui_click_aux* (click, botão, número))
else (_, _, Click click)

inclui_teleponteiro ((Posição-Atual empty-mapping,_,_), id-usuário, posição-usuário) =
(Posição-Atual *inclui_usuário_aux* (id-usuário, posição-usuário, empty-mapping), _,_)

inclui_teleponteiro ((Posição-Atual posição,_,_), id-usuário, posição-usuário) =
if **not** *existe_usuário* ((Posição-Atual posição,_,_), id-usuário)
then (Posição-Atual *inclui_usuário_aux* (id-usuário, posição-usuário, posição), _,_)
else (Posição-Atual posição,_,_)

exclui_teleponteiro ((Posição-Atual empty-mapping,_,_), _) =
(Posição-Atual empty-mapping), _, _)

exclui_teleponteiro ((Posição-Atual posição,_,_), id-usuário) =
if *existe_usuário* ((Posição-Atual posição,_,_), id-usuário)

then (Posição-Atual *exclui_usuário_aux* (id-usuário, posição), _,_)

else (Posição-Atual posição,_,_)

6.4 ATO Ambiente de Percepção

Este ATO permite a criação de um Ambiente de Percepção para auxiliar aos usuários no momento que estiverem desenvolvendo uma atividade cooperativa, assim é possível identificar quais participantes estão prontos para começar uma determinada atividade ou simplesmente para um encontro social.

A instanciação desse ATO é vista na Fig. 6.5

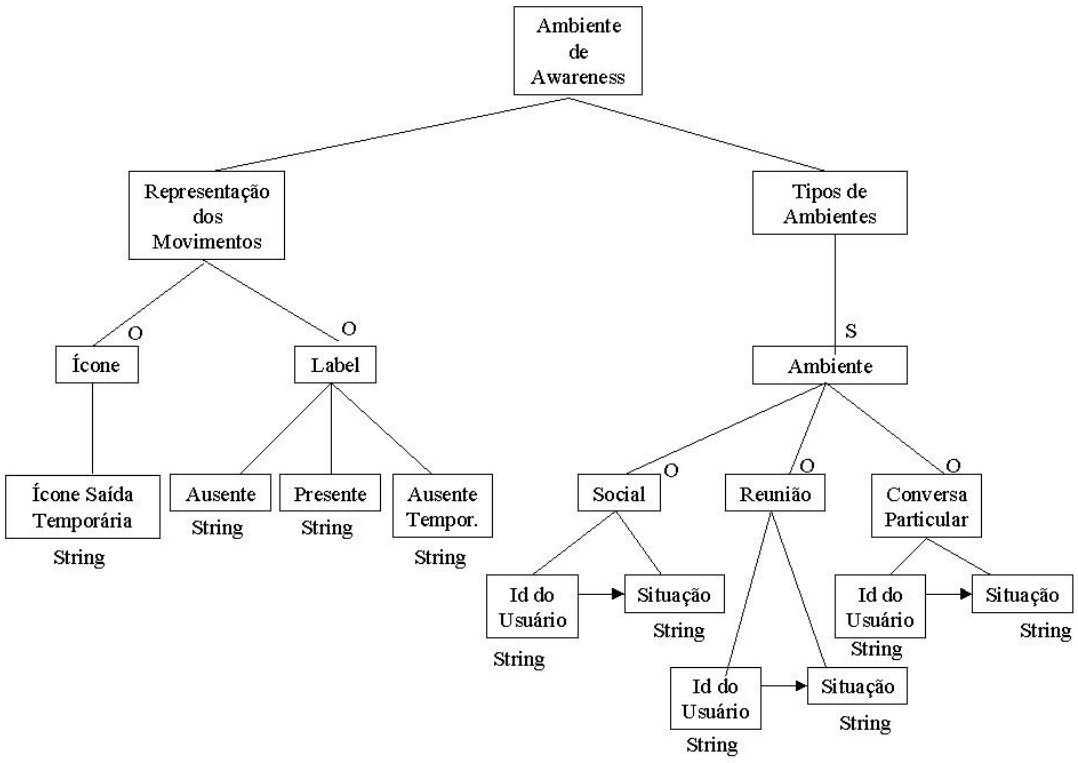


FIGURA 6.5 - Instanciação do ATO Ambiente de Percepção

6.4.1 Operação de criação

Esta operação cria o Ambiente de Percepção. Da mesma forma que os demais componentes, essa operação cria um ambiente vazio, para posterior preenchimento com os usuários que irão efetivamente participar de uma atividade cooperativa.

Interface

cria () :→ AA

Operação

cria = (Representação-Movimentos _, Tipos-Ambientes empty-set)

6.4.2 Operações Observadoras

As operações observadoras pertencentes a essa classe são:

- *quais_tipos_ambientes* – esta operação fornece os tipos de ambientes que formam o Ambiente de Percepção;
- *usuário_pertence_ambiente* – verifica se um determinado usuário está presente em um ambiente;
- *tipo_representação_movimento* – devolve qual a forma de representar o movimento (entrada e saída de usuários) no Ambiente de Percepção;
- *existe_ambiente* – verifica se um ambiente pertence ao Ambiente de Percepção;
- *ícone_saída_temp* – esta operação devolve o ícone que representa a saída temporária de um usuário em um ambiente.

Interface

quais_tipos_ambientes (_): AA → CONJUNTO
usuário_pertence_ambiente (_,_,_): AA, STRING, MAPEAMENTO → BOOL
tipo_representação_movimento (_): AA → STRING
existe_ambiente (_,_): AA, CONJUNTO → BOOL
ícone_saída_temp (_): AA → STRING

Operações

Variáveis formais: ..... tipos-ambientes: CONJUNTO
id-usuário: STRING

quais_tipos_ambientes ((_, Tipos-Ambientes empty-set)) =
empty-set

quais_tipos_ambientes ((_, Tipos-Ambientes tipos-ambientes)) =
    tipos-ambientes

usuário_pertence_ambiente ((_, Tipos-Ambientes empty-set), _,_) =
    FALSE

usuário_pertence_ambiente ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Social) =
    if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Social)
    then **if** id-usuário ∈ dom (Social)
        then TRUE
        else FALSE
    else FALSE

usuário_pertence_ambiente ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Reunião) =
    if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Reunião)
    then **if** id-usuário ∈ dom (Reunião)
        then TRUE
        else FALSE
    else FALSE

usuário_pertence_ambiente ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Conversa-Particular) =
    if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Conversa-Particular)
    then **if** id-usuário ∈ dom (Conversa-Particular)
        then TRUE
        else FALSE
    else FALSE

tipo_representação_movimento ((Representação-Movimento Ícone _, _)) =
    "Ícone"

tipo_representação_movimento ((Representação-Movimento Label _, _)) =
    "Label"

existe_ambiente ((_, Tipos-Ambientes tipos-ambientes), Social) =
    if Social ∈ tipos-ambientes
    then TRUE
    else FALSE

existe_ambiente ((_, Tipos-Ambientes tipos-ambientes), Reunião) =
    if Reunião ∈ tipos-ambientes
    then TRUE
    else FALSE

existe_ambiente ((_, Tipos-Ambientes tipos-ambientes), Conversa-Particular) =

if Conversa-Particular ∈ tipos-ambientes
then TRUE
else FALSE

ícone_saída_temporária ((Representação-Movimento Label _, _)) =
"Representaçao-por-Label"

ícone_saída_temporária ((Representação-Movimento Ícone _, _)) =
"Não-definido"

ícone_saída_temporária ((Representação-Movimento Ícone ícone-saída-temp, _)) =
ícone-saída-temp

6.4.3 Operações Modificadoras

As operações modificadoras pertencentes a essa classe são:

- *define_representação_movimento_ícone* – esta função define que os movimentos dos usuários (entrada e saída do ambiente) vão ser representados por ícones;
- *define_representação_movimento_label* – esta função define que os movimentos dos usuários (entrada e saída do ambiente) vão ser representados por labels significativos;
- *altera_representação_movimento* – esta função altera o tipo de movimentos dos usuários (entrada e saída do ambiente);
- *inclui_tipo_ambiente* – inclui um tipo de ambiente (social, reunião ou conversa particular) no Ambiente de Percepção;
- *exclui_tipo_ambiente* – exclui um tipo de ambiente (social, reunião ou conversa particular) do Ambiente de Percepção;
- *entrada_usuário_ambiente* – esta operação permite que um usuário esteja presente em um determinado tipo de ambiente do Ambiente de Percepção;
- *saída_usuário_ambiente* – esta operação permite que um usuário se ausente de um determinado tipo de ambiente do Ambiente de Percepção;
- *altera_ícone_saída_temp* – esta operação altera o ícone que é usado para representar a saída temporária de um usuário de um determinado ambiente;
- *saída_temp_usuário_ambiente* – esta operação permite que um usuário se ausente temporariamente de um determinado tipo de ambiente do Ambiente de Percepção;

Interface

define_representação_movimento_ícone (_,_):AA, STRING → AA
define_representação_movimento_label (_,_,_,_):
AA, STRING, STRING, STRING → AA

altera_ representação_movimento (_,_):AA, STRING → AA
inclui_tipo_ambiente (_,_):AA, MAPEAMENTO → AA
exclui_tipo_ambiente (_,_):AA, MAPEAMENTO → AA
entrada_usuário_ambiente (_,_,_): AA, STRING, MAPEAMENTO → AA
saída_usuário_ambiente (_,_,_): AA, STRING, MAPEAMENTO → AA
altera_ícone_saída_temp (_,_):AA, STRING → AA
saída_temp_usuário_ambiente (_,_,_): AA, STRING, MAPEAMENTO → AA

Operações

Variáveis formais: ..... ícone-saída-temp, ausente, presente, ausente-temp: STRING
id-usuário: STRING
tipos-ambientes: CONJUNTO

define_representação_movimento_ícone ((Representação-Movimentos _, _), ícone-saída-temp) =
(Representação-Movimentos Ícone ícone-saída-temp, _)

define_representação_movimento_label ((Representação-Movimentos _, _), ausente, presente, ausente-temp) =
(Representação-Movimentos Label (Ausente ausente, Presente presente, Ausente-Temp ausente-temp), _)

altera_representação_movimento ((Representação-Movimentos _, _), "Ícone") =
(Representação-Movimentos Ícone _, _)

altera_representação_movimento ((Representação-Movimentos _, _), "Label") =
(Representação-Movimentos Label _, _)

inclui_tipo_ambiente ((_, Tipos-Ambientes empty-set), Social) =
(_, Tipos-Ambientes *inclui_tipo_ambiente_aux* (empty-set, Social))

inclui_tipo_ambiente ((_, Tipos-Ambientes tipos-ambientes), Social) =
if **not** *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Social)
then (_, Tipos-Ambientes *inclui_tipo_ambiente_aux* (tipos-ambientes, Social))
else (_, Tipos-Ambientes tipos-ambientes)

inclui_tipo_ambiente ((_, Tipos-Ambientes empty-set), Reunião) =
(_, Tipos-Ambientes *inclui_tipo_ambiente_aux* (empty-set, Reunião))

inclui_tipo_ambiente ((_, Tipos-Ambientes tipos-ambientes), Reunião) =
if **not** *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Reunião)
then (_, Tipos-Ambientes *inclui_tipo_ambiente_aux* (tipos-ambientes, Reunião)),
else (_, Tipos-Ambientes tipos-ambientes)

inclui_tipo_ambiente ((_, Tipos-Ambientes empty-set), Conversa-Particular) =

(_, Tipos-Ambientes *inclui_tipo_ambiente_aux* (empty-set, Conversa-Particular))

inclui_tipo_ambiente ((_, Tipos-Ambientes tipos-ambientes), Conversa-Particular) =
if **not** *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Conversa-Particular)
then (_, Tipos-Ambientes *inclui_tipo_ambiente_aux* (tipos-ambientes, Conversa-Particular)),
else (_, Tipos-Ambientes tipos-ambientes)

exclui_tipo_ambiente ((_, Tipos-Ambientes empty-set), _) =
(_, Tipos-Ambientes empty-set)

exclui_tipo_ambiente ((_, Tipos-Ambientes tipos-ambientes), Social) =
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Social)
then (_, Tipos-Ambientes *exclui_tipo_ambiente_aux* (tipos-ambientes, Social))
else (_, Tipos-Ambientes tipos-ambientes)

exclui_tipo_ambiente ((_, Tipos-Ambientes tipos-ambientes), Reunião) =
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Reunião)
then (_, Tipos-Ambientes *exclui_tipo_ambiente_aux* (tipos-ambientes, Reunião))
else (_, Tipos-Ambientes tipos-ambientes)

exclui_tipo_ambiente ((_, Tipos-Ambientes tipos-ambientes), Conversa-Particular) =
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Conversa-Particular)
then (_, Tipos-Ambientes *exclui_tipo_ambiente_aux* (tipos-ambientes, Conversa-Particular))
else (_, Tipos-Ambientes tipos-ambientes)

entrada_usuário_ambiente ((_, Tipos-Ambientes empty-set), _, _) =
(_, Tipos-Ambientes empty-set)

entrada_usuário_ambiente ((_,Tipos-Ambientes tipos-ambientes), id-usuário, Social) =
if *existe_ambiente* ((_,Tipos-Ambientes tipos-ambientes), Social)
then **if** **not** *usuário_pertence_ambiente* ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Social)
then *entrada_usuário_ambiente_aux* ((_, usuários, Tipos-Ambientes tipos-ambientes), id-usuário, Social)
else (_, Tipos-Ambientes tipos-ambientes)
else (_, Tipos-Ambientes tipos-ambientes)

entrada_usuário_ambiente ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Reunião) =
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Reunião)
then **if** **not** *usuário_pertence_ambiente* ((_, Tipos-Ambientes tipos-

ambientes), id-usuário, Reunião)
then *entrada_usuário_ambiente_aux* ((_, usuários, Tipos-Ambientes tipos-ambientes), id-usuário, Reunião)
else (_, Tipos-Ambientes tipos-ambientes)
else (_, Tipos-Ambientes tipos-ambientes)

entrada_usuário_ambiente ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Conversa-Particular) =
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Conversa-Particular)
then **if** **not** *usuário_pertence_ambiente* ((_, usuários, Tipos-Ambientes tipos-ambientes), id-usuário, Conversa-Particular)
then *entrada_usuário_ambiente_aux* ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Conversa-Particular)
else (_, Tipos-Ambientes tipos-ambientes)
else (_, Tipos-Ambientes tipos-ambientes)

saída_usuário_ambiente ((_, Tipos-Ambientes empty-set), _, _) =
(_, Tipos-Ambientes tipos-ambientes)

saída_usuário_ambiente ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Social) =
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Social)
then **if** *usuário_pertence_ambiente* ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Social)
then *saída_usuário_ambiente_aux* ((_, usuários, Tipos-Ambientes tipos-ambientes), id-usuário, Social)
else (_,Tipos-Ambientes tipos-ambientes)
else (_, Tipos-Ambientes tipos-ambientes)

saída_usuário_ambiente ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Reunião) =
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Reunião)
then **if** *usuário_pertence_ambiente* ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Reunião)
then *saída_usuário_ambiente_aux* ((_, Representação-Usuários rep-usuários, Tipos-Ambientes tipos-ambientes), id-usuário, Reunião)
else (_, Tipos-Ambientes tipos-ambientes)
else (_, Tipos-Ambientes tipos-ambientes)

saída_usuário_ambiente ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Conversa-Particular) =
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Conversa-Particular)
then **if** *usuário_pertence_ambiente* ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Conversa-Particular)
then *saída_usuário_ambiente_aux* ((_, Representação-Usuários rep-usuários, Tipos-Ambientes tipos-ambientes), id-usuário, Conversa-Particular)
else (_, Tipos-Ambientes tipos-ambientes)
else (_, Tipos-Ambientes tipos-ambientes)

altera_ícone-saída-temp ((Representação-Movimentos Ícone _,_),novo-ícone-saída-temp) =
(Representação-Movimentos Ícone novo-ícone-saída-temp, _)

altera_ícone-saída-temp ((Representação-Movimentos Ícone ícone-saída-temp, _), novo-ícone-saída-temp) =
(Representação-Movimentos Ícone novo-ícone-saída-temp, _)

saída_temp_usuário_ambiente ((_, Tipos-Ambientes empty-set), _, _) =
(_, Tipos-Ambientes tipos-ambientes)

saída_temp_usuário_ambiente ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Social) =
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Social)
then **if** *usuário_pertence_ambiente* ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Social)
then *saída_temp_usuário_ambiente_aux* ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Social)
else (_, Tipos-Ambientes tipos-ambientes)
else (_, Tipos-Ambientes tipos-ambientes)

saída_temp_usuário_ambiente ((_, Tipos-Ambientes tipos-ambientes),id-usuário,Reunião)=
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Reunião)
then **if** *usuário_pertence_ambiente* ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Reunião)
then *saída_temp_usuário_ambiente_aux* ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Reunião)
else (_, Tipos-Ambientes tipos-ambientes)
else (_, Tipos-Ambientes tipos-ambientes)

saída_temp_usuário_ambiente ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Conversa-Particular) =
if *existe_ambiente* ((_, Tipos-Ambientes tipos-ambientes), Conversa-Particular)
then **if** *usuário_pertence_ambiente* ((_, Tipos-Ambientes tipos-ambientes), id-usuário, Conversa-Particular)
then *saída_temp_usuário_ambiente_aux* ((_, usuários, Tipos-Ambientes

```
                tipos-ambientes), id-usuário, Conversa-Particular)
        else    (_, Tipos-Ambientes tipos-ambientes)
else    (_, Tipos-Ambientes tipos-ambientes)
```

6.5 ATO Bloco de Notas

Este ATO permite a utilização de um Bloco de Notas, onde os usuários podem se comunicar, tal como um *chat*.

