PUC
RIO

Raquel Oliveira Prates

A Engenharia Semiótica de Linguagens de Interfaces Multi-Usuário

TESE DE DOUTORADO

Departamento de Informática
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Rio de Janeiro, Outubro de 1998

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Rua Marquês de São Vicente, 225 – Gávea
CEP 22453-900 Rio de Janeiro RJ Brasil
http://www.puc-rio.br

Raquel Oliveira Prates

A Engenharia Semiótica de Linguagens de Interfaces Multi-Usuário

Tese apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para obtenção do título de Doutor em Informática.

Orientadores : Clarisse Sieckenius de Souza

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, Outubro de 1998

Sonhos

Raio de sol, manhã despertada...
Olhos que de longe brilham
Com fé o caminho trilham,
Na esperança da chegada.

Há pedras no caminho
E o medo traz a escuridão,
Mas junto vem a mansidão.
Não travas luta sozinho.

Chega à conquista um dia
E à luz que sempre via,
Um sonho se realizou.

A vida segue em frente
E o leva sempre crente
De que um novo sonho começou.

Ao meu pai, Antonio Augusto P. Prates, que plantou em meu coração a semente deste sonho.

Agradecimento

Ao meu marido, João Luiz E. Campos, por ter estado ao meu lado durante esta caminhada, sempre me oferecendo apoio, amor, carinho e companheirismo. Agradeço ainda pelas inúmeras vezes que me ouviu e discutiu comigo questões deste trabalho, colaborando para que eu clarificasse e sedimentasse várias das idéias aqui contidas. Finalmente, lhe sou grata pela editoração eletrônica deste texto.

Aos meus pais, Antonio Augusto e Maria Helena, e meus irmãos, Henrique e Marcos, pelo seu eterno encorajamento, apoio, amor e carinho que permitiram, não só que eu chegasse ao fim deste trabalho, mas também que eu o iniciasse.

A Clarisse Sieckenius de Souza, minha orientadora e amiga, que sempre se empenhou para que eu pudesse desenvolver um trabalho de qualidade, trabalhando ao meu lado e oferecendo sempre o profissionalismo, seriedade e qualidade do seu próprio trabalho. Agradeço por ter me orientado, não só na minha formação como doutora, mas também como pesquisadora e profissional. Além disso, lhe sou grata por toda sua amizade, apoio e carinho durante todo este tempo.

A Isa Haro Martins por ter dedicado tanto do seu tempo e sempre com tão boa vontade ao teste do SERG apresentado aqui, e sem o qual eu não teria sido capaz de concluir este trabalho. Além disso, agradeço também a sua participação em discussões sobre as idéias deste trabalho e seus comentários, que ajudaram a enriquecê-lo.

A Cecília Kremer Vieira da Cunha pela sua amizade e companheirismo presentes desde o início deste trabalho. Agradeço também seu tempo e disposição para realizar comigo um segundo teste para a versão final deste trabalho.

A Jair Cavalcanti Leite pelo seu trabalho em Engenharia Semiótica que abriu caminho para este trabalho. Agradeço também por ter me deixado participar das discussões sobre o seu trabalho durante o desenvolvimento deste e pelas nossas discussões sempre tão produtivas.

Aos demais membros do SERG, Simone D. J. Barbosa, Sérgio R. P. da Silva, Gilda Dahis, Denise A. de Oliveira e Flávio M. Varejão por nossos seminários e por seus comentários sempre relevantes às idéias aqui apresentadas. À Simone agradeço ainda a leitura e revisão deste texto.

À Professora Ana Cristina Bicharra Garcia pela oportunidade de trabalhar no projeto ADDProc que permitiu que eu tivesse experiência no desenvolvimento de uma interface multi-usuário para um sistema real e conhecimento prático de várias questões apresentadas neste trabalho. Agradeço também as conversas e discussões que tivemos, juntamente com a Clarisse, e que resultaram no primeiro artigo desta pesquisa.

Ao Professor Rick Kazman que me ofereceu a oportunidade de fazer bolsa sanduíche na Universidade de Waterloo.

Ao Professor Tom Carey que me "adotou" durante minha estadia na Universidade de Waterloo e me ofereceu a oportunidade de trabalhar com ele e seu grupo. Agradeço também pelas várias conversas que tivemos sobre este trabalho e suas inúmeras colaborações para o seu desenvolvimento.

Aos Professores Carlos J. P. Lucena e Hugo Fuks pela oportunidade de usar o AulaNet como teste piloto deste trabalho. Agradeço também seu interesse e colaborações no desenvolvimento deste trabalho.

A Marcelo Blois Ribeiro, Marcus Felipe M. C. da Fontoura e Adelaide Letícia S. Lukowiecki pela sua ajuda com o AulaNet, permitindo que eu entendesse melhor o seu funcionamento e as idéias que o motivaram.

A Beatriz Castier pela amizade, apoio, carinho e companheirismo não só durante este doutorado, mas desde minha chegada ao Rio. Agradeço também pelos nossos "cafés virtuais", tão importantes na minha rotina neste último ano de trabalho, e pela revisão de parte deste texto.

A cada um dos meus amigos e familiares agradeço a amizade, carinho, amor e alegria que contribuiram imensamente para que durante todo este trabalho a alegria, descontração e bom humor estivessem sempre presentes.

Aos colegas do TeCGraf, ADDLabs, Departamento de Informática e BrasWat pelo companheirismo, amizade e apoio, muitas vezes técnico, durante este trabalho.

Aos Professores Virgílio A. de Almeida, Marcelo Gattass e Luiz Henrique de Figueiredo que embora não tenham participado diretamente deste trabalho, me incentivaram e contribuiram para que eu trilhasse este caminho.

Ao Departamento de Informática e TeCGraf que me proporcionaram a infra-estrutura necessária para a realização deste trabalho.

À banca examinadora que avaliou este trabalho e fez sugestões relevantes para a sua melhoria para a versão final.

Ao CNPq pela bolsa de estudos durante os quatro anos deste doutorado.

Finalmente, ao meu anjo da guarda que me protege, me ilumina e guia o meu caminho.

Resumo

Atualmente, *designers* de interfaces multi-usuário contam com o apoio de diretrizes, *frameworks*, modelos de descrição e *toolkits* durante o desenvolvimento destas. *Toolkits* dão suporte à construção da interface, enquanto diretrizes e *frameworks* levantam questões a serem consideradas durante a sua definição. Os modelos, por sua vez, permitem a descrição dos modelos conceituais do grupo a serem representados nesta interface, ou parte deles. No entanto, nenhum destes fornece ao projetista um ambiente que lhe permita planejar e definir sua interface.

Adotamos neste trabalho a abordagem da Engenharia Semiótica, na qual a interface é vista como uma mensagem unilateral sendo enviada pelo projetista aos usuários. Esta mensagem é considerada um artefato de meta-comunicação, uma vez que ela mesma é capaz de trocar mensagens com o usuário. Em um ambiente multi-usuário ela é também um artefato de mediação de comunicação, já que serve de *medium* para os usuários se comunicarem entre si. Esta abordagem enfatiza a expressão do *designer* e cria a necessidade de se desenvolver instrumentos de apoio a esta expressão.

Neste trabalho buscamos dar apoio ao *designer* de interfaces multi-usuário, dentro do quadro teórico da Engenharia Semiótica. Assim, apresentamos um modelo abstrato de meta-comunicação que serve de base para o desenvolvimento de instrumentos de apoio à descrição de interfaces multi-usuários. Este modelo propõe que se ofereça ao projetista, não apenas uma forma de descrever sua mensagem, mas também indicadores qualitativos sobre esta descrição. A partir deste modelo, criamos um modelo de arquitetura de suporte ao *design* de interfaces multi-usuário. Este modelo dá suporte a projetistas fazendo o *design* de interfaces de forma *top-down*. Assim, este trabalho é o primeiro passo na direção de um ambiente de suporte ao *design* de interfaces multi-usuário com qualidade.

Abstract

Nowadays there are guidelines, frameworks, models and toolkits that support the development of multi-user interfaces. Toolkits assist designers in buiding the interface, whereas guidelines and frameworks direct the designer towards issues to be considered during interface design. Models, on their turn, allow for the description of the conceptual model of the group (or part of it) to be represented in the interface. Nevertheless, none of these provide the designer with an environment in which to plan and define the system's interface.

We take a Semiotic Engineering approach, in which the interface is perceived as a one-shot message being sent from the designer to the users. In particular, interfaces can be viewed as meta-communication artifacts, since they can also exchange messages with the users. Furthermore, in multi-user environments, they are also communication embedding artifacts, since they allow for the communication among users. This approach focuses on the expression of the designer and requires the development of means to support this expression.

In this work, we intend to provide support to multi-user interface designers, in a Semiotic Engineering framework. In order to meet our goal, we developed a meta-communication model from which models and techniques to support the development of multi-user interfaces can be developed. The meta-communication model proposes that designers should be provided not only with means to describe their messages, but also some qualitative indications on their description. Based on this model, we developed a model of an architecture to support the design of multi-user interfaces. This model assists designers that develop multi-user interfaces top-down. Therefore, this work is the first step in the direction of an environment to support the design of multi-user interfaces with quality.

Contents

List of Figures

Lista de Símbolos

Chapter 1

Introdução

O avanço tecnológico e o conseqüente crescimento de uso de redes de computadores permitiram e motivaram o desenvolvimento de aplicações de *software* destinadas a servir vários usuários ao mesmo tempo, numa replicação de um dos ambientes mais comuns de trabalho — o grupo ou a equipe. A este desafio se vem chamando, com freqüência cada vez maior, Trabalho Cooperativo Auxiliado por Computador, ou CSCW (*Computer Supported Cooperative Work*) (Grudin, 1994b). Os novos sistemas desenvolvidos nesta direção têm sido denominados sistemas multi-usuário, sistemas de/para grupos ou ainda *groupware*.

O objetivo de uma aplicação multi-usuário é permitir aos usuários trabalhar colaborativamente em uma tarefa. Para isto, os usuários interagem não apenas com o sistema, mas também entre si, utilizando-se para isto do sistema como infra-estrutura de comunicação. Um outro fator essencial para que as pessoas possam trabalhar em equipe é a coordenação deste trabalho. Assim, o desenvolvimento de sistemas de grupo traz para a indústria de *software* novas questões, que não são relevantes no desenvolvimento de sistemas mono-usuário, e novos desafios (Grudin, 1994b). Um destes desafios é o *design* de interfaces multi-usuário (Ellis et al., 1991).

Para desenvolver interfaces para grupos o programador da interface pode se utilizar de ferramentas de desenvolvimento de interfaces mono-usuário. Neste caso, fica a cargo dele dar suporte aos aspectos colaborativos da interface, como por exemplo fazer o controle de acesso a *widgets* compartilhados e *feedthrough*, ou seja, projetar os efeitos observados por um certo usuário sobre um elemento de interface quando outro usuário executa uma ação sobre este elemento (Dix et al., 1993). No entanto, esta tarefa pode vir a ser árdua e frustrante e torna-se necessária, então, a criação de novas ferramentas, técnicas e metodologias que apóiem o desenvolvimento de interfaces multi-usuário (Greenberg e Roseman, 1998; Crow et al., 1997). Hoje há propostas de novos elementos de interface específicos para grupos (Hazemi e Macaulay, 1996; Greenberg e Roseman, 1998; Gutwin et al., 1996; Ackerman e Starr, 1996; Greenberg et al., 1996a; Greenberg et al., 1996b), diretrizes (Hamalainen et al., 1991; Mandviwalla e Olfman, 1994) e *frameworks* (Lim e Benbasat, 1991; Dewan e Choudary, 1992). Estes novos *widgets* e *frameworks* permitem ao *designer* definir a interação dos usuários com a aplicação e a linguagem de comunicação entre os membros do grupo, enquanto diretrizes chamam a sua atenção para questões a serem consideradas durante o projeto da interface.

Estes recursos dão apoio ao desenvolvimento da interface, mas não são suficientes, uma vez que que não apóiam importantes passos do processo de *design*. Segundo Winograd (1996), o design de um *software*, ou qualquer outro artefato, envolve a tomada de decisões e a sua

construção e, uma vez construído, o artefato leva consigo a marca de intenção do *designer* sobre o que ele faz e como deve ser entendido e usado. Esta posição vai ao encontro da abordagem da Engenharia Semiótica (de Souza, 1993), e ambas apontam para a necessidade de se fornecer ao *designer* não apenas ferramentas, modelos e técnicas que o apóiem na etapa de construção de interfaces, mas também aqueles que o auxiliem nas tomadas de decisões durante a etapa de planejamento destas interfaces.

Em abordagens semióticas (Nadin, 1988; Andersen et al., 1993; Nake, 1994; de Souza, 1993; Jorna e Van Heusden, 1996), toda aplicação computacional é concebida como um ato de comunicação que inclui o *designer* no papel (explícito ou implícito) de emissor de uma mensagem para os usuários dos sistemas por ele criados. Na Engenharia Semiótica, em particular, a interface de um sistema é explicitamente tratada como uma mensagem unilateral enviada pelo designer aos usuários; e, como esta mensagem também é capaz de se comunicar com os usuários através da troca de mensagens, ela é considerada um artefato de meta-comunicação (de Souza, 1993). O conteúdo desta mensagem é a resposta para duas perguntas fundamentais: (1) Qual a interpretação do *designer* para o tipo de problema que a aplicação é capaz de resolver? e (2) Como os usuários devem interagir com a aplicação para alcançar estas soluções dentro desta interpretação? No caso de sistemas de grupo, o *designer* também tem que "dizer" aos membros do grupo qual a estrutura do grupo em que eles estão inseridos, as possibilidades de distribuição de tarefas entre os membros do grupo, os protocolos e linguagens de comunicação entre eles e como suas ações afetam e são afetadas pelas ações de outros membros (Prates et al., 1997).

A Engenharia Semiótica adota uma posição complementar às atuais abordagens cognitivas para *design* de Interação Humano-Computador (IHC), à medida que ela troca o foco do *design* de interfaces do **aprendizado** para o **ensino**. Isto significa que o ponto principal para os *designers* de interfaces passa a ser como se expressarem melhor na transmissão da sua mensagem para os usuários, ao invés de como os usuários vão entender esta mensagem[1]. Esta troca de foco implica em se fornecer para os projetistas de interface meios de se expressarem que os ajudem, não apenas a enunciar sua mensagem, mas também a planejá-la com consciência e qualidade. Expressando-se mais claramente, o *designer* atinge potencialmente maior usabilidade para seu sistema. Por usabilidade entenda-se o uso do *software* de forma criativa e eficaz[2] (Adler e Winograd, 1992). Isto é possível, pois à medida que o projetista se expressa com maior clareza e qualidade, aumentam as chances de os usuários entenderem a sua mensagem e serem capazes de usar o sistema com maior eficiência, produtividade e criatividade.

Neste trabalho apresentamos um modelo desenvolvido dentro do quadro teórico de Engenharia Semiótica que apóia o *designer* na especificação de interfaces multi-usuário. O objetivo deste modelo é dar suporte a métodos e ferramentas que permitam ao *designer* o planejamento da colaboração e comunicação de grupos de usuários reunidos em um mesmo sistema de forma mais consciente e consistente.

[1] Esta troca de foco não implica que a necessidade de o usuário entender o *software* não seja fundamental. Ela apenas chama a atenção para a questão, igualmente fundamental, de o *designer* deixá-lo claro.

[2] Adler e Winograd advogam que este é o critério que se deveria avaliar para julgar a usabilidade de um sistema, ao invés de facilidade de uso e minimização da possibilidade de cometer erros que levam ao *design* de sistemas "anti-idiotas" (*idiot-proof*).

1.1 Objetivos e Limites

Em Engenharia Semiótica, a interface de um sistema pode ser entendida como sendo um "texto" em uma linguagem interativa cujo conteúdo é a mensagem que o *designer* transmite aos usuários. Assim como na geração de um texto em linguagem natural, a criação da interface envolve dois passos: o de planejamento e o de realização (Paris et al., 1991). No planejamento o projetista define o que ele vai dizer, enquanto que na realização ele define como vai dizê-lo. Este trabalho se enquadra na fase de planejamento de interfaces multi-usuário.

Seu objetivo é ser o primeiro passo na direção da construção de um ambiente que guie o projetista durante todo o processo de *design* de interfaces multi-usuário (planejamento e realização), sem, no entanto, restringir a sua criatividade e poder de decisão. O modelo geral, ou modelo abstrato de meta-comunicação, desenvolvido neste trabalho propõe que se permita ao *designer* expressar o seu próprio modelo conceitual particular do grupo através da especificação de um conjunto de requisitos que define os modelos de comunicação e colaboração deste grupo único e que representa parte do conteúdo da interface. Este modelo abstrato de meta-comunicação é composto por um conjunto de características básicas que definem o grupo e as condições para que regras heurísticas chequem as relações semânticas entre estas características.

Para que o modelo abstrato de meta-comunicação possa ser o mais geral possível, as características básicas que definem o grupo podem ser descritas intensional ou extensionalmente, independentemente do domínio, embora os valores que lhes serão atribuídos pelo *designer* sejam dependentes de domínio. O conjunto de regras que atua sobre estas dimensões pode, então, ser caracterizado como separável de contexto. Estas regras que atuam sobre tais características estruturam o espaço de soluções possíveis, uma vez que descartam as combinações que não fazem sentido.

Porém, em um determinado domínio, uma situação descartada pelo modelo pode perfeitamente fazer sentido. Fica, então, a cargo do *designer* nestes casos julgar as "inconsistências" identificadas pelo modelo. Podemos dizer que o modelo é descritivo, e não prescritivo, uma vez que ele não forma juízo de valor sobre o que é definido pelo *designer* e, em última instância, deixa a cargo do *designer* decidir o que faz ou não sentido e o que é descartado ou não.

O processo computacional ao longo do tempo pode fazer com que um elemento de determinada categoria de um sistema troque de categoria. Assim, o nosso modelo permite a descrição tanto de elementos que independem do tempo (descrição estática), quanto daqueles que se transformam no tempo (descrição dinâmica).

Para ser capaz de usar nosso modelo geral, o *designer* deve desenvolver o seu projeto de forma *top-down*. Ele deve começar por uma especificação de alto-nível e ir refinando-a, sucessivamente, até obter a interface final. Apesar de esta não ser a única forma possível de se fazer o *design* de um sistema, não discutiremos aqui os processos alternativos de *design* existentes, e construiremos toda a proposta dentro destes limites.

Tendo como objetivo o apoio à fase de planejamento em um processo de *design* como o descrito, derivamos do modelo abstrato de meta-comunicação um modelo de arquitetura de suporte ao *design* de interfaces multi-usuário. Este modelo propõe uma arquitetura para ferramentas que permitam ao projetista criar seu modelo de grupo e lhe fornecem indicadores qualitativos sobre sua criação. Para isso, além das características básicas e das regras semânticas separáveis de contexto, este modelo oferece ao projetista uma base de conhecimento, um simulador de cenários e um "conselheiro" de *widgets*.

Com o objetivo de obtermos alguns indícios sobre a viabilidade dos modelos que estamos propondo, implementamos um protótipo do modelo de arquitetura de suporte e executamos dois testes. O primeiro teste consiste na modelagem de um sistema multi-usuário existente, o *site* da *intranet* do SERG, (*Semiotic Engineering Research Group*, ou Grupo de Pesquisa em Engenharia Semiótica) (SERG, sd). Nossa intenção com este teste é mostrar que é possível implementar uma ferramenta de apoio à especificação do modelo de grupo de um sistema multi-usuário utilizando os nossos modelos, e termos um retorno preliminar sobre estes modelos. O segundo teste consiste da remodelagem da interface multi-usuário do sistema Qualitas. O objetivo deste teste é ter indícios de melhorias na modelo conceitual de grupo do *designer* decorrentes do uso dos nossos modelos. Não está no escopo desta tese testar extensivamente os modelos propostos e nem avaliar a sua utilização em ambientes reais de *design* de sistemas de grupo ou em domínios específicos, até porque isto requer a implantação do modelo de arquitetura de suporte em métodos e ferramentas bem-acabadas, o treinamento dos *designers* para o uso dos mesmos, e a avaliação de uma amostragem de produtos de *design* desenvolvidos.

1.2 Organização da Tese

Neste capítulo introduzimos o problema da falta de apoio adequado a projetistas no desenvolvimento de interfaces multi-usuário. O nosso objetivo neste trabalho é tentar dar um primeiro passo nesta direção. Adotamos a perspectiva da Engenharia Semiótica, e dentro deste quadro teórico apresentamos modelos que permitem o desenvolvimento de ferramentas, métodos e técnicas de apoio ao *designer*.

No próximo capítulo, apresentamos os conceitos básicos de aplicações multi-usuário e suas interfaces. Apresentamos e discutimos *frameworks*, diretrizes e *toolkits* existentes que buscam apoiar o desenvolvimento de tais aplicações e interfaces. Finalmente, terminamos o capítulo com um levantamento de requisitos para o apoio de desenvolvimento de interfaces multi-usuário.

No capítulo 3, apresentamos a abordagem da Engenharia Semiótica e os principais conceitos de semiótica necessários para o seu entendimento. Em seguida, estendemos a proposta original de Engenharia Semiótica para incluir interfaces multi-usuário. Apresentamos também um método de *design* que favorece a abordagem da Engenharia Semiótica, e que guia o apoio que pretendemos fornecer ao *designer* com este trabalho. Terminamos o capítulo com os requisitos de apoio ao *designer* gerados pela abordagem da Engenharia Semiótica.

Estes requisitos de Engenharia Semiótica requerem o desenvolvimento de apoio à expressão do *designer*. Com o intuito de permitir o desenvolvimento de instrumentos que dêem suporte aos projetistas de interfaces multi-usuário durante a fase de planejamento desta, desenvolvemos o modelo abstrato do componente multi-usuário do artefato de meta-comunicação. Este modelo propõe que seja oferecida ao *designer* uma linguagem de *design* separável de contexto, que lhe permita expressar sua descrição, estática e dinâmica do grupo. Apresentamos este modelo no capítulo 4.

O modelo abstrato de meta-comunicação permite o desenvolvimento de várias formas de suporte ao *designer* de interfaces multi-usuário, dentre elas modelos de arquitetura de ferramentas, diretrizes e técnicas. Para aplicar as idéias apresentadas no modelo abstrato de meta-comunicação, criamos o modelo de arquitetura de suporte ao *design* de interfaces multi-usuário, que está apresentado no capítulo 5. Este modelo permite o desenvolvimento de ferramentas de

suporte a projetistas cujo método de *design* é o descrito no capítulo 3.

Um protótipo derivado do modelo de arquitetura foi implementado e é apresentado no capítulo 6. O nosso objetivo ao desenvolver este protótipo era mostrar que era possível implementar uma instância de ferramenta derivada do modelo de arquitetura e testar a validade dos nossos modelos. Dois testes foram feitos com este protótipo e são apresentados neste capítulo, juntamente com os resultados obtidos em cada um deles.

No capítulo 7 discutimos os modelos apresentados, mostrando suas vantagens e limitações, e os testes feitos com a intenção de demonstrar a sua validade. Como estes testes são apenas preliminares, fazemos então uma proposta de testes a serem executados para se obter uma avaliação extensiva destes modelos. Ainda neste capítulo comparamos as ferramentas que podem ser derivadas do modelo de arquitetura com outros sistemas especialistas em *design* de interfaces.

Finalmente, no capítulo 8, terminamos nosso trabalho fazendo um breve resumo do que nos propusemos a fazer, do que foi desenvolvido e das contribuições do nosso trabalho. Em seguida, apresentamos trabalhos futuros que podem ser desenvolvidos a partir deste trabalho.

Anexadas a este trabalho, em forma de apêndices, incluimos ainda descrições de alguns componentes deste trabalho. No Apêndice A, apresentamos uma descrição detalhada das regras semânticas que compõem a linguagem de *design* do modelo de arquitetura. Em seguida, no Apêndice B, apresentamos o relatório gerado pelo protótipo para o teste do SERG. Finalmente, o Apêndice C contém o Manual do Usuário deste protótipo.

Chapter 2

Interfaces Multi-usuário

Neste capítulo introduzimos novos conceitos, questões e desafios trazidos à tona pelo surgimento de sistemas de grupo. Em seguida, discutimos diretrizes, *frameworks* e modelos criados para o desenvolvimento de aplicações de grupo e suas interfaces. Terminamos o capítulo com a apresentação de requisitos relevantes para o desenvolvimento de interfaces de sistemas CSCW.

2.1 Conceitos

Aplicações de grupo replicam o ambiente de grupo, dando apoio à troca de informações entre os membros do grupo e à sua colaboração durante a execução de uma tarefa. Neste trabalho usaremos colaboração no sentido mais geral apresentado por Lim e Benbasat (1991), no qual o termo significa a aplicação dos esforços individuais de duas ou mais pessoas na execução conjunta de uma tarefa. A execução desta tarefa pode envolver tanto objetivos comuns e esforços unificados, quanto objetivos distintos e ações conflitantes.

Assim como em sistemas mono-usuário, os usuários de um sistema de grupo interagem com o sistema. No entanto, eles passam a interagir também entre si. Conseqüentemente, os sistemas multi-usuário passam a ter como noções centrais a comunicação, colaboração e coordenação (Ellis et al., 1991). A comunicação diz respeito à troca de informações entre os membros, enquanto que a colaboração se refere à interdependência das tarefas entre os diversos membros e a coordenação lida com a integração e controle destas tarefas e trocas de informações (Lim e Benbasat, 1991; Ellis et al., 1991).

A forma como os usuários de uma aplicação multi-usuário podem colaborar pode variar no tempo e no espaço (Ellis et al., 1991; Grudin, 1994a). A colaboração entre os membros pode ocorrer sincronamente, ou seja, em tempo-real (por exemplo, em sistemas de conferência), ou assincronamente, onde as tarefas de cada usuário não dependem da "presença" simultânea dos demais (como no envio de *email*). Ortogonalmente, a colaboração pode ser classificada como local, quando os usuários se encontram no mesmo local físico, ou distribuída, quando eles estão em lugares diferentes. A determinação de onde um novo sistema deve se encaixar nesta taxonomia depende do domínio em que ele está inserido e dos tipos de colaboração aos quais ele deve dar suporte.

Uma outra questão a ser definida são as formas de interação entre os usuários, ou os seus protocolos de comunicação (Ellis et al., 1991). No caso em que estes protocolos são embutidos no sistema eles são chamados de protocolos tecnológicos. Se, porém, eles são deixados a

cargo dos usuários, então eles são protocolos sociais. Para ilustrar, imagine-se um sistema de conferência onde apenas um usuário pode falar de cada vez. Isto impõe sobre o grupo o processo de alternância da palavra e é um protocolo tecnológico. Se estivesse a cargo do grupo a decisão de como seria organizada a conversa, então o protocolo seria social. Alguns sistemas podem ter um protocolo misto, definindo parte do protocolo tecnologicamente, e deixando uma outra parte a cargo dos usuários.

A escolha do protocolo a ser implementado é uma importante decisão do *designer* e cada protocolo tem suas vantagens e desvantagens. O protocolo tecnológico garante que o processo definido será seguido, estrutura mais as atividades do grupo e, muitas vezes, é útil para os usuários novatos. Por outro lado, ele pode fazer imposições que sejam excessivamente restritivas e não atender a todas as necessidades do grupo. O protocolo social se adapta às necessidades do grupo e deixa a cargo dos usuários definir os passos do processo. Desta forma, não existe nenhuma garantia de quais serão estes passos. Além disso, esta definição pode criar uma sobrecarga de tempo e esforço para os usuários, que pode não ser desejável.

Embora a classificação da aplicação de grupo em relação ao tempo e espaço e a definição dos protocolos de comunicação influenciem profundamente a interação entre os membros de um grupo, elas não são os únicos fatores a defini-la. Os membros de um grupo podem assumir diferentes papéis e ter diferentes tarefas dentro deste grupo (Ellis et al., 1991; Smith e Rodden, 1995; Salvador et al., 1996). Além disso, o grupo pode ter uma estrutura hierárquica ou ser divido em subgrupos (Grudin, 1994b). Assim, o papel de cada membro e sua posição na estrutura do grupo são determinantes na resolução de questões como: que membros podem se comunicar com que outros, como esta comunicação se dá e qual a relação entre as tarefas deles.

Muitas vezes as tarefas de um grupo requerem que seus membros façam algumas atividades juntos e outras individualmente. Sistemas que pretendem dar suporte a este tipo de grupo têm que permitir que os participantes trabalhem ora em grupo, ora individualmente, e integrar estes modos de trabalho permitindo que usuários passem de um para outro de forma simples e natural (Ellis et al., 1991; Lim e Benbasat, 1991; Mandviwalla e Olfman, 1994; Hazemi e Macaulay, 1996). Às partes do sistema, onde mais de um membro trabalha em uma mesma tarefa, dá-se o nome de ambiente compartilhado virtual (Hamalainen et al., 1991; Greenberg et al., 1996a).

Em ambientes compartilhados físicos, quando uma pessoa interage com a outra, ela pode observar (muitas vezes até perifericamente) as ações da outra pessoa, o que lhe provê uma série de indicações sobre o que a outra pessoa está fazendo, qual o seu nível de interesse pelo que está se passando, qual o seu foco de atenção, qual o seu próximo passo, entre outras. Estas indicações lhe permitem coordenar suas ações com as da outra pessoa. Quando porém este ambiente é "transferido" para o computador, as pessoas não têm mais acesso a tais indicações. Cabe ao *designer* identificar as informações relevantes e, explicitamente, disponibilizá-las para os usuários para que eles consigam coordenar suas interações com os demais (Greenberg et al., 1996b; Greenberg, 1997; Gutwin et al., 1996; Roseman e Greenberg, 1996b; Greenberg e Roseman, 1998).

A determinação das informações necessárias e relevantes não é trivial. Se por um lado informações de menos dificultam a coordenação de ações entre os membros do grupo, por outro, informações de mais podem se tornar altamente distrativas e interruptivas. Assim, o balanceamento destas informações é essencial para o trabalho do grupo e depende do domínio e das interdependências entre os usuários e entre suas tarefas.

Se o *design* de interfaces mono-usuário já é uma tarefa complexa, as novas considerações

a serem feitas e decisões a serem tomadas introduzidas por grupos tornam o projeto de interfaces ainda mais complexo (Hewitt e Nigel Gilbert, 1993). Sistemas de grupos, onde existem membros exercendo diferentes papéis, podem requerer diferentes interfaces para cada tipo de membro, com partes individuais e partes compartilhadas. Além disso, a interface deve deixar claro como os usuários podem se comunicar entre si, como seus trabalhos se relacionam e dar todas as informações necessárias para que eles se coordenem.

Com o intuito de auxiliar o *designer* na criação de interfaces e aplicações multi-usuário surgem então diretrizes, *frameworks*[1], modelos e *toolkits*. Algumas propostas atuam no nível de planejamento, enquanto outras atuam no nível de realização destas aplicações e suas interfaces. Nas próximas seções apresentaremos algumas destas propostas e discutiremos o suporte que cada uma delas oferece ao *designer*.

2.2 *Frameworks* e Diretrizes

O objetivo de um *framework* é organizar e classificar as idéias e conceitos de uma disciplina ou área (Lim e Benbasat, 1991). Diretrizes, por sua vez, são princípios que devem ser levados em conta ou seguidos para se alcançar um determinado objetivo. Assim, eles são normalmente os primeiros a serem criados quando uma nova área emerge.

Lim e Benbasat (1991) criaram um *framework* para a comunicação de interfaces de grupo. O objetivo de seu trabalho é apresentar um método que guie pesquisadores na análise de requisitos de comunicação para grupos e seus membros, em sistemas de suporte ao trabalho colaborativo. Baseando-se em uma análise do fluxo de comunicação e tendo em foco a necessidade de se acomodar em um mesmo grupo diferentes estilos e preferências, bem como possibilitar transições suaves entre o trabalho individual e o de grupo, os autores propõem dimensões básicas de comunicação. Para cada uma destas dimensões, eles definem um conjunto de possíveis valores e discutem algumas das repercussões de cada um deles no projeto de interface.

Este trabalho trata de uma questão fundamental de sistemas multi-usuário, e permite um melhor entendimento do fenômeno da comunicação nestes sistemas, tanto por parte de pesquisadores, quanto de *designers*. No entanto, os autores não mostram no seu trabalho como as dimensões propostas estão relacionadas umas com as outras e como os valores de uma dimensão afetam, ou não, as demais. Assim, embora o *framework* permita ao *designer* descrever um modelo de comunicação de um grupo, ele (1) não guia o *designer* na escolha dos valores a serem atribuídos a cada dimensão, (2) não apresenta nenhuma garantia ou indicador de qualidade das escolhas feitas pelo *designer* e, (3) não oferece indícios de como as outras características do grupo, como por exemplo a estrutura do grupo, interdependência das tarefas, entre outras, afetam e são afetadas por esta descrição da comunicação.

Por sua vez, mais do que possibilitar o entendimento de fenômenos e abrir caminho para pesquisa, as diretrizes buscam aplicar resultados de pesquisas. No caso de sistemas de grupo, elas apresentam um conjunto de princípios genéricos cujo objetivo é criar balizas para as escolhas do *designer*.

[1]Em Engenharia de Software a palavra *framework* é usada para se referir a descrições abstratas que facilitam o re-uso de soluções de *design* e código. No entanto, neste trabalho usaremos *framework* para referir a estruturas que organizam conceitos e orientam pesquisa em alguma área (Lim e Benbasat, 1991).

Hamalainen et allii (1991) propõem diretrizes para o *design* de interfaces de sistemas de suporte a times. Eles consideram times como sendo duas ou mais pessoas trabalhando em uma tarefa comum, com um objetivo comum[2] em um ambiente organizacional. Seus princípios são baseados na análise de questões e critérios de avaliação para este tipo de interfaces e têm como objetivo auxiliar *designers* na satisfação destes requisitos.