A instanciação desse ATO é vista na Fig. 6.6.

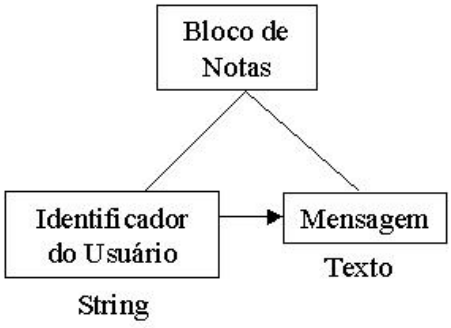


FIGURA 6.6 - Instanciação do ATO Bloco de Notas

6.5.1 Operação de criação

Esta operação cria um Bloco de Notas vazio, através de um mapeamento vazio. Assim os usuários só serão inseridos aqui quando enviarem uma mensagem.

Interface

cria : $\rightarrow$ BN

Operação

cria = (Corpo-Mensagem empty-mapping)

6.5.2 Operação Observadora

A operação observadora definida para esta classe é:

- *usuário_enviou_mensagem* – esta operação verifica se um determinado usuário já enviou alguma mensagem.

Interface

usuário_enviou_mensagem (_,_): BN, STRING → BOOL

Operação

Variáveis formais: ..... id-usuário: STRING
bloco-notas: MAPEAMENTO

usuário_enviou_mensagem ((Bloco-Notas bloco-notas), id-usuário) =
if id-usuário ∈ dom (bloco-notas)
then TRUE
else FALSE

6.5.3 Operação Modificadora

A operação modificadora definida para esta classe é:

- *envia_mensagem* – possibilita o envio de mensagens de um usuário participante da sessão do Bloco de Notas.

Interface

envia_mensagem (_,_,_): BN, STRING, TEXTO → BN

Operação

Variáveis formais: ..... id-usuário: STRING
mensagem: TEXTO
bloco-notas: MAPEAMENTO
envia_mensagem ((Bloco-Notas bloco-notas), id-usuário, mensagem) =
if mensagem ≠ " "
then (Bloco-Notas *envia_mensagem_aux* (bloco-notas, id-usuário, mensagem))
else (Bloco-Notas bloco-notas)

6.6 ATO Papel para os Usuários

Este ATO permite que os usuários que estão participando da atividade cooperativa possuem papéis (por exemplo, editor, leitor,...) e que podem ser representados através de ícones particulares a cada papel. A instanciação do ATO é mostrada na Fig. 6.7.

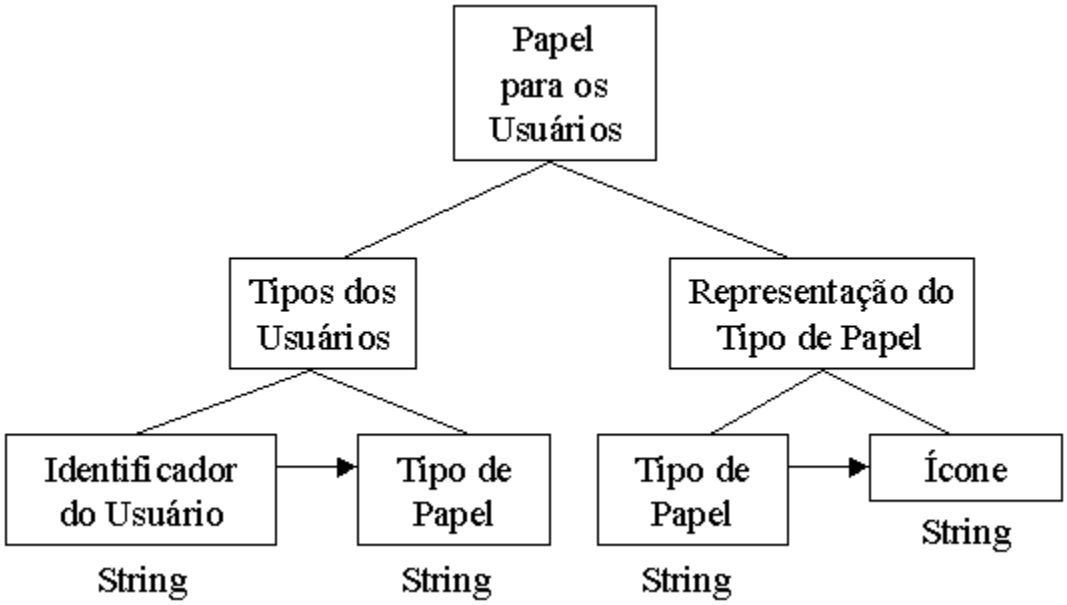


FIGURA 6.7 - Instanciação do ATO Papel para os Usuários

6.6.1 Operação de criação

Esta operação cria os papéis dos usuários. Assim, como a necessidade de tipos diferentes de papéis podem surgir durante o desenvolvimento das atividades cooperativas, esse ATO é criado vazio para posterior preenchimento.

Interface

cria (): → PPU

Operação

cria = (Tipos-Usuários empty-mapping, Representação-Tipos-Papéis empty-mapping)

6.6.2 Operações Observadoras

As operações observadoras definidas para esta classe são:

- *existe_usuário* – verifica se um determinado usuário (seu identificador) possui um papel;
- *papel_usuário* – esta operação fornece o papel que um determinado usuário possui;
- *existe_papel* – verifica se um papel pertence ao conjunto de Tipos de Papéis que os usuários podem assumir;
- *ícone_papel* – fornece qual ícone está sendo usado para representar um determinado papel;
- *existe_ícone* – verifica se um determinada ícone já pertence a algum papel.

Interface

existe_usuário (_,_): PPU, STRING → BOOL
papel_usuário (_,_): PPU, STRING → STRING
existe_papel (_,_): PPU, STRING → BOOL
ícone_papel (_,_): PPU, STRING → STRING
existe_ícone (_,_): PPU, STRING → BOOL

Operações

Variáveis formais: ..... tipos-usuário, rep-tipo-papel: MAPEAMENTO
id-usuário, papel, ícone: STRING

existe_usuário ((Tipos-Usuários empty-mapping, _), _) =
FALSE

existe_usuário ((Tipos-Usuários tipos-usuários, _), id-usuário) =
if *existe_usuário_aux* (tipos-usuários, id-usuário)
then TRUE
else FALSE

papel_usuário ((Tipos-Usuários empty-mapping, _), _) =
"Usuário_não_existe"

papel_usuário ((Tipos-Usuários tipos-usuários, _), id-usuário) =
if *existe_usuário* ((Tipos-Usuários tipos-usuários, _), id-usuário)
then *papel_usuário_aux* (tipos-usuários, id-usuário)
else "Usuário_não_existe"

existe_papel ((_, Representação-Tipo-Papel empty-mapping), _) =
FALSE

existe_papel ((_, Representação-Tipo-Papel rep-tipo-papel), papel) =
if *existe_papel_aux* (rep-tipo-papel, papel)
then TRUE
else FALSE

ícone_papel ((_, Representação-Tipo-Papel rep-tipo-papel), papel) =
if *existe_papel* ((_, Representação-Tipo-Papel rep-tipo-papel), papel)
then *ícone_ papel_aux* (rep-tipo- papel, papel)
else "Papel_não_existe"
existe_ícone ((_, Representação-Tipo-Papel empty-mapping), _) =
FALSE

existe_ícone ((_, Representação-Tipo-Papel rep-tipo- papel), ícone) =
if ícone ∈ rng (rep-tipo- papel)
then TRUE
else FALSE

6.6.3 Operações Modificadoras

As operações modificadoras definidas para esta classe são:

- *inclui_usuário_papel* – inclui um usuário e o papel que ele representa na sessão cooperativa que está participando;
- *exclui_usuário_papel* – elimina um usuário e seu papel;
- *inclui_papel_ícone* – inclui um papel e sua forma de representação (através do ícone);
- *exclui_papel_ícone* – exclui um papel e sua representação;
- *altera_papel_usuário* – esta operação modifica o papel de um determinado usuário;
- *altera_ícone_papel* – esta operação modifica o ícone de um papel.

Interface

inclui_usuário_papel (_,_,_): PPU, STRING, STRING → FU
exclui_usuário_papel (_,_): PPU, STRING → FU
inclui_papel_ícone (_,_,_): PPU, STRING, STRING → FU
exclui_papel_ícone (_,_): PPU, STRING → FU
altera_papel_usuário (_,_,_): PPU, STRING, STRING → FU
altera_ícone_papel (_,_,_): PPU, STRING, STRING → FU

Operações

Variáveis formais: ..... rep-tipo-papel, tipos-usuários: MAPEAMENTO
id-usuário, papel, ícone, novo-papel, novo-ícone: STRING

inclui_usuário_papel ((Tipos-Usuários empty-mapping, Representação-Tipo-Papel empty-mapping), _, _) =
(Tipos-Usuários empty-mapping, Representação-Tipo-Papel empty-mapping)

inclui_usuário_papel ((Tipos-Usuários empty-mapping, Representação-Tipo-Papel rep-tipo-papel), id-usuário, papel) =
if *existe_papel* ((Tipos-Usuários empty-mapping, Representação-Tipo-Papel rep-tipo-papel), papel)
then (Tipos-Usuários *inclui_usuário_papel_aux* (empty-mapping, id-usuário, papel), Representação-Tipo-Papel rep-tipo-papel)
else (Tipos-Usuários empty-mapping, Representação-Tipo-Papel rep-tipo-papel)

inclui_usuário_papel ((Tipos-Usuários tipos-usuários, Representação-Tipo-Papel rep-tipo-papel), id-usuário, papel) =
if **not** *existe_usuário* ((Tipos-Usuários tipos-usuários, Representação-Tipo-Papel rep-tipo-papel), id-usuário) **and**
existe_papel ((Tipos-Usuários tipos-usuários, Representação-Tipo-Papel rep-tipo-papel), papel)
then (Tipos-Usuários *inclui_usuário_papel_aux* (tipos-usuários, id-usuário, papel), Representação-Tipo-Papel rep-tipo-papel)
else (Tipos-Usuários tipos-usuários, Representação-Tipo-Papel rep-tipo-papel)

exclui_usuário_papel ((Tipos-Usuários empty-mapping, _), _) =
(Tipos-Usuários empty-mapping, _)

exclui_usuário_papel ((Tipos-Usuários tipos-usuários, _), id-usuário) =
if *existe_usuário* ((Tipos-Usuários tipos-usuários, _), id-usuário)
then (Tipos-Usuários *exclui_usuário_função_aux* (tipos-usuários, id-usuário), _)
else (Tipos-Usuários tipos-usuários, _)

inclui_papel_ícone ((_, Representação-Tipo-Papel empty-mapping), papel, ícone) =
(_, Representação-Tipo-Papel *inclui_papel_ícone_aux* (papel, ícone, empty-mapping))

inclui_papel_ícone ((_, Representação-Tipo-Papel rep-tipo-papel), papel, ícone) =

if **not** *existe_papel* ((_, Representação-Tipo-Papel rep-tipo-papel), papel) **and**
not *existe_ícone* ((_, Representação-Tipo-Papel rep-tipo-papel), ícone)
then (_, Representação-Tipo-Papel *inclui_papel_ícone_aux* (papel, ícone, rep-tipo-papel))
else (_, Representação-Tipo-Papel rep-tipo-papel)

exclui_papel_ícone ((_,Representação-Tipo-Papel empty-mapping), _) =
(_,Representação-Tipo-Papel empty-mapping)

exclui_papel_ícone ((Tipos-Usuários tipos-usuários,Representação-Tipo-Papel rep-tipo-papel), papel) =
if *existe_papel* ((Tipos-Usuários tipos-usuários, Representação-Tipo-Papel rep-tipo-papel), papel) **and**
not *existe_usuário_papel_excluída_aux* (tipos-usuários, papel)
then (Tipos-Usuários tipos-usuários, Representação-Tipo-Papel *exclui_papel_ícone_aux* (rep-tipo-papel, papel))
else (_, Representação-Tipo-Papel rep-tipo-papel)

altera_papel_usuário ((Tipos-Usuários tipos-usuários,Representação-Tipo-Papel rep-tipo-papel), id-usuário, novo-papel) =
if *existe_usuário* ((Tipos-Usuários tipos-usuários, Representação-Tipo-Papel rep-tipo-papel), id-usuário) **and**
existe_papel ((Tipos-Usuários tipos-usuários, Representação-Tipo-Papel rep-tipo-papel), novo-papel)
then (Tipos-Usuários *altera_papel_usuário_aux* (tipos-usuários, id-usuário, novo-papel), Representação-Tipo-Papel rep-tipo-papel)
else (Tipos-Usuários tipos-usuários,Representação-Tipo-Papel rep-tipo-papel)

altera_ícone_papel ((_,Representação-Tipo-Papel rep-tipo-papel), papel, novo-ícone) =
if *existe_papel* ((_,Representação-Tipo-Papel rep-tipo-papel), papel) **and**
not *existe_ícone* ((_,Representação-Tipo-Papel rep-tipo-papel), novo-ícone)
then (_,Representação-Tipo-Papel *altera_ícone_papel_aux* (papel, novo-ícone, rep-tipo-papel)
else (_,Representação-Tipo-Papel rep-tipo-papel)

6.7 ATO Barra de Rolagem de Sincronização

Este ATO permite a utilização de uma barra de rolagem de sincronização onde os usuários participantes podem se localizar em relação aos outros usuários. A Fig. 6.8 mostra a instanciação do ATO Barra de Rolagem de Sincronização.

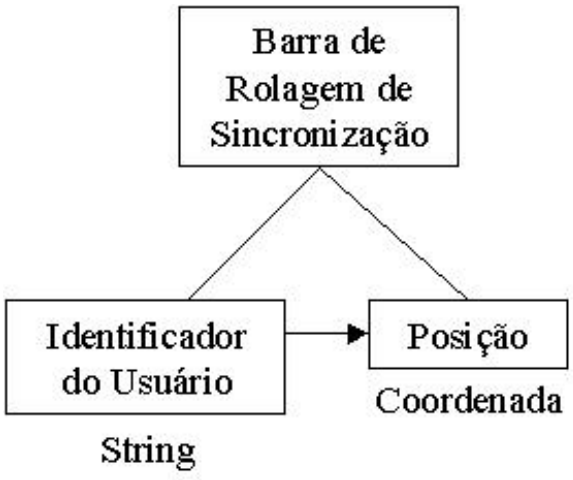


FIGURA 6.8 - Instanciação do ATO Barra de Rolagem de Sincronização

6.7.1 Operação de criação

A operação utilizada para criar uma Barra de Rolagem de Sincronização não necessita de argumentos pois é criada apenas um mapeamento vazio, no qual vai ser inserido os usuários que irão fazer parte da Barra.

Interface

cria (): $\rightarrow$ BRS

Operação

cria = (Barra-Rol-Sinc empty-mapping)

6.7.2 Operações Observadoras

As operações observadoras pertencentes a essa classe são:

- *existe_usuário* – verifica se um determinado usuário faz parte da Barra de Rolagem de Sincronização;
- *posição_barra_usuário* – esta operação verifica qual a posição que está localizada a Barra de Rolagem de Sincronização de um determinado usuário.

Interface

existe_usuário (_,_): BRS, STRING → BOOL
posição_barra_usuário (_,_): BRS, STRING → COORDENADA

Operações

Variáveis formais: ..... barra: MAPEAMENTO
id-usuário: STRING

existe_usuário ((Barra-Rol-Sinc empty-mapping),_) =
FALSE

existe_usuário ((Barra-Rol-Sinc barra), id-usuário) =
if *existe_usuário_aux* (barra, id-usuário)
then TRUE
else FALSE

posição_barra_usuário ((Barra-Rol-Sinc empty-mapping),_) =
(_)

posição_barra_usuário ((Barra-Rol-Sinc barra), id-usuário) =
if *existe_usuário_aux* (barra, id-usuário)
then *posição_barra_usuário_aux* ((Barra-Rol-Sinc barra), id-usuário)
else (_)

6.7.3 Operações Modificadoras

As operações modificadoras que pertencem a essa classe são:

- *inclui_usuário* – inclui um determinado usuário e sua posição na Barra de Rolagem de Sincronização
- *exclui_usuário* – exclui um determinado usuário e sua cor representativa na Barra de Rolagem de Sincronização
- *movimenta_barra* – coloca a barra de um usuário em uma determinada posição, possibilitando que este usuário possa visualizar onde os outros participantes do grupo estão trabalhando

Interface

inclui_usuário (_,_,_):BRS, STRING, COORDENADA → BRS
exclui_usuário (_,_):BRS, STRING → BRS
movmenta_barra (_,_,_): BRS, STRING, COORDENADA → BRS

Operações

Variáveis formais: ..... id-usuário: STRING
posição, nova-posição: COORDENADA
barra: MAPEAMENTO

inclui_usuário ((Barra-Rol-Sinc empty-mapping), id-usuário, posição) =
(Barra-Rol-Sinc *inclui_usuário_aux* (empty-mapping, id-usuário, posição))

inclui_usuário ((Barra-Rol-Sinc barra), id-usuário, posição) =
if **not** *existe_usuário* ((Barra-Rol-Sinc barra), id-usuário)
then (Barra-Rol-Sinc *inclui_usuário_aux* (barra, id-usuário, posição))
else (Barra-Rol-Sinc barra)

exclui_usuário ((Barra-Rol-Sinc empty-mapping), _) =
(Barra-Rol-Sinc empty-mapping)

exclui_usuário ((Barra-Rol-Sinc barra), id-usuário) =
if *existe_usuário* ((Barra-Rol-Sinc barra), id-usuário)
then (Barra-Rol-Sinc *exclui_usuário_aux* (barra, id-usuário))
else (Barra-Rol-Sinc barra)

movimenta_barra ((Barra-Rol-Sinc barra), id-usuário, nova-posição) =
if *existe_usuário* ((Barra-Rol-Sinc barra), id-usuário)
then (Barra-Rol-Sinc *movimenta_barra_aux* (barra, id-usuário, nova-posição))
else (Barra-Rol-Sinc barra)

6.8 ATO Janelas Auxiliares

O ATO Janelas Auxiliares permite que os usuários visualizem a área de trabalho de acordo com a janela escolhida (WYSIWID, *Radar View* ou *Miniature View*). A instanciação deste ATO é vista na Fig. 6.9.

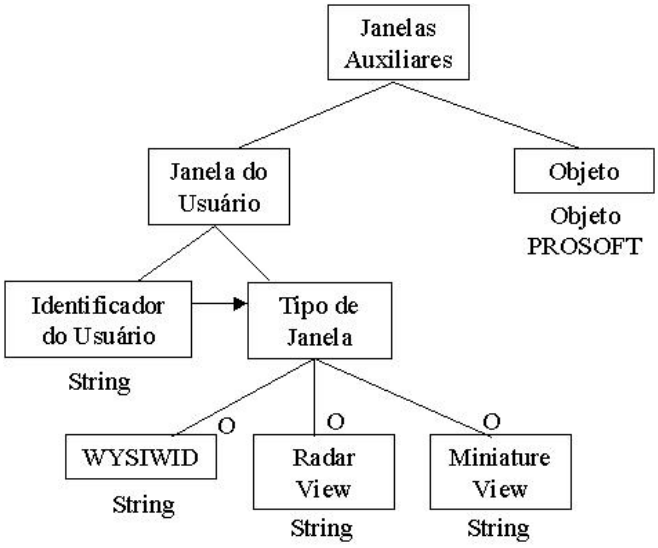


FIGURA 6.9 - Instanciação do ATO Janelas Auxiliares

6.8.1 Operação de criação

Esta operação cria janelas para os usuários de acordo com o tipo que deseja. Porém, neste primeiro instante não há escolha do tipo de janela de cada usuário, mas o objeto que eles irão visualizar já será escolhido.

É importante ressaltar que o objeto PROSOFT passado nesta operação como argumento, não pode ser do tipo do ATO Janelas Auxiliares, como também não pode ter um objeto do tipo Janela Auxiliares dentro do objeto. Isso ocorre porque se o objeto passado for do tipo Janelas Auxiliares ou contiver um objeto deste tipo, então iria acontecer uma recursividade e os usuários visualizariam janelas dentro de janelas.

Interface

cria (_): OBJETO-PROSOFT $\rightarrow$ JA

Operação

cria (objeto) = (Janelas-Auxiliares empty-mapping, Objeto objeto)

6.8.2 Operações Observadoras

As operações observadoras pertencentes a essa classe são:

- *tipo_janela_usuário* – esta operação devolve o tipo de janela que um determinado usuário está usando (podem ser WYSIWID, Radar View ou Miniature View);

- *existe_usuário* – verifica se um determinado usuário possui uma janela auxiliar.

Interface

tipo_janela_usuário (_,_): JA, STRING → STRING
existe_usuário (_,_):JA, STRING → BOOL

Operações

Variáveis formais: ..... id-usuário: STRING
janela-usuário: MAPEAMENTO

tipo_janela_usuário ((Janela-Usuário janela-usuário, _), id-usuário) =
if *existe_usuário* ((Janela-Usuário janela-usuário, _), id-usuário)
then *tipo_janela_usuário_aux* ((Janela-Usuário janela-usuário, _), id-usuário)
else "Não_existe_usuário"

existe_usuário ((Janela-Usuário janela-usuário, _), id-usuário) =
if id-usuário ∈ dom (janela-usuário)
then TRUE
else FALSE

6.8.3 Operações Modificadoras

As operações modificadoras que pertencem a essa classe são:

- *inclui_usuário* – inclui um usuário e o seu tipo de janela auxiliar;
- *exclui_usuário* – exclui um usuário;
- *altera_tipo_janela* – esta operação altera o tipo de janela de um usuário.

Interface

inclui_usuário (_,_,_): JA, STRING, STRING → JA
exclui_usuário (_,_): JA, STRING → JA
altera_tipo_janela (_,_,_): JA, STRING, STRING → JA

Operações

Variáveis formais: ..... id-usuário, tipo-janela, novo-tipo-janela: STRING
janela-usuário: MAPEAMENTO

inclui_usuário ((Janela-Usuário empty-mapping, _), id-usuário, tipo-janela) =
if novo-tipo-janela = "WYSIWID" **or**
novo-tipo-janela = "Radar-View" **or**
novo-tipo-janela = "Miniature-View"
then (Janela-Usuário *inclui_usuário_aux* (empty-mapping, id-usuário, tipo-janela), _)
else (Janela-Usuário empty-mapping, _)

inclui_usuário ((Janela-Usuário janela-usuário, _), id-usuário, tipo-janela) =
if **not** *existe_usuário* ((Janela-Usuário janela-usuário, _), id-usuário) **and**
(novo-tipo-janela = "WYSIWID" **or**
novo-tipo-janela = "Radar-View" **or**
novo-tipo-janela = "Miniature-View")
then (Janela-Usuário *inclui_usuário_aux* (janela-usuário, id-usuário, tipo-janela), _)
else (Janela-Usuário janela-usuário, _)

exclui_usuário ((Janela-Usuário empty-mapping, _), _) =
(Janela-Usuário empty-mapping, _)

exclui_usuário ((Janela-Usuário janela-usuário, _), id-usuário) =
if *existe_usuário* ((Janela-Usuário janela-usuário, _), id-usuário)
then (Janela-Usuário *exclui_usuário_aux* (janela-usuário, id-usuário), _)
else (Janela-Usuário janela-usuário, _)

altera_tipo_janela ((Janela-Usuário empty-mapping, _), _, _) =
(Janela-Usuário empty-mapping, _)

altera_tipo_janela ((Janela-Usuário janela-usuário, _), id-usuário, novo-tipo-janela) =
if *existe_usuário* ((Janela-Usuário janela-usuário, _), id-usuário) **and**
(novo-tipo-janela = "WYSIWID" **or**
novo-tipo-janela = "Radar-View" **or**
novo-tipo-janela = "Miniature-View")
then (Janela-Usuário *altera_tipo_janela_aux* (janela-usuário, id-usuário, novo-tipo-janela), _)
else (Janela-Usuário janela-usuário, _)

7 Conclusão

A Engenharia de Software utiliza várias tecnologias para que seu desempenho melhore cada vez mais. O uso de CSCW torna ainda mais eficiente o desenvolvimento de sistemas realizado por um grupo de pessoas. Para que haja a interação entre os membros do grupo se faz necessário meios que possibilitem a visualização das tarefas que estão sendo realizadas por todas as pessoas participantes, sendo que esta visão deve ser a mais parecida com a realidade.

A forma encontrada para que o grupo consiga identificar as atividades que estão ocorrendo e quem as estão realizando é através de componentes de interface homem-computador. Esses componentes conseguem, na área limitada de uma tela de computador, fornecer a percepção necessária para que o trabalho em grupo se torne mais produtivo e eficiente.

7.1 Contribuições

Uma importante contribuição deste trabalho é a especificação formal de componentes de interface homem-máquina, o que ainda não é explorado suficientemente a fim de mostrar os benefícios encontrados. A escolha do Ambiente de Desenvolvimento de Software PROSOFT Cooperativo traz, portanto, uma oportunidade para o aperfeiçoamento de especificações formais na área de componentes voltados para Interfaces Cooperativas, sendo que neste trabalho a especificação utilizada foi o PROSOFT-Algébrico.

O PROSOFT-Algébrico possui vantagens principalmente na forma de comunicação entre os ATOs construídos, pela ICS – Interface de Comunicação do Sistema. Pois permite uma fácil chamada de operações entre os ATOs que estão juntamente cooperando.

A construção das classes, interfaces e operações que formam os ATOs ainda é um processo trabalhoso, por isso a introdução neste trabalho de um capítulo que mostra como se realiza a construção de ATOs cooperativos, facilitando assim os próximos desenvolvedores de ATOs e também às pessoas que desejam estudar os ATOS de Componentes de Interfaces Cooperativas especificadas neste trabalho.

Os componentes propostos neste trabalho não só englobam aqueles que podem ser usados em sistemas que estão sendo construídos como também há alguns, tal como Ambiente de Percepção e Bloco de Notas, que podem ser usados individualmente sem que sejam parte de algum outro sistema *groupware*. Sendo assim, pode-se chamar esses componentes de sistemas *groupware* que já estão prontos e especificados para serem

utilizados. Esta característica se torna um diferencial para os outros ambientes de desenvolvimento que não proporcionam essa facilidade.

7.2 Dificuldades encontradas

A dificuldade inicial encontrada neste trabalho, foi o entendimento do paradigma do Ambiente PROSOFT e consequentemente do PROSOFT Cooperativo. Este obstáculo foi o causador da mudança de requisitos durante o desenvolvimento deste trabalho. Primeiramente, foi idealizada uma ferramenta que fornecesse um conjunto de componentes para o desenvolvedor de acordo com as suas necessidades e características do *groupware* que ele estaria desenvolvendo. Sendo essa idéia inicial descartada, deu-se a criação de uma "biblioteca" de componentes em que o desenvolvedor teria a oportunidade de "encaixá-los" no lugar mais apropriado durante o decorrer do trabalho.

7.3 Trabalhos Futuros

As seguintes tarefas complementariam este trabalho:

- Definir componentes de Inrterface Homem-Computador como forma de visualização para a Memória de Grupo, a qual faz parte da dissertação de Mestrado que está sendo desenvolvida atualmente pelo aluno Ronnie Alves nesta instituição;
- Abordar outros tipos de percepção, tais como as percepções organizacional ou funcional, possibilitando assim que os participantes de uma atividade cooperativa tenham acesso a informações de várias ramos da organização a que pertencem e da tarefa que estão realizando;
- Implementar os ATOs aqui propostos para melhor visualizar os componentes disponíveis para a Percepção do Espaço de Trabalho. Este trabalho é proposto aqui por não ter sido possível a sua realização pois o PROSOFT está atualmente passando por uma reengenharia da sua Interface Homem-Computador [RAB 2001];
- Realizar uma avaliação empírica que possibilite a quantificação dos benefícios trazidos pela utilização dos Componentes de Percepção especificados neste trabalho.

ANEXO 1 OPERAÇÕES INTERNAS PARA OS COMPONENTES DE PERCEPÇÃO

Neste anexo são apresentadas as operações internas referentes às operações dos Componentes de Percepção para o Ambiente PROSOFT Cooperativo. Estas operações são utilizadas somente nas especificações dos ATOs e não estão disponíveis nas suas interfaces.

1.1 ATO PREFERÊNCIAS DOS USUÁRIOS

As operações internas para o ATO Preferências dos Usuários estão divididas seguindo a especificação das operações citadas na interface, em observadoras e modificadoras.

1.1.1 Operações internas observadoras

As operações internas são as seguintes:

- *representação_usuário_aux* – auxilia a operação *representação_usuário* no retorno da forma que um determinado usuário é representado;
- *ícone_usuário_aux* – auxilia a operação *ícone_usuário* no retorno do ícone pelo qual um determinado usuário é representado;
- *existe_ícone_aux* – auxilia a operação *existe_ícone* na verificação se um determinado ícone já pertence a algum usuário;
- *existe_label_aux* – auxilia a operação *existe_label* na verificação se um determinado label já pertence a algum usuário;
- *existe_cor_aux* – auxilia a operação *existe_cor* na verificação se uma determinada cor já pertence a algum usuário.