Um conjunto de diretrizes mais genéricas que as apresentadas por Hamalainen et allii, é apresentado por Mandviwalla e Olfman (1994) . Embora suas diretrizes não sejam exclusivamente destinadas ao *design* de interfaces, muitas delas se relacionam diretamente com a interface das aplicações. Seu trabalho delineia as limitações de sistemas de *groupware* correntes e propõe-se a criar diretrizes que as resolvam através da adoção de uma visão mais ampla de trabalho colaborativo. As diretrizes são destinadas a aplicações *groupware* que suportem o trabalho colaborativo de grupos em organizações e têm como objetivo serem amplas e genéricas.

Mandviwalla e Olfman explicitam no seu trabalho que a intenção é apoiar o projetista no início do processo de *design*, possibilitando que ele se certifique de que todas as necessidades do grupo foram consideradas. O trabalho também deixa claro que subconjuntos de diretrizes para grupos específicos devem ser desenvolvidos a partir do conjunto genérico.

É fácil ver que os dois conjuntos de diretrizes anteriormente apresentados são fundamentados em diferentes problemas de sistemas de grupo. Diretrizes apresentadas por Grudin (1994) servem para frisar justamente a diversidade de problemas identificados nestes sistemas e perspectivas para melhorar a qualidade dos mesmos. No seu trabalho, Grudin mostra oito problemas que podem levar ao fracasso de um *groupware*. Para cada um destes problemas, aos quais ele se refere como desafios de *design*, ele apresenta sugestões de como contorná-lo. Seu trabalha resulta em métodos que devem ser seguidos para que estes desafios sejam vencidos.

Apesar de se proporem a resolver diferentes problemas, as diretrizes têm por objetivo final dar suporte ao *designer* na fase de planejamento do *groupware* como um todo, ou de sua interface especificamente. O projetista pode utilizá-las para garantir que levará em consideração todos os quesitos que elas levantam, como sugerem Mandviwalla e Olfman. No entanto, elas não tornam evidente as opções de solução disponíveis para se atender a estes quesitos e, mais do que isso, não mostram como a definição de um quesito pode afetar, entrar em conflito, ou sustentar o outro. Assim, em casos onde o *designer* não consegue atender a todos os princípios, fica difícil ele comprometer um em favor de outro, uma vez que não são deixadas claras todas as implicações envolvidas na decisão.

2.3 Modelos

O objetivo dos modelos disponíveis neste campo de conhecimento é permitir uma descrição estruturada dos sistemas de grupo. Esta descrição explicita as características tidas como relevantes pelos criadores do modelo, e permite que diferentes aplicações de grupo sejam comparadas com base nas distinções existentes no modelo. Nesta seção apresentamos dois modelos exemplares de *groupware*.

O primeiro destes modelos é apresentado por Ellis e Wainer (1994) . Nele o *groupware* é descrito sob três aspectos: o modelo ontológico, o modelo de coordenação e o modelo da interface com usuário. Ao modelo ontológico cabe a descrição estática dos objetos que são

[2] A definição de times os classifica como um tipo especial de grupos, onde os membros têm um mesmo objetivo e tarefa.

disponibilizados para os usuários para manipulação. As atividades que podem ser realizadas pelos usuários são representadas no modelo de coordenação. Finalmente, o modelo de interface fica encarregado da descrição da interface entre usuários e sistemas e, principalmente, da interface entre usuários. Para cada um destes submodelos, o modelo apresenta dimensões e uma descrição do que cada uma significa.

O outro modelo exemplar é o chamado modelo Denver, desenvolvido por Salvador, Scholtz e Larson (1996). Este modelo foi criado a partir de um workshop de *groupware* e é apresentado mais como uma proposta de modelo do que como modelo acabado. Esta proposta divide o processo de construção de *groupware* em três planos: requisitos, *design* e tecnologia. O plano de requisitos representa os requisitos para e objetivos da aplicação, e o plano de tecnologia está relacionado com a sua implementação. O plano de *design* fica encarregado da descrição funcional do sistema e é o de maior interesse. Eles identificam cinco dimensões que caracterizam um sistema de grupo: (1) as pessoas que participam do grupo e seus papéis dentro deste; (2) os artefatos ou objetos que podem ser produzidos ou consumidos durante a interação; (3) as tarefas e atividades que podem ser executadas pelos usuários; (4) as situações interativas do grupo, ou seja, as relações entre os membros do grupo; e (5) as seqüências de informações que podem ser trocadas e que determinam e identificam intenções e conflitos.

Os dois modelos exemplares identificam dimensões de caracterização bastante distintas, mas ambos permitem uma representação de aplicações de *groupware*. Eles atuam na fase de planejamento e auxiliam o *designer* ao fornecer uma forma de definir conceitos e distinções importantes para um sistema. No entanto, estes modelos não apresentam as interdependências entre as suas dimensões e nem são capazes de fornecer ao *designer* uma indicação de como avaliar a qualidade da sua descrição.

2.4 Apoio à Construção de Interfaces

Assim como surgiram métodos para apoiar o *designer* durante o planejamento da sua interface multi-usuário, também surgiram *toolkits* para dar suporte ao projetista durante a fase de realização da interface. Greenberg e Roseman em um de seus trabalhos (Greenberg e Roseman, 1998) discutem quatro *toolkits* e comparam-nos em relação aos componentes que julgam importantes de serem oferecidos ao *designer*: a arquitetura em tempo de execução, abstrações de programação para *groupware*, gerentes de sessão e *widgets* para grupos. Neste trabalho aternos-emos à apresentação e discussão de *widgets* e fatores que influenciam diretamente o *design* da interface.

A construção de interfaces multi-usuário usando *toolkits* e *widgets* mono-usuário pode ser uma tarefa árdua e frustrante (Roseman e Greenberg, 1996b; Greenberg e Roseman, 1998). Isto, não só por causa do custo de programação para se conseguir coordenar a interação dos diversos usuários, mas também porque os *widgets* mono-usuário não dão conta das novas interações entre usuários e das necessidades que surgem com elas. É necessário que o *designer* possa definir se um elemento de interface é privado ou compartilhado e, para os compartilhados, ele precisa ainda definir quem pode acessá-los e quando, e se existe alguma prioridade de acesso, por exemplo.

Quando várias pessoas têm acesso a um mesmo *widget*, cada uma deve poder observar as ações das outras sobre ele, ou seja, ter *feedthrough* (Dix et al., 1993). As questões que surgem na definição do *feedthrough* incluem: Quando deve um usuário receber o *feedthrough* da ação

do outro? A cada passo da ação (por exemplo, em um campo texto compartilhado, a cada caracter) ou ao final da ação (ou ao final da palavra)? Dewan e Choudary (1992) definem o compartilhamento dos estados de interação de um elemento de interface entre seus usuários como acoplamento estreito ou frouxo[3]. O acoplamento é estreito se as ações de um usuário são atualizadas imediatamente na tela dos demais, e é frouxo se as ações de uma pessoa só são propagadas quando um evento crítico ocorre (por exemplo, a digitação da tecla <ENTER> ao final de uma palavra). Neste caso, o estado final dos *widgets* é o mesmo, mas os intermediários não.

Stefik e seus colaboradores anteriormente propuseram uma classificação, não dos elementos de interface, mas da relação entre os ambientes virtuais compartilhados e o que os usuários vêem (Stefik et al., 1987). Quando todos os usuários que compartilham um ambiente vêem sempre exatamente a mesma imagem deste ambiente, então ele é classificado como WYSIWIS (*What You See Is What I See*, ou O Que Você Vê É O Que Eu Vejo) rígido. Se, no entanto, podem existir diferenças entre os ambientes dos diversos usuários, como por exemplo, diferentes tempos de atualização da informação, ou se nem todos podem ver as mesmas coisas, então o ambiente é classificado como WYSIWIS relaxado. Embora Stefik e colaboradores tenham proposto que a classificação seja feita para a totalidade da interface, em nosso trabalho vamos relaxar esta restrição e nos permitir referir a uma parte da interface como WYSIWIS rígida, e a outra como WYSIWIS relaxada[4].

Todas estas novas questões causaram a adaptação dos *widgets* existentes para levarem-nas em conta. No entanto, os resultados ainda não foram suficientes para dar suporte a várias das atividades de grupos e surgiram, então, novos *widgets* específicos para grupos (Roseman e Greenberg, 1996b; Greenberg e Roseman, 1998; Hazemi e Macaulay, 1996; Ackerman e Starr, 1996). Dentre estes novos elementos de interface, destacam-se os elementos de *awareness* que são aqueles que fornecem ao usuário informação sobre o que acontece "à sua volta", ou seja, sobre o ambiente compartilhado virtual e a interação dos outros usuários com este ambiente (Gutwin et al., 1996; Ackerman e Starr, 1996). O GroupLab (s.d.) é um dos grandes responsáveis pela pesquisa de *widgets* de *awareness* e eles criaram diversos *widgets* deste tipo, mostrando as diferentes informações que podem ser úteis de se apresentar e as formas distintas de o fazer. Ainda não se sabe qual seria o conjunto mínimo de informações deste tipo que deveria ser disponibilizado para o *designer* e nem a melhor forma de apresentação de cada um. À medida que novas aplicações de *groupware* são desenvolvidas, novas informações se mostram necessárias, e *widgets* são criados ou adaptados (Gutwin et al., 1995).

Em suma, os *toolkits* fornecem meios ao *designer* de construir a interface usando *widgets* novos e adaptados e possibilitando a definição de propriedades relevantes como controle de acesso e *feedthrough*. Novos elementos de interface e informações continuam sendo identificados como necessários e incluídos nestes *toolkits*. O apoio destes *toolkits* é fundamental, mas existe ainda espaço para que diferentes formas de suporte os complementem. Afinal, fica a cargo do *designer* tomar todas as decisões sobre que elementos escolher, qual a melhor maneira de controlar o acesso ou definir o acoplamento em uma determinada situação de colaboração.

[3]Em inglês é *tight x loose*.

[4]No trabalho orginal um ambiente com esta descrição seria considerado automaticamente WYSIWIS relaxado.

2.5 Requisitos para o Apoio ao Desenvolvimento de Interfaces Multi-Usuário

Como foi ressaltado neste capítulo, a interface de um sistema multi-usuário envolve tanto a interação usuário-sistema, quanto interações entre usuários. Isto implica que qualquer ferramenta ou técnica de apoio ao *design* de interfaces multi-usuário deve contemplar todas as questões consideradas no *design* de interfaces mono-usuário e ainda complementá-las com os novos quesitos surgidos para grupos. O usuário tem que entender via interface as estruturas do grupo e ter informação suficiente e relevante para interagir de forma eficiente com ele.

Revimos *frameworks*, diretrizes, modelos e *toolkits* que endereçam as novas questões de grupo e buscam dar suporte ao *designer* durante as fases de planejamento e realização de aplicações de grupo e suas interfaces. Estas técnicas e ferramentas apóiam o projetista durante as suas considerações, descrições e definições da interface e durante a construção desta. Contudo, nenhuma delas auxilia o *designer* no processo de escolhas e decisões sobre a interface. Ademais, uma vez feitas as escolhas do *designer*, nenhuma delas apresenta formas de se ter indicações sobre a qualidade destas escolhas.

Os projetistas de interfaces multi-usuário necessitam de um ambiente que apóie todo o processo de desenvolvimento destas interfaces, uma vez que é uma tarefa complexa e requer que o projetista defina a colaboração, comunicação e coordenação entre os membros, além da sua interação com a aplicação. Durante a fase de planejamento desta interface, o *designer* precisa de suporte para especificar o seu modelo de grupo e ter indicadores qualitativos sobre esta descrição. Para isso, não basta que este ambiente lhe ofereça dimensões básicas de descrição, ele precisa lhe fornecer também informações sobre como valores atribuídos a uma dimensão podem afetar ou não as demais. Assim, o *designer* pode avaliar as conseqüências das suas decisões ao tomá-las.

Uma vez especificado o modelo conceitual do grupo, o ambiente deve dar suporte também à fase de realização desta interface. A fase de realização envolve mais do que a implementação da interface. O *designer* deve ter apoio na escolha de *widgets* para representar os elementos desejados e nas decisões de como combiná-las. Como o conjunto de elementos de interfaces multi-usuário ainda está em desenvolvimento, o *designer* precisa saber quando já existe um elemento para representar a informação desejada, e quando um novo elemento precisa ser criado. Quando um novo *widget* é necessário, fica a cargo do *designer* defini-lo. Embora o ambiente não possa criar novos *widgets* para o *designer*, ele pode apoiar o projetista na construção deste novo elemento e permitir a sua inclusão em uma biblioteca de *widgets*.

Este ambiente descrito é idealizado, e ainda requer muito trabalho e pesquisa para que seja viável, e mais ainda para que seja construído. Neste trabalho, damos um passo nesta direção, propondo modelos que apóiem a especificação em alto-nível de uma parte do modelo de grupo durante a fase de planejamento.

Chapter 3

Engenharia Semiótica

A Semiótica é uma disciplina que estuda signos, sistemas de significação e comunicação, e os processos culturais envolvidos nestes sistemas (Eco, 1976). As abordagens semióticas de IHC (Nadin, 1988; Andersen et al., 1993; Nake, 1994; de Souza, 1993; Jorna e Van Heusden, 1996; de Souza, 1996), por sua vez, aplicam a teoria Semiótica ao processo de desenvolvimento de interfaces, criando assim um quadro referencial teórico para a exploração de propriedades comunicativas no meio computacional.

Neste capítulo apresentaremos a Engenharia Semiótica (de Souza, 1993; de Souza, 1996), que é a abordagem semiótica que utilizamos neste trabalho, e os conceitos de Semiótica necessários para o seu entendimento. Nossa contruibuição é a de estender o modelo original da Engenharia Semiótica para incluir o *design* de interfaces multi-usuário. Ao final do capítulo, discutiremos um método de *design* sob a perspectiva da Engenharia Semiótica.

3.1 A Abordagem da Engenharia Semiótica

A Engenharia Semiótica entende o processo de *design* de interfaces como sendo um ato de comunicação entre o projetista e o usuário. Para que haja comunicação entre duas pessoas, é necessário que uma delas seja a emissora, a outra a receptora, que elas compartilhem um código e que exista um meio de comunicação. O processo se completa quando o emissor cria uma mensagem no código compartilhado por ele e o receptor e envia esta mensagem pelo meio de comunicação ao receptor, que então a decodifica (Jackobson, 1970). A mensagem criada é formada por signos, onde **signo** é tudo aquilo que representa algo para alguém (Peirce, 1958). Ao receber a mensagem, o receptor interpreta estes signos, decodificando a mensagem, e cria mentalmente uma idéia do que o emissor quis dizer. Esta idéia que ele cria é chamada de **interpretante**, e pode ela mesma ser a semente para a derivação de novos interpretantes na mente do receptor, em uma cadeia indefinidamente longa de associações[1]. O processo de geração desta cadeia se chama **semiose ilimitada** (Eco, 1976) (ver Figura 3.1) . Embora a cadeia de interpretações seja potencialmente ilimitada, na realidade, em algum momento, o

[1]Para ilustrar o processo de comunicação, imagine-se que uma pessoa diz a outra a palavra <cachorro>. A palavra <cachorro> é um signo que representa o animal cachorro na língua portuguesa, que é o código compartilhado pelo emissor e receptor. A pessoa que ouve a palavra cachorro, imagina um cachorro, que pode ser, por exemplo, um pastor alemão, que lhe faz pensar no pastor alemão que teve quando criança, que lhe faz pensar no canil onde ficava este cachorro e assim indefinidamente.

receptor interrompe esta cadeia. A interrupção acontece quando o receptor acredita que ele tem uma boa hipótese sobre o que o emissor quis dizer, ou quando ele conclui que ele não é capaz de produzi-la, ou ainda quando não está disposto a criar tal hipótese. Nestes casos, o receptor pode dar continuidade (ou não) ao processo de comunicação com o emissor, trocando mensagens com ele e revezando-se entre os papéis de emissor e receptor.

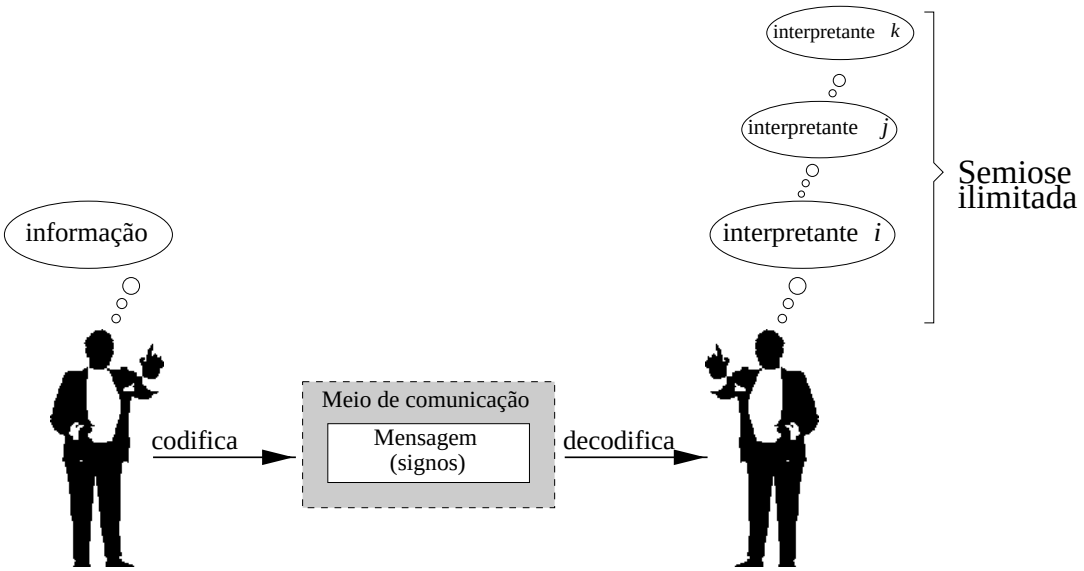

Figure 3.1 – Processo de comunicação entre duas pessoas.

Na comunicação entre *designer* e usuário, o meio é o computador e o código pode ser, por exemplo, uma linguagem de comandos ou, no caso de interfaces gráficas, o padrão WIMP[2]. O objetivo desta comunicação é o *designer* transmitir ao usuário o modelo de usabilidade de uma aplicação, ou seja, como o usuário pode com ela interagir para fazer um uso eficiente, produtivo e criativo dela (Leite, 1998). Para atingir seu objetivo, o *designer* primeiramente forma uma idéia do que ele quer comunicar ao usuário. Esta idéia é o seu modelo conceitual de usabilidade potencial da aplicação e resulta das suas decisões de projeto baseadas em: (1) sua interpretação sobre o problema que o usuário quer resolver e (2) sua racionalização sobre como ele pode resolvê-lo. O produto desta interpretação e racionalização é o que o projetista pretende comunicar ao usuário através de sua mensagem, que é de fato a interface. À medida que o usuário interage com a interface, ele próprio cria o seu interpretante sobre a mensagem do projetista, o que se torna o seu modelo adquirido de usabilidade da aplicação (ver Figura 3.2). O modelo adquirido de usabilidade da aplicação do usuário será quase que inevitavelmente diferente do modelo conceitual de usabilidade potencial do *designer*. No entanto, o sucesso desta comunicação depende apenas de estes modelos serem consistentes um com o outro (de Souza, 1996). Por exemplo, também na comunicação humana, quando ouvimos alguém dizer <Meu filho foi mordido por um cachorro.>, dificilmente teremos em mente a imagem exata do cachorro agressor, mas com certeza, a despeito disto, nos penalizaremos com o fato e a comunicação terá sucesso.

A interface é então a mensagem do *designer* para o usuário e ela deve transmitir a resposta a duas perguntas fundamentais: (1) Qual a interpretação do *designer* sobre o problema do

[2]WIMP é o acrônimo de *Windows, Icons, Menus and Pointers*, ou seja, Janelas, Ícones, Menus e Apontadores.

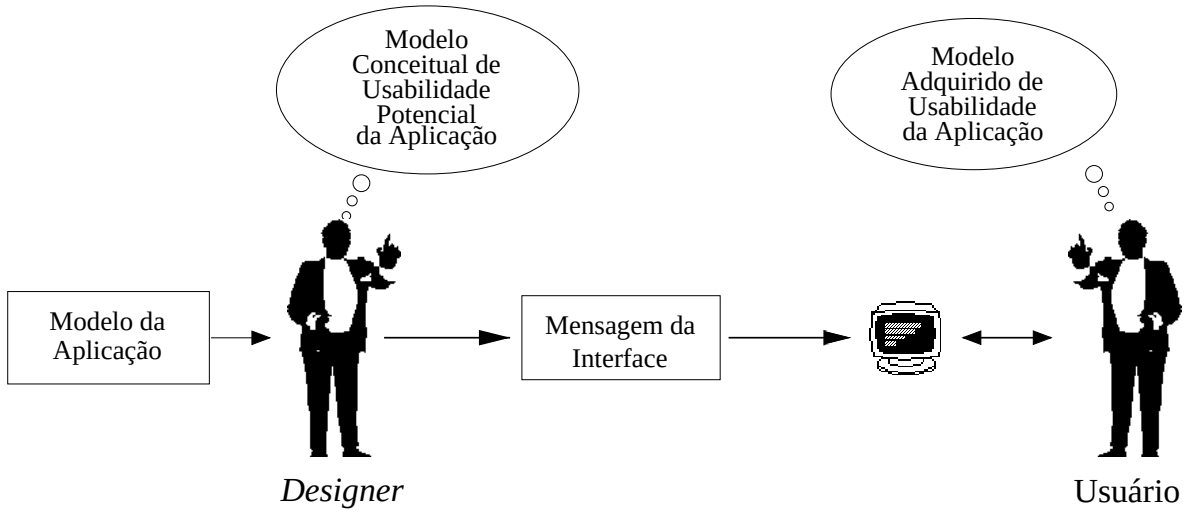

Figure 3.2 – Comunicação entre designer e usuário.

usuário? (2) Como o usuário pode interagir com a aplicação para resolver este problema? Como a interface é capaz de ela própria trocar mensagens com o usuário, ela é então uma meta-mensagem do *designer* para o usuário. Além disso, ela é uma mensagem unilateral, uma vez que o usuário não pode dar continuidade ao processo de comunicação entre ele e o projetista (de Souza, 1993) naquele mesmo contexto de interação.

Como toda mensagem, a interface pode ser decomposta em expressão e conteúdo. A sua expressão é o seu modelo de interação, que é o conjunto de comandos que o *designer* oferece ao usuário para interagir com a aplicação e é a resposta para a pergunta (2). O seu conteúdo é a sua funcionalidade, ou seja, o que a aplicação é capaz de fazer, e a resposta para a pergunta (1) (Leite, 1998). O modelo de usabilidade da aplicação é então formado pela combinação dos modelos de interação e funcionalidade.

A Engenharia Semiótica prioriza a expressão do projetista sobre a interpretação do usuário, mudando o foco do *design* de interfaces do seu aprendizado[3] para o seu ensino. A idéia é que com um projetista capaz de expressar com mais clareza as suas motivações, intenções e decisões, aumentam as chances de o usuário entender o que ele "quis dizer" e fazer melhor uso da aplicação. A teoria Semiótica é uma base apropriada, uma vez que trata o fenômeno de comunicação de forma integrada, onde tanto a expressão quanto a interpretação são igualmente importantes.

3.2 A Comunicação entre *Designer* e Grupo

O modelo original de Engenharia Semiótica foi desenvolvido para a comunicação entre o *designer* e os usuários de uma aplicação mono-usuário. Enquanto todos os pontos levantados neste modelo são relevantes para o *design* de interfaces multi-usuário, estas últimas levantam novas questões que o modelo original não considera. Assim, nesta seção estendemos a proposta

[3]As atuais perspectivas cognitivas estudam como criar interfaces que facilitem sua interpretação por parte do usuário (Norman, 1986). Contudo, não abordam, senão secundariamente, o problema complementar que aqui elegemos como focal.

inicial da Engenharia Semiótica para incluir tais questões.

Ao criar uma aplicação de grupo o projetista tem que levar em conta, não só o problema que o grupo quer resolver, e como fazê-lo, mas também a interação entre os membros deste grupo. Assim, parte do modelo conceitual de usabilidade potencial da aplicação do *designer* é o seu modelo conceitual do grupo. Em um grupo, diferentes membros podem assumir diferentes papéis, relacionando-se e comunicando-se de diferentes formas com os demais. Desta forma, para criar o seu modelo conceitual do grupo, o *designer* deve tomar decisões em relação aos papéis que os membros do grupo podem assumir, a estrutura do grupo em que tais membros estão inseridos, as tarefas de cada um e como elas se correlacionam, com que outros membros cada um pode comunicar-se e através de que linguagem e protocolos o fazem. As respostas a todas estas perguntas adicionais devem também ser transmitidas aos membros do grupo via interface (Figura 3.3).

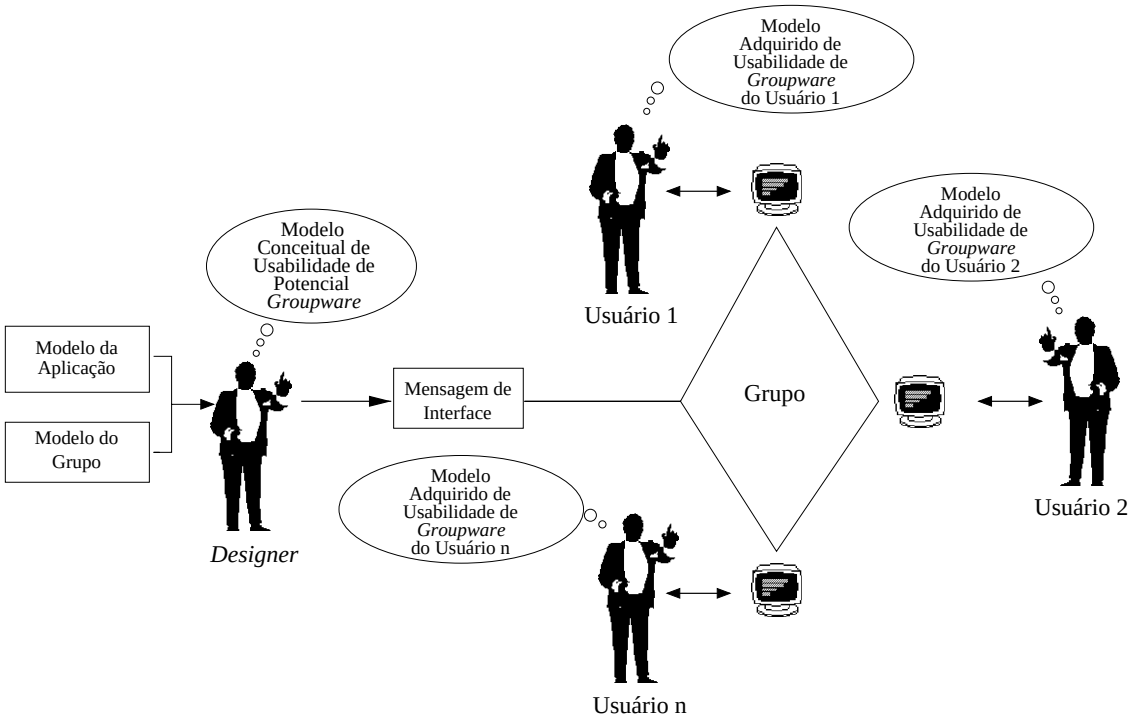

Figure 3.3 – Comunicação entre designer e os membros de um grupo.

Diferentes membros do grupo, com papéis e tarefas distintas, podem ter diferentes interfaces. Isto significa que o *designer* está enviando diferentes mensagens a estes membros ou subgrupos de membros. Assim, eles definitivamente atribuirão diferentes significados à interface. Não obstante, o significado atribuído por cada um à aplicação como um todo, à sua participação no grupo e aos modelos de comunicação e colaboração do grupo deve ser coeso e consistente com o dos demais membros, assim como com o modelo conceitual de usabilidade potencial da aplicação do *designer*. Isto só é possível se as diferentes mensagens enviadas pelo projetista a diferentes membros forem consistentes entre si e se os mecanismos de comunicação entre os membros do grupo assegurarem a infra-estrutura de canais para se manter a coesão.

O modelo de usabilidade da aplicação passa a ter então, além dos modelos de interação e funcionalidade, o modelo de grupo. A interdependência entre os modelos de interação e funcionalidade do modelo de usabilidade mono-usuário é mantido e, além disso, surge a interação

de cada um deles com o modelo de grupo. Isto acontece porque tanto o que o usuário pode fazer, quanto o modo como ele pode fazê-lo dependem do modelo do grupo e do papel do membro dentro deste modelo.

Como visto, as interfaces de aplicações mono-usuário são artefatos de meta-comunicação, já que são mensagens capazes de enviar e receber, elas próprias, mensagens. No cenário de aplicações multi-usuário elas assumem mais uma função comunicativa ao permitirem a comunicação entre membros do grupo. A interface multi-usuário passa a ser também um artefato de mediação de comunicação, ou seja um "medium" ou meio.

3.3 Método de *Design* para Usabilidade

Ao criar seu modelo conceitual da usabilidade potencial da aplicação, o *designer* está, na verdade, limitando e estruturando o seu espaço de soluções. Para um mesmo problema, podem existir inúmeras interpretações e soluções. À medida que o *designer* especifica e refina o seu modelo conceitual de usabilidade potencial, ele limita cada vez mais este espaço de soluções, até finalmente alcançar a solução final.

O processo de limitação do espaço de soluções é o próprio processo de *design*. Este processo pode ser subdividido em duas etapas: planejamento e realização[4]. Na etapa de planejamento, o *designer* projeta e decide o conteúdo da sua mensagem, ou seja, o modelo de funcionalidade da aplicação, e no caso de sistemas de grupo, também o modelo de grupo. Na etapa de realização, o *designer* define a expressão da sua mensagem, ou seja, o seu modelo de interação (Prates et al., 1998). Observe que estamos considerando o processo de *design* como sendo *top-down*.

O número de passos ou refinamentos sucessivos necessários para se atingir a interface final depende do *designer* e da aplicação. Sob a perspectiva da Semiótica, podemos pensar nos refinamentos sucessivos como sendo a própria semiose do *designer*. O *designer* alcança a interface final quando ele se dá por satisfeito (ou algum outro fator atua, como por exemplo, o prazo de entrega se esgota) e limita pragmaticamente esta semiose teoricamente ilimitada. Esta interface final é então o "congelamento" do seu interpretante, uma vez que mesmo que ele mude de idéia em relação às suas decisões de projeto, a interface (ou meta-mensagem) não pode mais ser mudada (exceto pela liberação de uma nova versão).

A interface final obtida pelo projetista pode ser vista como um *template* (ou molde) de interfaces, no sentido de que ela é uma descrição em alguma linguagem de programação de todos os estados possíveis pelos quais ela pode passar e dos valores válidos que seus elementos podem assumir em tempo de execução (Gelernter e Jagannathan, 1990). A cada vez que a interface é executada ela pode passar por caminhos distintos, de acordo com os dados fornecidos e escolhas feitas pelo usuário. No caso de aplicações multi-usuário, membros utilizando interfaces iguais (ou seja, geradas pelo mesmo *template*) podem ter experiências distintas. No momento da execução, a interface passa então de uma descrição a uma **virtualidade**, ou seja um "mundo", no qual o usuário percebe, age e responde a experiências (Winograd, 1996). Em aplicações multi-usuário, esta virtualidade inclui também a experiência de interação que os usuários têm uns com os outros.

A Figura 3.4 mostra o processo de *design* conforme descrito aqui. Os planos na vertical

[4]Estas etapas são análogas às etapas de geração de texto (Paris et al., 1991), como mencionado na seção 1.1.

representam os passos de refinamentos sucessivos do projetista até que ele chegue à interface final. Na horizontal, o plano mostra um instrumento de execução desta interface, contendo virtualidades distintas.

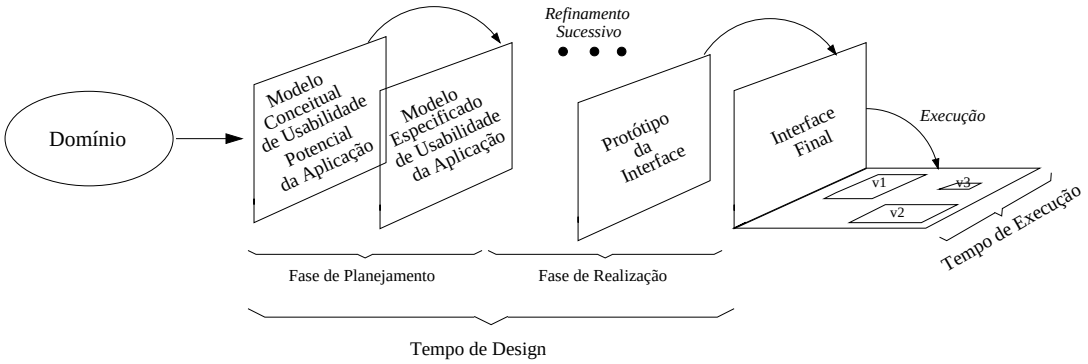

Figure 3.4 – Processo de design de uma interface e suas virtualidades.

3.4 Requisitos de Apoio ao *Designer*

O foco da Engenharia Semiótica na expressão do *designer* tem como conseqüência direta gerar expectativas sobre o desenvolvimento de ferramentas, técnicas e modelos que apóiem esta expressão. Como foi mostrado na seção anterior o processo de *design* tem vários passos, e o ideal seria que o projetista tivesse suporte em todos estes passos, desde a concepção do seu modelo conceitual de usabilidade potencial até a construção da interface.

Toolkits e bibliotecas de *widgets* que dão suporte à construção de interfaces já estão disponíveis para os projetistas. No entanto, para os outros passos do processo existe pouco ou nenhum suporte. O espectro de modelos e técnicas que podem auxiliar o *designer* é amplo e precisa ser explorado, mas certamente nele cabe a investigação tanto de modelos e técnicas genéricos, quanto daqueles dependentes de domínio.