Interface

representação_usuário_aux (_,_): PU, STRING → STRING
ícone_usuário_aux (_,_): PU, STRING → STRING
existe_ícone_aux (_,_):PU, STRING → BOOL
existe_label_aux (_,_):PU, STRING → BOOL

existe_cor_aux (_,_):PU, STRING → BOOL

Operações

Variáveis Formais: identificador-usuário, id-usuário, forma, ícone: STRING
ícone-usuário, label, label-usuário, cor, cor-usuário: STRING
preferências-usuários-restantes: MAPEAMENTO

representação_usuário_aux ((Preferências-Usuários modify (identificador-usuário, (Preferência (Forma-Representação (forma, _)), preferências-usuários-restantes), id-usuário) =
if identificador-usuário = id-usuário
then forma
else *representação_usuário_aux* ((Preferências-Usuários preferências-usuários-restantes), id-usuário)

ícone_usuário_aux ((Preferências-Usuários modify (identificador-usuário, (Preferência (Forma-Representação (_, ícone)), preferências-usuários-restantes), id-usuário) =
if identificador-usuário = id-usuário
then ícone
else *ícone_usuário_aux* ((Preferências-Usuários preferências-usuários-restantes), id-usuário)

existe_ícone_aux ((Preferências-Usuário empty-mapping), _) =
FALSE

existe_ícone_aux ((Preferências-Usuário modify (_, Preferências (_, Ícone ícone-usuário), preferências-usuários-restantes)), ícone) =
if ícone = ícone-usuário
then TRUE
else *existe_ícone_aux* (Preferências-Usuário preferências-usuários-restantes)

existe_label_aux ((Preferências-Usuário empty-mapping), _) =
FALSE

existe_label_aux ((Preferências-Usuário modify (_, Preferências (Label label-usuário, _), preferências-usuários-restantes)), label) =
if label = label-usuário
then TRUE
else *existe_label_aux* (Preferências-Usuário preferências-usuários-restantes)

existe_cor_aux ((Preferências-Usuário empty-mapping), _) =
FALSE

existe_cor_aux ((Preferências-Usuário modify (_, Preferências (Cor cor-usuário, _), preferências-usuários-restantes)), cor) =

if cor = cor-usuário
then TRUE
else *existe_cor_aux* (Preferências-Usuário preferências-usuários-restantes)

1.1.2 Operações internas Modificadoras

As operações internas são as seguintes:

- *inclui_usuário_label_aux* – auxilia a operação *inclui_usuário* na inclusão de um novo usuário;
- *exclui_usuário_aux* – auxilia a operação *exclui_usuário* na exclusão de um usuário;
- *altera_cor_aux* – auxilia a operação *altera_cor* na alteração da cor de representação de um usuário;
- *altera_label_aux* – auxilia a operação *altera_label* na alteração do label de representação de um usuário;
- *altera ícone_aux* – auxilia a operação *altera ícone* na alteração o ícone representativo de um usuário.

Interface

inclui_usuário_label_aux (_,_,_,_): PU, STRING, STRING, STRING → PU
exclui_usuário_aux (_,_): PU, STRING → PU
altera_cor_aux (_,_,_): PU, STRING, STRING → PU
altera_label_aux (_,_,_): PU, STRING, STRING → PU
altera_ícone_aux (_,_,_): PU, STRING, STRING → PU

Operações

Variáveis Formais: preferências-usuários: MAPEAMENTO
preferências-usuários-restantes: MAPEAMENTO
id-usuário, identificador-usuário, label, ícone, cor: STRING
nova-cor, novo-label, novo-ícone: STRING
pu: PU

inclui_usuário_label_aux ((Preferências-Usuários preferências-usuários)), id-usuário, label, ícone) =
(Preferências-Usuários modify (id-usuário, Preferências (Label label, Ícone ícone), preferências-usuários))

inclui_usuário_cor_aux ((Preferências-Usuários preferências-usuários)), id-usuário, cor, ícone) =

(Preferências-Usuários modify (id-usuário, Preferências (Cor cor, Ícone ícone), preferências-usuários))

exclui_usuário_aux (pu, id-usuário) =
pu \ id-usuário

altera_cor_aux ((Preferências-Usuários modify (identificador-usuário, (Forma-Rep Cor cor, _), preferências-usuários-restantes)), id-usuário, nova-cor) =
if identificador-usuário = id-usuário
then (Preferências-Usuários modify (identificador-usuário, (Forma-Rep Cor nova-cor, _), preferências-usuários-restantes))
else *altera_cor_aux* ((Preferências-Usuários preferências-usuários-restantes), id-usuário, nova-cor)

altera_label_aux ((Preferências-Usuários modify (identificador-usuário, (Forma-Rep Label label, _), preferências-restantes)), id-usuário, novo-label) =
if identificador-usuário = id-usuário
then (Preferências-Usuários modify (identificador-usuário, (Forma-Rep Label novo-label, _), preferências-restantes))
else *altera_label_aux* ((Preferências-Usuários preferências-restantes), id-usuário, novo-label)

altera_ícone_aux ((Preferências-Usuários modify (identificador-usuário, (_, Icone ícone), preferências-restantes)), id-usuário, novo-ícone) =
if identificador-usuário = id-usuário
then (Preferências-Usuários modify (identificador-usuário, (_, Ícone novo-ícone), preferências-restantes))
else *altera_ícone_aux* ((Preferências-Usuários preferências-restantes), id-usuário, novo-ícone)

1.2 ATO MÚLTIPLOS CURSORES

As operações internas para o ATO Múltiplos Cursores estão divididas seguindo a especificação das operações citadas na interface, em observadoras e modificadoras.

1.2.1 Operações internas Observadoras

As operações internas são as seguintes:

- *retorna_posição_nula* – esta operação auxilia a operação *posição_cursor* no caso de que não exista o usuário solicitado;
- *posição_cursor_aux* – auxilia a operação *posição_cursor* a verificar a posição de um determinado cursor;
- *existe_usuário_aux* – auxilia a operação *existe_usuário* a verificar se existe um determinado usuário.

Interface

retorna_posição_nula : → COORDENADA
posição_cursor_aux (_,_): MC, STRING → COORDENADA
existe_usuário_aux (_,_): MAPEAMENTO, STRING → BOOL

Operações

Variáveis formais: posição: COORDENADA
identificador-usuário, id-usuário: STRING
múltiplos-cursores-restantes, múltiplos-cursores: MAPEAMENTO

retorna_posição_nula =
(_,_)

posição_cursor_aux ((Múltiplos-Cursores modify (identificador-usuário, posição, múltiplos-cursores-restantes), id-usuário) =
if identificador-usuário = id-usuário
then posição
else *posição_cursor_aux* ((Múltiplos-Cursores múltiplos-cursores-restantes), id-usuário)

existe_usuário_aux (múltiplos-cursores, id-usuário) =
if id-usuário ∈ múltiplos-cursores
then TRUE
else FALSE

1.2.2 Operações internas Modificadoras

As operações internas são as seguintes:

- *movimenta_cursor_aux* – auxilia a operação *movimente_cursor* a deslocar o cursor de uma posição para outra;
- *inclui_usuário_aux* - auxilia a operação *inclui_usuário* na inclusão de um novo usuário e a posição do seu cursor;
- *exclui_usuário_aux* - auxilia a operação *exclui_usuário* na exclusão de um usuário do seu cursor.

Interface

movimenta_cursor_aux (_,_,_): MC, STRING, COORDENADA → MC
inclui_usuário_aux (_,_,_):
STRING, COORDENADA, MAPEAMENTO → MAPEAMENTO
exclui_usuário_aux (_,_): STRING, MAPEAMENTO → MAPEAMENTO

Operações

Variáveis formais: identificador-usuário, id-usuário: STRING
posição, nova-posição: COORDENADA
múltiplos-cursores-restantes, múltiplos-cursores: MAPEAMENTO

movimenta_cursor_aux ((Múltiplos-Cursores modify (identificador-usuário, posição, múltiplos-cursores-restantes)),.id-usuário, nova-posição) =
if identificador-usuário = id-usuário
then (Múltiplos-Cursores modify (identificador-usuário, nova-posição, múltiplos-cursores-restantes)
else *movimenta_cursor_aux* ((Múltiplos-Cursores múltiplos-cursores-restantes)),.id-usuário, nova-posição)

inclui_usuário_aux (id-usuário, posição, múltiplos-cursores) =
modify (id-usuário, posição, múltiplos-cursores)

exclui_usuário_aux (id-usuário, múltiplos-cursores) =
múltiplos-cursores \ id-usuário

1.3 ATO TELEPONTEIROS

As operações internas para o ATO Teleponteiros estão divididas seguindo a especificação das operações citadas na interface, em observadoras e modificadoras.

1.3.1 Operações internas Observadoras

As operações internas são as seguintes:

- *existe_operação_aux* – auxilia a operação *existe_operação* verificar se um teleponteiro possui uma representação para uma determinada operação;
- *formato_operação_aux* – auxilia a operação *formato_operação* no retorno da forma que o teleponteiro assume quando realiza uma determinada operação;
- *retorna_posição_nula_tele* – retorna uma coordenada sem valores (nula);
- *posição_teleponteiro_aux* – auxilia a operação *posição_teleponteiro* a verificar a posição de um teleponteiro pertencente a um determinado usuário;
- *existe_click_aux* – auxilia a operação *existe_click* na verificação se existe um determinado click definido.

Interface

existe_operação_aux (_,_): MAPEAMENTO, STRING → BOOL
formato_operação_aux (_,_): MAPEAMENTO, STRING → STRING
retorna_posição_nula_tele : → COORDENADA
posição_teleponteiro_aux (_,_): TP, STRING → COORDENADA
existe_click_aux (_,_,_):MAPEAMENTO, STRING, INTEGER → BOOL

Operações

Variáveis formais: **formato, formatos-restantes, posição-restantes: MAPEAMENTO**
click-restantes: MAPEAMENTO
operação, ícone, operação-a-verificar: STRING
identificador-usuário, id-usuário, botão, b: STRING
pos: COORDENADA
número, n: INTEGER

existe_operação_aux (formato, operação) =
if operação ∈ dom (formato)
then TRUE
else FALSE

formato_operação_aux (modify (operação, ícone, formatos-restantes), operação-a-verificar)=
if operação = operação-a-verificar
then ícone
else *formato_operação_aux* (formatos-restantes, operação-a-verificar)

retorna_coordenada_nula_tele =
(_,_)

posição_teleponteiro_aux ((Posição-Atual modify (identificador-usuário, pos, posição-restantes), _, _), id-usuário) =
if identificador-usuário = id-usuário
then pos
else *posição_teleponteiro_aux* ((Posição-Atual posição-restantes),_,_),id-usuário)

existe_click_aux (empty-mapping, _, _) =
FALSE

existe_click_aux (modify (b, n, click-restantes), botão, número) =
if b = botão **and** n = número
then TRUE
else *existe_click_aux* (click-restantes, botão, número)

1.3.2 Operações internas Modificadoras

As operações internas são as seguintes:

- *movimenta_teleponteiro_aux* – auxilia a operação *movimenta_teleponteiro* a movimentar o teleponteiro de um determinado usuário de uma posição para outra;
- *define_formato_operação_aux* – auxilia a operação *define_operação_formato* na definição da representação (ícone) do teleponteiro quando for realizar uma determinada operação;
- *altera_formato_operação_aux* – auxilia a operação *altera_formato_operação* a alterar a forma de representação (ícone) que o teleponteiro terá quando for realizar uma determinada operação;
- *inclui_usuário_aux* – auxilia a operação *inclui_usuário* na inserção de um usuário e a posição do seu teleponteiro;
- *exclui_usuário_aux* – auxilia a operação *exclui_usuário* na exclusão de um usuário e o seu teleponteiro;
- *exclui_click_aux* – auxilia a operação *exclui_click* na exclusão de um determinado click já definido.

Interface

movimenta_teleponteiro_aux (_,_,_):TP, STRING, COORDENADA → TP
define_formato_operação_aux (_,_,_):
STRING, STRING, MAPEAMENTO → MAPEAMENTO
altera_formato_operação_aux (_,_,_):
MAPEAMENTO, STRING, STRING → MAPEAMENTO
inclui_usuário_aux (_,_,_): STRING, STRING, MAPEAMENTO → MAPEAMENTO
exclui_usuário_aux (_,_): STRING, MAPEAMENTO → MAPEAMENTO
exclui_click_aux (_,_,_):MAPEAMENTO, STRING, INTEGER → MAPEAMENTO

Operações

Variáveis formais: identificador-usuário, id-usuário: STRING
operação, ícone, op, novo-ícone, botão, b: STRING
posição, nova-posição: COORDENADA
formato, formatos-restantes, posição-restantes: MAPEAMENTO
posições-usuários, click-restantes: MAPEAMENTO
número, n : INTEGER

movimenta_teleponteiro_aux ((Posição-Atual modify (identificador-usuário, posição, posiçao-restantes), _, _), id-usuário, nova-posição) =
if identificador-usuário = id-usuário
then (Posição-Atual modify (identificador-usuário, nova-posição, posiçao-restantes), _, _)
else *movimenta_teleponteiro_aux* ((Posição-Atual posiçao-restantes), _, _), id-usuário, nova-posição)

define_formato_operação_aux (operação, ícone, formato) =
modify (operação, ícone, formato)

altera_formato_operação_aux (modify (op, ícone, formatos-restantes), operação, novo-ícone) =
if op = operação
then modify (op, novo-ícone, formatos-restantes)
else *altera_formato_operação_aux* (formatos-restantes, operação, novo-ícone)

inclui_usuário_aux (id-usuário, posição, posições-usuários) =
modify (id-usuário, posição, posições-usuários)

exclui_usuário_aux (id-usuário, posições-usuários) =
posição \ id-usuário

exclui_click_aux (modify (b, n, click-restantes), botão, número) =
if b = botão **and** n = número
then modify (b, n, click-restantes) \ b
else *exclui_click_aux* (click-restantes, botão, número)

1.4 ATO AMBIENTE DE PERCEPÇÃO

.....As operações internas para o ATO Ambiente de Percepção pertencem à especificação das operações modificadoras.

1.4.1 Operações internas Modificadoras

As operações internas são as seguintes:

- *inclui_tipo_ambiente_aux* – auxilia a operação *inclui_tipo_ambiente* na inclusão de um tipo de ambiente (social, reunião ou conversa particular) no Ambiente de Percepção*;*
- *exclui_tipo_ambiente_aux* – auxilia a operação *exclui_tipo_ambiente* na exclusão de um tipo de ambiente (social, reunião ou conversa particular) do Ambiente de Percepção*;*
- *entrada_usuário_ambiente_aux* – auxila a operaçao *entrada_ambiente_aux* para que um usuário esteja presente em um determinado tipo de ambiente do Ambiente de Percepção*;*
- *saída_usuário_ambiente_aux* – auxila a operaçao *saída_ambiente_aux* para que um usuário se ausente de um determinado tipo de ambiente do Ambiente de Percepção*;*
- *saída_temp_usuário_ambiente_aux* – auxila a operaçao *saída_ambiente_aux* para que um usuário se ausente temporariamente de um determinado tipo de ambiente do Ambiente de Percepção.

Interface

inclui_tipo_ambiente_aux (_,_):CONJUNTO, MAPEAMENTO→ CONJUNTO
exclui_tipo_ambiente_aux (_,_):CONJUNTO, MAPEAMENTO→ CONJUNTO
entrada_usuário_ambiente_aux (_,_,_,_):
AA, STRING, MAPEAMENTO, MAPEAMENTO → AA
saída_usuário_ambiente_aux (_,_,_): AA, STRING, MAPEAMENTO → AA
saída_temp_usuário_ambiente_aux (_,_,_): AA, STRING, MAPEAMENTO → AA

Operações

Variáveis formais: tipos-ambientes: CONJUNTO
ambiente: MAPEAMENTO
id-usuário: STRING

inclui_tipo_ambiente_aux (tipos-ambientes, ambiente) =
add (tipos-ambientes, ambiente)

exclui_tipo_ambiente_aux (tipos-ambientes, ambiente) =
delete (tipos-ambientes, ambiente)

entrada_usuário_ambiente_aux ((_, Tipos-Ambientes tipos-ambientes), id-usuário, ambiente) =
(_, Tipos-Ambientes add (tipos-ambientes, modify (id-usuário, "Presente", ambiente)))

saída_usuário_ambiente_aux ((_, Tipos-Ambientes tipos-ambientes), id-usuário, ambiente) =
(_, Tipos-Ambientes add (tipos-ambientes, modify (id-usuário, "Ausente", ambiente)))

saída_temp_usuário_ambiente_aux ((_, Tipos-Ambientes tipos-ambientes), id-usuário, ambiente) =
(_, Tipos-Ambientes add (tipos-ambientes, modify (id-usuário, "Ausente-Temp", ambiente)))

1.5 ATO BLOCO DE NOTAS

A operação interna para o ATO Bloco de Notas pertence a uma operação modificadora da especificação.

1.5.1 Operação interna Modificadora

A operação interna é a seguinte:

- *envia_mensagem_aux* – esta operação auxilia a operação *envia_mensagem* inserindo uma mensagem no Bloco de Notas.

Interface

envia_mensagem_aux (_,_,_) : MAPEAMENTO, STRING, TEXTO → MAPEAMENTO

Operação

Variáveis formais: id-usuário: STRING
bloco-notas: MAPEAMENTO
mensagem: TEXTO

envia_mensagem_aux (bloco-notas, id-usuário, mensagem) =
modify (id-usuário, mensagem, bloco-notas)

1.6 ATO PAPEL PARA OS USUÁRIOS

As operações internas para o ATO Papel para os Usuários estão divididas seguindo a especificação das operações citadas na interface, em observadoras e modificadoras.

1.6.1 Operações internas observadoras

As operações internas são as seguintes:

- *existe_usuário_papel_excluído_aux* – verifica se existe um usuário que tenha um papel que foi excluído;
- *existe_usuário_aux* – auxilia a operação *existe_usuário* na verificação se um determinado usuário (seu identificador) possui uma papel;
- *papel_usuário_aux* – auxilia a operação *papel_usuário* na busca pelo papel de um usuário;
- *ícone_papel_aux* – auxilia a operação *ícone_papel* na busca pelo ícone que representa um determinado papel;
- *existe_papel_aux* – auxilia a operação *existe_papel* na verificação se um determinado papel já faz parte do contexto.

Interface

existe_usuário_papel_excluído_aux (_,_): MAPEAMENTO, STRING → BOOL
existe_usuário_aux (_,_): MAPEAMENTO, STRING → BOOL
papel_usuário_aux (_,_): MAPEAMENTO, STRING → STRING

ícone_papel_aux (_,_): MAPEAMENTO, STRING → STRING
existe_papel_aux (_,_): MAPEAMENTO, STRING → BOOL

Operações

Variáveis formais: tipos-usuários, tipos-usuários-restantes: MAPEAMENTO
rep-tipo-papel-restantes: MAPEAMENTO
id-usuário, identificador-usuário: STRING
papel, tipo-papel, ícone: STRING

existe_usuário_papel_excluído_aux (empty-mapping, _) =
FALSE

existe_usuário_papel_excluído_aux (tipos-usuários, papel) =
if papel ∈ rng (tipos-usuários)
then TRUE
else FALSE

existe_usuário_aux (tipos-usuários, id-usuário) =
if id-usuário ∈ dom (tipos-usuários)
then TRUE
else FALSE

papel_usuário_aux (modify (identificador-usuário, tipo-papel, tipos-usuários-restantes), id-usuário) =
if id-usuário = identificador-usuário
then tipo-papel
else *papel_usuário_aux* (tipos-usuários-restantes, id-usuário)

ícone_papel_aux (modify (tipo-papel, ícone, rep-tipo-papel-restantes), papel) =
if tipo-papel = papel
then ícone
else *ícone_papel_aux* (rep-tipo-papel-restantes, papel)

existe_papel_aux (empty-mapping, _) =
FALSE

existe_papel_aux (modify (tipo-papel, _, rep-tipo-papel-restantes), papel) =
if tipo-papel = papel
then TRUE
else *existe_papel_aux* (rep-tipo-papel-restantes, papel)

1.6.2 Operações internas Modificadoras

As operações internas são as seguintes:

- *inclui_usuário_papel_aux* – auxilia a operação *inclui_usuário_papel* na inclusão de um novo usuário e seu papel;
- *exclui_usuário_papel_aux* – auxilia na operação *exclui_usuário_papel* na eliminação de um usuário e seu papel;
- *inclui_papel_ícone_aux* – auxilia a operação *inclui_papel_ícone* na inclusão de um papel e sua forma de representação (através do ícone);
- *exclui_papel_ícone_aux* – auxilia a operação *exclui_papel_ícone* na exclusão de um papel e sua representação;
- *altera_papel_usuário_aux* – auxilia a operação *altera_papel_usuário* na alteração do papel de um usuário;
- *altera_ícone_papel_aux* – auxilia a operação *altera_ícone_papel* na alteração do ícone de um papel.

Interface

inclui_usuário_papel_aux (_,_,_):
MAPEAMENTO, STRING, STRING → MAPEAMENTO
exclui_usuário_papel_aux (_,_): MAPEAMENTO, STRING → MAPEAMENTO
inclui_papel_ícone_aux (_,_,_):
MAPEAMENTO, STRING, STRING → MAPEAMENTO
exclui_ papel_ícone_aux (_,_): MAPEAMENTO, STRING → MAPEAMENTO
altera_papel_usuário_aux (_,_,_):
MAPEAMENTO, STRING, STRING → MAPEAMENTO
altera_ícone_papel_aux (_,_,_):
MAPEAMENTO, STRING, STRING → MAPEAMENTO

Operações

Variáveis formais: id-usuário, papel, ícone, tipo-papel, identificador-usuário: STRING
novo-papel, novo-ícone: STRING
tipos-usuário, rep-tipo-papel: MAPEAMENTO
rep-tipo-papel-restantes: MAPEAMENTO

inclui_usuário_papel_aux (tipos-usuários, id-usuário, papel) =
modify (id-usuário, papel, tipos-usuários)

exclui_usuário_papel_aux (tipos-usuários, id-usuário) =
tipos-usuários \ id-usuário

inclui_papel_ícone_aux (rep-tipo-papel, papel, ícone) =
modify (papel, ícone, rep-tipo-função)

exclui_papel_ícone_aux (modify (tipo-papel, _, rep-tipo-papel-restantes), papel) =
if tipo-papel = papel
then rep-tipo-papel \ papel
else *exclui_papel_ícone_aux* (rep-tipo-papel-restantes, papel)

altera_papel_usuário_aux (modify (identificador-usuário, tipo-papel, tipos-usuários-restantes), id-usuário, novo-papel) =
if identificador-usuário = id-usuário
then modify (id-usuário, novo-papel, tipos-usuários-restantes)
else *altera_papel_usuário_aux* (tipos-usuários-restantes, id-usuário, novo-papel)

altera_ícone_papel_aux (papel, novo-ícone , rep-tipo-papel,) =
modify (papel, novo-ícone, rep-tipo-papel)

1.7 ATO BARRA DE ROLAGEM DE SINCRONIZAÇÃO

As operações internas para o ATO Barra de Rolagem de Sincronização estão divididas seguindo a especificação das operações citadas na interface, em observadoras e modificadoras.

1.7.1 Operações internas observadoras

As operações internas são as seguintes:

- *existe_usuário_aux* – auxilia a operação *existe_usuário* na verificação se um determinado usuário faz parte da Barra de Rolagem de Sincronização;
- *posição_barra_usuário_aux* – verifica qual a posição da Barra de Rolagem de Sincronização de um determinado usuário.

Interface

existe_usuário_aux (_,_): MAPEAMENTO, STRING → BOOL
posição_barra_usuário_aux (_,_): BRS, STRING → COORDENADA

Operações

Variáveis formais: barra, barra-restantes: MAPEAMENTO
id-usuário, identificador-usuário: STRING]
posição: COORDENADA

existe_usuário_aux (barra, id-usuário) =
if id-usuário ∈ dom (barra)
then TRUE
else FALSE

posição_barra_usuário_aux ((Barra-Rol-Sinc modify (identificador-usuário, posição, barra-restantes)), id-usuário) =
if identificador-usuário = id-usuário
then posição
else *posição_barra_usuário_aux* ((Barra-Rol-Sinc barra-restantes), id-usuário)

1.7.2 Operações internas Modificadoras

As operações internas são as seguintes:

- *inclui_usuário_aux* – auxilia a operação *inclui_usuário* na inclusão de um determinado usuário e sua cor representativa na Barra de Rolagem de Sincronização;
- *exclui_usuário_aux* – auxilia a operação *exclui_usuário* na exclusão de um determinado usuário e sua cor representativa na Barra de Rolagem de Sincronização;
- *movimenta_barra_aux* – auxilia a operação *movimenta_barra* a movimentar a barra de um determinado usuário de uma posição para outra.

Interface

inclui_usuário_aux (_,_,_):
MAPEAMENTO, STRING, COORDENADA → MAPEAMENTO
exclui_usuário_aux (_,_):MAPEAMENTO, STRING → MAPEAMENTO
movimenta_barra_aux (_,_,_):
MAPEAMENTO, STRING, COORDENADA → MAPEAMENTO

Operações

Variáveis formais: barra, barra-restantes: MAPEAMENTO
id-usuário, identificador-usuário: STRING
posição, nova-posição: COORDENADA

inclui_usuário_aux (barra, id-usuário, posição) =
modify (id-usuário, posição, barra)

exclui_usuário_aux (rep-usuários, id-usuário) =
rep-usuários \ id-usuário

movimenta_barra_aux (modify (identificador-usuário, posição, barra-restantes), id-usuário, nova-posição) =
if identificador-usuário = id-usuário
then modify (identificador-usuário, nova-posição, barra-restantes)
else *movimenta_barra_aux* (barra-restantes, id-usuário, nova-posição)

1.8 ATO JANELAS AUXILIARES

As operações internas para o ATO Teleponteiros estão divididas seguindo a especificação das operações citadas na interface, em observadoras e modificadoras.

1.8.1 Operação interna Observadora

A operação interna é a seguinte:

- *tipo_janela_usuário_aux* – auxilia a operação *tipo_janela_usuário* a verificar o tipo de janela que um determinado usuário está usando.

Interface

tipo_janela_usuário_aux (_,_): MAPEAMENTO, STRING → STRING

Operação

Variáveis Formais: id-usuário, identificador-usuário, tipo-janela: STRING
janela-usuário-restantes: MAPEAMENTO

tipo_janela_usuário_aux (modify (identificador-usuário, tipo-janela, janela-usuário-restantes), id-usuário) =
if identificador-usuário = id-usuário
then tipo-janela
else *tipo_janela_usuário_aux* (janela-usuário-restantes), id-usuário)

1.8.2 Operações internas Modificadoras

As operações internas são as seguintes:

- *inclui_usuário_aux* – auxilia a operação *inclui_usuário* na inclusão de um usuário na sessão de trabalho cooperativa;
- *exclui_usuário_aux* – auxilia a operação *exclui_usuário* na exclusão de um usuário da sessão de trabalho cooperativa;
- *altera_tipo_janela_aux* – auxilia a operação *altera_tipo_janela* a trocar o tipo de janela de um determinado usuário.

Interface

inclui_usuário_aux (_,_,_): MAPEAMENTO, STRING, STRING → MAPEAMENTO
exclui_usuário_aux (_,_): MAPEAMENTO, STRING → MAPEAMENTO
altera_tipo_janela_aux (_,_,_):MAPEAMENTO, STRING, STRING → MAPEAMENTO

Operações

Variáveis formais: janela-usuário, janela-usuário-restantes: MAPEAMENTO
id-usuário, tipo-janela, novo-tipo-janela: STRING
identificador-usuário: STRING

inclui_usuário_aux (janela-usuário, id-usuário, tipo-janela) =
modify (id-usuário, tipo-janela, janela-usuário)

exclui_usuário_aux (janela-usuário, id-usuário) =
janela-usuário \ id-usuário

altera_tipo_janela_aux (modify (identificador-usuário, tipo-janela, janela-usuário-restantes), id-usuário, novo-tipo-janela) =
if identificador-usuário = id-usuário
then modify (identificador-usuário, novo-tipo-janela, janela-usuário-restantes)
else *altera_tipo_janela_aux* (janela-usuário-restantes, id-usuário, novo-tipo-janela)

ANEXO 2 OPERAÇÕES PARA O DESENVOLVEDOR

Neste anexo são apresentadas as operações referentes ao desenvolvedor. Estas operações devem ser construídas quando o desenvolvedor utilizar um ou mais Componentes de Percepção juntamente com o ATO Preferências dos Usuários.

2.1 OBSERVAÇÕES GERAIS

Algumas observações são essenciais na construção de ATOs cooperativos utilizando Componentes de Percepção. São elas:

- A ferramenta que está sendo construída pode utilizar mais de um Componente de Percepção que utiliza ícones para representar alguma atividade (tal como os Teleponteiros e Papel para os Usuários), neste caso as operações construídas devem verificar também com estes ATOs, além do ATO Preferências dos Usuários, se um determinado ícone já está sendo utilizado;
- Se desejar incluir um usuário em algum componente e ele não pertencer ao objeto de Preferências dos Usuários, deve primeiramente incluí-lo no Preferências dos Usuários para depois ser possível incluir este usuário em qualquer componente desejado;
- Quando excluir um usuário de um componente, normalmente, será necessário excluí-lo do ATO Preferências dos Usuários como também de qualquer outro componente de percepção que a ferramenta estiver utilizando;
- No momento de inclusão de um usuário no Preferências dos Usuários deve também verificar se suas representações já estão sendo utilizadas por outro usuário ou por representações de atividades nos componentes.

Essas considerações podem ficar a critério do desenvolvedor, sendo seguidas ou não. Porém, para uma maior integração das ferramentas utilizadas no PROSOFT Cooperativa, seria interessante o uso dessas observações.

2.2 CRIAÇÃO DOS COMPONENTES

No momento que o desenvolvedor criar sua ferramenta (operação *cria*), deve também criar os componentes de percepção que utilizar e a classe Preferências dos Usuários. Isso é feito chamando a operação *cria* de cada componente de percepção utilizado através de uma chamada **ICS**.

2.3 OPERAÇÕES PARA AS PREFERÊNCIAS DOS USUÁRIOS

É necessário especificar as seguintes operações:

Observadoras

- *existe_usuário_pu* – esta operação verifica se um determinado usuário possui representação. Para isso é necessário chamar a operação *existe_usuário*, via **ICS**, do ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário;
- *representação_usuário* – verifica qual a forma de representação de um determinado usuário. Para tanto é necessário chamar a operação *representação_usuário*, via **ICS**, do ATO Preferências dos Usuários e passar como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário;
- *ícone_usuário* – esta operação informa qual o ícone representativo de um determinado usuário. Para tanto é necessário chamar a operação *ícone_usuário*, via **ICS**, do ATO Preferências dos Usuários e passar como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário;
- *existe_ícone* – verifica se um determinado ícone já pertence a algum usuário. Para tanto é necessário chamar a operação *existe_ícone*, via **ICS**, do ATO Preferências dos Usuários e passar como seletor o objeto do tipo Preferências dos Usuários e como argumento o ícone a ser verificado;
- *existe_label* – verifica se um determinado label já pertence a algum usuário. Para tanto é necessário chamar a operação *existe_label*, via **ICS**, do ATO Preferências dos Usuários e passar como seletor o objeto do tipo Preferências dos Usuários e como argumento o label a ser verificado;
- *existe_cor* – verifica se uma determinada cor já pertence a algum usuário. Para tanto é necessário chamar a operação *existe_cor*, via **ICS**, do ATO Preferências dos Usuários e passar como seletor o objeto do tipo Preferências dos Usuários e como argumento a cor a ser verificada.