Os primeiros passos nesta direção já estão sendo dados. Em seu trabalho Martins (Martins, 1998) criou um formalismo para projetistas de interfaces gráficas de modeladores geométricos se expressarem. Já Leite (1998) criou uma linguagem de *design* de usabilidade genérica para auxiliar projetistas de interfaces mono-usuário no passo de realização, ou seja, na definição do modelo de interação da interface e na sua relação com o modelo de funcionalidade. Em nosso trabalho seguimos a mesma linha de investigação de Martins e Leite, e propomos uma linguagem de *design* genérica que dá suporte ao projetista de interfaces multi-usuário no processo de especificação do seu modelo conceitual do grupo. Esta linguagem reflete as premissas de nosso modelo geral sobre o projeto deste tipo de interface e incorpora um sistema de valores sobre o que é destacado e o que é abstraído oriundo do estado atual de conhecimento em Engenharia Semiótica.

Chapter 4

Modelo Abstrato do Componente Multi-Usuário do Artefato de Meta-Comunicação

No capítulo 2, apresentamos métodos existentes para apoiarem *designers* de interfaces multi-usuário no desenvolvimento destas. Mostramos também a necessidade que existe de se auxiliar o projetista não apenas com ferramentas que permitam a descrição e construção das interfaces, mas também que permitam ao *designer* ter indicações da qualidade do seu projeto. A abordagem de Engenharia Semiótica, por sua vez, requer a criação de metodologias que permitam ao *designer* se expressar com clareza, na tentativa de facilitar a compreensão por parte dos usuários dos seus objetivos e motivações ao tomar decisões no projeto.

Neste trabalho nossa intenção é unir a motivação da Engenharia Semiótica com os requisitos de desenvolvimento de interfaces multi-usuário. Assim, propomos um modelo que tem como objetivo servir de base para o desenvolvimento de instrumentos de apoio ao *designer*, que lhe permitam expressar melhor o seu modelo de grupo a ser transmitido na interface para os membros deste grupo, fornecendo a ele indicadores qualitativos a respeito da sua mensagem. Chamamo-no de modelo abstrato do componente multi-usuário do artefato de meta-comunicação, e referiremos a ele como modelo abstrato de meta-comunicação.

O modelo abstrato de meta-comunicação propõe que se forneça ao projetista uma linguagem de *design* (LD) que lhe permita expressar seu modelo conceitual do grupo. Onde, uma LD é toda aquela na qual o projetista pode expressar o seu *design* (Winograd, 1996). A parte léxica desta linguagem deve ser formada por unidades descritivas básicas que permitam a definição do grupo, enquanto que a parte semântica deve ser formada por regras heurísticas que atuem sobre estas unidades. O modelo abstrato de meta-comunicação define um conjunto de características básicas a partir do qual as unidades descritivas da LD devem ser definidas. Para as regras o modelo estabelece condições a serem atendidas, uma vez que a definição destas regras depende das unidades descritivas da LD.

O conjunto de características básicas do modelo abstrato de meta-comunicação permite tanto a descrição dos elementos do grupo que independem do tempo, ou seja, não mudam com o tempo, quanto daqueles que sofrem transformações no decorrer do tempo. Estas dimensões estão definidas no modelo de forma intensional ou extensional, independentemente de qualquer domínio. À medida que o *designer* descreve seu modelo conceitual de um grupo, ele atribui valores às dimensões, fazendo então a sua ligação com o domínio.

As regras heurísticas devem atuar sobre a descrição do *designer*, expressando inferências semânticas com base nas unidades descritivas da linguagem e nos valores atribuídos a elas. Ao fazer estas inferências, as regras atribuem significado às combinações de valores das dimensões. O objetivo destas regras é chamar a atenção do *designer* para combinações que aparentemente não fazem sentido.

Neste capítulo apresentamos em detalhe o modelo abstrato de meta-comunicação que estamos propondo. Na próxima seção apresentamos as dimensões básicas, através das quais o *designer* descreve um grupo. Em seguida mostramos que tipo de variações no tempo, ou temporalidades, o *designer* pode descrever usando este modelo. Por fim, descrevemos as condições que as regras do componente semântico da linguagem de *design* devem atender.

4.1 Dimensões de Descrição

Nesta seção apresentamos o conjunto de características básicas do nosso modelo abstrato de meta-comunicação que devem servir de base para as unidades descritivas da linguagem de *design*. Estas dimensões básicas foram definidas baseadas na literatura de sistemas multi-usuário e nas suas características, conforme apresentado no Capítulo 2. Para algumas destas características estamos propondo uma classificação que, ao nosso ver, permite ao projetista uma descrição mais precisa do grupo.

Papéis e Hierarquia

Em um mesmo grupo diferentes pessoas podem assumir diferentes responsabilidades, e ter tarefas ou subtarefas distintas. Assim, os papéis de um grupo podem exigir qualificações, habilidades ou experiências específicas. Ao fazer o *design* de uma aplicação de grupo, o projetista precisa definir os diferentes papéis necessários para que o grupo funcione com eficiência e atinja seus objetivos.

O relacionamento entre os membros, que assumem os diversos papéis do grupo, e a distribuição da autoridade entre eles define a estrutura hierárquica do grupo. Assim, o *designer* pode definir desde grupos onde não existe hierarquia, ou onde a hierarquia é deixada para o protocolo social, até aqueles que possuem vários níveis rígidos de hierarquia.

Níveis de Interação

Nas aplicações de grupo os usuários interagem com o sistema e com os demais usuários via interface. Assim, podemos identificar três níveis em que a interação em um sistema de grupo pode se suceder: nível individual, nível interpessoal e, finalmente, nível de contexto. A Figura 4.1 mostra estes níveis em um sistema de grupo.

No **nível individual** o usuário interage apenas com a sua parte privada da aplicação, com o objetivo de executar alguma tarefa individual que lhe foi conferida. Neste nível, a interação do usuário é exclusivamente com o sistema. O que pode diferenciá-la de uma interação mono-usuário é a informação relativa ao trabalho do grupo ou de outros membros que, porventura, possa ser necessária para executar-se uma tarefa.

A comunicação entre os usuários é capturada no **nível interpessoal**. É neste nível que cada membro interage com um ou mais outros membros através da interface. A freqüência desta

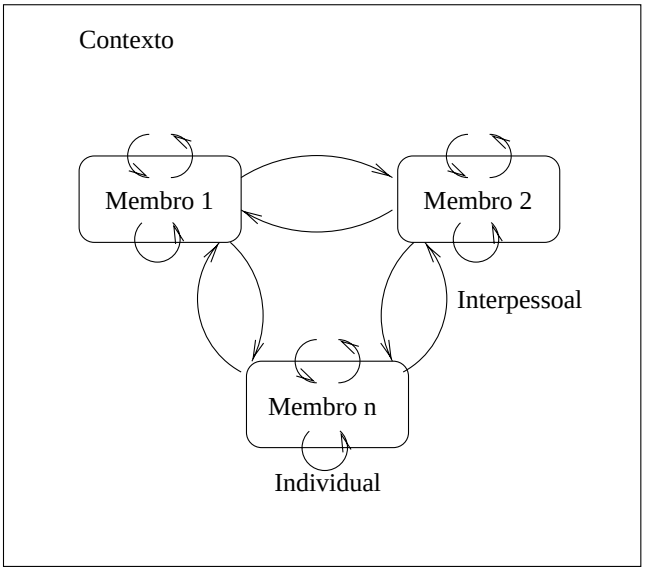

Figure 4.1 – Níveis de interação em um sistema de grupo.

interação e a sua importância para o trabalho do grupo podem variar de acordo com o domínio e objetivos do grupo. Assim, a interação entre membros pode ir de breve e opcional a longa e obrigatória (Lim e Benbasat, 1991).

Finalmente, o **nível de contexto** representa o contexto do grupo em que os membros estão inseridos. Este contexto inclui tanto informações sobre a organização do grupo, quanto sobre seu dinamismo. As informações sobre a organização do grupo se referem à estrutura hierárquica do grupo, a relação entre as tarefas dos diversos membros, etc. O dinamismo, por sua vez, abrange as informações de *awareness* como quem está "presente", em que cada membro está trabalhando, quanto da tarefa do grupo já foi completada, que oportunidades ou necessidade de providências há, e assim por diante.

Objetos

Chamamos de objeto tudo aquilo que faz parte da aplicação e sobre o que o usuário pode agir. Por agir entenda-se aplicar transformações arbitrárias e manipular seus atributos. O traço de propriedade dos objetos pode ser descrito como privado, compartihado ou público. Um objeto ser **privado** significa que ele pertence a apenas um membro, que é o único a ter controle sobre ele e poder agir sobre ele. O objeto **compartilhado** pertence a dois ou mais membros, ou seja, um subconjunto de membros pode agir sobre ele. Finalmente, o **público** pertence a todo o grupo. Para os objetos compartilhados e públicos é necessário que o *designer* defina controles e prioridades de acesso. Embora os donos dos objetos sejam os únicos capazes de agir sobre eles, isto não necessariamente significa que eles sejam os únicos a ter conhecimento ou habilidades de comunicação (ver próxima dimensão) sobre os objetos.

Habilidades de Comunicação

As habilidades de comunicação se referem às formas que os membros do grupo têm disponíveis para se comunicarem a respeito de objetos e de outras coisas. Nós identificamos três eixos

básicos constitutivos de comunicação: visão, discurso e ação (Prates et al., 1997). Quando um membro pode agir sobre um objeto, ele automaticamente deve poder ver aquele objeto. De outra forma, ele teria que agir às cegas sobre o objeto. Contudo seus objetos não são os únicos que ele pode ver. O *designer* pode permitir a membros do grupo verem objetos pertencentes a outros membros, ou permitir que os donos de um objeto decidam se o mostram ou não para outros membros. Além de poderem ver um objeto, os membros podem também falar sobre objetos. Assim, como no caso da visão, donos e não donos podem ter a habilidade de falar sobre um objeto. Em contrapartida, não é essencial que o dono de um objeto sempre possa falar sobre ele, pois o objeto pode ser confidencial. Quando o objeto é compartilhado ou público, é recomendável que os seus donos possam falar sobre ele, para que possam coordenar suas ações.

Enquanto ver e falar decorrem de atos de comunicação diretos, agir pode ser considerado (quando dissociado do discurso ou da observação) uma forma de comunicação indireta. Quando membros compartilham um objeto, e não possuem nenhuma outra forma de comunicação um com o outro, é através dos atos de cada membro sobre o objeto, que os demais podem fazer inferências sobre suas intenções e objetivos.

Todas estas habilidades comunicativas possibilitam aos membros ganhar conhecimento sobre os objetos. Em função delas podemos fazer uma apreciação qualitativa do conhecimento que pode ser adquirido. A visão, permite que o membro tenha uma exposição direta ao objeto, ou seja, a partir do que ele vê ele pode criar sua própria interpretação sobre o seu significado. A fala, por sua vez, favorece o conhecimento indireto. Quando alguém fala sobre algo, esta pessoa dá a sua interpretação sobre o tema do discurso. Assim, se um membro não pode ver um objeto, mas pode ouvir sobre ele, ele terá acesso apenas à interpretação de outros membros sobre aquele objeto, que pode ser errônea, conflitante, ou ainda pode vir acompanhada por um julgamento de valores sobre o objeto. Se também puder falar sobre este conhecimento indiretamente adquirido a outros, potencializa-se na rede de comunicação a emergência de um grande colapso de interação.

Modelos de Colaboração

Um fator decisivo sobre o quanto os membros devem trabalhar em cada um dos níveis de interação, quanto eles devem comutar de um nível para outro e quais são as suas necessidades comunicativas, é o modelo de colaboração do grupo. O modelo de colaboração representa a relação e interdependência entre as tarefas dos membros do grupo. Nós classificamos as possíveis colaborações entre os membros em cinco modelos distintos: o modelo de ilha, os modelos de encaixe rígido e nebuloso, o modelo de sobreposição e, finalmente, o modelo único ou coincidente.

No **modelo de ilha**, mostrado na Figura 4.2, as tarefas dos membros são completamente independentes umas das outras. Assim, os usuários tenderão a passar a maior parte do seu tempo, se não todo ele, no nível individual. Além disso, não será necessário que eles se comuniquem entre si. Observe-se que, embora as tarefas não exijam a comunicação entre os membros, o *designer* pode fornecer-lhes os meios de se comunicarem, permitindo assim que um protocolo social de comunicação, e até mesmo de colaboração, seja estabelecido se os membros acharem conveniente.

Nos modelos de encaixe as tarefas dos membros têm uma interseção entre si, e o trabalho de um membro afeta o trabalho de um ou mais outros. Se o *designer* define que existe uma

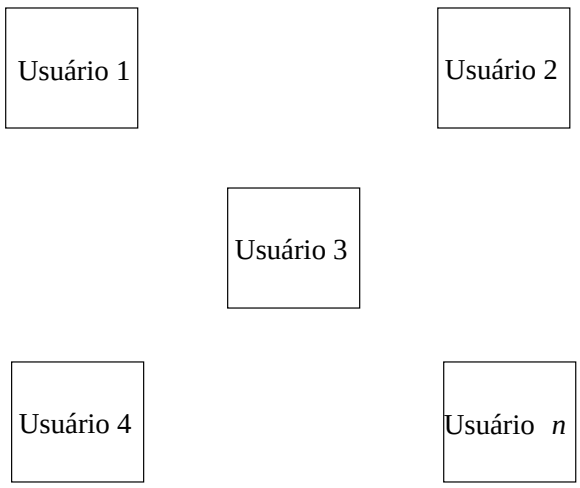

Figure 4.2 – Modelo de colaboração de ilhas.

interação entre as tarefas dos membros, mas deixa a definição de como ela se dá para o protocolo social, então este modelo é o de **encaixe nebuloso**. Neste caso (ver Figura 4.3), a tarefa será executada no nível individual, mas ter-se-á muita atividade nos níveis interpessoal e de contexto, uma vez que a negociação da interação entre as tarefas é necessária. Para que esta negociação aconteça, os membros terão que se comunicar.

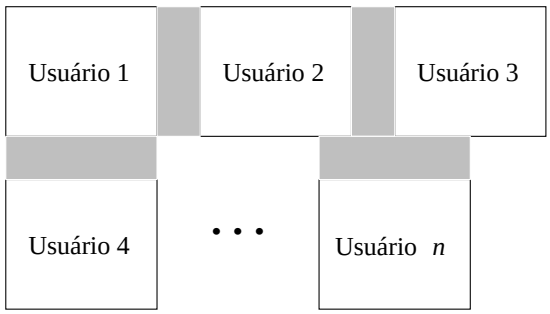

Figure 4.3 – Modelo de colaboração de encaixe nebuloso.

Se, porém, em vez de deixá-la a cargo dos usuários, o projetista definir a interação entre as tarefas no protocolo tecnológico, então o modelo é de **encaixe rígido** (ver Figura 4.4). Neste caso, o nível de interação predominante será o individual. Não obstante, o nível interpessoal possivelmente será necessário, ao menos para que os membros possam informar o *status* da tarefa àqueles que estão interessados. Assim, a necessidade de formas de comunicação será pequena, mas existente.

Para ilustrar a diferença entre os modelos de encaixe nebuloso e rígido, imagine-se um editor de texto colaborativo, onde diferentes autores escrevem diferentes capítulos de um livro, e suponha-se que cada capítulo depende do capítulo anterior. Além disso, antes de começar a escrever cada autor deve fazer um esquema de seu capítulo. Se o projetista deste editor definir que o autor de cada capítulo recebe como referência do capítulo anterior este esquema, então o modelo de colaboração é o de encaixe rígido. Entretanto, se ele deixar a cargo dos autores dos capítulos resolverem entre si se a referência do capítulo anterior será o esquema, o capítulo

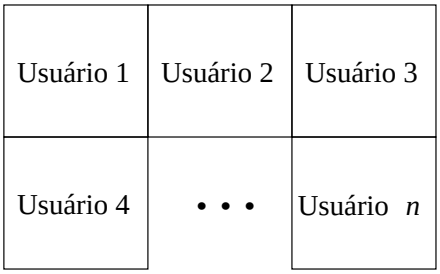

Figure 4.4 – Modelo de colaboração de encaixe rígido.

inteiro ou até mesmo um resumo deste capítulo, então o modelo de colaboração é o de encaixe nebuloso.

O próximo modelo representa a situação onde existe uma parte da tarefa de dois ou mais membros, na qual eles têm que trabalhar juntos. Chamamos este modelo de **modelo de sobreposição** e a Figura 4.5 o ilustra. Neste modelo, parte do trabalho do membro é feito no nível individual, enquanto outra parte é compartilhada no nível interpessoal. Além disso, para que estes membros possam coordenar seu trabalho em equipe, é preciso que eles tenham ao menos as informações de contexto relevantes. A execução de uma tarefa compartilhada sem a habilidade de falar, para planejar e discutir ações, pode não ser impossível, mas é bastante difícil e potencialmente ineficiente.

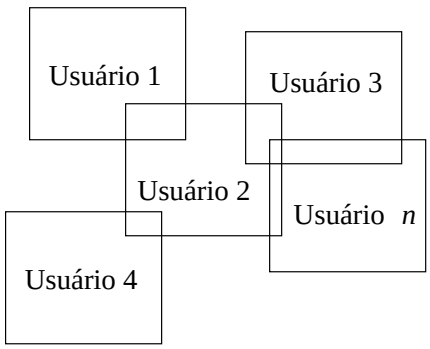

Figure 4.5 – Modelo de colaboração de sobreposição.

Vale a pena distinguir o caso específico do modelo de sobreposição, onde toda a tarefa é compartilhada entre dois ou mais membros. Este caso específico, ao qual damos o nome de **modelo único ou coincidente**, está representado na Figura 4.6. Neste situação os níveis interpessoal e de contexto são os predominantes. O nível individual ou é inexistente, ou existe com o objetivo de permitir ao usuário fazer rascunhos, ou coisas do gênero. Assim como no modelo de sobreposição, a comunicação tem fundamental importância no planejamento e coordenação das tarefas.

Em um mesmo grupo pode haver um ou mais modelos de colaboração sendo usados. Isto acontece porque subgrupos de participantes podem ter requisitos e relacionamentos distintos. Note-se que se um mesmo usuário tem que agir em diferentes situações em modelos de colaboração muito distintos, ele tem propensão a ficar confuso e a agir inconsistentemente em cada uma destas situações.

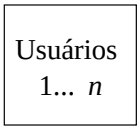

Figure 4.6 – Modelo de colaboração único ou coincidente.

Widgets

Para que o usuário entenda a mensagem sendo enviada pelo *designer*, ele deve interagir com a interface. Através desta interação com a expressão da mensagem, ele entende o seu conteúdo. Assim, o *designer* deve cuidadosamente escolher que *widgets* farão parte do modelo de interação da aplicação. Estes *widgets* devem ter características relevantes à mensagem sendo passada e devem motivar o interpretante desejado na mente dos usuários. Nós classificamos o conjunto existente de elementos de interfaces para grupo em três categorias: indicadores, acionadores e *applets*.

Os **indicadores** são elementos de interface que fornecem ao usuário informações relevantes sobre o contexto. Eles não oferecem meios de o usuário interagir com outros elementos do grupo, embora possam oferecer informações sobre eles. As únicas ações que o usuário pode executar sobre indicadores são relacionadas à apresentação, como trocar o estilo de representação da informação ou requisitar informação mais ou menos detalhada. Exemplos de indicadores são *widgets* como os apresentados por Ackerman e Starr (1996), que informam um usuário sobre a quantidade de atividade que está ocorrendo no ambiente compartilhado, ou os radares simples (*radar-views*) que mostram uma representação miniaturizada do ambiente compartilhado, indicando onde cada membro se encontra naquele momento (Gutwin et al., 1996). A Figura 4.7[1] mostra um exemplo de tela de um editor de texto para grupos e do lado direito desta tela um radar simples, que mostra em que parte do texto cada pessoa (diferenciada por cor) está trabalhando.

Os elementos de interface **acionadores** são aqueles que permitem ao usuário agir sobre a informação sendo apresentada e, comunicar-se com a aplicação e com os demais usuários. Exemplos deste tipo de elementos incluem botões, campos de texto (compartilhados ou não), *whiteboards*[2], e radares sofisticados, que não somente mostram a informação para o usuário, mas também permitem que ele atue sobre ela (por exemplo alterando a sua posição no ambiente, através do indicador dela no radar).

Finalmente, *applets* (Roseman e Greenberg, 1996a) são de fato aplicações de *groupware* com objetivos específicos, que podem ser acopladas ao sistema (sistemas *plug-in*). Um exemplo de *applet* seria um calendário que pode ser acoplado a qualquer outro *groupware*, como por exemplo a um sistema de suporte a reuniões. Um fator muito importante que deve ser considerado quando o *designer* se utiliza de um *applet* é que os *applets* já vêm com a sua própria interface. Isto significa que o projetista daquele *applet* já definiu a sua mensagem para o usuário, e a forma de expressá-la. Embora o objetivo de um *applet* seja justamente permitir que se incorpore uma aplicação a outra de maneira simples, do ponto de vista da interface, seu acoplamento não deve ser encarado como um processo trivial. Muito pelo contrário, ele pode

[1] A figura foi tirada do *site* do GroupLab (s.d.).

[2] Janelas que funcionam como um quadro-branco virtual onde cada membro tem uma "caneta" e todos podem escrever ao mesmo tempo (Roseman e Greenberg, 1996a; Santos, 1995).

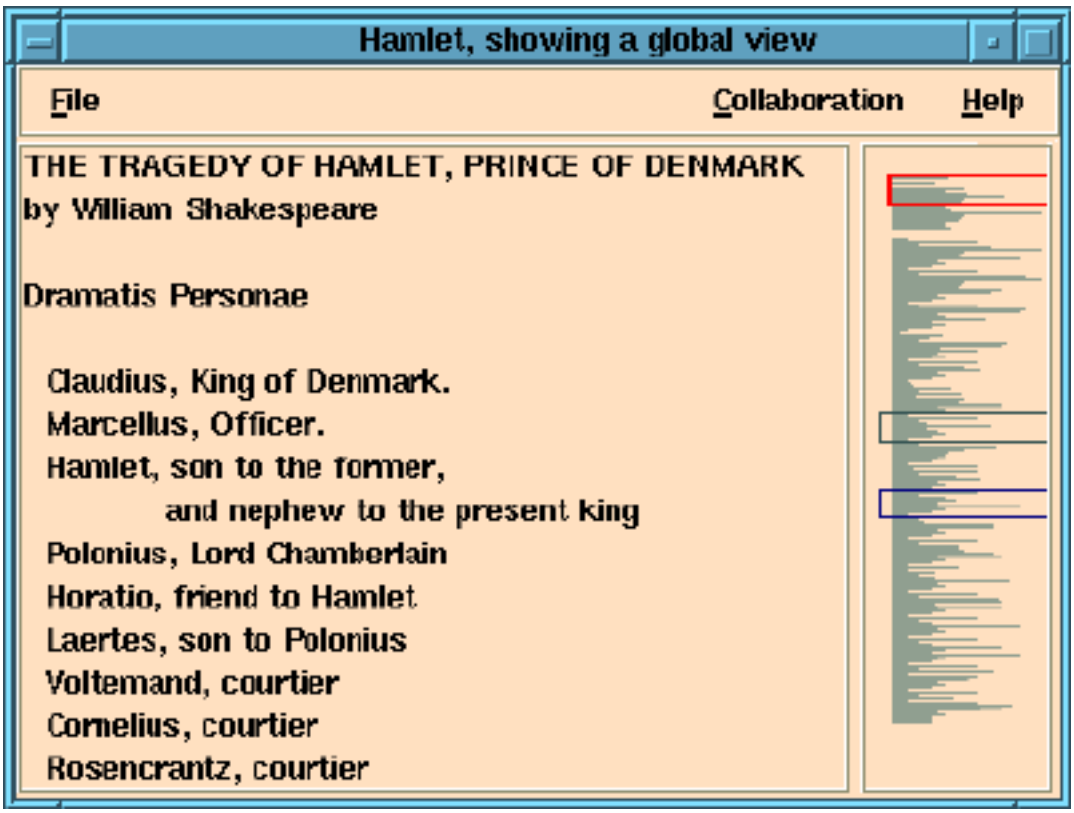

Figure 4.7 – Radar simples (*radar-view*) para textos.

ser o causador de quebras de comunicação *designer*-usuário. Isto acontece se o conteúdo desta mensagem projetada pelo *designer* do applet não estiver de acordo com a mensagem que o *designer* que vai acoplá-lo pretende passar, ou ainda se a escolha de significação para os seus elementos de interface, e conseqüentemente a expressão da mensagem do *applet*, não for consistente com as escolhas e com a expressão da aplicação à qual ele será acoplado. Vale notar que os riscos de que estamos falando aqui podem afetar não só a consistência da interação global, mas também a coesão do sistema, provocando rupturas e pontos negros de atuação em grupo.

Tempo *x* Espaço

Propondo um paradigma para o que denominaram "lingüística de programação", Gelernter e Jagannathan (1990) introduziram os conceitos de mapas de espaço e tempo[3] como parâmetros úteis para se comparar linguagens de programação. Mapas de espaço se referem a entidades que acontecem em diferentes posições de memória, mas no mesmo intervalo no tempo, enquanto que mapas de tempo se referem àqueles que acontecem em diferentes períodos de tempo, mas na mesma posição de memória. Embora o tempo e espaço a que eles se referem estivessem na memória do computador e na CPU, nós acreditamos que podemos aplicar estes conceitos ao *design* de interfaces multi-usuário. Quando o projetista de uma interface para grupos cria sua mensagem para os usuários, algumas das coisas que ele deve definir são por quantos períodos

[3]No original: *space-maps* e *time-maps*.

no tempo, ou eras, o grupo pode passar e quais são as transições válidas para cada membro ou subgrupo; ou seja, que partes do ambiente os membros ou subgrupos podem "freqüentar".

Para ilustrar como estes conceitos de intervalos de espaço e tempo se aplicam a *groupware*, pensemos no TeamRooms (Roseman e Greenberg, 1996a; TeamWave Software Ltd., sd). O TeamRooms é um sistema de grupos que permite a definição de salas de trabalho nas quais as pessoas podem atuar juntas. Cada uma destas salas é um sub-ambiente distinto. Se o *designer* tivesse definido que cada grupo tinha uma hierarquia e que membros do grupo só podiam entrar na sala do seu superior após completar sua tarefa, então ele teria definido dois intervalos no tempo. O primeiro é o tempo durante o qual o membro executa sua tarefa, e o segundo vigora após ele a completar.

Apesar de estarmos falando de tempo e espaço, vale salientar que os conceitos sendo apresentados aqui não são equivalentes à taxonomia de tempo *x* espaço apresentada no Capítulo 2. A taxonomia do Capítulo 2 classificava como os usuários de um grupo iriam usá-lo no tempo (síncrono ou assíncrono) e no espaço (local ou distribuído). Os intervalos de tempo e espaço apresentados aqui se referem aos estados pelos quais os usuários podem passar e os sub-ambientes que eles podem freqüentar "dentro" da virtualidade criada pela aplicação. Estas duas taxonomias são, porém, interdependentes. Por exemplo, se os membros do grupo têm que estar em um mesmo ambiente do sistema em um mesmo intervalo de tempo, isto implica que a aplicação deve ser síncrona. Mas, a relação de causa e conseqüência entre as duas taxonomias não é fixa. Em uma aplicação, o requisito de especificação pode definir que os membros do grupo devem usar o *groupware* sincronamente, isto implica que estes membros vão se encontrar nos ambientes compartilhados virtuais. Por outro lado, o requisito pode definir que os membros precisam executar uma tarefa juntos, o que vai implicar que pelo menos durante um período de tempo, a aplicação deve ser síncrona.

4.2 Temporalidade

As dimensões de caracterização apresentadas na seção 4.1 permitem ao *designer* fazer a descrição estática do grupo, ou seja, daquilo que não se altera com o tempo. No entanto, existem vários motivos pelos quais o projetista pode querer ou precisar definir mudanças no tempo. Por exemplo, a tarefa pode precisar ser feita em etapas e cada uma delas exigir que o grupo trabalhe de uma forma distinta, ou os usuários, à medida que usam a aplicação, podem desejar definir novas funções, modificar sua forma de trabalho, ou adaptar o sistema ao seu contexto específico.

Para exemplificar uma mudança no tempo, imaginemos um *software* educacional, que chamaremos de EduWare, no qual um professor trabalha com um grupo de alunos no aprendizado de alguns conceitos. A atividade de aprendizado é definida como tendo duas partes. Na primeira, os alunos devem estudar os conceitos e, na segunda, eles são testados para verificar o aprendizado. Na etapa de aprendizagem, os alunos trabalham em uma sala de aula virtual compartilhada. Nesta sala eles podem interagir uns com os outros, acessar o material disponível ou tirar dúvidas com o professor. Quando o professor acredita que os alunos já têm domínio dos conceitos, ele os interrompe e aplica um teste. O teste é feito individualmente pelos alunos e eles não têm acesso ao material, colegas ou professor. É fácil ver que de uma etapa para outra, o grupo passa por várias alterações. Os objetos que estavam disponíveis para o usuários deixam de estar e eles passam a agir sobre um novo objeto (o teste). As habilidades comunicativas dos

membros e o modelo de colaboração também são alterados.

Para que qualquer alteração ocorra, ela precisa ter sido prevista pelo *designer*. Durante o tempo de *design*, o projetista tem que definir, por exemplo, que modificações podem ocorrer, o que ou quem pode dispará-las e em que situações elas podem ocorrer. As mudanças definidas pelo projetista podem variar desde a configuração de alguns parâmetros da aplicação até a possibilidade de criar novas tarefas ou permitir a entrada de novos membros no grupo.

O modelo abstrato de meta-comunicação faz a distinção entre dois tipos de variações no tempo que podem ser definidas para uma aplicação: as mudanças e as meta-mudanças. As **mudanças** são aquelas nas quais o projetista especifica exatamente que transformações são aplicadas sobre que elementos do grupo, ou seja, o projetista define o "antes e o depois" da transformação. Ao definir as **meta-mudanças** o *designer* não define como o grupo muda, mas como os usuários podem mudá-lo. Neste caso, o *designer* prevê que mudanças pode ser necessário ou interessante fornecer ao usuário e disponibiliza para ele um **conjunto transformacional**, ou seja, um conjunto de primitivas de transformação que o usuário pode usar para definir mudanças no grupo. No EduWare, as mudanças da etapa de aprendizado para a etapa de teste foram definidas pelo *designer* e disparadas pelo professor ao aplicar um teste. Exemplos de meta-mudanças que o *designer* poderia fornecer ao usuário professor do EduWare seriam a inscrição de um novo aluno na turma, definição de novas atividades (como exercícios) durante a fase de aprendizagem, mudança do método de teste, entre outras.

Para definir mudanças, o *designer* deve decidir por quantas e quais etapas o grupo deve ou pode passar. Ele deve então fazer a descrição estática do grupo para a primeira etapa. Para as outras etapas, o *designer* pode ou descrever cada uma delas estaticamente, ou definir o que muda de uma etapa para a próxima.

Os objetivos do *designer* para definir meta-mudanças variam. O projetista pode querer que cada usuário do grupo seja capaz de configurar seu ambiente individual, ou a representação do seu ambiente compartilhado conforme suas preferências pessoais, pode querer que o grupo ou seus membros modifiquem a tarefa ou forma de trabalho, ou, até mesmo, que eles incluam novos participantes, criem novas tarefas ou comunidades, dentre outras coisas (Grudin, 1994b; Hamalainen et al., 1991). O AulaNet (s.d.) pode ser usado para ilustrar algumas modificações que o *designer* pode fornecer aos usuários. O AulaNet permite que uma pessoa ofereça um curso em WWW e que outras pessoas se inscrevam neste curso e freqüentem-no. Os papéis básicos do sistema são: professor, aluno, aluno co-autor e administrador. O administrador pode criar novos alunos e professores e o professor pode criar novos cursos[4] e pode transformar alunos em alunos co-autores de um curso. Além disso, ao criar um novo curso e aceitar alunos para este curso, o professor cria uma hierarquia, onde os alunos estão subordinados a ele.

As alterações feitas por um usuário em um sistema de grupo podem afetá-lo individualmente, caso sejam feitas no nível de interação individual. No entanto, se as modificações forem feitas nos níveis interpessoais ou públicos, elas podem afetar vários usuários. Assim, uma mudança em um sistema de grupo pode precisar ser definida em conjunto por todos os usuários envolvidos (Hamalainen et al., 1991). Os poderes de alteração que o *designer* oferece aos usuários têm que estar de acordo com estrutura do grupo. No AulaNet, a decisão de tornar o aluno um aluno co-autor cabe ao professor, embora afete o aluno[5]. Assim, o professor tem

[4]Cursos podem ser criados como privados ou compartilhados com outros professores (professores co-autores).

[5]Quando isto acontece, normalmente, professor e aluno já entraram em um acordo sobre a co-autoria. No entanto, este acordo é feito fora do AulaNet e não está representado no sistema.

autoridade sobre os alunos. Neste caso esta autoridade está de acordo com a estrutura do grupo, onde os alunos de um curso são subordinados a seu professor.

O nosso modelo abstrato de meta-comunicação classifica o conjunto de primitivas de transformação que o *designer* pode oferecer aos usuários de acordo com o poder de alteração que este conjunto os fornece. Assim, o conjunto transformacional pode ser classificado como: de configuração, aditivo e irrestrito. O conjunto transformacional de **configuração** permite apenas que os usuários alterem a forma de visualização e representação de elementos e informações do grupo. Assim, este conjunto atua sobre a expressão da mensagem do projetista. Quando o *designer* permite aos usuários acrescentar novos elementos e relações ao grupo, então ele lhes fornece um conjunto transformacional **aditivo**. Este conjunto de primitivas é monotônico, no sentido que ele não permite ao usuário destruir as definições do *designer*, mas apenas acrescentar novas definições, ou substituir as existentes por outras equivalentes. No entanto, se o projetista decidir fornecer aos usuários não apenas primitivas monotônicas, mas também aquelas que permitem a destruição do que existe, então o conjunto oferecido será **irrestrito**.

Quando o *designer* fornece aos usuários um conjunto transformacional aditivo ou irrestrito, isto pode implicar fornecer também um conjunto de configuração. Quando novas situações ou elementos são criados, o *designer* tem que permitir que eles sejam incluídos na interface da aplicação. Para isto, o *designer* tem que ou prever o que precisará ser incluído na interface e oferecer *templates* para os usuários, ou permitir que os usuários decidam como fazer esta inclusão[6].

4.3 Condições para as Regras Semânticas

As dimensões de descrição estáticas e dinâmicas compõem a estrutura sintática da linguagem de *design* do modelo abstrato de meta-comunicação. As regras heurísticas que atuam sobre elas formam o componente semântico desta linguagem. O objetivo de tal conjunto de regras é dar significado à descrição do grupo sendo feita pelo *designer* e lhe indicar as definições que possivelmente estão em conflito, ou que não são consistentes, sem, no entanto, restringir seu poder de expressão. Para que isto seja possível é necessário que as regras sejam separáveis de contexto e descritivas.

Regras **separáveis de contexto** são aquelas que são capazes de expressar inferências semânticas a partir da estrutura sintática, mas que não levam em conta o domínio. As unidades descritivas da LD são fundamentadas nas dimensões básicas apresentadas nas seções anteriores, e assim tanto elas quanto os valores que elas podem assumir podem ser descritos independentemente do domínio. As regras devem se basear nas possíveis combinações destes valores, e expressar inferências a partir deles, sem considerar o contexto do grupo.

Por não levarem em conta o contexto do grupo, as regras separáveis de contexto só são capazes de identificar **inconsistências em potencial**. Pode ser que uma situação que elas caracterizem como "sem sentido" em um domínio específico faça perfeito sentido. Assim, estas regras devem ser **descritivas** e não prescritivas. Isto significa que as regras não devem retratar julgamento de valor absoluto sobre as situações que identificam, e devem deixar a cargo do *designer* decidir se a inconsistência em potencial, no contexto em questão, é realmente uma inconsistência ou não. Quando o *designer* decidir que não se trata de uma inconsistência, ele

[6] Uma discussão mais profunda sobre este assunto é apresentada no Capítulo 7.

deve ser capaz de “passar por cima” dela, ou até mesmo desligar a regra que a identificou.

Os requisitos de que as regras sejam separáveis de contexto e descritivas garantem que a linguagem de *design* coloca o *designer* como responsável pela qualidade do seu projeto e não cerceia a sua criatividade.

* * *

O modelo abstrato de meta-comunicação estrutura o espaço de soluções de interfaces multi-usuário, e cria uma proposta de linguagem de *design* que as expresse. Com base neste modelo, podem ser desenvolvidos diferentes modelos de ambientes de apoio ao *designer* de interfaces multi-usuário. Assim, com base no modelo abstrato de meta-comunicação e também no método de *design* apresentado na seção 3.3, desenvolvemos o modelo de arquitetura de suporte ao *design* de interfaces multi-usuário, que será apresentado no próximo capítulo. No âmbito da arquitetura deste modelo, a proposta para uma linguagem de *design* do modelo abstrato de meta-comunicação adquire uma instanciação clara que torna mais fácil de apreciar a sua expressividade e a sua função.

Chapter 5

Modelo de Arquitetura de *Design* de Interfaces Multi-Usuário

O modelo abstrato de meta-comunicação propõe o desenvolvimento de linguagens de *design* que permitam ao *designer* descrever o seu modelo conceitual de um grupo. Como vimos no capítulo 3, o modelo conceitual do grupo faz parte do conteúdo da mensagem do projetista de uma aplicação de grupos para seus usuários. Assim, como o modelo de arquitetura de arquitetura ao *design* de interfaces multi-usuário (que chamaremos daqui por diante de modelo de arquitetura) dá suporte à especificação deste modelo conceitual de grupo do projetista, ele apóia o projetista durante a etapa de planejamento da interface multi-usuário.

O objetivo do modelo de arquitetura que estamos propondo, e que apresentamos neste capítulo, é permitir ao projetista especificar a parte do conteúdo da sua mensagem para os usuários referente ao seu modelo conceitual de grupo, fornecendo-lhe indicativos sobre a qualidade desta especificação. Baseado na descrição resultante, o modelo produz diretrizes e sugestões iniciais para a etapa de realização da interface. Para atingir este objetivo o modelo é composto de uma linguagem de *design*, uma base de conhecimento, um simulador de cenários e um conselheiro de *widgets*.

A linguagem de *design* (LD) do modelo de arquitetura segue as especificações de linguagem de *design* definidas no modelo abstrato de meta-comunicação. Assim, a sua parte léxica é composta por construtores que permitem ao projetista expressar as definições do seu modelo conceitual do grupo. Por sua vez, a parte semântica é formada por regras heurísticas separáveis de contexto e descritivas que atuam sobre a definição do grupo feita a partir destes construtores. Os significados atribuídos pelas regras semânticas à descrição do *designer*, juntamente com a explicação destas regras, permitem ao *designer* entender as implicações das suas decisões na mensagem sendo passada para o usuário e agir sobre elas.

A explicação das regras heurísticas compõem a base de conhecimento. À medida que o *designer* toma decisões, ele acrescenta a esta base de conhecimento as justificativas, motivações e intenções destas decisões. Assim, ao final da sua especificação o *designer* possui também a lógica desta especificação.

O modelo abstrato de meta-comunicação enfatizou os aspectos estáticos e dinâmicos de um grupo que devem ser expressos pelo *designer* na especificação de uma interface para este grupo. Assim, a LD instanciada no modelo de arquitetura permite tanto a descrição estática do grupo, quanto a dinâmica, abrangendo mudanças e meta-mudanças. As alterações descritas pelas meta-mudanças só acontecem em tempo de execução e, qualquer inconsistência em potencial

que elas porventura introduzam no modelo de grupo também só acontecem neste momento. Assim, para se informar sobre estas possíveis inconsistências, o projetista pode interagir com o componente cenógrafo para criar simulações sobre os estados que podem vir a ser definidos para o grupo em tempo de execução.

Enquanto a LD, a base de conhecimento e o cenógrafo dão suporte à fase de planejamento, o conselheiro de *widgets* já dá um passo na direção da realização da interface. Para isto, ele usa como entrada a especificação do projetista no modelo e gera uma série de diretrizes específicas para a escolha de *widgets* e decisões de interface.

A Figura 5.1 representa o modelo de arquitetura que estamos propondo e a sua interação com o *designer*. Neste capítulo, apresentamos em detalhe este modelo. Nas próximas seções, apresentamos cada um dos componentes do modelo. Em seguida, descrevemos o relatório que o *designer* recebe como resultado da sua especificação. Por fim, mostramos como este modelo se integra ao método de *design*.

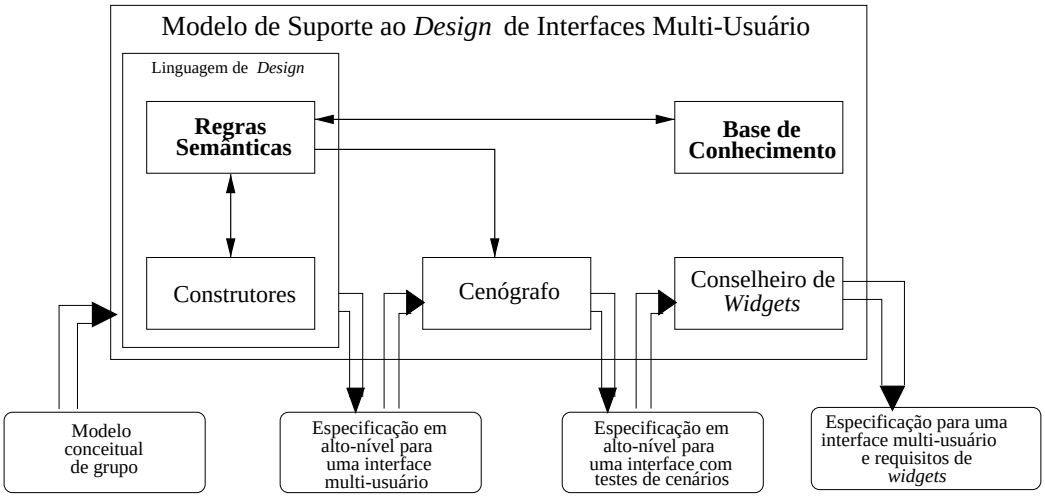

Figure 5.1 – Modelo de Arquitetura de Suporte ao *Design* de Interfaces Multi-Usuário.

5.1 Construtores

Os construtores são fundamentados nas dimensões básicas do modelo abstrato de meta-comunicação, apresentadas na seção 4.1. Cada construtor corresponde, ou a uma dimensão básica, ou à aplicação de restrições sobre a descrição intensional de uma dimensão básica. Na descrição do seu modelo conceitual do grupo, o *designer* usa estes construtores para definir quais são os membros deste grupo, os objetos sobre os quais eles podem agir, suas habilidades comunicativas sobre estes objetos, a hierarquia do grupo e os modelos de colaboração de suas comunidades. Estes construtores estão classificados em três categorias: os Tipos de Entidades (TE), os de Definição de Instâncias (DI) e os de Processos e Relacionamentos (P&R).

Tipos de Entidades

Para descrever um grupo, o *designer* deve definir quais são os membros deste grupo e os objetos sobre os quais eles podem atuar. Os membros podem assumir diferentes papéis dentro do grupo

e atuar sobre diferentes tipos de objetos. Assim, antes de definir os membros e os objetos, o projetista deve criar os tipos destes membros e objetos, ou seja, os papéis que os membros podem assumir e as classes de objetos que podem existir.

Para definir um papel o *designer* especifica o nome deste papel (por exemplo: professor) e a sua descrição. A descrição pode incluir as suas responsabilidades, o perfil do usuário exigido, dentre outras características que o projetista achar relevantes. Analogamente, para criar uma classe de objetos, o *designer* define o nome da classe e a sua descrição, que no caso significa o que os objetos daquela classe representam, o que pode ser feito com eles, etc.

Definição de Instâncias

Após definir os papéis e classes, o *designer* pode então criar as instâncias destes tipos, ou seja, os membros do grupo (por exemplo, definir um membro prof_ihc do tipo professor) e seus objetos. Cada membro do grupo pode ser definido como sendo um indivíduo, um subgrupo ou uma população. Um **indivíduo** significa que aquele membro representa um usuário final da aplicação. O **subgrupo** significa que um membro representa um número limitado de usuários finais, cujas definições são idênticas, ou seja, todos têm os mesmos objetos, o mesmo superior, etc. Finalmente, a **população** representa um conjunto ilimitado de usuários finais "idênticos". O objetivo desta distinção é permitir ao *designer* descrever grupos de usuários que têm a mesma definição de uma forma mais concisa, além de permitir a descrição de um grupo de usuários cujo o número de participantes só será definido em tempo de execução. Note-se que aqui diferenciamos **membro** de **usuário**. Por usuário nos referimos à pessoa que usará a aplicação de grupo, enquanto que por membro nos referimos à descrição do projetista deste usuário e de suas funções dentro do grupo.

Os objetos são relacionados à aplicação, e definem sobre o que os membros podem agir. Cada objeto pode ser privado, compartilhado ou público. O objeto **privado** pertence a um membro e apenas este membro pode agir sobre ele. O **público** pertence a todo o grupo, e qualquer membro pode agir sobre ele. Finalmente, o **compartilhado** pertence ou a um membro do tipo subgrupo ou população, o que significa que os usuários representados por eles compartilharão este objeto, ou a uma comunidade de membros, ou ainda a uma sub-hierarquia de membros. No caso do objeto compartilhado existe uma leve distinção entre agir sobre e possuir o objeto. Por exemplo, se um objeto é compartilhado por uma comunidade, a comunidade possui o objeto, mas quem age sobre ele não é a comunidade propriamente dita, mas sim os seus membros.

O *designer* pode criar dois tipos de objetos: o de tarefa e o de contexto. Os de **tarefa** devem ser usados para descrever os objetos que estão relacionados com a tarefa do membro ou do grupo. Os de **contexto**, por sua vez, devem ser usados para descrever a informação de contexto necessária para que os membros coordenem suas tarefas e colaborem. Para exemplificar, se o projetista define um objeto de tarefa compartilhado entre vários membros, é aconselhável que ele defina também um objeto de contexto compartilhado por estes mesmos membros que indique se o objeto de tarefa está livre ou em uso por outro membro.

Processos e Relacionamentos

Os construtores de P&R permitem ao *designer* descrever como os membros de um grupo se relacionam, ou seja, a estrutura hierárquica do grupo, suas comunidades, modelos de colaboração e habilidades comunicativas. A hierarquia do grupo permite que cada membro tenha um

ou mais subordinados, mas apenas um superior. Se um membro tivesse mais de um superior, os seus superiores poderiam entrar em conflito, e um acabar desautorizando o outro. Além disso, esta restrição se faz necessária para que possamos analisar as relações de autoridade entre os membros. Como no mundo físico uma pessoa pode assumir mais de uma tarefa, e fazer parte de diferentes subgrupos e hierarquias, nós criamos o construtor de sobreposição que permite ao *designer* definir que um mesmo usuário vai fazer o papel de dois ou mais membros. A definição da hierarquia do grupo não é compulsória, uma vez que nem todo grupo tem uma estrutura hierárquica. De toda forma, este caso pode ser visto como um caso particular da hieraraquia onde todos os membros estão no mesmo nível.

Os subordinados de um mesmo superior formam uma comunidade. Note-se que uma comunidade é então um construtor que deriva da dimensão básica de hierarquia, criando a restrição de que seus membros têm o mesmo superior. Para toda comunidade o *designer* tem que definir a interdependência das tarefas dos seus membros, ou seja, o modelo de colaboração da comunidade. As possíveis formas de colaboração são aquelas apresentadas na seção 4.1: modelo de ilha, encaixe rígido, encaixe nebuloso, sobreposição e único. Dentro de uma mesma comunidade a colaboração entre colegas pode diferir. Assim, o projetista pode definir subcomunidades dentro destas comunidades, e atribuir a cada uma delas diferentes modelos de colaboração. Atribuir formas de trabalho extremamente diferentes a um mesmo membro pode torná-lo propenso a agir de forma não apropriada em uma situação ou em outra. Assim, as relações entre os modelos de colaboração de um membro são verificadas por regras semânticas[1]. O *designer* pode ainda relaxar a restrição de os membros da comunidade terem o mesmo superior e definir comunidades livres. Para estas comunidades ele também tem que definir modelos de colaboração, mas as regras que envolvem verificação sobre as posições dos membros e suas relações não são checadas.

A comunicação entre os membros do grupo deve ser definida através dos construtores: ver, mostrar, falar e conversar. **Ver** permite que o *designer* defina quais objetos de outros membros um membro pode ver. **Mostrar** dá ao usuário o poder de decisão sobre se ele vai deixar outro membro ver ou não um determinado objeto. Quando o projetista define que um membro pode **falar** ao outro, isto significa que o outro é apenas ouvinte e não pode responder àquele que falou. Já **conversar** significa que ambos os membros podem falar e ouvir. Além disso, o *designer* pode definir o assunto de um discurso ou conversa como sendo ou um objeto específico, ou tema livre. Enquanto que falar ou conversar sobre um objeto restringe a interação dos membros ao trabalho, o assunto livre permite tanto que eles falem de trabalho, quanto que eles interajam socialmente.

As habilidades comunicativas não se aplicam apenas aos membros, mas também às comunidades. O projetista pode definir que uma comunidade tem a habilidade de ver, ouvir ou conversar. A comunidade ter uma habilidade significa que todos os seus membros herdam aquela habilidade enquanto fizerem parte da comunidade. Observe que uma comunidade não pode ser capaz de falar sobre ou mostrar um objeto. Isto porque falar e mostrar envolvem decisões dos membros sobre o que falar, quando mostrar, ou quem será o encarregado de falar ou mostrar, ou seja, quem será o seu porta-voz. Embora o nosso modelo não inclua este tipo de definição ele deixa o caminho aberto para este tipo de extensão[2].

Ao definir a interdependência entre as tarefas dos membros, o modelo de colaboração es-

[1] Ver regras 2, 3 e 4 no Apêndice A.

[2] Este ponto é discutido em mais detalhe no Capítulo 7.

pecifica requisitos para a interação entre estes membros. Acontece, porém, que esta interdependência das tarefas é determinada pelo compartilhamento de objetos de tarefa, enquanto que a interação entre os membros é determinada pelos objetos de contexto e habilidades comunicativas. Assim, para verificar se as definições do *designer* para o modelo de colaboração, objetos e habilidades comunicativas estão consistentes, e chamar sua atenção para aquelas que não estão, criamos regras semânticas que checam a relação entre estas dimensões[3].

Nesta seção, à medida que falávamos sobre construtores, introduzimos algumas das regras que fazem verificações semânticas sobre as definições do *designer*. Na próxima seção apresentamos estas regras em detalhes.

5.2 Regras Semânticas

As regras semânticas da linguagem de *design* apresentadas aqui preenchem os requisitos de serem separáveis de contexto e descritivas estabelecidos pelo modelo abstrato de meta-comunicação. Assim elas verificam definições especificadas pelo *designer*, suas combinações, ou variações no tempo que não fazem sentido, identificando inconsistências em potencial. Conforme apresentado na seção 4.3, uma inconsistência em potencial pode fazer sentido em um determinado domínio. Por exemplo, existe uma regra[4] que define que uma comunidade que tem o modelo de colaboração de ilha não deve compartilhar nenhum objeto de tarefa. No entanto, pode ser que o *designer* esteja criando um grupo onde os membros estão competindo para ver quem completa toda a tarefa em menos tempo. Assim, o *designer* pode definir o modelo de colaboração como sendo o de ilha, já que o trabalho deles é independente um do outro, e pode definir um objeto compartilhado que eles acessem ao fim de cada sub-tarefa com o objetivo de completar os dados desta sub-tarefa que terminaram. O objetivo deste objeto seria que os membros tivessem conhecimento do desempenho dos demais, estimulando a competição.

As regras heurísticas do modelo de arquitetura podem ser classificadas de acordo com dois eixos independentes. O primeiro as classifica de acordo com o tipo de regra que elas são, ou seja, o tipo de inconsistência que elas são capazes de apontar. Já o segundo, as classifica de acordo com as características do grupo que elas avaliam.

Classificação por Tipo

Quanto ao tipo, as inconsistências se classificam como: de pendências, primárias, de alerta, de confirmação ou de verificação. As regras **de pendências** identificam tipos ou instâncias de tipos que foram declaradas, mas que não foram utilizadas durante a especificação. Um exemplo de regra de pendência é: “Toda classe deve ter pelo menos uma instância.”[5].

As regras **primárias**, **de alerta** e de **confirmação** são aquelas que identificam situações que podem ser totalmente avaliadas pelo sistema. A diferença entre elas é o grau de certeza na afirmação de que a situação encontrada é realmente uma inconsistência. As primárias são aquelas que consideramos que identificam inconsistências. As de alerta são as que consideramos que na maior parte das vezes a situação identificada é uma inconsistência, mas que reconhecemos que em muitos domínios ela não o é. Finalmente, as de confirmação não necessariamente

[3] As regras são explicadas em detalhes na próxima seção (5.2).

[4] Ver regra 5 no Apêndice A.

[5] Ver regra 25 do Apêndice A.

são inconsistências, mas passam uma mensagem específica “curiosa” para os usuários. Neste caso, esta situação é identificada e o *designer* é informado da mensagem que ele está passando para o usuário. Para clarificar a diferença entre as regras primárias, de alerta e de confirmação comparemos as três regras abaixo:

- Regra 1: Para ser capaz de trabalhar em um grupo, o membro deve ou possuir objetos ou ter habilidades comunicativas.
- Regra 28: Para participar ativamente em um grupo, o membro deve possuir ao menos um objeto.
- Regra 35: Membros de uma comunidade que têm o modelo de colaboração de ilha só devem ter a habilidade de conversar se o *designer* pretende lhes dar a oportunidade de serem espontaneamente colaborativos.

A Regra 1 é uma regra de inconsistência primária, uma vez que se o membro não pode atuar sobre nenhum objeto, não pode ver nada, não pode falar, ouvir ou conversar com ninguém, então na verdade ele não consegue participar de forma alguma do grupo e não tem sentido ele fazer parte deste grupo. Já a Regra 28 verifica uma condição para o membro participar **ativamente** do grupo. No entanto, não é incomum os grupos terem um visitante ou supervisor, que não possui objetos, mas que pode ver objetos ou conversar com as pessoas. Assim, a Regra 28 se classifica como sendo de alerta. Por fim, a Regra 35 se classifica como sendo de confirmação, uma vez que ela simplesmente chama a atenção do *designer* para as suas intenções ao definir a situação identificada.

As regras de **verificação**, identificam situações em que uma das condições que as tornam inconsistências em potencial não pode ser avaliada pelo mecanismo de inferência das regras. Assim, o modelo informa o *designer* de qual deveria ser o valor daquela condição, para que a situação não seja uma inconsistência. Abaixo temos um exemplo de uma regra de verificação:

- Regra 39: Membros que compartilham um objeto e que têm habilidades comunicativas ativas (mostrar, falar ou conversar) sobre ele deveriam entrar em um acordo de quando e como usá-las.

O modelo consegue avaliar que os membros que compartilham um objeto têm habilidades comunicativas ativas sobre ele, mas ele não tem como saber se os membros são obrigados a entrar em um acordo sobre seu uso ou não. O objetivo destas regras é informar estas condições ao *designer* para que ele tome as providências para que a inconsistência não aconteça. No caso acima, baseado na mensagem recebida, o projetista poderia decidir criar um mecanismo que garantisse que todos os membros tivessem concordado com o uso a ser feito antes que qualquer um dos membros pudesse fazer uso de suas habilidades.

Classificação por Característica Avaliada

Esta classificação agrupa as regras sob algumas características relevantes. O objetivo deste agrupamento é tanto facilitar o entendimento do *designer* sobre os principais pontos sendo avaliados no grupo, quanto permitir que ele defina quais destes grupos de características são relevantes para o seu projeto. Esta classificação se faz nos eixos de generalidades, comunicação, colaboração, coordenação, conhecimento, hierarquia, autoridade e independência. Cada regra

pode pertencer a uma ou mais destas categorias[6]. Assim, se o *designer* está modelando um grupo que não tem hierarquia, ele pode configurar o conjunto de regras desligando o subconjunto delas que lida com hierarquia.

As principais categorias são a de comunicação, colaboração e generalidade. Toda regra está em uma ou mais destas categorias. Na verdade, as outras categorias são subconjuntos de uma destas categorias, ou uma combinação de subconjuntos delas. **Comunicação** engloba todas as regras que lidam com as habilidades comunicativas de um membro ou comunidade, enquanto que **colaboração** reúne aquelas relacionadas ao modelo de colaboração atribuído a uma comunidade. Por fim, as regras de **generalidades** são aquelas que tratam de características genéricas do grupo e não se encaixam em nenhuma das outras duas categorias, como por exemplo as regras do tipo pendente.

As regras de **coordenação** são aquelas que checam como os membros de uma comunidade coordenam seu trabalho. Como a quantidade de coordenação necessária entre os membros está diretamente relacionada com o tipo de colaboração entre eles, estas regras são um subconjunto das regras de colaboração. A próxima categoria é a de **conhecimento** e diz respeito às regras que verificam se o conhecimento que os membros têm dos objetos, é adequado ou não, é de qualidade, etc. Como o conhecimento é adquirido vendo, ouvindo ou conversando sobre os objetos, estas regras são um subconjunto das regras de comunicação. As regras de **hierarquia** verificam se as habilidades comunicativas dos membros estão de acordo com a sua posição na hierarquia e também são um subconjunto das regras de comunicação. As regras de **autoridade** são ainda mais particulares e relacionam as habilidades que os membros têm sobre os objetos, sua posição na hierarquia e sua relação com o(s) dono(s) do objeto. A maior parte destas regras faz parte também do conjunto de regras de hierarquia. Finalmente, as regras de **independência** são um subconjunto das regras de colaboração e garantem que os membros de uma comunidade, cuja colaboração é o modelo de ilha, são realmente independentes. Esta categoria se torna relevante porque muitas vezes, apesar de as tarefas dos membros não serem interdependentes, o *designer* deseja lhes oferecer a oportunidade de colaborar espontaneamente.

Para exemplificar, abaixo apresentamos algumas das regras e suas respectivas classificações por características avaliadas. Note-se como cada regra, exceto a de generalidade, pertence a mais de uma categoria.

- Regra 6: Comunidades cujo modelo de colaboração é o de ilha não devem conceder a seus membros a visão dos objetos uns dos outros.
  - Classificação: Colaboração, comunicação, e independência.
- Regra 13: Comunidades que têm o modelo de colaboração de sobreposição devem possuir objetos de contexto.
  - Classificação: Colaboração e coordenação.
- Regra 17: Um membro só deve poder ver objetos pertencentes a um descendente ou ascendente seu, se todos os membros entre eles na hierarquia também o podem.
  - Classificação: Hierarquia, comunicação e autoridade.

[6] A maior parte das regras pertence a mais de uma categoria.

- Regra 22: Para ser capaz de mostrar um objeto, o membro deve ter conhecimento direto prévio deste objeto.
  - Classificação: Comunicação e conhecimento.
- Regra 26: Todo objeto deve pertencer a alguém.
  - Classificação: Generalidade.

Representação Formal

Apesar de as habilidades comunicativas do nosso modelo serem restritas a ver, mostrar, falar, ouvir e conversar, algumas das regras verificam se os membros podem negociar, planejar ou informar. Estes conceitos envolvem mais do que as nossas unidades comunicativas básicas, porém a regra apenas checa se estes membros têm o requisito básico de comunicação. Por exemplo, o requisito básico para negociar ou planejar é ter a habilidade de conversar, enquanto que para informar pode ser mostrar ou falar. Ao atribuirmos significado às dimensões básicas e suas combinações, muitas vezes criamos estes tipos de abstrações.

Para que seja possível entender o que significam as abstrações que expressamos nas regras, é preciso entender a combinação de características sendo verificadas pelas regras. O texto da regra nem sempre explicita estas características, e para expressá-las com precisão e clareza necessitamos de uma representação formal. Dentre as possíveis representações formais que poderíamos utilizar, nos decidimos pelas DCG's (*Definite Clause Grammar* ou Gramática de Cláusulas Definidas).

As cláusulas das DCG's (Pereira e Warren, 1980) permitem que se descreva uma estrutura através de predicações. Os nossos construtores de descrição estática podem ser vistos como predicações que permitem a definição da estrutura de um grupo. Além disso, as DCG's (Pereira e Warren, 1980) são equivalentes às ATN's (Woods, 1970) (*Augmented Transition Network* ou Redes de Transição Ampliadas), e suportam a descrição da transição de estruturas no tempo, dando conta também dos nossos construtores de descrição dinâmica.

Os nossos construtores permitem a geração de estruturas de representação de grupo, enquanto que as regras de inferência controlam a formação de "textos" sobre estas estruturas. Ou seja, elas conferem as combinações definidas, avaliando o grau de aceitabilidade delas. Assim a nossa linguagem de *design* executa tanto a tarefa de geração de estruturas, quanto a de reconhecimento, ambas suportadas pelas DCG's. O uso de DCG's tem, portanto, o papel de homogeneizar as regras de especificação como um ambiente linguístico descritivo e como uma base de axiomas e regras de inferência que pode ser avaliada por linguagens de programação como PROLOG.

Para testar nossa linguagem de *design* representamo-la em DCG e criamos um protótipo de ambiente de apoio à expressão do *designer* em PROLOG. Apresentamos este protótipo na subseção 6.1. Abaixo mostramos a representação formal de uma regra e a sua descrição em linguagem natural. As cláusulas do lado direito são condições a serem provadas e estão também representadas. Apenas as provas mais básicas, como (`pode_ver, pode_agir, pode_falar, é_dono,` etc.) não estão representadas aqui, mas podem ser consultadas no dicionário de cláusulas na seção A.1 do Apêndice A.

Regra 33: Membros que podem ver um objeto ou podem agir sobre um objeto compartilhado e podem informar outros membros de ações e decisões sobre o objeto podem dar apenas a sua interpretação dos fatos.

- Representação formal da sua verificação:

```
Regra33(Participante1,Objeto,Participante2)→
    tem_conhecimento_direto(Participante1,Objeto),
    não_é_porta_voz(Participante1,Objeto),
    pode_informar(Participante1,Objeto,Participante2).

tem_conhecimento_direto(Participante,Objeto)→
    pode_ver(Participante,Objeto).

tem_conhecimento_direto(Participante,Objeto)→
    pode_agir(Participante,Objeto).

não_é_porta_voz(Participante,Objeto)→
    não(é_dono(Participante,Objeto)).

não_é_porta_voz(Participante,Objeto)→
    é_dono(Participante,Objeto),
    é_compartilhado(Objeto).

pode_informar(Informante,Objeto,Informado)→
    pode_falar(Informante,Objeto,Informado),
    não(é_dono(Informado,Objeto)).

pode_informar(Informante,Objeto,Informado)→
    pode_conversar(Informante,Objeto,Informado),
    não(é_dono(Informado,Objeto)).
```

A descrição completa das regras do modelo, contendo o seu enunciado em linguagem natural, sua explicação, suas classificações e representação formal são apresentadas no Apêndice A.

5.3 Descrição Dinâmica e Cenógrafo

Os construtores apresentados na seção 5.1 permitem que o *designer* faça a descrição estática do seu grupo. No entanto, o modelo de arquitetura permite ao *designer* expressar também mudanças e meta-mudanças. Nesta seção apresentamos como o *designer* faz descrições dinâmicas, e as regras que atuam sobre elas. Em seguida apresentamos o cenógrafo, que permite ao *designer* fazer simulação de inconsistências que podem vir a ser introduzidas pelas meta-mudanças.

Definição de Mudanças

Para definir mudanças, o *designer* deve primeiro descrever o grupo estaticamente. Em seguida, ele parte para definir novas eras ou situações e que transformações o grupo sofre ao entrar em cada uma delas. Tanto eras, quanto situações definem um novo período de tempo para o

sistema de grupo. A diferença entre elas é que uma **era** é uma mudança definitiva, a aplicação não volta mais para a era que estava em vigor quando esta iniciou. A **situação**, por sua vez, é uma mudança temporária. A situação acontece dentro de uma era e, após um evento ou o decorrer de um certo tempo, ela termina, ou retornando para a era corrente, ou dando início a uma nova situação.

No exemplo do EduWare (apresentado na seção 4.2, onde professor e alunos trabalhavam na aprendizagem e avaliação de novos conceitos), a etapa de aprendizagem é uma era, pois é a primeira (e não poderia voltar para nenhuma outra). Já a etapa de teste, é uma situação, uma vez que após completar o teste os alunos voltam para a etapa de aprendizagem. No entanto, se após o teste os alunos seguissem para uma nova etapa, por exemplo de aplicação dos conceitos aprendidos, e não retornassem mais às etapas já concluídas, então todas as etapas seriam eras.

A definição de uma nova era sempre tem como ponto de partida a definição da era anterior. Já o ponto de partida de uma situação pode ser tanto a era corrente, quanto uma outra situação. De todo jeito, a sua descrição inicial é a mesma da era ou situação da qual ela se origina. Em cima da definição inicial, o projetista define as mudanças usando construtores absolutos ou relativos.

Os **construtores absolutos** são aqueles que descrevem uma característica do grupo e foram definidos na seção 5.1. Já os **construtores relativos** são os que alteram uma característica existente. As alterações definidas pelos construtores relativos poderiam ser efetuadas com os construtores absolutos, mas envolveriam destruição de uma definição e criação de uma nova. Dentre as modificações que podem ser definidas através destes construtores estão a troca de papel de um membro, a troca de classe de um objeto, a troca do(s) dono(s) de um objeto, a troca da posição na hierarquia de um membro, etc.

Antes de iniciar uma nova era ou situação, o *designer* deve checar e resolver todas as inconsistências do período corrente. Por resolver queremos dizer solucioná-las através de mudanças nas suas definições, ou mantê-las, mas explicá-las. Se o *designer* cria uma nova era ou situação a partir de uma inconsistente, ele propaga as inconsistências para a era sendo criada. Uma vez propagadas, as inconsistências de tempos diferentes têm que ser resolvidas isoladamente (ou seja, independentemente da mesma inconsistência em outro tempo). À medida que o projetista altera a descrição de uma nova era ou situação ele tanto pode criar novas inconsistências em potencial, quanto resolver aquelas que porventura tenham sido propagadas. O modelo armazena as mudanças feitas pelo *designer* (por construtores relativos ou absolutos) e calcula que classes de regras devem ser verificadas para as mudanças feitas. A este conjunto ele adiciona todas as regras cujas inconsistências foram propagadas.

Definição de Meta-Mudanças

O nosso modelo de arquitetura permite que o projetista descreva o conjunto transformacional que ele pretende fornecer aos usuários. No entanto, como nosso modelo suporta apenas a etapa de planejamento da mensagem e permite ao *designer* a definição de somente parte do seu conteúdo, sua linguagem de *design* não oferece construtores para definição de primitivas de transformação de configuração. Ela oferece, porém, construtores para a definição de primitivas aditivas e irrestritas.