Modificadoras

- *inlcui_usuário_label* - esta operação inclui um usuário, seu label e ícone representativos. Para isso é necessário verificar se esse usuário pertence ao Prosoft Cooperativo, para isso deve-se chamar a operação *existe*_usuário, via **ICS**, do ATO Prosoft Cooperativo. Caso o usuário exista então é chamada a operação *inclui_usuário_label*, via **ICS**, do ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como argumentos o identificador do usuário, seu label e seu ícone. Caso contrário o objeto é devolvido sem modificações;
- *inlcui_usuário_cor* - esta operação inclui um usuário, sua cor e ícone representativos. Para isso é necessário verificar se esse usuário pertence ao Prosoft Cooperativo, para isso deve-se chamar a operação *existe*_usuário, via **ICS**, do ATO Prosoft Cooperativo. Caso o usuário exista então é chamada a operação *inclui_usuário_cor*, via **ICS**, do ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como argumentos o identificador do usuário, sua cor e seu ícone. Caso contrário, o objeto é devolvido sem modificações;
- *exclui_usuário_pu* – esta operação exclui um determinado usuário. Para isso é necessário chamar a operação *exclui_usuário*, via **ICS**, do ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário;
- *altera_cor* – altera a cor representativa de um determinado usuário. Primeiro deve-se verificar se o usuário existe, através da operação *existe_usuário_pu*. Caso afirmativo, deve-se chamar a operação *altera_cor*, via **ICS**, do ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como argumentos o identificador do usuário e a nova cor. Caso contrário o objeto retorna sem modificações;
- *altera_label* – altera o label representativo de um determinado usuário. Primeiro deve-se verificar se o usuário existe, através da operação *existe_usuário_pu*. Caso afirmativo, deve-se chamar a operação *altera_label*, via **ICS**, do ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como argumentos o identificador do usuário e o novo label. Caso contrário o objeto retorna sem modificações;
- *altera_ícone* – altera o ícone representativo de um determinado usuário. Primeiro deve-se verificar se o usuário existe, através da operação *existe_usuário_pu*. Caso afirmativo, deve-se chamar a operação *altera_ícone*, via **ICS**, do ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como argumentos o identificador do usuário e o novo ícone. Caso contrário o objeto retorna sem modificações.

2.4 OPERAÇÕES PARA OS MÚLTIPLOS CURSORES

Para este componente é necessário especificar as seguintes operações:

OBSERVADORAS

- *posição_cursor* – esta operação retorna a posição do cursor de um determinado usuário. Para isso deve chamar a operação *posição_cursor*, via **ICS**, do ATO Múltiplos Cursores passando como seletor o objeto do tipo Múltiplos Cursores e como argumento o identificador do usuário;
- *existe_usuário_mc* – verifica se um determinado usuário possui um múltiplo cursor. Para tanto é necessário enviar uma chamada para operação *existe_usuário*, via **ICS**, para o ATO Múltiplos Cursores passando como seletor o objeto do tipo Múltiplo Cursor e como argumento o identificador do usuários.

MODIFICADORAS

- *movimenta_cursor* – esta operação movimenta o cursor de um determinado usuário de uma posição para outra. Para isso é necessário verificar se esse usuário possui sua representação nas Preferências dos Usuários enviando uma chamada para operação *existe_usuário*, via **ICS**, para o ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário. Caso esse usuário exista então é enviada uma chamada para a operação *movimenta_cursor*, via **ICS**, do ATO Múltiplos Cursores passando como seletor o objeto do tipo Múltiplos Cursores e como argumentos o identificador do usuário e a nova posição que o cursor estará localizado;
- *inclui_usuário_mc* – esta operação torna possível que um usuário tenha um múltiplo cursor. Para isso é necessário verificar se este usuário já possui um cursor através da operação *existe_usuário_mc* e também verificar se ele está presente nas Preferências dos Usuários, através da chamada da operação *existe_usuário*, via **ICS**, do ATO Preferências dos Usuários, passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário. Caso este usuário não possua um cursor e tenha sua representação nos Preferências dos Usuários é chamada então a operação *inclui_usuário*, via **ICS**, do ATO Múltiplos Cursores passando como seletor o objeto do tipo Múltiplos Cursores e como argumento o identificador do usuário e a posição do cursor. Caso contrário, o objeto retorna sem modificações;
- *exclui_usuário_mc* - esta operação exclui o cursor de um usuário. Para isso é necessário verificar se este usuário possui um cursor através da operação *existe_usuário_mc*. Caso este usuário tenha um cursor é chamada então a operação *exclui_usuário*, via **ICS**, do ATO Múltiplos Cursores passando como seletor o objeto do tipo Múltiplos Cursores e como argumento o identificador do usuário. Caso contrário, o objeto retorna sem modificações.

2.5 OPERAÇÕES PARA OS TELEPONTEIROS

Para este componente é necessário especificar as seguintes operações:

OBSERVADORAS

- *posição_teleponteiro* - esta operação retorna a posição do telponteiro de um determinado usuário. Para isso deve chamar a operação *posição_teleponteiro*, via **ICS**, do ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumento o identificador do usuário;
- *existe_operação* – esta operação verifica se existe um operação que tenha uma forma diferente de representação no teleponteiro. Para isso é necessário enviar uma chamada **ICS** para a operação *existe_operação* do ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumento a operação que se deseja verificar;
- *formato_operação* – esta operação verifica qual o formato (ícone) do teleponteiro quando o usuário estiver realizando uma determinada operação. Para tanto deve-se chamar a operação *formato_operação*, via **ICS**, do ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumento a operação que se deseja verificar;
- *características_clikc* - esta operação verifica quais as características do click do mouse. Para tanto deve-se chamar a operação *características_click*, via **ICS**, do ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros;
- *existe_formato* - esta operação verifica se um formato (ícone) já existe. Para tanto deve-se chamar a operação *existe_formato*, via **ICS**, do ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumento o formato que se deseja verificar;
- *existe_usuário_tp* – verifica se um determinado usuário possui teleponteiro. Para isso é necessário enviar uma chamada para a operação *existe_usuário*, via **ICS**, do ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumento o identificador do usuário;
- *existe_click* – verifica se um determinado formato de click já foi definido. Para isso é enviada uma chamada para a operação *existe_click*, via **ICS**, do ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumentos o botão e o número de clicks.

MODIFICADORAS

- *movimenta_teleponteiro* – esta operação movimenta o teleponteiro de um determinado usuário. Para isso é necessário verificar se esse usuário possui sua representação no Preferências dos Usuários, enviando uma chamada, via **ICS**, para a operação *existe_usuário* do ATO Preferências dos Usuários, passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário. Caso ele já tenha sua representação então é enviada, via **ICS**, uma chamada para a operação

movimenta_teleponteiro do ATO Teleponteiros, passando como seletor o objeto do tipo Teleponteiros e como argumentos o identificador do usuário e a nova posição que o teleponteiro irá assumir. Caso contrário, o objeto retorna sem modificações;

- *define_formato_operação* – esta operação define o formato (ícone que o teleponteiro irá assumir) quando o usuário for realizar uma determinada tarefa. Para isso, deve-se primeiro verificar se o ícone já não pertence a algum usuário. Esta verificação se faz enviando uma chamada para a operação *existe_ícone*, via **ICS**, para o ATO Preferências dos Usuários. Caso este ícone não seja a representação de algum usuário é então enviada uma chamada para a operação *define_formato_operação*, via **ICS**, para o ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumentos a operação e o formato que ela irá assumir. Caso contrário, o objeto é retornado sem modificações;
- *altera_formato_operação* - esta operação altera o formato (ícone que o teleponteiro irá assumir) quando o usuário for realizar uma determinada tarefa. Para isso, deve-se primeiro verificar se o ícone já não pertence a algum usuário. Esta verificação se faz enviando uma chamada para a operação *existe_ícone*, via **ICS**, para o ATO Preferências dos Usuários. Caso este ícone não seja a representação de algum usuário é então enviada uma chamada para a operação *altera_formato_operação*, via **ICS**, para o ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumentos a operação e o novo formato que ela irá assumir. Caso contrário, o objeto é retornado sem modificações;
- *inclui_click* - esta operação inclui um formato de click do mouse quando o usuário estiver usando o teleponteiro. Primeiramente é verificado se já existe esse click definido, a partir da operação *existe_click*. Se não existe, então é enviada uma chamada para a operação *inclui_click*, via **ICS**, para o ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumentos o botão do mouse que irá ser pressionado (direito ou esquerdo) e o número de clicks no mouse (um ou dois). Caso contrário, o objeto é retornado sem modificações;
- *exclui_click* – esta operação exclui um dos formatos adotas pelo click. Primeiro é necessário verificar se este formato de click existe, através da operação *existe_click*. Se existe então é chamada a operação *exclui_click*, via **ICS**, do ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumentos o botão e o número de clicks. Caso contrário, o objeto é devolvido sem modificações;
- *inclui_teleponteiro* - esta operação torna possível que um usuário tenha um teleponteiro. Para isso é necessário verificar se este usuário está presente nas Preferências dos Usuários, através da chamada da operação *existe_usuário*, via **ICS**, do ATO Preferências dos Usuários, passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário. Caso este usuário tenha sua representação nos Preferências dos Usuários é chamada então a operação *inclui_teleponteiro*, via **ICS**, do ATO Teleponteiros passando como seletor o objeto do tipo Teleponteiros e como argumento o identificador do usuário e a posição do teleponteiro. Caso contrário, o objeto retorna sem modificações;
- *exclui_teleponteiro* - esta operação o teleponteiro de um determinado usuário. Para isso é chamada a operação *exclui_teleponteiro*, via **ICS**, do ATO Teleponteiros passando

como seletor o objeto do tipo Teleponteiros e como argumento o identificador do usuário. Caso contrário, o objeto retorna sem modificações.

2.6 OPERAÇÕES PARA O AMBIENTE DE PERCEPÇÃO

Para este componente é necessário especificar as seguintes operações:

Observadoras

- *quais tipos ambientes* – verifica quais os tipos de ambientes que formam o Ambiente de Percepção. Para isso, é necessário chamar a operação *quais_tipos_ambientes*, via **ICS**, do ATO Ambiente de Percepção, passando somente o seletor Ambiente de Percepção;
- *usuário_pertence_ambiente* – verifica se um determinado usuário pertence a algum ambiente. Para isso, basta enviar uma mensagem, via **ICS**, para o ATO Ambiente de Percepção chamando a operação *usuário_pertence_ambiente*, passando como seletor o objeto do Ambiente de Percepção e como parâmetros o identificador do usuário a ser verificado e o ambiente;
- *tipo_representação_movimento* – verifica qual a forma de representação do movimento de entrada, saída e saída temporária dos ambientes do Ambiente de Percepção. Para isso, é necessário chamar a operação *tipo_representação_ambiente*, via **ICS**, passando como seletor o objeto Ambiente de Percepção;
- *existe_ambiente* – verifica se um determinado ambiente pertence ao Ambiente de Percepção. Sendo assim, é necessário chamar a operação *existe_ambiente*, via **ICS**, do ATO Ambiente de Percepção, passando como seletor o objeto Ambiente de Percepção e como parâmetro o ambiente que se deseja verificar;
- *ícone_saída_temp* – verifica qual o ícone que representa a saída temporária de um usuário de um dos ambientes que formam o Ambiente de Percepção. Assim, é necessário chamar a operação *ícone_saída_temporária*, via **ICS**, passando como seletor o objeto do tipo Ambiente de Percepção.

Modificadoras

- *define_representação_movimento_ícone* – esta operação define como forma de representação dos movimentos (entrada, saída e saída temporária), ícones representativos. Para tanto, é necessário primeiramente verificar se o ícone que irá representar a saída temporária já não pertence a algum usuário, isso através da operação *existe_ícone*, chamada via **ICS**, do ATO Preferências dos Usuários, passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o ícone a ser verificado. Caso o ícone não pertença a um usuário, é então chamada, via **ICS**, a operação *define_representação_movimento_ícone* do ATO Ambiente de Percepção, passando como seletor o objeto do tipo Ambiente de Percepção e como argumento o

ícone de saída temporária. Caso já exista o ícone então o objeto retornará sem modificações;

- *define_representação_movimento_label* – esta operação define como forma de representação dos movimentos (entrada, saída e saída temporária), labels. Para tanto, é necessário primeiramente verificar se os labels que irão representar a entrada, saída e saída temporária já não pertencem a algum usuário, isso através da operação *existe_label*, chamada via **ICS**, do ATO Preferências dos Usuários. Para cada chamada é passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o label a ser verificado (um de cada vez, para presente, ausente e ausente temporariamente). Caso os label não pertencerem a um usuário, é então chamada, via **ICS**, a operação *define_representação_movimento_label* do ATO Ambiente de Percepção, passando como seletor o objeto do tipo Ambiente de Percepção e como argumentos os labels de presente, ausente e de saída temporária. Caso já existam pelo menos um dos labels então o objeto retornará sem modificações;

- *altera_ representação_movimento* – esta operação altera o tipo de representação dos movimentos (entrada, saída e saída temporária) dos usuários nos ambientes que formam o Ambiente de Percepção. Para isso é necessário verificar se o tipo de representação atual não é o mesmo para o qual está sendo modificado. Isso é feito através da operação *tipo_representação_movimento* que verifica o tipo de representação do movimento. Caso a nova representação não seja igual a atual, então é chamada a operação *altera_representação_movimento*, via **ICS**, do ATO Ambiente de Percepção passando como seletor o objeto do tipo Ambiente de Percepção e como parâmetro o novo tipo de representação. Caso contrário, o objeto é devolvido sem modificações;

- *inclui_tipo_ambiente* – esta operação inclui um tipo de ambiente no Ambiente de Percepção. Para isso primeiramente verifica-se se o novo tipo de ambiente já não existe, através da operação *existe_ambiente*. Caso o ambiente ainda não pertença ao Ambiente de Percepção, então é chamada, via **ICS**, a operação *inclui_tipo_ambiente* do ATO Ambiente de Percepção e passado como seletor o objeto do tipo Ambiente de Percepção e como argumento o tipo do ambiente a ser inserido. Caso contrário, o objeto é retornado sem modificações;

- *exclui_tipo_ambiente* – esta operação exclui um tipo de ambiente do Ambiente de Percepção. Para isso primeiramente verifica-se se o ambiente já existe, através da operação *existe_ambiente*. Caso o ambiente pertença ao Ambiente de Percepção, então é chamada, via **ICS**, a operação *exclui_tipo_ambiente* do ATO Ambiente de Percepção e passado como seletor o objeto do tipo Ambiente de Percepção e como argumento o tipo do ambiente a ser excluído. Caso contrário, o objeto é retornado sem modificações;

- *entrada_usuário_ambiente* – esta operação insere um usuário em um dos ambientes que formam o Ambiente de Percepção. Para isso é necessário verificar se o usuário já possui sua representação no ATO Preferências dos Usuários, através da chamada da operação *existe_usuário*, via **ICS**, passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário a ser verificado. Caso o usuário já exista no Preferências dos Usuários então é chamada a operação *entrada_usuário_ambiente*, via **ICS**, do ATO Ambiente de Percepção passando como seletor o objeto do tipo Ambiente de Percepção e como argumentos o identificador do

usuário e o ambiente no qual ele será inserido. Caso contrário, o objeto é retornado sem modificações;

- *saída_usuário_ambiente* - esta operação exclui um usuário de um determinado ambiente pertencente ao Ambiente de Percepção. Para isso é necessário verificar se o usuário pertence ao ambiente pelo qual irá ser retirado através da operação *usuário_pertence_ambiente*. Caso o usuário pertença ao ambiente então é chamada a operação *saída_usuário_ambiente*, via **ICS**, do ATO Ambiente de Percepção passando como seletor o objeto do tipo Ambiente de Percepção e como argumentos o identificador do usuário e o ambiente no qual ele será excluído. Caso contrário, o objeto é retornado sem modificações;
- *saída_temp_usuário_ambiente* - esta operação tira temporáriamente um usuário de um determinado ambiente pertencente ao Ambiente de Percepção. Para isso é necessário verificar se o usuário pertence ao ambiente pelo qual irá ser retirado temporariamente através da operação *usuário_pertence_ambiente*. Caso o usuário pertença ao ambiente então é chamada a operação *saída_temp_usuário_ambiente*, via **ICS**, do ATO Ambiente de Percepção passando como seletor o objeto do tipo Ambiente de Percepção e como argumentos o identificador do usuário e o ambiente no qual ele será retirado temporariamente. Caso contrário, o objeto é retornado sem modificações;
- *altera_ícone_saída_temp* – esta operação altera o ícone que representa a saída temporária de um usuário de um ambiente pertencente ao Ambiente de Percepção. Para isso é necessário verificar se o novo ícone é a representação de algum usuário, então é chamada, via **ICS**, a operação *existe_ícone* do ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como parâmetro o novo ícone. Caso não exista, então é chamada a operação *altera_ícone_saída_temp*, via **ICS**, do ATO Ambiente de Percepção passando como seletor o objeto do tipo Ambiente de Percepção e como argumento o novo ícone. Caso contrário, o objeto é devolvido sem modificações.