Nos construtores aditivos, o *designer* tem que definir o que pode ser criado ou substituído e por quem, e que restrições se aplicam à mudança definida. Por exemplo, para o projetista definir

que um usuário pode criar novos membros, ele tem que especificar quem (que membro(s) ou comunidade(s)) pode criá-los, membros de que papéis e de que tipos (indivíduo, subgrupo ou população) podem ser criados e que restrições se aplicam a esta criação. O projetista do EduWare poderia definir que novos membros no papel de aluno e do tipo indivíduo podem ser criados pelo membro professor, (apenas) durante a etapa de aprendizagem. Outras primitivas de transformação que poderiam ser definidas por um projetista usando os construtores incluem a criação de novos papéis, classes, membros, objetos, comunidades e relações hierárquicas e a substituição de modelos de colaboração vigentes e membros de uma sobreposição.

O conjunto transformacional irrestrito inclui todas as primitivas do conjunto aditivo, mais aquelas que permitem a destruição do que já está definido. A definição de primitivas para destruição de elementos ou relações existentes é análoga àquelas de criação. O projetista deve definir quem pode destruir o que e sob que condições.

Regras Semânticas Dinâmicas

A verificação da consistência das mudanças e meta-mudanças definidas pelo *designer* também é feita pelo conjunto de regras separáveis de contexto. Ao definir uma nova era ou situação, para checar sua consistência o projetista conta com basicamente as mesmas regras semânticas definidas para a descrição estática. Estas regras dão conta da nova definição do grupo para aquele tempo. A descrição estática de cada tempo pode ser consistente, mas a passagem de uma para outra gerar inconsistências. Desta forma, regras semânticas são necessárias para checar esta passagem. Um exemplo de uma regra semântica seria: “Quando um membro troca de superior (de um tempo para outro), então este novo superior deve ser ou o superior ao seu superior, ou então não deve estar diretamente relacionado a ele na hierarquia.”. O objetivo desta regra é identificar mudanças de hierarquia que gerem quebra da hierarquia, como seria o caso de um membro passar a ser o superior do membro que era seu superior no tempo anterior.

Quando o projetista define meta-mudanças é preciso verificar as relações entre os membros que podem executar mudanças e os membros afetados por elas. Um exemplo destas regras seria: “Um membro que pode alterar o modelo de colaboração de uma comunidade deve ser o seu superior.”. O objetivo desta regra é garantir que o único membro que tem autoridade para trocar a forma de trabalho de uma comunidade é o seu superior. No caso de conjuntos transformacionais irrestritos, atuam também regras que verificam a relação entre os membros que podem criar e destruir os mesmos elementos do grupo.

As regras heurísticas que verificam a consistência de mudanças e meta-mudanças formam uma nova categoria de regras classificadas por características avaliadas[7], à qual damos o nome de **dinâmicas**. As regras desta categoria, como as demais, pertencem também a outras categorias[8]. As regras dinâmicas checam as definições dinâmicas do *designer*, mas nenhuma delas verifica se o poder de modificação sendo concedido aos usuários lhes permite introduzir inconsistências ao grupo. A dificuldade de checar as inconsistências que podem ser criadas pelos usuários é que elas só acontecem em tempo de execução e é inviável testar todo o conjunto de possíveis estruturas que podem ser criadas, uma vez que este conjunto pode ser ilimitado. Na tentativa de maximizar o conhecimento do projetista sobre o poder que ele está fornecendo

[7] Esta classificação foi apresentada na seção 5.2.

[8] A regra de mudança apresentada é classificada como: dinâmica, hierarquia e autoridade; enquanto que a de meta-mudança é classificada como sendo: dinâmica, colaboração, hierarquia e autoridade.

aos usuários, o nosso modelo testa a consistência de cenários possíveis de serem criados pelos usuários. A estes cenários damos o nome de **cenários típicos** e ao componente que os cria de cenógrafo.

Cenógrafo

Para criar um cenário típico, o cenógrafo parte da descrição do grupo para a era ou situação em que o usuário pode definir alterações, que chamamos de **cenário concreto**. Tendo como ponto inicial o cenário concreto, o cenógrafo interage com o *designer* e simula a execução de mudanças que os usuários podem definir, tendo como resultado o cenário típico. Para o cenário típico são checadas as regras semânticas. Se alguma inconsistência é encontrada o *designer* é informado. Cabe a ele, então, a decisão de se é necessário alterar a primitiva de transformação fornecida ou, se no domínio da aplicação aquela inconsistência faz sentido. O *designer* pode interagir com o cenógrafo para criar tantos cenários típicos quanto ele ache necessário. Observe-se que se nenhuma inconsistência for encontrada em um cenário típico, isto não garante que elas não acontecerão em tempo de execução.

5.4 Base de Conhecimento

A base de conhecimento contém a lógica das regras, ou seja, a explicação do porquê das regras, e conseqüentemente, o de uma situação ser considerada uma inconsistência. Toda vez que uma inconsistência é identificada na definição do *designer*, ele é informado e recebe uma mensagem contendo a regra violada e a sua explicação. O objetivo desta base de conhecimento é fornecer ao *designer* informação suficiente, quando uma inconsistência em potencial é encontrada, para que ele possa decidir se no caso em questão se trata de uma inconsistência ou não. Se ele determinar que a situação não representa uma inconsistência, então ele é solicitado a oferecer uma explicação sobre a sua decisão.

Para exemplificar, voltemos ao exemplo dado na seção 5.2, no qual o *designer* define uma comunidade que tem modelo de colaboração de ilha e que compartilha um objeto de tarefa, e o modelo identifica uma inconsistência em potencial. O *designer* recebe então uma mensagem identificando a regra que foi violada, sua explicação e a instância que a violou, como mostrado abaixo.

Inconsistência potencial primária:

Regra 5: Comunidades que têm o modelo de colaboração de ilha, não devem possuir objetos de tarefa.

Explicação: Quando membros trabalham em um modelo de ilha, isto significa que o trabalho deles é completamente independente um do outro e que, conseqüentemente, eles não deveriam compartilhar nenhum objeto relacionado a seus trabalhos.

Instância que violou a regra: <*Nome da Comunidade, Objeto*>

Como o projetista está criando um ambiente competitivo e tem como objetivo acirrar a competição dos membros através daquele objeto, ele conclui que este caso específico não é uma inconsistência e fornece a seguinte explicação:

Explicação para instância: Os membros desta comunidade estão competindo pelo melhor tempo, e o objetivo deste objeto é incentivar esta competição.

Unindo as explicações do modelo para as regras e as do *designer* para as suas decisões que não estão de acordo com as regras, obtemos a lógica do modelo conceitual de grupo sendo especificado. Afinal, as regras explicam as situações que são consideradas consistentes, e as explicações do *designer* dão conta das que são consideradas inconsistentes. Assim, ao final da especificação, o *designer* obtém não apenas uma descrição do seu modelo, mas também uma explicação para ele, contendo as justificativas das suas decisões.

5.4.1 Conselheiro de *Widgets*

Este componente consiste em um conjunto de regras que relacionam a descrição do modelo do grupo do *designer* a características que devem estar presentes nos *widgets* a serem escolhidos para a interface final deste modelo. Este componente então serve como primeiro mapeamento da fase de planejamento para a fase de realização. Neste ponto do processo de *design* as definições estão ainda em alto-nível e uma escolha definitiva de *widgets* seria precipitada. Além do mais, o conjunto de elementos de interface que podem ser usados em uma interface multi-usuário é ainda um conjunto aberto, e não é possível sistematizar esta escolha, pelo menos por enquanto. Assim, o conselheiro de *widgets* gera uma série de requisitos para e sugestões sobre os *widgets* a serem escolhidos.

A entrada do conselheiro de *widgets* é a descrição final do grupo feita pelo *designer*. Ele dá suas sugestões e requisitos baseado na relação entre os modelos de colaboração, os objetos existentes, e as habilidades comunicativas dos membros sobre estes objetos. O *designer* pode escolher entre dois modos de uso para este componente: o modo pós-processador e o interativo. No modo pós-processador, o componente olha as definições do *designer* e lhe dá um conjunto de diretrizes para que fatores considerar e como eles deveriam influenciar suas decisões para cada tipo de situação definida. Quando o projetista escolhe o modo interativo, ele define fatores que não estão previamente definidos no modelo e que influenciam as sugestões do conselheiro e as decisões do próprio *designer* sobre os *widgets*. Assim, ao invés de o conselheiro dar todo o conjunto de diretrizes possíveis para uma situação, ele só dá as relevantes.

Estes fatores que influenciam as escolhas dos *widgets* e que não estão expressos no modelo incluem indicação de se a aplicação será usada sincronamente ou não, se os membros de um modelo único terão sempre o mesmo foco ou não, se nos modelos únicos e nas partes compartilhadas do modelo de sobreposição todos podem agir ao mesmo tempo, ou apenas um de cada vez, entre outros (Ellis et al., 1991; Santos, 1995). Assim, se o *designer* já sabe definir todos estes fatores, ele pode interagir com o conselheiro e obter um conjunto de requisitos mais específicos e a sua descrição destes fatores. Por outro lado, se muitos destes fatores ainda não estão definidos, o *designer* pode consultar o modo pós-processador e obter as diretrizes gerais configuradas para o seu modelo.

Para exemplificar mostramos as sugestões dos modos interativo e pós-processador do conselheiro para um mesmo modelo de entrada. Neste grupo, a comunidade *X* tem o modelo de colaboração único. No modo interativo o *designer* tem que definir:

- Se todos os membros desta comunidade têm sempre o mesmo foco.

- Se têm o mesmo foco, se todos podem interagir com o sistema em paralelo ou apenas um de cada vez (ou seja, se tem *floor-control*).
- Se existe prioridade entre os membros para acessar cada um dos objetos compartilhados.

Suponhamos que o *designer* então defina:

- O foco pode não ser o mesmo.
- Não existe prioridade de acesso aos objetos.

O conselheiro então lhe daria as seguintes sugestões:

Como o modelo da comunidade *X* é único, os membros devem ter um ambiente compartilhado entre eles. Para permitir focos em diferentes partes este ambiente deve ser WYSIWIS-relaxado, mas com *feedthrough* imediato para membros com mesmo foco. Um dos objetos de contexto definidos deve ser um indicador dos focos dos membros. Um *widget* que exerce esta função por exemplo é o *radar-view*. Cada membro deve ter seu próprio cursor, discriminado dos cursores dos demais. Para isso podem ser usados *telepointers* semânticos[9]. Os objetos *O1, O2 e O3* compartilhados entre os membros devem ter indicador de uso associados. O Membro *M* possui o objeto privado *OP*. O acesso a este objeto deve ser feito fora do ambiente compartilhado, já que os outros membros não têm um objeto equivalente.

...

No caso, do modo pós-processador as sugestões seriam:

Como o modelo da comunidade *X* é único, as seguintes diretrizes devem ser levadas em consideração:

- Se os membros têm sempre o mesmo foco, o ambiente compartilhado deve ser WYSIWIS-rígido, senão ele deve ser WYSIWIS-relaxado.
- Se o foco for o mesmo e se apenas um membro pode interagir com o sistema por vez (ou seja, tem *floor-control*), então deve existir um mecanismo de controle e se necessário um mecanismo para pedir a "palavra".
- Se o foco puder ser diferente, ou se o foco for o mesmo, mas os membros puderem interagir com o sistema em paralelo, então, cada membro deve ter seu próprio cursor discriminado dos demais. Para isso podem ser usados *telepointers* semânticos. Nestes casos, o *feedthrough* deve ser imediato para membros com mesmo foco.
- Se o foco puder ser diferente, é necessário que um dos objetos de contexto definidos seja um indicador do foco dos membros. Um *widget* que exerce esta função por exemplo é o *radar-view*.
- Os objetos *O1, O2 e O3* compartilhados entre os membros devem ter indicador de uso associados, além disso se alguns membros têm prioridade sobre outros, o controle de acesso deve levar em conta esta prioridade.
- O Membro *M* possui o objeto privado *OP*. O acesso a este objeto deve ser feito fora do ambiente compartilhado, já que os outros membros não têm um objeto equivalente.

[9] *Telepointers* é o nome dado aos cursores dos membros de um *groupware*. Os *telepointers* semânticos são aqueles que têm alguma informação associada, como por exemplo, a que participante eles pertencem ou a tarefa sendo executada por eles (Greenberg et al., 1996b).

- …

Observe-se que nas sugestões dadas pelo conselheiro em ambos os modos, ele engloba tanto as definições do modelo de grupo, quanto os novos fatores. No interativo, ele já dá uma resposta específica para a situação, enquanto que no modo de pós-processamento ele dá todo o conjunto de diretrizes. Na verdade, este conjunto de diretrizes é a lógica do conselheiro para dar as respostas específicas do modo interativo.

O conselheiro de *widgets* não é capaz de dar muitas sugestões sobre como representar mudanças e meta-mudanças na interface. Desta forma, ele se restringe a explicitar as variações possíveis no tempo e dar sugestões bem alto nível. No caso de mudanças, ele informa ao *designer* que tem que ficar claro para os usuários o que mudou e o que causou esta mudança. Quando os usuários podem fazer alterações, o conselheiro de *widgets* avisa ao projetista que é preciso deixar claro para os usuários as alterações que cada um pode realizar, em que circunstâncias eles podem realizá-las e as suas conseqüências.

5.5 Saída do Modelo

Ao final da especificação do seu modelo conceitual do grupo, o *designer* obtém um relatório[10]. O relatório é separado por eras e situações. Para cada um destes intervalos de tempo é descrita a causa do seu início (a menos da primeira era), e no caso de situações também do seu término. Em seguida, são descritas as mudanças feitas para a era ou situação corrente em relação a anterior. Finalmente, é apresentada toda a descrição daquele intervalo de tempo, feita pelo *designer*.

A descrição do modelo conceitual do grupo para um intervalo de tempo, mostra os valores atribuídos aos construtores e as relações entre eles. Assim, para cada membro por exemplo, o relatório contém o papel do membro, seu tipo (indivíduo, subgrupo ou população), seu superior, seus subordinados, as comunidades a que pertence, os objetos sobre os quais pode agir e as suas habilidades comunicativas. Além disso, todas as inconsistências encontradas para aquele intervalo são listadas, juntamente com a regra que violaram, a explicação da regra e a explicação do projetista para sua decisão de manter tal inconsistência, caso ele tenha fornecido uma.

Caso o projetista tenha pedido ao conselheiro de widgets um parecer sobre a realização da interface multi-usuário descrita, este também é anexado ao relatório. Assim, o relatório contém a especificação em alto-nível de parte do conteúdo da mensagem que o *designer* pretende passar para os usuários do seu sistema de grupo, as justificativas, intenções e motivações do *designer* para as suas decisões, e as diretrizes para a criação da expressão desta mensagem.

5.6 Integração do Modelo de Arquitetura ao Método de *Design* para Usabilidade

Em um processo de *design top-down*, o projetista inicia o seu *design* a partir da avaliação dos requisitos do domínio e inicia a etapa de planejamento gerando seu modelo conceitual de usabilidade potencial da aplicação de grupo, que inclui o seu modelo conceitual do grupo. Feito

[10]Um exemplo do relatório gerado pelo protótipo implementado do modelo de arquitetura é apresentado no Apêndice B.

isto, o projetista parte para a especificação deste modelo conceitual. Esta especificação, normalmente, é feita em vários passos, nos quais cada passo é um refinamento do passo anterior. A etapa de planejamento termina no momento em que o modelo de usabilidade potencial do *designer*, ou seja, o conteúdo da sua mensagem para os usuários, está completamente especificado.

Inicia-se então a fase de realização da mensagem, na qual o projetista define a sua expressão. A expressão da mensagem é definida escolhendo-se os elementos de interface que a comporão e então construindo-a. Esta etapa também pode passar por refinamentos sucessivos, onde os passos deste refinamento são testes de usabilidade junto aos usuários.

Nosso modelo de arquitetura se encontra na fase de planejamento e auxilia o *designer* na fase de especificação do seu modelo conceitual do grupo, particularmente, na especificação da estrutura do grupo e dos seus modelos de comunicação e colaboração. Além de apoiar esta especificação o modelo fornece ao *designer* indicativos de qualidade da sua descrição de grupo, através da identificação de inconsistências potenciais e a explicação das regras que as identificaram. Finalmente, o conselheiro de *widgets* dá o primeiro passo na direção da fase de realização ao gerar diretrizes para decisões de interface e sugestões para seus *widgets*. A Figura 5.2 mostra a integração do modelo de arquitetura ao método de *design*para usabilidade.

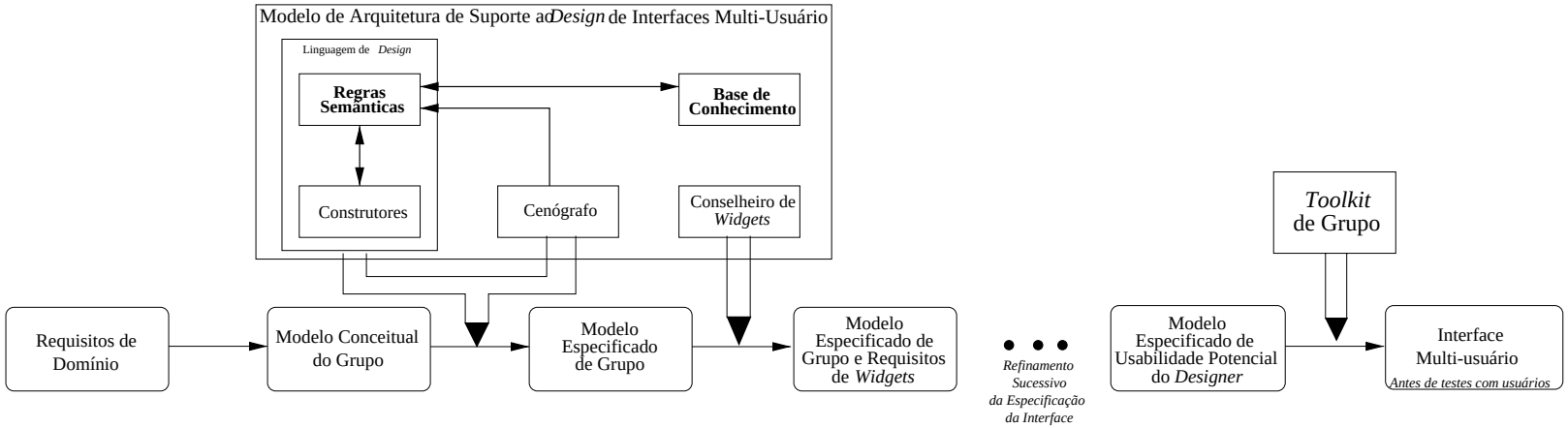

Figure 5.2 – Integração do modelo de arquitetura ao método de *design* para usabilidade.

Chapter 6

Validação

O modelo de arquitetura apresentado no capítulo 5 cria a base para o desenvolvimento de ferramentas que dêem suporte ao projetista durante o fase de planejamento da interface multi-usuário, mais especificamente do seu modelo conceitual do grupo. Para mostrar que é possível desenvolver uma ferramenta que implemente o modelo proposto, desenvolvemos um protótipo. Usando este protótipo fizemos testes de modelagem de grupos que nos permitiram obter uma avaliação preliminar dos componentes propostos no modelo de arquitetura.

Como o modelo de arquitetura foi derivado do modelo abstrato de meta-comunicação e do método de design para usabilidade, estes testes fornecem também uma avaliação destes dois últimos. Assim, a análise dos construtores e regras semânticas da LD do modelo de arquitetura é também uma apreciação das dimensões descritivas e condições de criação de regras semânticas do modelo abstrato de meta-comunicação.

Neste capítulo apresentamos o protótipo implementado e os testes feitos usando-no. O primeiro teste consiste da modelagem do *site Intranet* do SERG. A modelagem foi feita a partir "do zero", uma vez que não existia ainda uma modelagem da interação do grupo. O segundo teste feito foi a remodelagem do modelo conceitual de grupo do sistema Qualitas, para o qual já tinha sido feita uma modelagem usando técnicas típicas e genéricas[1] de Engenharia de *Software*.

6.1 Protótipo

Para implementar o protótipo escolhemos a linguagem PROLOG, uma vez que as regras estavam formalizadas em DCG's e poderiam ser testadas diretamente nesta linguagem. O protótipo pode ser visto como uma instância do modelo de arquitetura proposto. Para que pudéssemos construi-lo, primeiramente especificamos os componentes deste modelo, e então partimos para sua implementação. Destes componentes, o protótipo tem implementado os construtores estáticos e de mudanças, as regras de inferência e a base de conhecimento. Os construtores de meta-mudança e as regras semânticas que atuam sobre eles estão especificados em detalhe, embora não tenham sido implementados. O cenógrafo e conselheiro de *widgets*, por sua vez, estão especificados em mais alto nível e, por conseguinte, ainda mais distantes de uma implementação.

Além do intuito de mostrar que é possıvel se implementar uma ferramenta com base no modelo de arquitetura, os nossos principais objetivos ao implementar este protótipo e testá-lo

[1] Por genéricas nos referimos às técnicas que não são específicas para modelagem de sistemas multi-usuário.

eram:

- Mostrar que é possível descrever um modelo conceitual de grupo na linguagem de *design* proposta pelo modelo de arquitetura.
- Avaliar a organização proposta do espaço de *design*, ou seja, dimensões de descrição do modelo abstrato de meta-comunicação e os construtores do modelo de arquitetura. O objetivo aqui era verificar se o *designer* seria capaz de descrever todos os aspectos desejados do grupo usando os construtores oferecidos.
- Verificar se a linguagem de *design* traz benefícios para o *designer*, ou seja, se o faz considerar dimensões e relações entre elas que de outra forma passariam despercebidas.
- Investigar se algum dos componentes do modelo de arquitetura não era essencial ao fornecimento de indicadores qualitativos sobre a descrição de um grupo.

Para utilizar o protótipo, o *designer* cria um novo modelo conceitual de grupo e o define, através dos construtores do modelo de arquitetura. A linguagem de *design* está expressa através de uma linguagem de comandos[2], onde os comandos permitem ao designer descrever o grupo através da atribuição de valores aos construtores e requisitar a verificação da sua descrição. Além disso, existem comandos que mostram para o *designer* as características já definidas por ele e as relações entre elas. A base de conhecimento atua quando o *designer* requisita ao sistema uma verificação da sua descrição e inconsistências em potencial são identificadas. O sistema então fornece ao projetista a explicação sobre porque a situação identificada é considerada uma inconsistência. Se o projetista então define que no domínio sendo modelado a situação em questão faz sentido, ele pode neste momento entrar sua explicação para ela na base de conhecimento.

Ao final da sua especificação o protótipo fornece ao *designer* um relatório em linguagem natural, contendo a sua especificação. O relatório mostra a descrição do grupo, as inconsistências encontradas pelas regras e as explicações, tanto para a regra quanto para a inconsistência, de cada uma das eras especificadas. Além disso, de uma era para outra ele descreve as mudanças feitas usando os construtores relativos.

6.2 Teste do SERG

O teste feito usando o protótipo tinha como objetivo modelar o *site* da *Intranet* do SERG. Este *site* foi inicialmente criado com o objetivo de facilitar a interação e cooperação entre os membros do grupo de pesquisa da PUC-Rio em Engenharia Semiótica. No entanto, esta o *site* não foi formalmente modelado, e à medida que qualquer um dos membros sentia necessidade de disponibilizar ou organizar algum tipo de informação ele mesmo o fazia no seu ambiente privado, que podia ser acessado por todos. Assim, os membros não tinham uma mesma visão do grupo e das formas de colaboração entre si. Com isto, muitas vezes a contribuição de um membro passava despercebida por outros, uma vez que ela não se encaixava no tipo de contribuição esperada por eles. Outras vezes um membro não conseguia encontrar a contribuição de um colega por não saber onde procurar ou como ela tinha sido enquadrada pelo primeiro.

[2]O Manual do Usuário contendo os comandos, suas definições e forma de uso, é apresentado no Apêndice C.

Dados os problemas causados pela participação "aleatória" dos membros do SERG, o objetivo da *designer* ao modelar o *site* da *Intranet* do SERG era buscar um sistema que permitisse aos membros ter uma visão consistente sobre ele e assim interagir e colaborar de forma eficiente. Em um segundo momento, a *designer* gostaria de expandir o *site* para permitir a participação de pesquisadores colaboradores externos à PUC. Assim sendo, a projetista decidiu modelar duas eras: a primeira correspondendo à descrição da colaboração dos membros da PUC, e a segunda contendo as modificações necessárias para incluir os membros externos à PUC.

A *designer* responsável por testar o protótipo com o teste do SERG é membro do mesmo e contou com o apoio da autora. Esta pessoa é especialista em interfaces e em Engenharia Semiótica, embora não em interfaces multi-usuário. A descrição do modelo conceitual do grupo gerada é apresentada no Apêndice B.

O objetivo deste teste como validação dos modelos propostos era avaliar tanto o protótipo quanto os modelos dos quais ele foi derivado. Avaliar o protótipo significa avaliar esta implementação ou instância do modelo de arquitetura. Em relação aos modelos, buscávamos verificar se a *designer* seria capaz de expressar seu modelo conceitual de grupo usando a LD, se alguma dimensão da LD nunca seria usada, se existiam características do grupo que não poderiam ser expressas com os construtores ou dimensões disponíveis e se o conjunto de regras semânticas proposto seria relevante.

6.2.1 Resultados

Apresentamos como resultados deste teste as observações da *designer* (participante do teste) e da autora. Elas se referem tanto ao protótipo, quanto ao modelo de arquitetura. Estas observações dão indícios sobre o uso, qualidade e acertos necessários no modelo, e sobre características a serem consideradas na implementação de um produto baseado neste modelo.

Referentes ao Protótipo

Após o teste, a visão geral da participante do teste foi de que a utilização do protótipo para a especificação foi útil, uma vez que permitiu a organização do seu modelo conceitual do grupo, armazenou suas decisões de *design*[3], facilitando que decisões consistentes fossem tomadas em situações similares e, finalmente, identificou algumas inconsistências de especificação e do seu modelo conceitual do grupo que teriam passado despercebidas. Durante o teste, vários pontos passíveis de alteração foram identificados. Alguns destes pontos são requisitos para o desenvolvimento de um produto que permita um uso eficiente do modelo implementado, enquanto outros são sugestões de formas de aproximar a linguagem de *design* dos objetivos de especificação do *designer*.

O primeiro e mais importante ponto levantado foi em relação à interface do protótipo. A interface de linguagem de comando não deixa à mostra os valores atribuídos às dimensões e as relações estabelecidas. Assim, ela requer grande esforço do *designer* para lembrar tudo o que já foi definido e como. Os comandos que permitiam que se listasse o que estava definido melhoravam esta limitação, mas ainda assim era solicitado grande esforço do *designer*. Por

[3] Aqui a participante não se refere à base de conhecimento, mas às suas decisões de como modelar determinadas situações.

exemplo, quando o *designer* vai definir os membros do seu grupo, o ideal é que ele possa consultar os papéis definidos. Embora as relações entre membros não sejam espaciais, nos parece que uma interface gráfica seria a melhor forma de apresentar as informações ao *designer* (de Souza et al., 1997).

Durante a especificação do seu modelo conceitual, a participante identificou que em alguns momentos para atingir a especificação desejada, ela deveria dividi-la em um grande número de passos. Assim, a participante sugeriu alguns acréscimos à linguagem que têm como objetivo aproximar a linguagem de *design* dos objetivos de expressão do *designer*. Os acréscimos sugeridos foram:

- Herança de objetos: Seria interessante que o *designer* pudesse, além de associar objetos a membros, associar descrição de objetos a papéis. Assim, quando um membro deste papel fosse declarado, ele automaticamente receberia um objeto conforme o descrito para a classe.

- Associação de objetos de contexto a objetos de tarefa: Algumas classes de objetos sempre requerem que seus objetos de tarefa tenham associados a eles objetos de contexto. Assim, seria interessante poder definir isto para a classe de objetos, de forma que toda vez que um objeto de tarefa fosse instanciado para aquela classe, um de contexto também o fosse.

- Sobreposição de papéis: Em alguns grupos, a sobreposição de certos membros acontece com freqüência. Assim, seria interessante poder definir um papel que fosse na verdade a sobreposição de outros dois, e quando uma instância deste papel fosse definida, seriam criados os membros dos papéis sobrepostos e a sua sobreposição.

- Habilidades comunicativas gerais: Algumas habilidades comunicativas podem envolver vários membros ou comunidades. Assim, seria interessante que se pudesse definir uma habilidade sobre um objeto como pública ou compartilhada.

- Regras de verificação: Como as regras de verificação identificam situações que o modelo não consegue provar como inconsistência, elas podem gerar "mensagens de inconsistência" para situações consistentes. No teste feito, uma delas acusou um grande número de situações que não eram inconsistências. Isto sugeriu, que estas regras deveriam ser tratadas pela interface de forma diferente das demais. Por exemplo, independente do número de situações a serem verificadas que ela encontrasse, ela só daria uma mensagem para o *designer*, e este poderia decidir se gostaria de ver as situações identificadas ou não.

Note-se que as observações feitas nesta subseção se referem apenas à implementação do modelo de arquitetura, e não ao modelo propriamente dito. Estas observações são importantes de serem documentadas, uma vez que permitem que sejam consideradas em futuras implementações do modelo.

Referentes ao Modelo de Arquitetura

Alguns dos comentários e sugestões apresentados pela participante não se referiam apenas à implementação, mas também ao modelo de arquitetura por trás dela. Um ponto de nosso interesse neste teste era observar a adequação do conjunto de regras que estamos propondo. No teste feito, conforme o esperado, a verificação de cada regra teve um dos três resultados:

- A especificação “obedecia” à regra e nenhuma inconsistência foi encontrada.
- A inconsistência foi encontrada e era relevante.
- A inconsistência foi encontrada, mas não era relevante.

A relevância das inconsistências encontradas foram julgadas pela participante com base nos seus objetivos de modelagem, que eram claramente dependentes do domínio, e na explicação das regras oferecidas pelo protótipo. As inconsistências relevantes foram classificadas como tendo duas possíveis causas: (1) a participante esqueceu de alguma definição embora a tivesse previsto e (2) a situação inconsistente tinha sido gerada por “efeitos colaterais” de descrição e não tinha sido prevista pela *designer*. Em ambos os casos, inconsistências relevantes causaram modificações da descrição pela *designer*. As inconsistências relevantes alcançaram o nosso objetivo ao propô-las, e oferecem um indício das vantagens que um sistema de apoio baseado no modelo de arquitetura podem oferecer a projetistas de interfaces multi-usuário.

Durante o teste, em algumas instâncias de relação entre superior e subordinado, a participante manifestou a intenção de criar relações entre eles equivalente às que ele podia criar para comunidades. Isto não aconteceu para todas as relações entre superior-subordinado e nem se aplicava ao superior e todos seus subordinados. Assim, estas observações sugerem que um construtor equivalente ao de subcomunidades deveria estar disponível para relações entre superior e subordinado e talvez até entre superior e uma determinada linha de descendentes. Mais testes devem ser realizados para apoiar este indício. De todo jeito, vale a pena notar que mesmo que se chegue à conclusão que tal construtor deva ser acrescentado ao modelo de arquitetura, este acréscimo não afetaria o modelo abstrato de meta-comunicação. Afinal, este novo construtor poderia ser definido através de restrições às suas dimensões básicas, mantendo assim a continuidade semiótica entre o modelo de arquitetura e o de meta-comunicação.

6.3 Teste do Qualitas

O sistema Qualitas tem como objetivo permitir aos funcionários de uma empresa manter a consistência e qualidade (conforme normas ISO) da avaliação de requisições de clientes e da criação e cumprimento de contratos que atendem a estas requisições. Diferentemente do *site* da *Intranet* do SERG, o sistema Qualitas já possuía uma modelagem formal. Esta modelagem tinha sido feita usando métodos típicos de Engenharia de *Software* baseados em conceitos de orientação a objetos. Contudo, nenhum dos métodos utilizados deixava claro para o projetista como uma decisão sobre um aspecto do modelo afetava (ou não) os demais. Tampouco estes métodos forneciam ao *designer* indicativos de qualidade sobre o modelo sendo criado. Assim, o objetivo deste teste, tanto para a equipe do Qualitas quanto para a validação dos nossos modelos, era verificar se a modelagem do grupo do Qualitas no nosso protótipo traria algum benefício para a qualidade do modelo do Qualitas, além daqueles já obtidos com a modelagem existente.

A projetista responsável pelo teste do Qualitas[4] era uma das projetistas da equipe e, assim como a projetista do *site* da *Intranet* do SERG, contou com o apoio da autora. Esta projetista é

[4]A descrição do modelo conceitual do grupo gerada não é mostrada neste trabalho por não ser de domínio público.

especialista em Engenharia Semiótica e interfaces, e além disto tem experiência como *designer* de sistemas e interfaces multi-usuário em ambientes WWW.

6.3.1 Resultados

No Qualitas membros têm a capacidade de criar objetos em tempo de execução, e de acordo com o objeto criado podem interagir de diferentes formas. Estes dois requisitos já caracterizavam um empecilho para a modelagem do Qualitas no protótipo, uma vez que este não tinha implementado os construtores de meta-mudanças do modelo de arquitetura. Por outro lado, eles mostram que os construtores que permitem apenas a descrição estática de um grupo não são suficientes e que, conforme proposto pelo modelo de arquitetura, eles devem existir também aqueles que dão conta da descrição dinâmica.

Decidiu-se então continuar o teste apesar deste obstáculo. Para isto, escolheu-se um estado possível e provável do sistema para se modelar estaticamente. Note-se que a solução adotada corresponde ao papel do cenógrafo no modelo de arquitetura. Assim, a avaliação do protótipo para este estado específico nos permite apreciar como o cenógrafo pode contribuir para a qualidade de um modelo conceitual de grupo sendo definido em uma ferramenta que implementasse todo o modelo de arquitetura.

A projetista do Qualitas conseguiu definir seu modelo conceitual de grupo na linguagem de *design* disponível. Em alguns momentos do *design*, a projetista pensava em termos de tarefas dos membros do grupo, mas não teve problemas em traduzi-las em termos de objetos, até mesmo porque já o tinha feito para a modelagem original do Qualitas. No momento de definir os modelos de colaboração entre os membros do grupo, a participante observou que não os tinha claro e teve que refletir e consultar a modelagem original para defini-los. Ela percebeu que isto se deu porque na modelagem original do Qualitas o modelo de colaboração não era explicitamente definido. Assim, esta dimensão dos nossos modelos levou a participante a refletir sobre uma característica do grupo que até então ela não tinha considerado, senão implicitamente. Neste caso, a organização do espaço de solução proposta em nossos modelos ampliou o espaço de solução original da projetista.

Ainda durante a especificação do grupo a projetista quis definir habilidades comunicativas para membros de uma sub-hierarquia, da mesma forma que se poderia definir um objeto sendo compartilhado por estes membros. Esta observação reforça a observação e sugestão da projetista do SERG de se acrescentar à linguagem de *design* do protótipo habilidades comunicativas gerais.

Após especificado o modelo conceitual do grupo da projetista, passou-se então para a verificação da descrição deste pelas regras semânticas do protótipo. Nesta fase, muitas das inconsistências em potencial apontadas pelo sistema foram consideradas como não sendo inconsistências e foram explicadas pela *designer*. Ao explicar estas situações a participante definiu a lógica das decisões de *design* tomadas na modelagem do sistema, que até então não tinham sido explicitamente documentadas.

A verificação da descrição do modelo conceitual do grupo da participante teve duas contribuições significantes na especificação e modelagem da participante. A primeira delas aconteceu quando o sistema apontou uma inconsistência em potencial na interação entre dois membros do grupo. A projetista não considerou a situação uma inconsistência. No entanto, ao explicar a decisão que levava a esta situação ela se deu conta que membros de um determinado papel

na verdade sempre assumiam dois papéis distintos dentro do grupo. A participante definiu este "insight" como sendo um ganho fornecido pelo nosso protótipo, uma vez que até este momento a equipe não tinha atentado para este fato.

Como conseqüência deste novo "insight" a projetista resolveu, ao invés de fornecer uma explicação sobre a situação, modificar a sua especificação do grupo e descrever estes dois papéis separadamente. O fato de no domínio um determinado tipo de membro sempre assumir ambos fez com que ela sugerisse que seria desejável poder fazer não apenas a sobreposição de membros, mas também de papéis. Sugestão que mais uma vez reforça uma sugestão da projetista do SERG.

A segunda contribuição importante se deu na verificação da nova descrição do modelo de grupo, já com a modificação descrita acima. O sistema apontou como inconsistência em potencial o fato de todos os membros da hierarquia serem capazes de ver objetos privados dos demais membros. Ao avaliar esta situação a participante se deu conta de que não estava muito certa de como explicar esta decisão de *design*. A sua conclusão foi de que a equipe não tinha conscientemente considerado a questão, mas simplesmente "transferido" a situação do mundo real para o virtual. Embora a participante pudesse imaginar porque esta habilidade poderia ser desejável, ela também conjeturava sobre os problemas que ela poderia trazer. No caso desta decisão de *design* a equipe não tinha refletido sobre (1) a necessidade da habilidade em questão no ambiente virtual, e (2) os problemas que esta habilidade poderia porventura vir a trazer, uma vez que no mundo real a concretização desta habilidade exigia um certo esforço por parte do membro, enquanto que no mundo virtual este custo passaria a ser praticamente nulo.

Esta inconsistência em potencial identificada não levou a participante a alterar a sua descrição do grupo, mas a explicá-la. A participante entrou como explicação os motivos pelos quais ela imaginava que esta situação poderia ser desejável, mas que deveriam ser confirmados com os usuários, e também os problemas que ela antevia que poderiam decorrer dela e que não tinham sido considerados. Note-se que esta informação, uma vez documentada pela projetista, poderia ser usada tanto por ela quanto por outros membros da equipe de *design* do Qualitas.

Este exemplo vivenciado pela projetista do Qualitas mostra claramente como as regras do modelo de arquitetura podem levar o *designer* de um sistema multi-usuário a considerar questões importantes que de outra maneira teriam passado despercebidas. No caso de problemas como os antevistos pela participante virem a se concretizar eles passariam incólumes apenas durante a etapa de *design*, pois uma vez entregue o sistema e colocado em uso eles rapidamente seriam sentidos pelos usuários. Neste caso, o problema teria que ser resolvido em tempo de manutenção do sistema, o que geralmente é mais caro para o usuário tanto em termos produtivos quanto financeiros (Preece et al., 1994; Hartson, 1998).

* * *

A avaliação apresentada neste capítulo é apenas preliminar, uma vez que foi feita apenas para dois casos. Além disso, os objetivos do teste em cada caso eram distintos. Não obstante, eles foram capazes de mostrar que é possível implementar uma instância do modelo de arquitetura proposto, que a linguagem de *design* deste modelo é capaz de descrever o modelo conceitual de diferentes grupos, e ainda que as regras semânticas conseguem apontar para o *designer* problemas que de outra forma teriam passado despercebidos, permitindo assim que o *designer* melhore a qualidade do seu modelo conceitual do grupo. Assim sendo, estes testes já demonstram a validade das nossas hipóteses apresentadas nos capítulos 4 e 5. No próximo capítulo discutimos em detalhe os resultados dos testes preliminares e propomos testes exten-

sivos para os nossos modelos.

Nos dois testes feitos as participantes eram especialistas em Engenharia Semiótica. No entanto, é importante ressaltar que este não é um pré-requisito para se usar o protótipo ou os modelos propostos. A Engenharia Semiótica serviu como base teórica para o desenvolvimento dos modelos e assim está “embutida” neles. Desta forma, qualquer projetista de interfaces multi-usuário pode usar ferramentas baseadas no modelo de arquitetura e projetá-las de acordo com princípios de Engenharia Semiótica, sem no entanto ter conhecimento desta.

Chapter 7

Discussões

Neste trabalho, apresentamos o modelo abstrato de meta-comunicação (Capítulo 4), que propõe o desenvolvimento de linguagens de *design* separáveis de contexto e descritivas como apoio à expressão de projetistas de interfaces multi-usuário. A parte léxica desta linguagem é composta por dimensões básicas e permite a descrição do modelo conceitual do grupo do projetista. A parte semântica é formada por um conjunto de regras separáveis de contexto que atuam sobre estas dimensões, com o intuito de fornecer ao *designer* indicadores qualitativos sobre sua descrição.

Este modelo foi desenvolvido dentro do quadro teórico de Engenharia Semiótica, que focaliza no ensino de interfaces e requer o desenvolvimento de ferramentas, técnicas e métodos de apoio à expressão dos projetistas de interfaces. Dentro deste quadro teórico, apresentamos também um método de *design*. Este método enfatiza o *design* de interfaces como criação de meta-mensagens que expressam o modelo conceitual de usabilidade potencial da aplicação do *designer*. O objetivo é que, consciente do seu papel de emissor de uma mensagem, o projetista consiga expressá-la com maior clareza, buscando facilitar o entendimento do usuário.

Aplicando o modelo abstrato de meta-comunicação ao método de *design*, desenvolvemos um modelo que propõe uma arquitetura de arquitetura ao *design* de interfaces multi-usuário. Este modelo apóia o projetista durante a etapa de planejamento de parte do conteúdo da sua mensagem aos usuários de um sistema de grupo. Além da instanciação de uma linguagem de *design* separável de contexto fundamentada na proposta do modelo abstrato de meta-comunicação, este modelo de arquitetura oferece ao *designer* um instrumento de avaliação e um instrumento de apoio inicial à etapa de realização.

Com base no modelo de arquitetura implementamos um protótipo. Apresentamos testes de descrição de modelos conceitual de grupos distintos usando este protótipo. Estes testes nos permitiram demonstrar a validade das nossas hipóteses e ter indícios de como nossos modelos podem ajudar *designers* de interfaces multi-usuário a projetar modelos de grupo de melhor qualidade.

Neste capítulo discutimos os modelos de meta-comunicação e arquitetura e os testes de validação feitos usando o protótipo. Em seguida, fazemos uma proposta para a avaliação extensiva destes modelos. Finalmente comparamos o um sistema de apoio ao designer baseado no modelo de arquitetura com sistemas especialistas em *design* de interface existentes.

7.1 Modelo do Componente Multi-Usuário do Artefato de Meta-Comunicação

Nosso modelo abstrato de meta-comunicação propõe que uma linguagem de *design* seja oferecida ao *designer* para apoiar a sua expressão do conteúdo da mensagem relativo ao modelo conceitual de grupo. Esta linguagem é uma linguagem de *design* de usabilidade de interfaces multi-usuário, uma vez que ela reflete para o *designer* o seu papel de emissor de mensagem para os usuários de um grupo e permite que ele descreva esta mensagem, fornecendo-lhe indicadores qualitativos sobre ela.

As dimensões de descrição desta linguagem representam uma síntese das características de grupos propostas até hoje. Embora as nossas dimensões permitam uma descrição geral do modelo de grupo, elas enfatizam os seus modelos de comunicação e colaboração. Esta ênfase é motivada pelo fato de interfaces multi-usuário serem artefatos de mediação de comunicação, e o seu papel como tal ser uma parte fundamental da mensagem do *designer* para os usuários.

Estas dimensões assumem também o importante papel de fio condutor entre o modelo abstrato de meta-comunicação e os modelos que podem ser derivados dele, como por exemplo o modelo de arquitetura. Por serem separáveis de contexto, elas garantem a continuidade semiótica (de Souza, 1998) entre estes modelos, ou seja, que a descrição destas dimensões e das unidades descritivas de modelos derivados do modelo abstrato de meta-comunicação, como por exemplo os construtores do modelo de arquitetura, mais a explicação das dimensões sejam suficientes para se entender o mapeamento entre estas dimensões e unidades.

As regras heurísticas que atuam sobre estas dimensões são as que fornecem ao projetista os indicadores qualitativos. Pelo fato de estes indicadores serem baseados em inferências separáveis de contexto, eles não levam em consideração o domínio da aplicação. Por isto, eles indicam ao *designer* potenciais inconsistências e não fazem julgamento de valor definitivo sobre elas. Assim, as linguagens de *design* derivadas da proposta do modelo abstrato de meta-comunicação não são capazes de garantir a qualidade da descrição gerada, mas garantem que as decisões tomadas pelo projetista terão levado em conta as suas repercussões na mensagem como um todo.

Na realidade, o trabalho consciente do *designer* e os indicativos de qualidade somente podem ser garantidos se o *designer* trabalha com as regras ligadas, ou pelo menos com as regras relevantes ligadas. No momento em que o projetista começa a desligar regras, ele pode, inadvertidamente, causar a perda de indicadores qualitativos relevantes. Como as regras são separáveis de contexto e não conseguem expressar inferências sobre o domínio, o modelo não é capaz de avaliar se as regras desligadas são realmente não-relevantes, e nem, se ao desligá-las, o projetista não impediu que inconsistências relevantes fossem apontadas. Se o *designer* desliga regras relevantes, ele causa um decaimento da avaliação do seu modelo de grupo. Não é possível calcular esta curva de decaimento, mas é evidente que se o projetista desliga todas as regras ele perde o apoio qualitativo da linguagem à sua descrição.

7.1.1 Expressão de Atos de Fala

Quando pessoas se expressam através de linguagem natural, elas comunicam a seus ouvintes, não apenas o conteúdo da mensagem, mas também o que se pretende atingir com aquela mensagem. Searle (1979) propôs uma taxonomia, na qual ele classifica atos de fala em cinco

categorias:

- Assertivos: O emissor se compromete (em vários graus) com a veracidade ou falsidade de uma proposição. Por exemplo, “O dia está bonito.” ou “Ele não foi a festa.”.
- Diretivos: O emissor tem a intenção de conseguir que o receptor siga um curso de ação futuro. Por exemplo, “Vá agora!” ou “Você pode pegar água para mim, por favor?”.
- Comissivos: O emissor se compromete (em vários graus) a um curso de ação futuro. Por exemplo, “Te encontro mais tarde no escritório.” ou “Prometo ir à festa.”.
- Expressivos: O emissor expressa um estado psicológico definido no conteúdo da mensagem. Por exemplo, “Me perdoe pelo esquecimento.” ou “Parabéns pela formatura.”.
- Declarativos: A fala do emissor instaura uma realidade no mundo. Por exemplo, “Eu vos declaro marido e mulher.” ou “Você está despedido.”.

Uma linguagem de *design* derivada da proposta do modelo abstrato de meta-comunicação permite que o *designer* expresse atos de fala declarativos, diretivos e assertivos. Quando em execução, uma aplicação de grupo é uma virtualidade ou “mundo virtual” no qual os usuários agem e interagem. Assim, cada definição do projetista é um ato declarativo, uma vez que o que foi definido passa a ser realidade neste mundo, que é a aplicação. Os atos diretivos são as especificações que definem a ação do usuário. Por exemplo, quando o projetista do EduWare define que o professor pode aplicar o teste nos alunos, o seu ato de fala pode ser entendido como: “Aplique o teste nos alunos, quando achar conveniente.”. Enquanto os atos declarativos e diretivos são atos expressos diretamente na linguagem de *design*, os assertivos são expressos indiretamente. Na mensagem que o projetista envia para o usuário ele transmite, entre outras coisas, a sua interpretação sobre o problema que os usuários querem resolver e como eles devem interagir entre si. Estas questões são asserções do projetista sobre o que ele acredita ser verdade sobre os usuários.

Na sua especificação o *designer* define os atos de fala que podem ser expressos pelos usuários através da interface. O projetista fornece aos usuários atos declarativos quando lhes fornece conjuntos transformacionais aditivos ou irrestritos, através dos quais os usuários podem instaurar novas realidades no mundo. Quando o *designer* define valores para as habilidades comunicativas, ele pode fornecer aos usuários os demais atos de fala. Quais atos de fala ele poderá oferecer vai depender das unidades descritivas de comunicação da linguagem.

Continuidade Mundo-Representação

A aplicação especificada pelo *designer* pode ser entendida como sendo uma representação de uma ou mais situações do mundo físico. Através das suas declarações na linguagem de *design*, o projetista cria esta representação. Assim, o que não é declarado pelo *designer* não faz parte desta representação e, simplesmente, não existe na virtualidade que resulta dela. Desta forma, um objeto que existe na situação do mundo físico sendo representada, se não for definido pelo *designer*, passa a não existir no mundo virtual “equivalente”.

O *designer* ao projetar seu modelo conceitual de grupo deve estar atento para que ele só crie dependências entre elementos que só podem ser manipulados via aplicação, evitando assim uma descontinuidade entre mundo e aplicação. Por exemplo, um curso do AulaNet pode requerer

que alunos façam e entreguem trabalhos. Os trabalhos devem ser enviados ao professor através do AulaNet e o sistema, automaticamente, armazena a informação de quem enviou ou não o trabalho[1]. Se estivesse definido no AulaNet que só alunos que entregam os trabalhos podem continuar a "assistir" o curso, e um professor permitisse que alunos entregassem trabalhos fora do AulaNet, então professor e aluno teriam um problema, uma vez que existiria uma quebra entre a realidade física e a realidade virtual. No entanto, esta quebra poderia ser evitada pelo *designer* se ele definisse que todo trabalho só pode ser feito dentro do ambiente virtual do AulaNet.

Descrições Dinâmicas

Toda e qualquer alteração que pode acontecer dentro de uma aplicação é prevista pelo *designer*. Mesmo que a alteração permita aos usuários destruir toda a aplicação definida e criar outra. Na seção 4.2, vimos que o modelo abstrato de meta-comunicação classifica os conjuntos transformacionais em os de configuração, os aditivos e os irrestritos.

Primitivas de configuração que permitam a alteração na forma de visualização ou representação de informação privada podem promover maior satisfação do usuário sem causar efeitos colaterais. No entanto, estas mesmas alterações em níveis interpessoais e públicos podem criar vários problemas. Por exemplo, imagine-se uma aplicação onde vários usuários que têm acesso a uma informação precisam tomar decisões sobre ela e cada usuário pode escolher a sua forma preferencial de visualizar esta informação. Um usuário poderia escolher o gráfico de torta, enquanto o outro escolheria o de barras. Quando estes dois usuários vão se comunicar sobre a informação eles podem ter problemas, uma vez que a representação a que eles se referem não é a mesma (Greenberg et al., 1996b; Ellis e Wainer, 1994) e existe uma quebra na coesão de suas interfaces.

No conjunto de mudanças de configuração entram também alterações que o usuário pode fazer sobre os elementos de interfaces. Alguns sistemas permitem que o usuário troque o texto ou figura de um botão, a ordem dos botões, dentre outras coisas. Do ponto de vista de interface, estas modificações são extremamente prejudiciais para a veiculação da mensagem do projetista para o usuário. Afinal, a interface é a expressão desta mensagem, e ao permitir que o usuário a altere, o *designer* abre mão da interface como meio de comunicação. No caso de interfaces multi-usuários este problema é ainda mais grave, uma vez que a mensagem não é enviada a apenas um usuário, mas a vários, e o sucesso da comunicação depende, entre outras coisas, da coesão e consistência entre as mensagens enviadas a cada um deles.

Os conjuntos transformacionais aditivos e irrestritos fornecem aos usuários finais "poderes" de *design*, e entram no âmbito de *End User Programming Languages* (EUPL's) ou Linguagens de Programação para Usuários Finais (Barbosa et al., 1998). As mudanças definidas pelos usuários podem gerar a introdução de inconsistências no modelo de grupo definido pelo *designer*. Assim, o ideal seria que o *designer* fornecesse aos usuários, não apenas as primitivas de transformação, mas também a linguagem de *design* (parcial ou completa). Os usuários passariam então a ser responsáveis pela qualidade das alterações que fizessem, uma vez que teriam indicadores qualitativos sobre elas.

Podemos comparar conjuntos transformacionais irrestritos às gramáticas irrestritas de lin-

[1] Este exemplo foi retirado de entrevista da autora com um dos responsáveis pelo desenvolvimento do AulaNet em 13 de julho de 1998.

guagens formais, onde produções e símbolos são potencialidades para rupturas, uma vez que podem gerar elementos nulos. Assim, quando o *designer* fornece aos usuários uma aplicação contendo transformações irrestritas, ele lhes está fornecendo apenas uma potencialidade de mensagem. Afinal, esta mensagem pode ser destruída, ou fundamentalmente alterada, pelos usuários.

Mudanças aditivas e irrestritas afetam tanto o conteúdo da mensagem, quanto a sua expressão. Afinal, quando os usuários criam novos elementos nos modelos de funcionalidade ou grupo de uma aplicação, estes elementos têm que ser representados também no seu modelo de interação. No caso da eliminação de um elemento existente, sua representação no modelo de interação também deve ser eliminada. No caso de interfaces multi-usuário, uma mudança no conteúdo da mensagem pode gerar a necessidade de alteração na interface de um ou mais membros. Assim, ao fornecer mudanças aditivas e irrestritas o projetista tem que prever também as mudanças de configuração necessárias para expressar as novas situações.

Quando mudanças em interfaces multi-usuário afetam mais de um membro do grupo, elas podem requerer que a sua definição também envolva mais de um membro. Neste caso, o *designer* tem que oferecer aos usuários não apenas uma ferramenta de *design*, mas uma ferramenta de *design* participatório (Hamalainen et al., 1991).

* * *

O modelo abstrato de meta-comunicação e o método de *design* permitem que vários tipos de apoio ao *designer* de interfaces multi-usuário sejam derivados, como por exemplo métodos de desenvolvimento, técnicas, diretrizes e modelos de ferramentas. No capítulo 5, apresentamos o modelo de arquitetura de suporte ao *design* de interfaces multi-usuário que desenvolvemos fundamentado no modelo abstrato de meta-comunicação e no método de *design*. Na próxima seção, discutimos este modelo de arquitetura.

7.2 Modelo de Arquitetura de Suporte ao *Design* de Interfaces Multi-Usuário

A nossa intenção com o modelo de arquitetura de suporte ao *design* de interfaces multi-usuário é dar suporte ao projetista durante a especificação do seu modelo conceitual do grupo que deverá estar expresso na sua mensagem para os usuários, ou seja, na sua interface multi-usuário. Comparamos o nosso modelo com outros apresentados no capítulo 2 e em seguida, discutimos cada um de seus componentes.

7.2.1 Comparação entre o Modelo de Arquitetura e Demais Modelos de Grupo

Nesta seção comparamos o modelo conceitual de *groupware* apresentado por Ellis e Wainer (1994) e o modelo Denver (Salvador et al., 1996), apresentados no capítulo 2, com o modelo de arquitetura que estamos propondo. A primeira grande diferença é que o nosso modelo de arquitetura foi construído dentro do quadro teórico de Engenharia Semiótica e tem como objetivo apoiar o *designer* de interfaces multi-usuário na sua expressão desta mensagem para os usuários. O modelo Denver e o conceitual de *groupware* não entendem a interface como meta-mensagem do *designer* para os usuários, e nem são baseados em nenhuma outra teoria.

Todos os modelos identificam características básicas para aplicações de grupo, e a descrição do grupo e do sistema colaborativo é feita através destas dimensões. As dimensões escolhidas por cada um dos modelos nos permitem determinar o foco de cada um destes modelos. As dimensões do modelo Denver são abrangentes e permitem uma descrição geral do modelo. Algumas destas dimensões são descritas intensionalmente, mas sua descrição não deixa claro que valores elas podem assumir, apenas os seus limites máximos e mínimos. Além disso, outras são descritas em linguagem natural e requerem que os projetistas as descrevam também em linguagem natural. Estas dimensões então dificultam o desenvolvimento de ferramentas de apoio à descrição do projetista, que as trate de maneira sistemática.

O modelo conceitual de *groupware* está dividido em três sub-modelos: o ontológico, o de coordenação e o de interfaces. Enquanto os modelos ontológico e de coordenação possuem um conjunto de características básicas que descrevem o modelo, o de interface é composto por um conjunto de pontos a serem observados ou que podem ser relevantes em uma interface de grupo. As características dos modelos ontológico e de coordenação focalizam-se nas tarefas do grupo. Assim, suas características permitem a descrição destas tarefas, e da relação dos membros com estas tarefas. Entretanto, algumas das características só podem ter seu conjunto de possíveis valores descritos em domínios específicos. Além disso, embora os autores chamem a atenção para o papel de aplicações de grupo como mediadores de comunicação entre os usuários, eles não oferecem dimensões que permitam a descrição da relação entre membros ou de sua comunicação mútua.

O modelo de arquitetura que estamos propondo tem como objetivo a expressão do *designer* sobre o seu modelo conceitual de grupo, e assim permite que ele descreva em alto-nível todo este modelo. Além disso, ele considera a interface multi-usuário um artefato de mediação de comunicação e, para que o projetista possa expressá-la como tal, focaliza nos modelos de colaboração e comunicação do grupo. Os valores que os construtores deste modelo podem assumir podem ser descritos intensional ou extensionalmente e, então, permitem que inferências semânticas separáveis de contexto sejam feitas a partir deles.

Todos estes modelos permitem uma descrição do modelo conceitual de grupo ou parte dele. Assim, eles fornecem a projetistas um meio de comparar diferentes modelos. No entanto, uma das limitações do modelo de Denver e do conceitual de *groupware* é que nenhum deles explicita como uma alteração do valor de uma das dimensões pode afetar as decisões em outras, e nem fornece uma maneira de avaliar qualitativamente a descrição feita. Nosso modelo de arquitetura lida com estas questões através das regras de inferência semântica e da base de conhecimento. As regras semânticas relacionam as dimensões e deixam claro para o *designer* como sua decisão sobre os valores de uma delas pode influenciar as demais. Associada a elas, a base de conhecimento fornece uma explicação sobre as regras e permite ao *designer* avaliar o custo e benefício de cada decisão para o seu domínio específico. Desta forma, o nosso modelo fornece ao *designer* indicadores qualitativos sobre sua descrição e garante que ao tomar decisões em pontos específicos, ele levou em conta as suas conseqüências no restante da descrição.

Uma outra distinção entre o nosso modelo e os outros apresentados é que, enquanto os outros não levam em consideração as mudanças do grupo e seu trabalho no tempo[2], o nosso

[2]Em um dos componentes do seu modelo de interface com o usuário, Ellis e Wainer mencionam a existência de um nível de informação, no qual os usuários podem inspecionar ou alterar o modelo de colaboração, mas não oferecem meios de se definir este nível ou estas alterações.

permite a descrição tanto da parte estática da aplicação, quanto da dinâmica. Para as descrições dinâmicas, as regras semânticas dinâmicas, a base de conhecimento e o cenógrafo fornecem indicadores qualitativos. Nós entendemos que é fundamental que o *designer* defina as mudanças e meta-mudanças que acontecem (ou podem acontecer) em uma aplicação de grupo, uma vez que elas são determinantes na mensagem sendo passada pelo *designer* e na usabilidade desta aplicação.

Os modelos de Denver e conceitual de *groupware* fazem considerações genéricas sobre a realização da interface. No entanto, tais considerações não levam em conta a descrição feita pelo projetista. O modelo de arquitetura fornece ao projetista, através do conselheiro de *widgets*, diretrizes e sugestões que são baseadas na descrição do seu modelo. Assim, o conselheiro de *widgets* apóia o *designer* na realização da interface multi-usuário que expressará um modelo de grupo específico.

7.2.2 Construtores

Os construtores do nosso modelo de arquitetura determinam que alguns aspectos de grupos sejam descritos explicitamente através deles, enquanto outros têm que ser expressos em termos de combinação destes construtores. Por exemplo, os modelos de colaboração e comunicação estão expressos explicitamente, enquanto que o de coordenação não está. No entanto a coordenação pode ser expressa através do modelo de colaboração, as habilidades comunicativas e a hierarquia dos membros. Assim, embora a coordenação não esteja explícita nos construtores, ela é verificada nas regras e faz parte da linguagem de *design*.

Os construtores são fundamentados nas dimensões básicas do modelo abstrato de meta-comunicação e, mais do que isso, são semioticamente contínuos a estas dimensões. Isto significa que existe um mapeamento do conjunto de construtores para o conjunto de dimensões e que este mapeamento pode ser abduzido[3] a partir da descrição destes conjuntos e de uma explicação sobre o conjunto de dimensões (de Souza, 1998). Assim, os construtores são baseados nas dimensões do modelo abstrato de meta-comunicação e balizados pelo objetivo da linguagem de *design* do modelo de arquitetura de colocar a comunicação e interação entre os usuários em primeiro plano.

Ao definir os construtores de comunicação, uma das questões que consideramos foi o seu nível de abstração. Em sistemas de grupo é comum que o *designer* deseje definir reuniões, transmissões (*broadcasts*), porta-vozes para o grupo ou subgrupos, assim como, a conversa particular entre dois colegas, uma ordem do superior a um de seus subordinados, entre outas coisas. Assim, decidimos definir construtores que representassem níveis básicos de comunicação, a partir dos quais atos de comunicação em níveis mais abstratos pudessem ser definidos.

A partir dos construtores de comunicação, ver, mostrar, falar e conversar, o *designer* pode definir atos de comunicação mais complexos. Para exemplificar, descrevemos abaixo como alguns destes atos poderiam ser descritos a partir dos nossos construtores comunicativos:

- **reunião**: é uma conversa onde todos os membros participantes podem conversar com

[3] Abduzir é o processo de a partir de fatos conhecidos tirar-se uma conclusão plausível e provisória, embora não provada. Por exemplo, suponha-se que é verdade que: "Um pacote de viagem para Maceió tem desconto." e "A agência de turismo RaqTur vende pacotes para o Nordeste do Brasil com desconto.". Assim, uma conclusão plausível (ou abdução) é que o pacote para Maceió em questão é oferecido pela RaqTur. Esta conclusão é provisória até que surjam evidências que a contradigam (o que pode nunca acontecer) (Eco e Sebeok, 1983).

todos os outros e, que acontece em um determinado momento e ambiente.

- **transmissão**: um membro fala a vários outros em um mesmo momento.
- **exposição**: um membro mostra algo a vários outros em um mesmo momento.
- **porta-voz**: um membro de uma comunidade de membros fala em nome de toda a comunidade.

Embora o modelo não inclua estas abstrações, as regras levam algumas em consideração como negociar e planejar (seção 5.2). Nestes casos, o modelo não verifica todas as condições destas abstrações, mas apenas se elas têm o construtor de comunicação que permite a sua realização (no caso, conversar). Podemos dizer, então, que estas regras não checam se os membros negociam ou planejam, e sim se eles têm potencial para isto.

Além disso, se diferentes construtores fossem derivados de cada tipo de habilidade comunicativa, então a linguagem de *design* poderia classificar os atos comunicativos. Por exemplo, ao invés dos construtores falar e conversar, a linguagem poderia ter construtores que representassem os atos de fala. Esta linguagem, então, poderia suportar estruturas como a ACCORD (Laufer e Fuks, 1995), de representação de diálogos, ou permitir modelagem de interfaces de sistemas como o Coordinator (Winograd e Flores, 1987), um sistema multi-usuário que coordena a comunicação em um ambiente de trabalho.

A descrição do *designer* do trabalho de cada membro e da interdependência entre o trabalho dos membros do grupo é feita através de objetos e modelo de colaboração. Não oferecemos ao *designer* formas de descrever explicitamente as tarefas, ou seja, a relação entre os objetos e o conjunto de ações que podem ser executadas sobre eles. Esta nossa decisão foi tomada com base nos objetivos da nossa linguagem de *design*. Primeiramente, as interdependências entre membros criadas por objetos ou tarefas são expressas através do modelo de colaboração. Além disso, a relação entre os objetos pertencentes a um mesmo membro não cria impactos na interação e comunicação entre ele e os demais membros do grupo. Assim, como a expressão da interação e comunicação entre os membros são o foco principal desta linguagem, ela não requer que tarefas estejam explicitamente representadas. O segundo ponto considerado foi que tarefas têm alta dependência do domínio, o que torna muito difícil (se não impossível) a sua representação de forma separável de contexto. E é justamente o fato de ser separável de contexto que permite à nossa linguagem fornecer aos projetistas indicadores de qualidade.

7.2.3 Regras Semânticas

Como foi apresentado na seção 5.2, as regras têm como objetivo relacionar as dimensões básicas que descrevem o modelo e, a partir desta descrição sintática, expressar inferências sobre o significado destas combinações. Assim, não tem como as regras darem conta de todos os possíveis domínios e aplicações, e nem de fazerem julgamento de valor dos significados encontrados. Por isso, e também para não restringir a criatividade do *designer*, o modelo é descritivo e o *designer* é o responsável. Como responsável, ele pode julgar que uma inconsistência em potencial identificada pelas regras como não sendo uma inconsistência, naquele domínio, ou caso específico, e pode desligar regras e grupo de regras que não considerar relevantes.

Por as regras serem separáveis de contexto e não levarem o domínio em consideração, elas não são capazes de avaliar a qualidade do *design*. No entanto, juntamente com a base de

conhecimento, elas dão indicativos de qualidade para o *designer*, à medida que elas apontam para inconsistências em potencial e as justificam para o projetista. Assim, elas fornecem ao *designer* informações sobre as conseqüências de suas decisões, permitindo que ele as leve em consideração ao tomar estas decisões. Embora as regras não garantam a qualidade do *design*, elas garantem (se estiverem ligadas) que qualquer decisão tomada pelo projetista terá sido consciente e avaliada.

Cabe aqui ressaltar que o conjunto de regras que propomos é ad hoc, baseado na nossa experiência em implementação de interfaces multi-usuário e na literatura existente. Assim, aquelas que empiricamente se mostrarem não relevantes podem ser alteradas, substituídas ou simplesmente eliminadas.

7.2.4 Linguagem de *Design*

A linguagem de *design* do modelo de arquitetura é formada pelos construtores e regras semânticas e, assim, permite a expressão do *designer* sobre o seu modelo conceitual de grupo, enfatizando a comunicação e colaboração entre os membros do grupo. Embora os construtores desta linguagem constituam características básicas que permitem a descrição de grupos, a linguagem não pode ser considerada uma linguagem de padrões (*pattern language*). Linguagens de padrões devem expressar padrões que conectam um problema freqüente a uma solução conhecida e recorrente para ele (Bayle et al., 1998; Winograd, 1996). A solução que estamos propondo para o problema de apoio ao *designer* durante a fase de planejamento de uma interface multi-usuário não é uma solução conhecida e recorrente, mas sim uma solução nova baseada no quadro teórico definido por Engenharia Semiótica e em conceitualizações destas interfaces encontradas na literatura da área.

Além de estarmos propondo uma nova solução para o problema de *design* de interfaces multi-usuários, esta solução tem como traço marcante ser separável de contexto. Assim, em diferentes domínios diferentes aspectos de caracterização podem ser relevantes, fazendo com que diferentes subconjuntos de regras sejam mantidas ligadas. Assim, embora a linguagem de *design* do modelo de arquitetura não seja uma linguagem de padrões, ela pode funcionar como geradora de tais linguagens. A geração de linguagens de padrões pode acontecer se o uso desta linguagem em diferentes domínios fizer emergir, em cada um deles, combinações recorrentes de valores atribuídos às dimensões básicas e subconjuntos de regras considerados relevantes.

Nesta linguagem, o *designer* pode explicitar atos declarativos através dos construtores dinâmicos que permitem a definição de meta-mudanças. Ao definir valores para as habilidades comunicativas, ele relaciona a comunicação dos usuários com o domínio. Neste nível de domínio ele pode, então, fornecer aos usuários qualquer um dos atos de fala, inclusive o de declaração (se por exemplo, um superior puder despedir seu subordinado). É interessante observar que a distinção entre falar e conversar aponta para atos de fala preferenciais. Falar favorece os atos de fala assertivos, no qual o usuário pode informar os demais sobre alguma coisa, e diretivos, onde os usuários podem fazer uma requisição a outros. Conversar permite qualquer ato de fala neste nível de especificação.