2.7 OPERAÇÕES PARA O BLOCO DE NOTAS

Para este componente é necessário especificar duas operações:

Observadora

- *usuário_enviou_mensagem* – verifica se um determinado usuário já enviou alguma mensagem para o Bloco de Notas. Neste caso, é necessário somente fazer uma chamada, via **ICS**, para a operação *usuário_enviou_mensagem* do Bloco de Notas, passando como seletor o Bloco de Notas e como parâmetro o identificador do usuário a ser verificado;

Modificadora

- *envia_mensagem* – esta operação possibilita a um usuário enviar uma mensagem para o Bloco de Notas. Para isso, se faz necessário que esta operação primeiro verifique se o usuário que enviará a mensagem está presente no objeto das Preferências dos Usuários, chamando a operação *existe_usuário* do ATO Preferências dos Usuários através de uma **ICS**. Caso afirmativo, é enviado uma chamada, via **ICS**, para o Bloco de Notas chamando a operação *envia_mensagem*, passando como seletor o Bloco de Notas e como parâmetros o identificador do usuário e a mensagem a ser enviada. Caso contrário, o objeto retorna sem modificações.

2.8 OPERAÇÕES PARA O PAPEL PARA OS USUÁRIOS

Para este componente é necessário especificar as seguintes operações:

Observadoras

- *existe_usuário_ppu* - verifica se um determinado usuário possui um papel específico. Sendo assim, é necessário chamar a operação *existe_usuário*, via **ICS**, do ATO Papel para Usuários, passando como seletor o objeto do tipo Papel para os Usuários e como parâmetro o identificador do usuário;
- *papel_usuário* – verifica qual o papel que um determinado usuário possui. Para tanto é necessário enviar uma mensagem, via **ICS**, para a operação *papel_usuário* do ATO Papel para os Usuários passando como seletor o objeto do tipo Papel para os Usuários e como argumento o identificador do usuário;
- *existe_papel* – verifica se um determinado papel já possui sua representação (ícone). Para isso é necessário fazer a chamada da operação *existe_papel*, via **ICS**, para o ATO Papel para os Usuários passando como seletor o objeto do tipo Papel para os Usuários e como argumento o papel que se quer verificar;
- *ícone_papel* – verifica qual o ícone que um determinado papel possui. Para tanto é necessário enviar uma mensagem, via **ICS**, para a operação *ícone_papel* do ATO Papel para os Usuários passando como seletor o objeto do tipo Papel para os Usuários e como argumento o papel que se quer verificar;
- *existe_ícone* – verifica se um determinado ícone já é a representação de algum papel. Para isso é necessário fazer a chamada da operação *existe_ícone*, via **ICS**, para o ATO Papel para os Usuários passando como seletor o objeto do tipo Papel para os Usuários e como argumento o ícone que se quer verificar.

Modificadoras

- *inclui_usuário_papel* - esta operação torna possível que um usuário tenha um papel e sua representação. Para isso é necessário verificar se este usuário está presente nas

Preferências dos Usuários, através da chamada da operação *existe_usuário*, via **ICS**, do ATO Preferências dos Usuários, passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário. Caso este usuário tenha sua representação nos Preferências dos Usuários é chamada então a operação *inclui_usuário_papel*, via **ICS**, do ATO Papel para os Usuários passando como seletor o objeto do tipo Papel para os Usuários e como argumento o identificador do usuário e o papel que ele irá assumir. Caso contrário, o objeto retorna sem modificações;

- *exclui_usuário_papel* - esta operação exclui o usuário e seu papel. Para isso é necessário verificar se este usuário possui um papel através da operação *existe_usuário_ppu*. Caso este usuário tenha um papel é chamada então a operação *exclui_usuário_papel*, via **ICS**, do ATO Papel para os Usuários passando como seletor o objeto do tipo Papel para os Usuários e como argumento o identificador do usuário. Caso contrário, o objeto retorna sem modificações;
- *inclui_papel_ícone* – esta operação inclui um tipo de papel e sua representação (ícone). Primeiro é necessário verificar se o ícone que vai ser inserido já não pertence a algum usuário, para isso se faz uma chamada para a operação *existe_ícone*, via **ICS**, do ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o ícone a ser verificado. Caso não exista, é então chamada a operação *inclui_papel_ícone*, via **ICS**, para o ATO Papel para os Usuários passando como seletor o objeto do tipo Papel para os Usuários e como argumentos o papel e o ícone que irá representá-lo. Caso contrário, o objeto é devolvido sem modificações;
- *exclui_papel_ícone* – esta operação exclui o papel e sua representação. Para tanto é necessário chamar a operação *exclui_papel_ícone*, via **ICS**, do ATO Papel para os Usuários, passando como seletor o objeto do tipo Papel para os Usuários e como argumento o papel a ser excluído;
- *altera_papel_usuário* - esta operação altera o papel de um determinado usuário. Para tanto é necessário chamar a operação *altera_papel_usuário*, via **ICS**, do ATO Papel para os Usuários, passando como seletor o objeto do tipo Papel para os Usuários e como argumentos o identificador do usuário a ter o papel alterado e o novo papel;
- *altera_ícone_papel* - esta operação altera o ícone (representação) de um determinado papel. Primeiro é necessário verificar se o ícone que vai ser inserido já não pertence a algum usuário, para isso se faz uma chamada para a operação *existe_ícone*, via **ICS**, do ATO Preferências dos Usuários passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o ícone a ser verificado. Caso não exista, é então chamada a operação *altera_ícone_papel*, via **ICS**, para o ATO Papel para os Usuários passando como seletor o objeto do tipo Papel para os Usuários e como argumentos o papel e o novo ícone que irá representá-lo. Caso contrário, o objeto é devolvido sem modificações.

2.9 OPERAÇÕES PARA A BARRA DE ROLAGEM DE SINCRONIZAÇÃO

Para este componente é necessário especificar as seguintes operações:

Observadoras

- *existe_usuário_brs* – esta operação verifica se um determinado usuário está presente na barra de rolagem de sincronização. Para isso é necessário enviar uma chamada para a operação *existe_usuário*, via **ICS**, para o ATO Barra de Rolagem de Sincronização passando como seletor o objeto do tipo Barra de Rolagem de Sincronização e como argumento o identificador do usuário;
- *posição_barra_usuário* – esta operação verifica onde está posicionada a barra de um determinado usuário. Para isso deve-se enviar uma chamada para operação *posição_barra_usuário*, via **ICS**, para o ATO Barra de Rolagem de Sincronização e passar como seletor o objeto do tipo Barra de Rolagem de Sincronização e como argumento o identificador do usuário.

Modificadoras

- *inclui_usuário_brs* - esta operação inclui um usuário na Barra de Rolagem de Sincronização. Para isso é necessário verificar se este usuário está presente nas Preferências dos Usuários, através da chamada da operação *existe_usuário*, via **ICS**, do ATO Preferências dos Usuários, passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário. Caso este usuário tenha sua representação nos Preferências dos Usuários é chamada então a operação *inclui_usuário*, via **ICS**, do ATO Barra de Rolagem de Sincronização passando como seletor o objeto do tipo Barra de Rolagem de Sincronização e como argumento o identificador do usuário e a posição que ele irá assumir. Caso contrário, o objeto retorna sem modificações;
- *exclui_usuário_brs* - - esta operação exclui o usuário da Barra de Rolagem de Sincronização. Para isso é necessário verificar se este usuário está presente na barra de rolagem de sincronização, através da operação *existe_usuário_brs*. Caso este usuário esteja presente é chamada então a operação *exclui_usuário*, via **ICS**, do ATO Barra de Rolagem de Sincronização passando como seletor o objeto do tipo Barra de Rolagem de Sincronização e como argumento o identificador do usuário. Caso contrário, o objeto retorna sem modificações;
- *movimenta_barra* - esta operação movimenta a barra de rolagem de um determinado usuário. Para isso é necessário verificar se esse usuário possui sua representação no Preferências dos Usuários, enviando uma chamada, via **ICS**, para a operação *existe_usuário* do ATO Preferências dos Usuários, passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário. Caso ele já tenha sua representação então é enviada, via **ICS**, uma chamada para a operação *movimenta_barra* do ATO Barra de Rolagem de Sincronização, passando como seletor o objeto do tipo Barra de Rolagem de Sincronização e como argumentos o identificador

do usuário e a nova posição que a barra irá assumir. Caso contrário, o objeto retorna sem modificações.

2.10 OPERAÇÕES PARA AS JANELAS AUXILIARES

Para este componente é necessário especificar as seguintes operações:

Observadoras

- *tipo_janela_usuário* – verifica qual o tipo de janela (WYSIWID, *Radar View* e *Miniature View*) pertence a um determinado usuário. Para isso, primeiro é verificado se o usuário possui um janela auxiliar, através da operação *existe_usuário_já*. Caso esse usuário tenha uma janela auxiliar então é enviada uma chamada para operação *tipo_janela_usuário*, via **ICS**, para o ATO Janelas Auxiliares passando como seletor o objeto do tipo Janelas Auxiliares e como argumento o identificador do usuário;
- *existe_usuário_já* – verifica se um determinado usuário possui uma janela auxiliar. Para isso é enviada uma chamada, via **ICS**, para a operação *existe_usuário* do ATO Janelas Auxiliares passando como seletor o objeto do tipo Janelas Auxiliares e como argumento o identificador do usuário.

Modificadoras

- *inclui_usuário_ja* – esta operação torna possível que um usuário possua uma janela auxiliar. Para isso é necessário verificar se este usuário já possui uma janela através da operação *existe_usuário_ja* e também verificar se ele está presente nas Preferências dos Usuários, através da chamada da operação *existe_usuário*, via **ICS**, do ATO Preferências dos Usuários, passando como seletor o objeto do tipo Preferências dos Usuários e como argumento o identificador do usuário. Caso este usuário já não possua uma janela auxiliar e tenha sua representação nos Preferências dos Usuários é chamada então a operação *inclui_usuário_ja*, via **ICS**, do ATO Janelas Auxiliares passando como seletor o objeto do tipo Janelas Auxiliares e como argumento o identificador do usuário e o tipo de janela que ele deseja. Caso contrário, o objeto é retorna sem modificações;
- *exclui_usuário,ja* - esta operação exclui um usuário e sua janela auxiliar. Para isso é necessário verificar se este usuário possui uma janela através da operação *existe_usuário_ja*. Caso este usuário tenha uma janela auxiliar é chamada então a operação *exclui_usuário_ja*, via **ICS**, do ATO Janelas Auxiliares passando como seletor o objeto do tipo Janelas Auxiliares e como argumento o identificador do usuário. Caso contrário, o objeto retorna sem modificações;
- *altera_tipo_janela* – esta operação altera o tipo de janela (WYSIWID, *Radar View* e *Miniature View*) de um usuário. Primeiramente é necessário verificar se este usuário possui uma janela auxiliar, através da operação *existe_usuário_ja* e também verificar se

o novo tipo é igual ao atual tipo de janela. Caso o usuário exista e o novo tipo seja diferente do atual então é chamada a operação *altera_tipo_janela*, via **ICS**, do ATO Janelas Auxiliares passando como seletor o objeto do tipo Janelas Auxiliares e como argumentos o identificador do usuário e o novo tipo de janela. Caso contrário, o objeto é devolvido sem modificações.

Bibliografia

[ALV 2000] ALVES, Ronnie C. Uma Ferramenta para Apoiar Decisões de Grupo para o Ambiente PROSOFT. In: SEMANA ACADÊMICA DO PPGC, 5., 2000, Porto Alegre. **Anais...** Porto Alegre: PPGC da UFRGS, 2000. p. 83 – 86.

[AND 2001] ANDERSON, Adriano. **Groupware.** Disponível em: <http://w3.openlink.com.br/adrianom/groupw.htm>. Acesso em: 20 jun. 2001.

[ARA 97] ARAÚJO, Renata M.; DIAS, Márcio de S.; BORGES, Marcos R. da S. Suporte por Computador ao Desenvolvimento Cooperativo de Software: Classificação e Propostas. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 11., 1997, Fortaleza, CE. **Anais...** Fortaleza:[s.n.], 1997. p. 299 – 314.

[BHA 95] BHARAT, Krishna; BROWN, Marc H. **Visual Obliq:** A System for Building Distributed, Multi-User Applications by Direct Manipulation. [S.l.: s.n.], 1995. 35 p. SRC Research Report.

[BIS 95] BISCHOFBERGER, W. et al. **Computer Supported Cooperative Software Engineering with Beyond-Sniff**. Trabalho apresentado na International Conference on Software Engineering, 17., 1995, Noordwijkerhout, Neth. Disponível em: <http://www.ubs.com/research/ubilab/Publications/Bis94c.html>. Acesso em: 11 dez. 2000.

[BJO 78] BJONER, D.; JONES, C. **The Vienna Development Method:** the meta language. Berlin: Springer-Verlag, 1978.

[BOR 93] BORGES, M. R. S. Suporte por Computador ao Trabalho Cooperativo. In: ESCOLA BRASIL-ARGENTINA DE INFORMÁTICA, 6., 1993. **Anais...** [S.l.: s.n.], 1993.

[BOR 95] BORGES, Marcos Roberto da Silva; CAVALCANTI, Maria Cláudia Reis; CAMPOS, Maria Luiza Machado. Suporte por Computador ao Trabalho Cooperativo. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 14., 1995, Canela. **Anais...** Canela: SBC, 1995. 45 p.

[BOR 96] BORGES, M. R. S. **Awareness Mechanisms for SISCO**. 1996. Disponível em: <http://www.nce.ufrj.br/~mborges/sisco3/aware/sumario2.html>. Acesso em: 15 jan. 2001.

[BOR 99] BORGES, MARCOS; ARAÚJO, RENATA. O USO DE GROUPWARE EM DESENVOLVIMENTO DE SOFTWARE. IN: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 1999, FLORIANÓPOLIS - SC. ANAIS... **FLORIANÓPOLIS: SBC, 1999.**

[BUX 92] BUXTON, William A. S. Telepresence: Integrating Shared Task and Person Spaces. In: GRAPHIC INTERFACE, 1992. **Proceedings**... [S.l.]: Morgan Kaufmann Publishers, 1992. p. 123 – 129.