Esta linguagem de *design* pode ser estendida, através da criação de novos construtores e regras. No entanto para manter as propriedades da linguagem e do modelo, algumas condições devem ser observadas. Primeiramente, todo novo construtor ou regra deve ser monotônico, ou seja, após a sua inserção todas as propriedades e relacionamentos válidos, devem continuar

válidos. Além disso, novos construtores têm que ser fundamentados nas dimensões básicas do modelo abstrato de meta-comunicação e manter a continuidade semiótica entre esta linguagem e a daquele modelo. Novas regras, ao serem inseridas, devem ser classificadas quanto ao seu tipo e às características que avaliam. Ademais, para toda nova regra, uma explicação sobre ela deve ser criada e armazenada na base de conhecimento.

7.2.5 Base de Conhecimento

A base de conhecimento contém a explicação das regras. Assim, ela permite ao *designer* entender porque uma determinada situação foi considerada uma inconsistência e as repercussões desta "inconsistência", fornecendo, assim, informações relevantes a serem consideradas pelo projetista ao tomar uma decisão em relação àquela situação. Além disso, a base de conhecimento permite ao *designer* armazenar as suas próprias explicações ao decidir manter uma inconsistência.

Ao final da especificação do modelo de grupo, a base de conhecimento contém a lógica do projetista para suas decisões de projeto. Assim, a base de conhecimento não é útil apenas durante a especificação do *designer*, mas nos passos seguintes do processo de *design*. Nos passos de refinamento da especificação, a lógica do *designer* pode ser fundamental na definição do detalhamento da descrição. Além disso, o conhecimento sobre o modelo de grupo pode ser útil também para a definição dos *widgets* a serem usados e a criação de *help*'s instrutivos. Chamamos de *help*'s instrutivos aqueles nos quais o projetista explica aos usuários, não apenas que ações podem ser executadas em um determinado momento, mas também o porquê de estas ações poderem, ou não, ser executadas (de Souza, 1997).

Para ilustrar um *help* instrutivo que poderia ser construído com uma base de conhecimento como a proposta no modelo de arquitetura, imagine-se uma aplicação multi-usuário de controle de notas de alunos de um departamento. O projetista define um grupo constituído por um coordenador, um grupo de professores subordinados a ele e um grupo de monitores subordinados a estes professores. O projetista define que a hierarquia do grupo é rígida e, embora o coordenador possa falar com os monitores para marcar reuniões gerais, eles só podem dar ordens a professores, que podem, por sua vez, dar ordens a monitores.

Neste ambiente, os monitores são os responsáveis por dar as notas dos alunos. Suponha-se que ao verificar as notas dos alunos, o coordenador descubra que enquanto um monitor dava as notas em algarismos, o outro as dava em conceitos, como mostrado na Figura 7.1. Assim, o coordenador resolve mandar uma mensagem de requisição aos monitores, pedindo-lhes que padronizem as notas. Ao acessar a tela de envio de mensagens representada na Figura 7.2, o coordenador descobre que quando seleciona a mensagem de requisição, a opção <monitores> no campo dos receptores de mensagens fica inativo, ou seja, que ele não é capaz de mandar aos monitores mensagens de requisição. Ao acessar o *help* típico, a resposta que ele obtém é: "Mensagens de requisição só podem ser enviadas aos membros listados no campo de receptores que estiverem ativos.". Isto o coordenador já tinha compreendido, e o *help* não lhe oferece nenhuma explicação sobre o porquê de os monitores estarem inativos. No entanto, se o projetista tivesse fornecido aos usuários um *help* instrutivo, então o coordenador receberia uma mensagem como: "Membros do grupo só podem enviar requisições para os seus subordinados imediatos.". Esta mensagem permite ao coordenador entender porque ele não consegue atingir o seu objetivo, e independente de se ele concorda ou não com a solução do *designer*, ele sabe

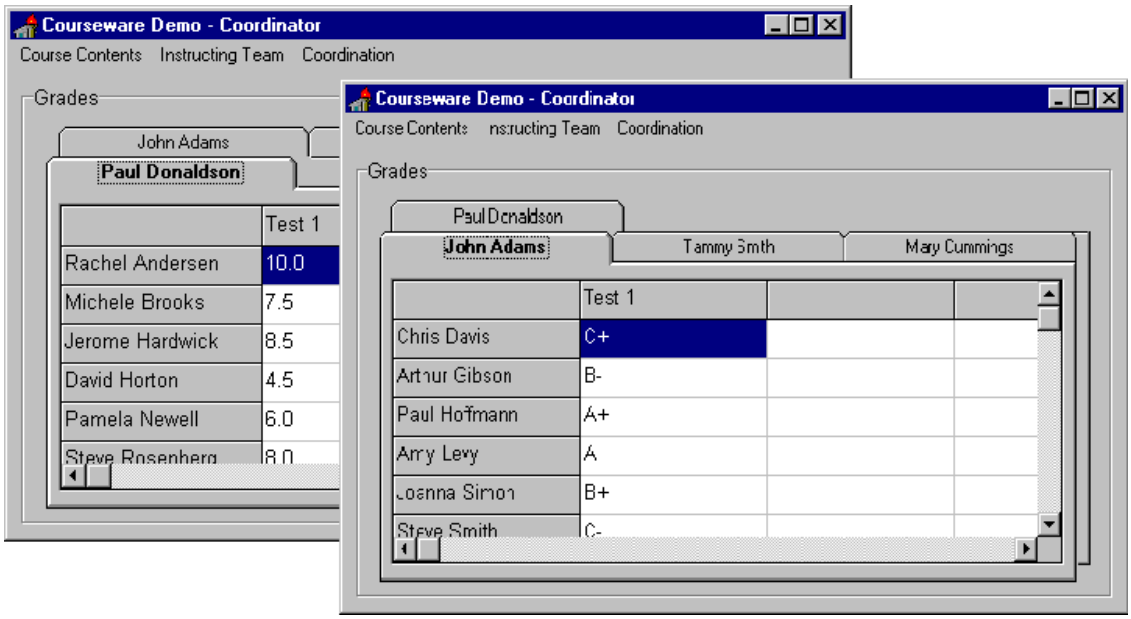

Figure 7.1 – Telas do coordenador de notas dos alunos.

decidir como agir para atingir seu objetivo.

7.2.6 Cenógrafo

O papel do cenógrafo no modelo de arquitetura é permitir que o *designer* tenha uma noção sobre o poder de alteração que ele está fornecendo aos usuários. Embora o cenógrafo aponte ao projetista inconsistências potenciais que os usuários podem introduzir em tempo de execução, ele não tem como saber ou informar o *designer* sobre quais são todas as possíveis inconsistências, se todas elas já foram encontradas, ou quantas faltam.

Quando o projetista fornece aos usuários poder de alteração ele, de fato, lhes permite trocar do papel de usuário para *designer*. Assim, é importante que tanto o *designer*, quanto os usuários estejam cientes desta troca de papel. Além disso, o projetista precisa fornecer ao usuário não apenas os conjuntos transformacionais, mas também o conhecimento necessário para que ele possa fazer um *design* de qualidade, ou seja, fazer alterações relevantes, sem no entanto introduzir quebras na mensagem original (Barbosa et al., 1998). Para isso, é preciso que o *designer* ofereça também ao usuário meios de obter indicadores qualitativos das mudanças que ele está realizando, como por exemplo a linguagem de *design* do modelo de arquitetura, ou parte dela, e também conhecimento sobre as motivações e intenções do projetista original, que poderia vir através de um *help* instrutivo.

7.2.7 Conselheiro de *Widgets*

O conselheiro de *widgets* é um instrumento de apoio à realização da interface multi-usuário sendo especificada pelo projetista. O apoio à realização que o conselheiro de *widgets* é capaz de oferecer ao projetista é em alto-nível, através de diretrizes e sugestões. Uma das razões para isto é que, no passo do planejamento da interface multi-usuário que o modelo de arquitetura

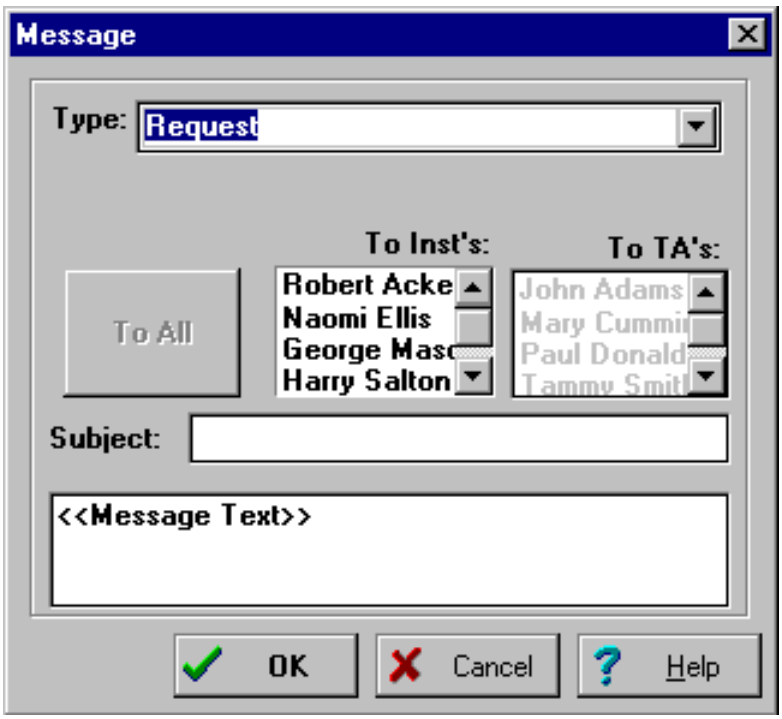

Figure 7.2 – Tela para envio de mensagem.

apóia o *designer*, muitas especificações ainda estão em alto-nível e pode não se ter informação suficiente para se definir totalmente a interface ou parte dela, tornando uma escolha de elementos de interface neste ponto precipitada.

No momento em que o projetista termina o passo de planejamento de uma interface multi-usuário e tem toda a especificação detalhada da mesma, ele inicia a realização desta interface. O ideal seria que o *designer* tivesse então um apoio como o modelo proposto por Leite (1998), porém para interfaces multi-usuário. No seu modelo, Leite propõe que uma linguagem de *design* que apóia o *designer* de interfaces mono-usuário seja acoplada a *toolkits* de construção destas interfaces. Esta linguagem dá arquitetura ao projetista no processo de mapeamento do seu modelo de funcionalidade (definido na fase de planejamento) para o seu modelo de interação. Para isto, esta linguagem fornece ao *designer* um vocabulário de elementos de interface, seus interpretantes preferenciais e um conjunto de regras heurísticas que os combinam para atingir uma determinada mensagem.

Embora uma extensão ao modelo de Leite para interfaces multi-usuário fosse o ideal, por enquanto não é possível estendê-lo. Esta impossibilidade fica por conta dos *toolkits* para interfaces multi-usuários ainda serem imaturos (Greenberg e Roseman, 1998) e o conjunto de elementos de interface ainda estar sendo criado sob demanda (Gutwin et al., 1995). Isto significa que o vocabulário para interfaces multi-usuário, equivalente ao proposto por Leite, ainda está sendo desenvolvido e a criação de uma linguagem, tendo-o como base, seria precoce.

O fato de o conjunto de elementos de interface ainda ser aberto e novas *widgets* estarem sendo criadas à medida que projetistas precisam expressar um novo tipo de interação ou informação restringe também a atuação do conselheiro de *widgets*. Apesar de o conselheiro atuar sobre a especificação alto-nível do projetista, ainda durante a fase de planejamento, ele já dá sugestões sobre *widgets* que poderiam ser usados para expressar uma informação ou

mensagem, baseado no interpretante preferencial dos *widgets*. No entanto, estas suas sugestões ficam limitadas aos *widgets* conhecidos.

7.3 Validação

Para demonstrar a validade dos modelos propostos, implementamos um protótipo baseado no modelo de arquitetura e, como teste, projetistas de dois sistemas multi-usuário descreveram seus modelos conceituais de grupo para cada um deles. A implementação do protótipo já demonstra que o modelo de arquitetura proposto é possível de ser implementado. Os testes, por sua vez, dão indícios de diferentes aspectos tanto do protótipo, quanto dos modelos em que ele foi baseado.

Embora distintos, os modelos de grupo de ambos sistemas multi-usuário puderam ser descrito na linguagem de *design* disponível. As projetistas dos dois testes foram capazes de descrever todas as características desejadas usando os construtores da LD. Durante a sua descrição, elas fizeram sugestões de construtores que poderiam ser acrescentados à LD com o objetivo de aproximá-la ainda mais dos objetivos de expressão do *designer*. A maior parte destas sugestões implicam mudanças para o protótipo, ou seja, para a instância do modelo de arquitetura e não para o modelo propriamente dito. Mesmo aquelas que gerariam acréscimo de novos construtores no modelo de arquitetura (e não só uma diferente implementação dos existentes) não afetariam o modelo abstrato de meta-comunicação, uma vez que eles poderiam ser definidos em termos das dimensões deste modelo. Além disto, todos os construtores da LD foram utilizados durante os testes, o que nos dá indicadores da sua relevância.

Enquanto o teste do SERG, até por ter sido mais extenso, nos forneceu mais informações sobre o protótipo e a LD, o teste do Qualitas foi capaz de nos mostrar a importância dos componentes que não estavam implementados no protótipo: conjunto de construtores de meta-mudanças e cenógrafo. A necessidade dos construtores de meta-mudanças ficou clara logo no início, uma vez que a maior parte do trabalho dos membros do grupo do Qualitas é sobre objetos que eles criam em tempo de execução. Foi decidido então se modelar um possível estado do sistema. Desta forma, o estado modelado corresponde a um cenário típico criado a partir de um cenário concreto (que incluiria as descrições dinâmicas) e poderia ter sido criado pelo cenógrafo. A verificação do estado modelado pelas regras semânticas chamou a atenção da projetista para importantes questões, que até então não tinham sido consideradas. Assim, podemos presumir que estas contribuições poderiam decorrer da interação da *designer* com o cenógrafo para criar um cenário típico e da sua verificação pelas regras.

Uma das questões que levantamos neste trabalho foi a necessidade de ferramentas que guiem o *designer* na direção de um *design* de maior qualidade. Como já foi colocado, o modelo de arquitetura proposto não tem como garantir a qualidade do *design*, mas apenas fazer com que *designers* sempre entendam e levem em consideração como cada uma de suas decisões afeta outros aspectos do grupo sendo modelado. Desta forma, o modelo de arquitetura fornece indicadores qualitativos para o *designer*, permitindo que ele os considere e melhore a qualidade do seu *design*. Os testes executados demonstram a validade desta nossa hipótese, uma vez que em ambos os testes inconsistências em potencial somadas à suas respectivas explicações permitiram ao *designer* considerar "efeitos colaterais" de suas decisões e aspectos do grupo que de outra forma não teriam sido levados em conta. Estas considerações algumas vezes fizeram com que as projetistas mudassem sua descrição do grupo, obtendo assim uma descrição que con-

sideraram de melhor qualidade. Além disso, no teste do Qualitas uma das inconsistências em potencial apontadas pelo protótipo levantou uma questão, cuja decisão dependia de discussão com os usuários da aplicação.

Os testes deram indícios de que o protótipo, e logo o modelo de arquitetura, ao apontar inconsistências em potencial permitem a *designers* perceber erros de modelagem, efeitos colaterais indesejáveis e até mesmo antever problemas operacionais de interação do grupo que podem vir a acontecer. Nestas situações, podemos dizer que o modelo de colaboração e as ferramentas dele derivadas agem como ferramentas de avaliação formativa, isto é, avaliação feita anteriormente à implementação do sistema e que influencia o seu desenvolvimento (Preece et al., 1994). Assim sendo, elas muitas vezes causam a redução tanto do custo financeiro do sistema, uma vez que permitem a correção do erro no início do processo de *design*, quanto de uso, na medida que permitem que *designers* ofereçam um sistema de melhor qualidade para os usuários, diminuindo a oportunidade de os usuários não entenderem a mensagem sendo passada (Hartson, 1998).

Embora a validação dos modelos propostos feita neste trabalho nos tenha permitido demonstrar como eles, e as ferramentas derivadas deles, podem apoiar o projetista e guiá-lo na direção de um *design* de melhor qualidade, ela é apenas preliminar. Uma avaliação extensiva dos modelos apresentados neste trabalho requer o desenvolvimento de pelo menos um produto de apoio ao *design* baseado nestes modelos, sua implantação em um ambiente real de *design* de interfaces multi-usuário e uma amostragem dos produtos desenvolvidos utilizando-o. Na próxima seção apresentamos uma proposta de testes extensivos do modelo de arquitetura que devem ser feitos, mas que estão fora de escopo deste trabalho.

7.3.1 Proposta de Testes Extensivos

Para que testes mais extensivos sejam feitos, é necessário que se tenha não apenas um protótipo deste modelo, mas também uma ferramenta bem-acabada, que possa ser usada com eficiência. Um dos requistos para esta ferramenta é que ela tenha uma interface que deixe claro para o *designer* as definições de características básicas já especificadas, e as relações entre elas. Antes de ser usada para testes do modelo, a própria ferramenta tem que ser submetida a testes de usabilidade.

Uma vez pronta a ferramenta, ela deve ser usada então para testar o modelo. Este teste deve ser feito em ambientes reais de desenvolvimento de interfaces multi-usuário. Para cada teste é importante documentar:

- Construtores que não foram usados para descrever grupo.
- Características de grupo que não se conseguiu descrever com os construtores existentes.
- Regras identificadas como irrelevantes e desligadas.
- Inconsistências relevantes identificadas e sua classificação como sendo de especificação (o *designer* esqueceu ou cometeu um erro ao especificar algo que estava previsto), ou de modelagem do grupo (*designer* não previu inconsistência).
- Modificações nos modelos de grupo decididas com base nas inconsistências.

Com base na informação obtida sobre os construtores pode-se avaliar se eles são suficientes ou não. Se não o forem, deve-se definir quais novos construtores devem ser fornecidos aos *designers*. Neste caso, para cada novo construtor deve-se avaliar se ele pode ser acrescentado no modelo de acordo com as condições apresentadas na seção 7.2.4. Se algum construtor não puder ser construído a partir das dimensões básicas do modelo abstrato de meta-comunicação, mantendo a continuidade semiótica entre eles, então esta situação implicaria na necessidade de alteração deste modelo também. Se existirem construtores que não tiverem sido usados, deve-se verificar se é para um domínio ou situação específica ou se ele nunca é usado.

Para as regras, deve-se verificar também se existem regras que são consideradas irrelevantes para todos os testes. Neste caso, verificar se é devido a um traço comum dos domínios, ou se porque a regra não é útil e pode ser retirada do componente semântico da linguagem. Para as inconsistências relevantes identificadas, deve-se classificar o grau de gravidade da inconsistência identificada e das modificações que ela gerou. Para que a ferramenta, e conseqüentemente o modelo, sejam considerados úteis, eles devem ser capazes de gerar uma melhoria de qualidade de modelos conceituais de grupo ou suas representações.

As informações acima permitem uma avaliação da linguagem de *design* oferecida. Além disso, deve-se avaliar também a base de conhecimento, o cenógrafo e o conselheiro de *widgets*. A base de conhecimento deve ser avaliada sob dois aspectos. O primeiro é se as explicações das regras facilitam o entendimento dos projetistas sobre porque as situações identificadas são consideradas inconsistentes, e lhes oferecem informações suficientes para julgar se naquele domínio a situação é relevante ou não. O outro aspecto é se as explicações oferecidas pelos projetistas são úteis durante a modelagem do grupo e passos seguintes do *design* da interface.

Para o conselheiro de *widgets* deve-se avaliar qual o modo preferencial dos projetistas, se o de pós-processamento ou o interativo, e a causa desta preferência. Este teste pode mostrar se em diferentes domínios diferentes modos são adequados, ou se um modo é sempre mais adequado que o outro. Além disso, deve-se testar o quanto as diretrizes e sugestões produzidas pelo conselheiro são úteis durante os passos seguintes do *design* e, principalmente, durante a fase de realização da interface.

A avaliação do cenógrafo envolve dois passos. O primeiro para verificar a quantidade de cenários típicos que os projetistas consideram suficientes de se testar, e a quantidade e qualidade das inconsistências relevantes que conseguem ser identificadas com o seu uso. O segundo passo do teste deve ser feito em tempo de execução, quando os usuários realizam as mudanças oferecidas. Neste passo, deve-se identificar inconsistências que os usuários conseguiram introduzir, apesar dos testes em cenários típicos.

Finalmente, deve-se avaliar a experiência do *designer* ao usar a ferramenta. Como o objetivo do modelo é apoiar os *designers* na fase de planejamento da interface, é importante que eles achem a ferramenta benéfica para seu *design*. Avaliar as críticas apresentadas pelos projetistas e verificar se elas se referem à ferramenta, ao modelo de arquitetura ou até mesmo ao modelo abstrato de meta-comunicação, e quais as suas implicações.

Os testes descritos até este ponto são relativos ao trabalho de *design* da interface. Outro ponto importante que pretendemos, ao usar o quadro teórico de Engenharia Semiótica para desenvolver este trabalho, é melhorar a comunicação entre *designer* e usuário e assim permitir a produção de interfaces que podem ser usadas com maior eficiência e qualidade. O teste ideal para este quesito seria ter-se diversos projetistas, com perfis semelhantes, desenvolvendo interfaces multi-usuário para uma mesma aplicação. Destes projetistas metade deveria utilizar

a ferramenta como apoio e a outra metade não. Após construídas as interfaces testes de usabilidade por inspeção e com usuários deveriam ser feitos para se avaliar se a mensagem do *designer* fica melhor entendida pelos usuários das interfaces desenvolvidas com a ferramenta de apoio.

No entanto, este teste ideal não pode ser feito em um ambiente real de *design*, uma vez que a situação de teste não é praticada. Assim, a opção seria aplicar testes de usabilidade por inspeção e com usuários em interfaces com e sem o apoio da ferramenta geradas e tentar observar se as que tiveram o apoio da ferramenta apresentam melhorias recorrentes em relação às outras. As principais melhorias pretendidas seriam no aspecto de entendimento pelos usuários das mensagens sendo passadas pelo *designer* e da capacidade destes usuários de usarem a aplicação de forma eficiente e criativa.

7.4 Comparação entre Ferramentas Baseadas no Modelo de Arquitetura e Outros Sistemas Especialistas de *Design* de Interfaces

Uma ferramenta desenvolvida a partir do nosso modelo de arquitetura, como por exemplo o protótipo apresentado, pode ser definida como sendo um sistema especialista em *design* de interfaces multi-usuário. Na verdade, especialista na etapa de planejamento destas interfaces, uma vez que é sobre esta etapa que ela é capaz de aconselhar o *designer* e guiá-lo na direção de um *design* de melhor qualidade. Apesar de vários sistemas especialistas em *design* de interfaces já terem sido propostos (Kim e Foley, 1990; Wiecha e Boies, 1990; Bleser e Sibert, 1990; Mark, 1986), nenhum deles é para interfaces multi-usuário ou para a etapa de planejamento da interface.

A maior parte dos sistemas estudados apóiam projetistas de interfaces mono-usuário em diferentes aspectos da etapa de realização destas interfaces. Bleser e Sibert (1990) apóiam o projetista no aspecto físico da interface, uma vez que ajudam-no a escolher o melhor dispositivo de interação para as ações a serem executadas pelo usuário. O sistema DON, desenvolvido por Kim e Foley (1990), auxilia o projetista na definição de sua interface, escolhendo os elementos de interface a serem usados e a sua disposição na tela, de acordo com preferências e prioridades definidas pelo projetista. A ajuda oferecida por Wiecha e Boies (1990) através do sistema de gerenciamento de interfaces com o usuário ITS é mais abstrata que a fornecida por DON. Eles se preocupam em ajudar o o projetista a manter a consistência da definição sintática da interface (sem no entanto entrar no mérito dos *widgets* a serem escolhidos), usando para isto um conjunto de regras de estilo.

Estes sistemas diferenciam dos sistemas derivados do modelo de arquitetura em outros aspectos além da etapa de *design* e do tipo de interface sendo projetada que eles apóiam. O principal ponto de divergência entre eles é que enquanto os primeiros aconselham, os outros agem. Isto é, os sistemas derivados do modelo de arquitetura apontam ao *designer* inconsistências em potencial e fornecem explicação sobre elas, ajudando-o então a pesar as conseqüências e implicações de suas decisões. Os sistemas apresentados acima, por sua vez, tomam controle do *design* e, baseado em preferências e prioridades definidas pelo *designer*, tomam decisões por ele.

A diferença acima é fundamental, uma vez que ela caracteriza os sistemas apresentados

como antagónicos à proposta da Engenharia Semiótica[4]. Afinal, ao tomar decisões de *design* pelo projetista, estes sistemas se tornam autores, ou pelo menos co-autores, da mensagem sendo passada pelo projetista aos usuários. No entanto, estes sistemas atuam apenas na definição da expressão da mensagem, sem ter conhecimento do seu conteúdo ou das intenções de *design*.

O sistema Consul desenvolvido por William Mark (1986) apresenta uma motivação em linha com a perspectiva da Engenharia Semiótica. O sistema permite que o *designer* expresse o seu modelo conceitual de usabilidade potencial para que assim ele aumente as chances de os usuários criarem um modelo adquirido de usabilidade consistente[5]. O sistema apóia projetistas de interfaces mono-usuário na etapa de planejamento, permitindo que ele defina e descreva objetos sobre os quais os usuários podem agir e que ações podem ser realizadas sobre cada um deles.

Apesar do sistema Consul se aproximar mais dos sistemas derivados do modelo de arquitetura que os demais sistemas discutidos nesta seção, ele ainda difere em muitos pontos. Consul é um sistema dependente de domínio uma vez que sua base de conhecimento contém termos (que se referem a objetos e ações) e descrições destes termos para um domínio específico, no caso automação de trabalho de escritório. As regras de inferência atuam para garantir que os novos termos sendo definidos pelo *designer* são consistentes com os já existentes. Quando uma inconsistência é encontrada o *designer* é informado e pode examinar as definições dos termos existentes e daquele que ele está criando. No entanto, Consul não oferece ao *designer* uma explicação de porque ele considerou a situação uma inconsistência. Além disso, ele é prescritivo e se o *designer* discordar da sua conclusão ele não permite que o *designer* passe por cima da regra de inferência e nem tampouco "o convença" do seu erro. Neste caso, a única saída para o projetista é mudar sua estratégia de definição do termo para uma com a qual o sistema "concorde".

Assim, os sistemas derivados do modelo de arquitetura, mais do que sistemas especialistas, podem ser considerados sistemas conselheiros. Afinal, diferentemente dos sistemas apresentados nesta seção, eles não impõem suas conclusões sobre o *designer*, mas simplesmente oferecem o seu conhecimento como recurso para o *designer* obter indicativos de qualidade sobre seu *design*. Uma outra distinção é o fato dos sistemas derivados do modelo de arquitetura fornecerem explicações sobre os seus conselhos, o que é considerado positivo por usuários de sistemas conselheiros (Carroll e McKendree, 1987).

[4]Em seu trabalho Leite (Leite, 1998) propõe um modelo de apoio à etapa de realização da interface sob a perspectiva da Engenharia Semiótica.

[5]Mark se refere ao que nós chamamos de modelo conceitual de usabilidade potencial e modelo adquirido de usabilidade como modelo conceitual de *design* e modelo do usuário, respectivamente.

Chapter 8

Conclusões e Trabalhos Futuros

Neste trabalho, propusemo-nos a criar modelos que permitissem o apoio a projetistas de interfaces multi-usuário durante o desenvolvimento destas. Para isso, usamos o quadro teórico de Engenharia Semiótica, estendido para interfaces multi-usuário. Os modelos, métodos, técnicas e ferramentas que podem ser derivados de nossos modelos visam dar suporte à expressão do *designer* sobre parte do conteúdo da sua mensagem sendo passada aos usuários, mais especificamente à parte relativa ao seu modelo conceitual do grupo.

Consideramos que os projetistas de interfaces multi-usuário desenvolvem-nas em duas etapas de um processo de *design top-down*, a de planejamento e a de realização. O modelo abstrato de meta-comunicação que apresentamos propõe que linguagens de *design* separáveis de contexto e descritivas sejam oferecidas aos projetistas. Estas linguagens devem ter como elementos léxicos unidades descritivas fundamentadas nas dimensões básicas de caracterização apresentadas. Além disso, este conjunto léxico deve incluir tanto elementos de descrição estáticos, quanto dinâmicos, permitindo então que o *designer* descreva aspectos do grupo dependentes e independentes do tempo. Estas linguagens devem oferecer também um componente semântico composto por regras de inferências que atuam sobre estas dimensões. Estas regras devem ser capazes de expressar inferências a partir da estrutura sintática descrita, mas não do domínio representado. Desta forma, linguagens de *design* derivadas da proposta do modelo abstrato de meta-comunicação apóiam o projetista, não apenas na sua descrição do grupo, mas no seu entendimento de como os valores atribuídos às características do grupo se relacionam. Assim, o modelo cria uma base para que se forneça ao *designer* informações sobre como uma decisão sobre um aspecto do grupo pode afetar os demais.

Para mostrar como a proposta do modelo abstrato de meta-comunicação poderia ser aplicada, apresentamos um modelo de arquitetura de suporte ao *design* de interfaces multi-usuários. Além de oferecer ao projetista uma linguagem de *design* separável de contexto que permite descrições estáticas e dinâmicas, este modelo estende seu apoio ao projetista ao lhe fornecer também uma base de conhecimento, um cenógrafo e um conselheiro de *widgets*. A base de conhecimento complementa as informações fornecidas pelo conjunto de regras sobre que dimensões estão relacionadas, explicando como elas estão relacionadas e como o valor de uma pode afetar a outra. Além disso, a base de conhecimento permite ao projetista entrar com suas próprias explicações sobre suas decisões, criando assim a lógica do *design* do grupo. O cenógrafo permite ao *designer* criar simulações de cenários que ele autoriza o usuário a criar através das meta-mudanças, e aplica a estes cenários as regras semânticas. Finalmente, o conselheiro de *widgets* dá o primeiro passo em direção à fase de realização ao fornecer ao *designer*

diretrizes e sugestões baseadas na descrição do grupo especificada. Estas diretrizes e sugestões incluem indicações de que considerar durante a fase de realização e de *widgets* que parecem apropriados para representar determinadas características descritas.

Através destes modelos atingimos nosso objetivo de apoiar o *designer* durante a fase de planejamento da sua interface multi-usuário, fornecendo-o com indicadores qualitativos sobre a sua especificação. Esta é uma contribuição importante do nosso trabalho, uma vez que as formas de apoio existentes atualmente não auxiliam o *designer* na tomada de decisões, nem lhes fornecem qualquer indicação sobre a qualidade da sua descrição. Vamos um pouco além da fase de planejamento, fornecendo ao projetista as primeiras considerações sobre a fase de realização. Assim, este trabalho é um primeiro passo na direção da construção de um ambiente de *design* de interfaces multi-usuário que dê suporte ao projetista durante todo o processo de *design* de interfaces com qualidade.

Além disso, o modelo abstrato de meta-comunicação pode gerar diferentes formas de apoio ao *designer*, como modelos de arquitetura e técnicas. O modelo de arquitetura, por sua vez, permite que dele sejam derivadas diretrizes e ferramentas distintas que apóiem o *design* de interfaces multi-usuário. Além destes modelos, o método de *design* para usabilidade favorece o foco do ensino de interfaces adotado pela Engenharia Semiótica e permite a criação de métodos de desenvolvimentos de interfaces, que permitem seu entendimento como meta-mensagens do *designer* para os usuários.

Trabalhos Futuros

Este trabalho é um primeiro passo na direção de um ambiente de apoio ao processo de *design* de interfaces multi-usuário e propõe modelos que dão suporte ao *designer* durante a fase de planejamento deste processo. Como vimos no capítulo 7, uma avaliação extensiva destes modelos é necessária. Para que esta avaliação seja possível, é preciso que alguns estudos apontados pelos nossos testes preliminares sejam realizados.

Um dos pontos levantados foi a necessidade de uma investigação sobre uma interface capaz de transmitir ao *designer* as dimensões a serem definidas e a relação entre elas de forma clara e eficiente. As sugestões da participante do teste do SERG indicam que a implementação da linguagem de *design* do modelo de arquitetura deveria ser orientada a objeto. Assim, um estudo de como seria esta implementação e a sua adequação se faz necessário.

Além disso, descrevemos condições a serem observadas para se incluir novos construtores e regras no modelo, assim como abstrações sobre os construtores apresentados. Seria interessante pesquisar um mecanismo que permitisse esta inclusão de forma automática. Para isto, este mecanismo deveria permitir a descrição de construtores a partir das dimensões básicas do modelo abstrato de meta-comunicação, garantindo a sua continuidade semiótica, a descrição de abstrações em termos dos construtores existentes, a definição de novas regras, e ainda, ser capaz de avaliar as regras afetadas pelas novas definições.

No início desta pesquisa, trabalhamos junto ao ADDLabs[1] (s.d.) no desenvolvimento de uma interface multi-usuário para o projeto ADDProc, um sistema multi-usuário para o *design* de plantas de processamento de óleo (Garcia e Vivácqua, 1996). Enquanto este trabalho apontou para várias das considerações dos modelos, a interface foi desenvolvida sem o apoio que

[1] Laboratório de Documentação e Design Inteligente

estamos propondo neste trabalho. Assim, uma avaliação da interface atual e um re-projeto usando uma ferramenta de apoio baseada no modelo de arquitetura seria um estudo inicial proveitoso.

Um dos pontos que colocamos que deveria ser avaliado é a qualidade da comunicação *designer*-usuário, e o suporte que nossos modelos oferecem para esta comunicação. Para isto devem ser usados testes de usabilidade de inspeção da interface e com os usuários. No entanto, os métodos de avaliação hoje existentes foram desenvolvidos dentro de uma abordagem cognitiva e não têm como um de seus objetivos avaliar o processo de comunicação *designer*-usuário. Assim, esta avaliação aponta para a necessidade de se ter testes de usabilidade que unam a abordagem semiótica e a cognitiva, incluindo em seus objetivos a avaliação da transmissão da mensagem entre *designer* e usuários.