[CAP 92] CARMO, Paulo R. S. **ATO Diretório.** Porto Alegre: CPGCC – UFRGS, 1992. Relatório de Pesquisa.

[CET 2000] CETUS TEAM. **Object-Oriented Language:** Tcl/Tk. 2000. Disponível em: <http://www.cetus-links.org/oo_tcl_tk.html#oo_tcl_tk_guis>. Acesso em: 13 dez. 2000.

[CHA 74] CHAPIN, N. New Format for Flowcharts. **Software Practice and Experience**, Chichester, v.4, n.4, p. 341-357, 1974.

[COA 90] COAD, P.; YORDAN, E. **Object-Oriented Analysis**. [S.l.]: Prentice-Hall, 1990.

[COC 93] COCKBURN, A.; GREENBERG, Saul. Making Contact: Getting the group communicating with groupware. In: ACM CONFERENCE ON ORGANIZATIONAL COMPUTER SYSTEMS, 1993, California. **Proceedings...** [S.l.]: ACM Press, 1993. Disponível em: <http://www.cpsc.ucalgary.ca:80/projects/grouplab/projects/telefreek/> . Acesso em: 15 mar. 2000.

[DEW 91] DEWAN, P.; CHOUDHARY, R. Primitives for Programming Multi-User Intefaces. In: ACM UIST, 1991. **Proceedings**... [S.l.: s.n.], 1991. p. 69-78.

[DIX 98] DIX, Alan et al. **Human-Computer Interaction**. [S.l.]: Prentice Hall Europe, 1998. 638 p.

[DOU 92] DOURISH, P.; BELLOTI, V. Awareness and coordination in shared workspace. In: CONFERENCE ON COMPUTER-SUPPORTED COOPERATIVE WORK, 4., 1992. **Proceedings**... [S.l.: s.n.], 1992. p. 107-114.

[DOU 92b] DOURISH, Paul; BLY, Sara. Portholes: Supporting Awareness in a Distributed Work Group. In: ACM CHI, 1992. **Proceedings**... [S.l.: s.n.], 1992. p. 541 – 547.

[ELL 91] ELLIS, C. A.; GIBBIS, S. J.; REIN, G. L. Groupware: Some issues and experiences. **Communications of the ACM**, New York, v. 34, n. 1, p. 38-58, Jan. 1991.

[ENS 90] ENSOR, Bob et al. How can we make groupware pratical? In: CHI CONFERENCE, 1990. **Proceedings**... [S.l.: s.n.], 1990. p. 87-89.

[GAJ 95] GAJEWSKA, Hania; MANASSE, Mark; REDELL, Dave. Argohalls: Adding Support for Group Awareness to the Argo Telecollaboration System. In: ACM UIST, 1995. **Proceedings**... [S.l.: s.n.], 1995. p. 157 – 158.

[GOL 96] GOLENDZINER, L.; SANTOS, C. Versions and configurations in object-oriented database systems: a uniform treatment. In: PRÊMIO CAMPAQ DE ESTÍMULO A PESQUISA E DESENVOLVIMENTO EM INFORMÁTICA, 1., 1996, São Paulo. **[Trabalhos Selecionados]**. São Paulo: Instituto Uniemp, 1996. p. 35-47.

[GRE 92] GREENBERG, Saul et al. GroupSketch. In: SPECIAL EDITION OF THE CSCW TECHNICAL VIDEO PROGRAM, 1992. **Proceedings**... [S.l.: s.n.], 1992.

[GRE 96] GREENBERG, Saul; ROSEMAN, Mark. **GroupWeb:** A WWW Browser as Real Time Groupware. 1996. Disponível em: <http://www.acm.org/sigchi/chi96/proceedings/shortpap/Greenberg4/sg3txt.html>. Acesso em: 02 set. 1999.

[GRE 97] GREENBERG, Saul. Collaborative Interfaces for the Web. In: FORSYTHE, Chris; GROSE, Eric; RATNER, Julie (Ed.). **Human Factors and Web Development**. [S.l.]: LEA Press, 1997.

[GRE 99] GREENBERG, Saul; ROSEMAN, Mark. Groupware Toolkits for Synchronous Work. In: BEAUDOUIN-LAFON, M. (Ed.). **Computer-Supported Cooperative Work** (Trends in Software 7). [S.l.]: John Wiley & Sons, 1999. 258p. p. 135-168.

[GRU 94] GRUDIN, JONATHAN. COMPUTER-SUPPORTED COOPERATIVE WORK: HISTORY AND FOCUS. IEEE COMPUTER**, LOS ALAMITOS, CA, V.27, N. 5, P. 19-26, MAY 1994.**

[GUG 85] GUTTAG, J. V.; HORNING, J. J.; WING, J. M. An Overview of the Larch Family of Specification Languages. **IEEE Software**, Los Alamitos, v. 2, n.5, p. 24-36, Sept. 1985.

[GUT 96] GUTWIN, Carl; GREENBERG, Saul. Workspace Awareness for Groupware. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEM, COMPANION, ACM SIGCHI, 1996. **Proceedings**... [S.l.: s.n.], 1996. p. 208-209.

[GUT 96b] GUTWIN, Carl; ROSEMAN, Mark; GREENBERG, Saul. **A Usability Study of Awareness Widgets in a Shared Workspace Groupware System.** [S.l.]: ACM, March 1996.

[GUT 96c] GUTWIN, Carl. **Workspace Awareness Research.** Feb. 1996. Disponível em:<http://www.cpsc.ucalgary.ca/projects/grouplab/people/carl/research/awareness.html>. Acesso em: 09 jan. 2000.

[GUT 96d] GUTWIN, Carl; GREENBERG, Saul; ROSEMAN, Mark. Workspace Awareness Support with Radar Views. In: ACM CHI, 1996. **Proceedings**... Vancouver: [s.n.], 1996. p. 210-211.

[GUT 97] GUTWIN, Carl; GREENBERG, Saul. Workspace Awareness. In: WORKSHOP ON AWARENESS IN COLLABORATIVE SYSTEMS, ACM CHI, 1997. **Proceedings**... Atlanta: [s.n.], 1997. p. 22-27.

[GUT 98] GUTWIN, Carl; GREENBERG, Saul. **Design for Individuals, Design for Groups:** Tradeoffs between power and workspace awareness. Calgary, Alberta , Canada: Department of Computer Science, University of Calgary, 1998. (Report 98-621-12).

[HAL 90] HALL, A. Seven Myths of Formal Methods. **IEEE Software**, New York, p. 11 – 20, 1990.

[JON 96] JONES, C. Formal Methods Light: A Rigorous Approach to Formal Methods. **IEEE Computer**, New York, v. 29, n. 4, p. 20 – 21, Apr. 1996.

[KAR 97] KARSENTY, Alain. Easing Interaction through User-Awareness. In: ACM IUI, 1997. **Proceedings**... [S.l.: s.n.], 1997. p. 225 – 228.

[KYN 95] KING, Morten. Making Representations Work. **Communications of the ACM**, New York, v. 38, n. 9, p. 46 – 55, Sept. 1995.

[KOC 97] KOCH, Michael; KOCH, Jürgen. Using Component Technology for Group Editors – The Iris Group Editor Environment. In: WORKSHOP ON OBJECT ORIENTED GROUPWARE PLATAFORMS, TELEMATVS RESEARCH CENTRE, 1997. **Proceedings**... Lancaster, UK: [s.n.], 1997. p. 44 – 49.

[KOR 96] KÖRBES, Fábio. **Implementação de um Mecanismo de Herança no PROSOFT.** 1996. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[MAO 99] MAOJUN, Zhang. VCS: A Virtual Environment Support for Awareness and Collaboration. In: ACM MULTIMEDIA, 1999. **Proceedings**... Orlando, USA: ACM, 1999. p. 163 – 165.

[MEN 89] MENDES, S.; AGUIAR, T. C. **Métodos para Especificação de Sistemas.** São Paulo: E. Blücher, 1989.

[MIL 92] MILLER, David S.; SMITH, John G.; MULLER, Michael J. TelePICTIVE: Computer-Supported Collaborative GUI, Design for Designers with Diverse Expertise. In: UIST, 1992. **Proceedings**... [S.l.: s.n.], 1992. p. 151 – 160.

[MIT 96] MITCHEL, Alex. **Calliope:** A Multi-User Shared Editor. 1996. Disponível em: <http://www.dgp.utoronto.ca/people/alex/thesis/calliope>. Acesso em: 09 jan. 2000.

[MOR 97] MORAES, Sílvia Maria Wanderley. **Um Ambiente Expert para Apoio ao Desenvolvimento de Software.** 1997. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[NAS 73] NASSI, I.; SCHNEIDERMAN, B. Flowchart Techniques for Strutured Programming. **SIGPLAN Notices**, New York, Aug. 1973.

[NAU 69] NAUR, P.; RANDELL, B. **Software Engineering:** A Report on a Conference Sponsored by the NATO Science Commitee. Roma: Scientif Affairs Division, 1969.

[NUR 91] NUNAMAKER, J. F. Eletronic meeting systems to support group work. **Communications of the ACM**, New York, v. 34, n. 7, p. 40-61, July 1991.

[NUN 89] NUNES, Daltro José. **PROSOFT:** Um ambiente de desenvolvimento de software extendivel. Porto Alegre: DI da UFRGS, 1989.

[NUN 92] NUNES, DALTRO JOSÉ. ESTRATÉGIA DATA-DRIVEN NO DESENVOLVIMENTO DE SOFTWARE. IN: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 1992, GRAMADO. ANAIS... **GRAMADO: SBC, 1992.**

[NUN 94] NUNES, Daltro José. **PROSOFT**. Relatório de Pesquisa Interno. 1994.

[NUS 2000] NUNES, Isabel Dillmann. **Estudo do Aspecto de Interação Humana para Ferramentas com Trabalho Cooperativo**. 2000. 54 p. Trabalho Individual 1 (Mestrado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[NUS 2000b] NUNES, Isabel D.; NUNES, Daltro J.; REIS, Rodrigo Q.; REIS, Carla A. Classificação e Avaliação da Interação Humana em Editores Cooperativos Síncronos. In: INTERNATIONAL SYMPOSIUM ON KNOWLEDGE MANAGEMENT/DOCUMENT MANAGEMENT, 2000. **Proceedings...** Curitiba, Brasil: [s.n.], 2000. p. 473 – 493.

[PAL 94] PALMER, James D.; FIELDS, N. Ann. Computer-Supported Cooperative Work. **IEEE Computer**, New York, p. 15 – 17, May 1994.

[PAT 90] PATTERSON, John F.; HILL, Ralph D.; ROHALL, Steven L. Rendezvous: Na Architecture for Synchronous Multi-User Applications. In: CSCW, 1990. **Proceedings...** [S.l.: s.n.], 1990. p. 317-328.

[POL 96] POLTROCK, Steven; GRUDIN, Jonathan. Groupware and workflow: a survey of systems and behavioral issues. In: ACM CHI, 1996. **Proceedings...**. Vancouver, Canadá: [s.n.], April 1996.

[PRE 95] PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: MAKRON Books do Brasil, 1995. 1056 p.

[RAB 2001] RABELO, Abraham; LIMA REIS, Carla A.; REIS, Rodrigo Quites; NUNES, Daltro José. **Interação Humana durante execução de processos de software:** Classificação e Exemplos. Porto Alegre: PPGC da UFRGS, 2001. (Relatório Técnico, 310).

[RAP 99] RAPOSO, Alberto B.; MAGALHÃES, Léo P.; RICARTE, Ivan L. M. **Interação na Web**. In: JAI, 1999. Disponível em: <http://www.dca.fee.unicamp.br/~alberto/pubs/JAI99/curso_jai99.html >. Acesso em: 20 jun. 2001.

[REN 91] REIN, H.; ELLIS, C. rIBIS: a real-time group hypertext system. **International Journal of Man-Machine Studies**, [S.l.], v. 34, 1991.

[RED 94] REINHARD, Walter; SCHWEITZER, Jean; VÖLKSEN, Gerd. CSCW Tools: Concepts and Architectures. **IEEE Computer**, New York, p. 28-36, May 1994.

[REC 98] REIS, Carla A. **Um Gerenciador de Processos de Software para o Ambiente PROSOFT.** 1998. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[REI 98] REIS, Rodrigo Quites. **Uma Proposta ao Desenvolvimento Cooperativo de Software no Ambiente PROSOFT.** 1998. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[REI 2001] REIS, Rodrigo Quites. **APSEE-StaticPolicy:** Sintaxe, semântica algébrica e exemplos de uma linguagem para verificação automática de políticas estáticas em modelos de processos de software. Porto Alegre: PPGC da UFRGS, 2001. Relatório Técnico.

[RIB 91] RIBEIRO, Leila. **Integração no PROSOFT de ambientes corretos obtidos a partir de especificações algébricas e executadas usando sistemas de reescrita.** 1991. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[ROS 92] ROSEMAN, M.; GREENBERG, S. GroupKit: A groupware toolkit for building real-time conferencing applications. In: ACM CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK, 1992. **Proceedings...** Toronto, Canada: ACM, 1992. p. 43-50.

[ROS 98] ROSEMAN, Mark. **Groupkit User Manual**. 1998. Disponível em: http://www.ie2.u-psud.fr/~roussel/docs/groupkit-5.1/> . Acesso em: 12 dez. 2000.

[ROD 77] ROSS, D.; SCHOMAN, K. Structured Analysis for Requirements Definition. **IEEE Transactions on Software Engineering**, New York, v. 3, n. 1, p. 6-15, Jan. 1977.

[ROD 85] ROSS, D. Applications and Extensions of SADT. **IEEE Computer**, New York, v. 18, n. 4, p. 25-35, Apr. 1985.

[SCH 94] SCHLEBBE, Heribert. **Distributed Prosoft.** Germany: University of Stuttgart, Aug. 1994.

[SCL 97] SCHLICHTER, Johan; KOCH, Michael; BÜRGER, Martin. Workspace Awareness for Distributed Teams. In: WORKSHOP COORDINATION TECHNOLOGY FOR COLLABORATIVE APPLICATIONS, 1997. **Proceedings...** Berlin: Springer-Verlag, 1997.

[SCU 94] SCHUM, S.; HAMMOND, N. Argumentation-Based design rationale: what use at what cost? **International Journal of Human-Computer Studies**, [S.l.], n. 40, 1994.

[SIL 99] SILVA, Paulo Sérgio. **Instalação do Tcl/Tk**. 1999. Disponível em: <psdasilva@yahoo.com>. Acesso em: 12 dez. 2000.

[SIL 2001] SILVA, Fábio. **Um Modelo de Simulação de Processo de Software baseado em Conhecimento para o Ambiente PROSOFT.** 2001. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[SLO 95] SLONNEGER, Kenneth. **Formal Syntax and Semantics of programming languages**. Reading: Addison-Wesley, 1995. 637 p.

[TCL 2000] TCL/TK. **TCL Developer Xchange**. 2000. Disponível em: <http://dev.scriptics.com/software/tcltk/>. Acesso em: 12 dez. 2000.

[WAT 91] WATT, D. **Programming Language Syntax and Semantics.** New York: Prentice-Hall, 1991.

[WOO 97] WOODCOCK, Andree. **CSCW and related issues.** July 1997. Disponível em: <http://dougal.derby.ac.uk/andree/cscwandrel.html>. Acesso em: 02 set. 1999.

[YAK 90] YAKEMOVIC, K. B.; CONKLIN, J. Report on a development project use of an issue-based information system. In: CONFERENCE ON COMPUTER-SUPPORTED COOPERATIVE WORK, 3., 1990. **Proceedings...** [S.l.: s.n.], 1990. p. 105-118.