À medida que a linguagem de *design* do modelo de arquitetura for sendo usada, prevemos o aparecimento de linguagens de padrões para domínios específicos. Assim, seria interessante desenvolver um mecanismo que permitisse que os padrões que emergissem fossem acoplados à linguagem de *design*. O acoplamento de um padrão deveria determinar o seu domínio específico, determinar a configuração da linguagem de *design*, desligando as regras que não fossem relevantes, e também permitir o acoplamento de um componente semântico dependente do domínio, ou seja, regras que fizessem inferências sobre a descrição levando o domínio em consideração. Ao permitir o acoplamento deste novo componente semântico, o mecanismo deve garantir que as novas regras não entram em conflito com as regras separáveis de contexto.

Os modelos que propomos permitem a modelagem da interação de um grupo, considerando que esta interação acontece dentro de um ambiente computacional. Seria interessante investigar se estes modelos permitiriam a modelagem da interação de um grupo fora do ambiente computacional. Por exemplo, no LCD Idea Kit (Kit, sd) um professor pode preparar uma atividade de ensino. Esta atividade pode incluir trabalhos em grupo por parte dos alunos. Ao definir este trabalho, o professor tem em mente seu objetivo e a interação desejada entre os alunos. Neste caso, poder-se-ia fornecer ao professor a linguagem de *design* proposta, para que o professor pudesse modelar o grupo desejado e suas interações, e avaliar a qualidade do apoio fornecido e o quão proveitoso os professores iriam considerar este apoio.

Appendix A

Descrição das Regras

Neste apêndice apresentamos a descrição completa de cada uma das regras. Esta descrição consiste do enunciado da regra em linguagem natural, sua explicação, suas classificações e a representação formal da sua verificação. A descrição ou prova das cláusulas utilizadas na representação formal das regras são apresentadas no dicionário de cláusulas na seção A.1.

A representação formal da verificação de uma regra identifica as situações que caracterizam inconsistências em potencial. O termo, ou predicado, do lado esquerdo da seta é o que se deseja provar e os termos do lado direito são as condições que devem ser verdadeiras para que o predicado seja verdadeiro. Um predicado pode ter mais de um conjunto de condições que o torna verdadeiro. Neste caso, cada um destes conjuntos é representado por uma prova distinta do predicado em questão. Quando o caracter '_' aparece em uma prova no lugar de uma das variáveis, isto significa que a condição é verdadeira para qualquer valor daquela variável. A negação de uma cláusula é representada por `não(Cláusula)`.

Segue abaixo a descrição de cada uma das regras:

Regras primárias:

- Regra 1: Para ser capaz de trabalhar em um grupo, o membro deve ou possuir objetos ou ter habilidades comunicativas.
  - Explicação: Quando um membro não pode agir sobre nenhum objeto e nem pode ver, falar, ouvir, mostrar ou conversar sobre estes objetos, então ele não é capaz de participar do grupo.
  - Classificação: Comunicação
  - Representação Formal:
    ```
    Regra1(Membro)→
        não(pode_agir(Membro,QualquerObjeto)),
        não(tem_habilidade(Membro,QualquerObjeto)).
    ```
- Regra 2: O modelo de colaboração de comunidades embutidas em outras comunidades deve sempre ser mais estreito que ou tão estreito quanto o modelo de colaboração das comunidades que as embutem.
  - Explicação: Membros já podem trabalhar tão juntos quanto definido pelo modelo de colaboração da comunidade que embute a outra. Assim, se o modelo de colaboração

da comunidade embutida for menos estreito que o já definido, então ele não significa nada, já que o outro é determinante.

  - Classificação: Colaboração
  - Representação Formal:
    `Regra2(Comunidade)`$\rightarrow$
    `está_embutida(Comunidade,ComunidadeEmbutidora),`
    `tem_modelo_menos_estreito(Comunidade,`
    `ComunidadeEmbutidora).`

- Regra 3: Membros que pertencem a mais de uma comunidade deveriam sempre trabalhar no mesmo modelo de colaboração, ou pelo menos em modelos similares.
  - Explicação: Quando um membro tem que trabalhar em comunidades que têm modelos de colaboração muito diferentes, ele deve comutar entre estes modelos. Estas transições podem ser de difícil adaptação e podem fazer com que o membro ajam inconsistentemente em algum dos modelos e, conseqüentemente, comunidades.
  - Classificação: Colaboração
  - Representação Formal:
    `Regra3(Membro)`$\rightarrow$
    `pertence_várias_comunidades(Membro,Comunidades),`
    `têm_modelos_discrepantes(Comunidades).`

- Regra 4: Usuários que sobrepõem membros do grupo devem trabalhar no mesmo modelo de colaboração, ou pelo menos em modelos similares.
  - Explicação: Quando um usuário acumula (ou sobrepõe) membros do grupo, e com isso é levado a trabalhar em diferentes modelos de colaboração, ele deve comutar entre estes modelos. Estas transições podem ser de difícil adaptação e podem fazer com que o usuário aja inconsistentemente quando estiver representando algum dos membros.
  - Classificação: Colaboração
  - Representação Formal:
    `Regra4(Usuário)`$\rightarrow$
    `sobrepõe_membros(Usuário,Membros),`
    `têm_modelos_discrepantes(Membros).`

- Regra 5: Comunidades que têm o modelo de colaboração de ilha, não devem possuir objetos de tarefa.
  - Explicação: Quando membros trabalham em um modelo de ilha, isto significa que o trabalho deles é completamente independente um do outro e que, conseqüentemente, eles não deveriam compartilhar nenhum objeto relacionado a seus trabalhos.
  - Classificação: Colaboração e independência

  - Representação Formal:
    ```
    Regra5(Comunidade)→
        tem_modelo_de_ilha(Comunidade),
        tem_objetos_de_tarefa(Comunidade,_).
    ```

- Regra 6: Comunidades cujo modelo de colaboração é o de ilha não devem conceder a seus membros a visão dos objetos uns dos outros.

  - Explicação: Quando a comunidade tem a habilidade de ver um objeto, então todos os seus membros podem ver este objeto. Quando estes membros trabalham no modelo de ilha, seus trabalhos são independentes uns dos outros e a habilidade de ver objetos privados uns dos outros lhes dá acesso a informação que não está relacionada com seu trabalho.
  - Classificação: Colaboração, comunicação, e independência.
  - Representação Formal:
    ```
    Regra6(Comunidade)→
        tem_modelo_de_ilha(Comunidade),
        pode_ver(Comunidade,Objeto),
        é_privado(Objeto),
        é_dono(Membro,Objeto),
        pertence_comunidade(Membro,Comunidade).
    ```

- Regra 7: Membros cujo modelo de colaboração mais estreito em que trabalham é o de ilha, não deveriam ser capazes de falar com nenhum outro membro das comunidades a que pertencem.

  - Explicação: Membro sempre trabalha de forma independente, uma vez que o modelo de colaboração mais estreito em que trabalha é o de ilha. Como a habilidade de falar não permite ao ouvinte ser colaborativo, neste caso ela geraria interrupção e distração.
  - Classificação: Colaboração e comunicação.
  - Representação Formal:
    ```
    Regra7(Membro)→
        pertence_várias_comunidades(Membro,Comunidades),
        tem_modelo_de_ilha(Comunidades),
        pode_falar(Membro,_,MembroOuvinte),
        é_colega(Membro,MembroOuvinte).
    ```

- Regra 8: Membros de uma comunidade cujo modelo de colaboração é o de ilha e que têm a habilidade de mostrar objetos uns aos outros, devem ter também a habilidade de conversar uns com os outros sobre aqueles objetos.

  - Explicação: O modelo em que os membros colaboram é o de ilha, o que significa que seus trabalhos são independentes uns dos outros. Assim, apenas mostrar um objeto a um colega pode ser gerar confusão e interrupção. Esta situação pode ser evitada se ao mostrar um objeto, o membro for capaz de explicar ao colega o que ele está mostrando e suas intenções ao mostrá-lo.

  - Classificação: Colaboração e comunicação.
  - Representação Formal:
    ```
    Regra8(Membro)→
        pertence_comunidade(Membro,Comunidade),
        tem_modelo_de_ilha(Comunidade),
        pode_mostrar(Membro,Objeto,MembroObservador),
        pertence_comunidade(MembroObservador,Comunidade),
        não(pode_conversar(Membro,Objeto,MembroObservador)).
    ```

- Regra 9: Comunidades que têm o modelo de colaboração de encaixe (rígido ou nebuloso) não devem possuir objetos de tarefa.
  - Explicação: Quando uma comunidade tem o modelo de colaboração de encaixe, isto significa que os trabalhos de seus membros tem um certo grau de interdependência. No entanto, este modelo não requer que os membros trabalhem juntos em um mesmo objeto. Se este for o caso, então o modelo de colaboração deveria ser o de sobreposição.
  - Classificação: Colaboração.
  - Representação Formal:
    ```
    Regra9(Comunidade)→
        tem_modelo_de_encaixe(Comunidade),
        tem_objetos_de_tarefa(Comunidade,_).
    ```

- Regra 10: Membros de uma comunidade que tem o modelo de colaboração de encaixe nebuloso deveriam poder conversar com pelo menos um outro membro desta comunidade.
  - Explicação: No modelo de encaixe nebuloso membros de uma comunidade precisam negociar a interseção de seus trabalho. Para serem capazes de fazer isto eles precisam conversar.
  - Classificação: Colaboração, comunicação e coordenação.
  - Representação Formal:
    ```
    Regra10(Comunidade)→
        tem_modelo_de_encaixe_nebuloso(Comunidade),
        não(membros_podem_conversar(Comunidade)).
    ```

- Regra 11: Comunidades que têm o modelo de colaboração de encaixe (rígido ou nebuloso) devem possuir objetos de contexto, ou seus membros devem ter contato (mostrar, falar ou conversar) uns com os outros.
  - Explicação: No modelo de encaixe membros que produzem informações que serão usadas por outros devem poder dar indicações sobre estas informações, como por exemplo se elas estão prontas para serem usadas ou não.
  - Classificação: Colaboração, comunicação e coordenação.

  - Representação Formal:
    ```
    Regra11(Comunidade)→
        tem_modelo_de_encaixe(Comunidade),
        não(tem_objetos_de_contexto(Comunidade,_)),
        não(membros_têm_contato(Comunidade)).
    ```

- Regra 12: Comunidades que têm o modelo de colaboração de sobreposição devem possuir pelo menos um objeto de tarefa.

  - Explicação: No modelo de sobreposição membros de uma comunidade trabalham juntos em alguma tarefa. Para que isto seja possível eles precisam compartilhar um objeto de tarefa. Se membros não precisam trabalhar juntos em uma tarefa, então o modelo de colaboração não devia ser o de sobreposição.
  - Classificação: Colaboração.
  - Representação Formal:
    ```
    Regra12(Comunidade)→
        tem_modelo_de_sobreposição(Comunidade),
        não(tem_objetos_de_tarefa(Comunidade,_)).
    ```

- Regra 13: Comunidades que têm o modelo de colaboração de sobreposição devem possuir objetos de contexto.

  - Explicação: No modelo de sobreposição membros de uma comunidade precisam trabalhar juntos em uma tarefa. Para que isto seja possível eles precisam de objetos de contexto para que consigam coordenar seu trabalho.
  - Classificação: Colaboração e coordenação.
  - Representação Formal:
    ```
    Regra13(Comunidade)→
        tem_modelo_de_sobreposição(Comunidade),
        não(tem_objetos_de_contexto(Comunidade,_)).
    ```

- Regra 14: Comunidades que têm o modelo de colaboração coincidente devem possuir objetos de contexto.

  - Explicação: No modelo coincidente os membros da comunidade executam toda a tarefa juntos. Para que isto seja possível eles precisam de objetos de contexto para que consigam coordenar seu trabalho.
  - Classificação: Colaboração e coordenação.
  - Representação Formal:
    ```
    Regra14(Comunidade)→
        tem_modelo_coincidente(Comunidade),
        não(tem_objetos_de_contexto(Comunidade,_)).
    ```

- Regra 15: Comunidades que têm o modelo de colaboração coincidente devem conceder a habilidade de conversar a seus membros.

  - Explicação: Membros têm que trabalhar juntos em um ou mais objetos, e se não puderem conversar, não conseguem coordenar ou planejar suas ações sobre estes objetos, tornando seu trabalho mais difícil.
  - Classificação: Colaboração e coordenação.
  - Representação Formal:
    ```
    Regra15(Comunidade)→
        tem_modelo_coincidente(Comunidade),
        não(membros_podem_conversar(Comunidade)).
    ```

- Regra 16: Membros só devem ter conhecimento do trabalho de ou contato com outros membros da comunidade quando eles são hierarquicamente relacionados.
  - Explicação: Os trabalhos de membros que não são hierarquicamente relacionados uns com os outros não é relacionado. Assim, estes membros não deveriam ter conhecimento ou contato uns com os outros, para que não tenham acesso à informação que não lhes diz respeito e para evitar uma quebra de hierarquia.
  - Classificação: Colaboração, conhecimento e autoridade.
  - Representação Formal:
    ```
    Regra16(Membro)→
        não(membros_relacionados(Membro,OutroMembro)),
        membros_têm_algum_contato(Membro,OutroMembro)).
    ```

- Regra 17: Um membro só deve poder ver objetos pertencentes a um descendente ou ascendente seu, se todos os membros entre eles na hierarquia também o podem.
  - Explicação: A habilidade de ver um objeto concede ao membro conhecimento direto sobre ele. Assim, se um descendente ou ascendente pode ver um objeto que outros membros que estão hierarquicamente mais perto do dono do objeto não podem, pode-se ter uma quebra na hierarquia. Especialmente no caso do descendente, o que significaria que ele teria mais conhecimento (ou pelo menos um conhecimento mais direto) do que seus ascendentes na hierarquia.
  - Classificação: Hierarquia, conhecimento e autoridade.
  - Representação Formal:
    ```
    Regra17(Membro)→
        membros_diretamente_relacionados(Membro,OutroMembro)),
        pode_ver(Membro,Objeto),
        é_dono(OutroMembro,Objeto),
        membros_intermediários(Membro,OutroMembro,
        MembrosIntermediários),
        não(todos_membros_podem_ver(MembrosIntermediários,
        Objeto)).
    ```

- Regra 18: Um membro só deve ter conhecimento indireto sobre um objeto pertencente a ascendente seu, se todos os membros entre eles têm pelo menos conhecimento indireto sobre este objeto.

  - Explicação: Um membro ter conhecimento indireto sobre um objeto significa que este membro poder ser mostrado este objeto, ou pode ouvir ou conversar sobre ele. Se um descendente tem conhecimento indireto sobre objetos de um ascendentes, e os membros entre eles não tem pelo menos este conhecimento (poderiam ter mais conhecimento, como por exemplo um conhecimento direto), então esta situação caracteriza uma quebra de hierarquia.
  - Classificação: Hierarquia, conhecimento e autoridade.
  - Representação Formal:
    `Regra18(Membro)`$\rightarrow$
    `é_descendente(Membro,Ascendente)),`
    `tem_conhecimento_indireto(Membro,Objeto),`
    `é_dono(Ascendente,Objeto),`
    `membros_intermediários(Membro,Ascendente,`
    `MembrosIntermediários),não(`
    `todos_membros_têm_algum_conhecimento(`
    `MembrosIntermediários,Objeto)).`

- Regra 19: Membros de uma comunidade ou subcomunidade que compartilham um objeto devem ter as mesmas habilidades sobre estes objetos.
  - Explicação: Membros de uma (sub)comunidade estão no mesmo nível hierárquico. Assim, se uns têm mais habilidades sobre os objetos que outros, então eles têm mais autoridade sobre os objetos que os outros.
  - Classificação: Hierarquia e autoridade.
  - Representação Formal:
    `Regra19(Comunidade)`$\rightarrow$
    `possui_objetos(Comunidade,Objeto)),`
    Para cada objeto que pertence à comunidade:
    `não(membros_têm_mesmas_habilidades(Comunidade,`
    `Objeto)).`

- Regra 20: A distribuição de habilidades entre os membros de uma sub-hierarquia que compartilham um objeto deve ser a mesma, ou então consistente com sua posição na hierarquia.
  - Explicação: Se alguns do membros de uma sub-hierarquia têm mais habilidades sobre um objeto compartilhado que seus ascendentes, então eles têm maior autoridade sobre os objetos que seus ascendentes. Esta situação caracteriza uma quebra na hierarquia.
  - Classificação: Hierarquia e autoridade.
  - Representação Formal:
    `Regra20(Sub-hierarquia)`$\rightarrow$
    `possui_objetos(Sub-hierarquia,Objeto)),`
    Para cada objeto que pertence à sub-hierarquia:
    `não(membros_têm_habilidades_condizentes(Comunidade,`
    `Objeto)).`

- Regra 21: Membros só devem ser capazes de mostrar e falar sobre seus próprios objetos ou aqueles que pertencem a seus descendentes.
  - Explicação: A habilidade de mostrar ou falar sobre objetos a outros membros, confere ao membro que possui a habilidade alguma autoridade sobre estes objetos e, conseqüentemente, sobre seus donos. Quando um membro pode falar ou mostrar um objeto e não é nem o dono, nem um ascendente do dono, esta autoridade pode ocasionar uma quebra de hierarquia.
  - Classificação: Hierarquia e autoridade.
  - Representação Formal:
    ```
    Regra21(Membro)→
        pode_falar_ou_mostrar(Membro,Objeto)),
        não(é_dono(Membro,Objeto)),
        é_dono(Dono,Objeto),
        não(é_ascendente(Membro,Dono)).
    ```
- Regra 22: Para ser capaz de mostrar um objeto, o membro deve ter conhecimento direto prévio deste objeto.
  - Explicação: Para que o membro saiba que o objeto existe e possa decidir mostrá-lo, ele deve antes poder vê-lo.
  - Classificação: Conhecimento.
  - Representação Formal:
    ```
    Regra22(Membro)→
        pode_mostrar(Membro,Objeto)),
        não(tem_conhecimento_direto(Membro,Objeto)).
    ```
- Regra 23: Para ser capaz de falar sobre um objeto, o membro deve ter algum conhecimento prévio deste objeto.
  - Explicação: Para ser capaz de falar sobre o objeto, o membro deve tê-lo visto ou ouvido falar sobre ele.
  - Classificação: Conhecimento.
  - Representação Formal:
    ```
    Regra23(Membro)→
        pode_falar(Membro,Objeto)),
        não(tem_algum_conhecimento(Membro,Objeto)).
    ```

Regras pendentes:

- Regra 24: Todo papel deve ter pelo menos uma instância.
  - Explicação: Se não existem membros que exerçam este papel, então ele não existe no grupo e, logo, não é necessário.
  - Classificação: Generalidade.

  - Representação Formal:
    ```
    Regra24(Papel)→
        não(existe_membro(Membro,Papel)).
    ```

- Regra 25: Toda classe deve ter pelo menos uma instância.
  - Explicação: Se não existem objetos desta classe, então ela não é usada neste grupo e, logo, não é necessária.
  - Classificação: Generalidade.
  - Representação Formal:
    ```
    Regra25(Papel)→
        não(existe_objeto(Objeto,Classe)).
    ```

- Regra 26: Todo objeto deve pertencer a alguém.
  - Explicação: Se um objeto não pertence a ninguém, então membros não podem agir, ver, mostrar, falar ou conversar sobre. Como o objeto não pode ser usado, ele não é necessário.
  - Classificação: Generalidade.
  - Representação Formal:
    ```
    Regra26(Objeto)→
        não(é_dono(Participante,Objeto)).
    ```

- Regra 27: Uma sobreposição de membros deve envolver dois ou mais membros.
  - Explicação: A sobreposição de membros define que o usuário final especificado assumirá todos os membros desta sobreposição. Assim, uma sobreposição de apenas um membro não significa nada.
  - Classificação: Generalidade.
  - Representação Formal:
    ```
    Regra27(Objeto)→
        sobrepõe_membros(Usuário,ListaDeMembros),
        não(tem_membros_suficientes(ListaDeMembros)).
    ```

Regras de alerta:

- Regra 28: Para participar ativamente em um grupo, o membro deve possuir ao menos um objeto.
  - Explicação: Se um membro não pode agir sobre nenhum objeto, então seu trabalho está limitado às habilidades que ele tem sobre os objetos.
  - Classificação: Generalidade.
  - Representação Formal:
    ```
    Regra28(Membro)→
        não(possui_objetos(Membro,Objetos)).
    ```

- Regra 29: Comunidades que têm o modelo de colaboração de sobreposição devem conceder a habilidade de conversar a seus membros.
  - Explicação: Membros têm que trabalhar juntos em um ou mais objetos, e se não puderem conversar, não conseguem coordenar ou planejar suas ações sobre eles, tornando seu trabalho mais difícil.
  - Classificação: Collaboração, comunicação e coordenação.
  - Representação Formal:
    ```
    Regra29(Comunidade)→
        tem_modelo_de_sobreposição(Comunidade),
        não(membros_podem_conversar(Comunidade)).
    ```

- Regra 30: Membros de um grupo só devem ter conhecimento e contato sobre objetos compartilhados que são compartilhados por seus colegas ou descendentes.
  - Explicação: Quando um membro tem conhecimento ou contato sobte algum objeto compartilhado que pertence a um ascendente seu, ele tem também informação sobre o trabalho de ou contato com outros membros que não apenas têm uma posição mais elevada na hierarquia, mas que também não têm nenhum relacionamento com ele.
  - Classificação: Hierarquia, autoridade e conhecimento.
  - Representação Formal:
    ```
    Regra30(Membro,Objeto)→
        é_compartilhado(Objeto),
        não(é_dono(Membro,Objeto)),
        é_dono(Dono,Objeto),
        é_ascendente(Dono,Membro)
    ,     tem_conhecimento_ou_contato(Membro,Objeto).
    ```

- Regra 31: Membros de um grupo só devem ter conhecimento de objetos compartilhados por sub-hierarquias formadas por seus descendentes.
  - Explicação: Quanto um membro tem conhecimento sobre um objeto compartilhado por uma sub-hierarquia, cujo membro raiz é um colega, ele também tem informação no trabalho dos descendentes deste colega, que não estão relacionados a ele.
  - Classificação: Hierarquia, autoridade e conhecimento.
  - Representação Formal:
    ```
    Regra31(Membro,Objeto)→
        é_compartilhado_subhierarquia(Objeto,Sub-hierarquia),
        não(é_dono(Membro,Objeto)),
        é_dono(Sub-hierarquia,Objeto),
        não(é_ascendente(Membro,Sub-hierarquia)),
        tem_algum_conhecimento(Membro,Objeto).
    ```

- Regra 32: Se membros devem ter conhecimento sobre um objeto, então eles devem ser capazes de ver o objeto.

  - Explicação: Quando o único conhecimento que o membro tem sobre o objeto é indireto, isto significa que alguém que tem conhecimento do objeto lhe falou sobre este objeto. Isto significa que este conhecimento já foi interpretado por alguém antes de chegar nele, e conseqüentemente, sua precisão não pode ser mais garantida.
  - Classificação: Comunicação e conhecimento.
  - Representação Formal:
    ```
    Regra32(Membro)→
        não(tem_conhecimento_direto(Membro,Objeto)),
        tem_conhecimento_indireto(Membro,Objeto).
    ```

- Regra 33: Membros que podem ver um objeto ou podem agir sobre um objeto compartilhado e podem informar outros membros de ações e decisões sobre o objeto podem dar apenas a sua interpretação dos fatos.

  - Explicação: A informação dada pelo membro pode não representar a opinião de (todos) os membros que podem agir sobre o objeto.
  - Classificação: Comunicação.
  - Representação Formal:
    ```
    Regra33(Membro1,Objeto,Membro2)→
        tem_conhecimento_direto(Membro1,Objeto),
        não_é_porta_voz(Membro1,Objeto),
        pode_informar(Membro1,Objeto,Membro2).
    ```

- Regra 34: Quando mais de um membro pode falar sobre um objeto, diferentes visões da informação podem ser contadas.

  - Explicação: Diferentes membros podem falar sobre as ações e decisões nos objetos. Se os membros têm interpretações distintas destes "fatos" então diferentes versões serão contadas.
  - Classificação: Comunicação.
  - Representação Formal:
    ```
    Regra34(Membro1,Membro2,Objeto)→
        pode_contar(Membro1,Objeto,_),
        pode_contar(Membro2,Objeto,_).
    ```

Regras de confirmação:

- Regra 35: Membros de uma comunidade que têm o modelo de colaboração de ilha só devem ter a habilidade de conversar se o *designer* pretende lhes dar a oportunidade de serem espontaneamente colaborativos.

  - Explicação: O trabalho dos membros não é interdependente e não requer que eles conversem. Assim, eles só devem ter a habilidade de conversar, se a intenção for a de permitir a eles decidir se querem colaborar entre si ou não.
  - Classificação: Colaboração, comunicação e independência.

  - Representação Formal:
    ```
    Regra35(Comunidade)→
        tem_modelo_de_ilha(Comunidade),
        membros_podem_conversar(Comunidade).
    ```

- Regra 36: Membros de uma comunidade que tem o modelo de colaboração de encaixe rígido só devem ter a habilidade de conversar se o *designer* pretende lhes dar a oportunidade de serem espontaneamente colaborativos.

  - Explicação: O trabalho dos membros tem uma interseção bem definida e é suficiente que membros tenham informação sobre esta interseção. A habilidade de conversar os permite mais do que a transmissão da informação e deveria ser concedida aos membros apenas se a intenção for a de permitir a eles decidir se querem colaborar mais entre si ou não.
  - Classificação: Colaboração e comunicação.
  - Representação Formal:
    ```
    Regra36(Comunidade)→
        tem_modelo_de_encaixe_rígido(Comunidade),
        membros_podem_conversar(Comunidade).
    ```

- Regra 37: Superiores e ascendentes de um membro que possui um objeto privado, e membro raiz de uma sub-hierarquia que compartilha um objeto e que têm as habilidades de mostrar e falar sobre estes objetos, exercem sua autoridade ao utilizarem-nas.

  - Explicação: A autoridade do superior ou ascendente de um membro é reforçada ao lhe conceder as habilidades de mostrar e falar, ou seja, algum controle, sobre os objetos deste membro. Isto também acontece quando o membro raiz de uma sub-hierarquia possui habilidades de mostrar ou falar sobre os objetos compartilhados pela sub-hieraraquia, e os demais membros não as tem.
  - Classificação: Hierarquia e autoridade.
  - Representação Formal:
    ```
    Regra37(Membro)→
        pode_falar_ou_mostrar(Membro,Objeto),
        não(é_dono(Membro,Objeto)),
        é_dono(Dono,Objeto),
        é_ascendente(Membro,Dono).
    Regra37(Membro)→
        pode_falar_ou_mostrar(Membro,Objeto),
        é_dono(Membro,Objeto),
        é_compartilhado_subhierarquia(Objeto,Sub-hierarquia),
        é_raiz_subhierarquia(Membro,Sub-hierarquia),
        não(pode_falar_ou_mostrar(DemaisMembros,Objeto)).
    ```

Regras de verificação:

- Regra 38: Membros que têm conhecimento sobre um objeto devem ser hierarquicamente superior ao seu dono ou seus trabalhos devem ser influenciado pelo objeto.

  - Explicação: Se o objeto não influencia o trabalho do membro e o membro não é ascendente ao dono do objeto, então esta informação não lhe diz respeito. Esta situação permite que o membro perturbe outros membros com a informação.
  - Classificação: Conhecimento e autoridade.
  - Representação Formal:
    `Regra38(Membro)`$\rightarrow$
    `tem_algum_conhecimento(Membro,Objeto),`
    `não(é_dono(Membro,Objeto)),`
    `é_dono(Dono,Objeto),`
    `não(é_ascendente(Membro,Dono)).`

- Regra 39: Membros que compartilham um objeto e que têm habilidades comunicativas ativas (mostrar, falar ou conversar) sobre ele deveriam entrar em um acordo de quando e como usá-las.
  - Explicação: Se os membros não tem que entrar em um acordo sobre como, quando e a quem mostrar, falar sobre ou conversar sobre o objeto, então um membro pode usar uma destas habilidades sem o conhecimento ou consentimento dos demais membros que podem agir sobre ele.
  - Classificação: Autoridade e comunicação.
  - Representação Formal:
    `Regra39(Comunidade)`$\rightarrow$
    `possui_objetos(Comunidade,Objetos),`
    `pertence_comunidade(Membro,Comunidade),`
    `tem_habilidade_ativa(Membro,Objeto,OutroMembro),`
    `não(pertence_comunidade(OutroMembro,Comunidade)).`

- Regra 40: Membros que só trabalham no modelo de colaboração coincidente, deveriam ter apenas objetos compartilhado ou públicos.
  - Explicação: Membros sempre têm que trabalhar com outros membros para executar a tarefa. Assim, todos seus objetos de tarefa devem ser compartilhados ou públicos. Objetos de tarefa privados podem ser fornecido aos membros para serem usados para rascunhos e outras formas de preparação para a tarefa conjunta.
  - Classificação: Colaboração.
  - Representação Formal:
    `Regra40(Comunidade)`$\rightarrow$
    `modelo_coincidente(Comunidade),`
    `pertence_comunidade(Membro,Comunidade),`
    `tem_objetos_de_tarefa(Membro,ListaObjetos),`
    Para os Objetos da lista: `é_privado(Objeto)).`

- Regra 41: Membros podem ter contato ou interagir com ascendentes ou descendentes se todos os membros entre eles na hierarquia também participa deste contato ou interação.

  - Explicação: Quando um ascendente (descendente) precisa ter contato ou interação com um descendente (ascendente), ele pode criar uma quebra na hierarquia. Para evitar esta quebra a interação ou contato deve ser restrito à participação, ou ao menos ao conhecimento, dos membros entre eles na hierarquia. When an ascendent (descendent) needs to have
  - Classificação: Comunicação, hierarquia e autoridade.
  - Representação Formal:
    ```
    Regra41(Membro1,Membro2)→
        é_ascendente(Membro1,Membro2),
        membros_intermediários(Membro1,Membro2,
          Intermediários),
        não(Intermediários==[]1),
        tem_habilidade_ativa(Membro1,_,Membro2).
    ```

- Regra 42: Membros que podem falar sobre um objeto e não são ascendentes ao seu dono, só devem ser capazes de falar para relatar um problema identificado neste objeto ou algo do genêro.

  - Explicação: Quando um membro fala a outro, o ouvinte não é capaz de respondê-lo. Assim, quando subordinados, descendentes ou colegas podem falar com um membro sobre seu objeto, seu discurso deve estar restrito a relato de problemas identificados e outros discursos que não requerem resposta. Senão esta situação pode caracterizar uma quebra de hierarquia.
  - Classificação: Comunicação, hierarquia e autoridade.
  - Representação Formal:
    ```
    Regra42(Membro1,Membro2)→
        pode_falar(Membro1,Objeto,Membro2),
        é_dono(Membro2,Objeto),
        não(é_ascendente(Membro1,Membro2)).
    ```

- Regra 43: Quando um membro só pode vir a ter conhecimento sobre um objeto se alguém lhe mostrar este objeto, e este membro tiver habilidade de falar ou mostrar este objeto, elas devem estar restritas à condição de ele já ter obtido o conhecimento sobre o objeto.

  - Explicação: As habilidades de mostrar ou falar sobre o objeto devem estar restritas ao membro tê-lo visto previamente, senão o membro teria controle sobre objetos sobre os quais ele, oficialmente, não tem conhecimento.
  - Classificação: Comunicação e conhecimento.
  - Representação Formal:
    ```
    Regra43(Membro,Objeto)→
        não(tem_conhecimento_direto(Membro,Objeto)),
        não(tem_conhecimento_indireto(Membro,Objeto)),
        pode_mostrar(_,Objeto,Membro),
        pode_falar_ou_mostrar(Membro,Objeto).
    ```

[1] [] significa vazio.

- Regra 44: Membros superiores e ascendentes a um membro que possui um objeto e membro raiz de uma sub-hierarquia que compartilha um objeto podem exercer sua autoridade através de conversas sobre o objeto das quais o dono ou demais donos não participam.

  – Explicação: Se o dono(s) de um objeto não participa de uma conversa que seu superior ou ascendente tem sobre este objeto, então a autoridade deste ascendente é reforçada. Afinal ele é capaz de conversar sobre objetos que pertencem a um descendente seu sem a participação, e até conhecimento, deste.

  – Classificação: Comunicação, hierarquia e autoridade.

  – Representação Formal:

```
Regra44(Membro,Objeto)→
    não(é_dono(Membro,Objeto)),
    é_dono(Dono,Objeto),
    é_ascendente(Membro,Dono),
    pode_conversar(Membro,Objeto,OutroMembro),
    não(OutroMembro == Dono).
Regra44(Membro,Objeto)→
    é_raiz_subhierarquia(Membro,Sub-hierarquia)),
    é_compartilhado_subhierarquia(Objeto,Sub-hierarquia),
    pode_conversar(Membro,Objeto,OutroMembro),
    não(é_dono(OutroMembro,Objeto)).
```

- Regra 45: Comunidade que tem o modelo de colaboração de ilha deve possuir apenas objetos de contexto que se refiram ao grupo como um todo, e não aos trabalhos dos membros.

  – Explicação: Trabalhos dos membros da comunidade são independentes uns dos outros. Assim, estes membros deveriam compartilhar objetos de contexto se estes transmitirem informações gerais sobre o grupo

  – Classificação: Collaboração.

  – Representação Formal:

```
Regra45(Comunidade)→
    tem_modelo_de_ilha(Comunidade),
    tem_objetos_de_contexto(Comunidade).
```

- Regra 46: Membros devem ser capazes de conversar apenas sobre objetos que eles possuem, que pertencem a seus descendentes ou quando o dono ou ascendente do dono participam da conversa.

  – Explicação: Membros podem ter conversas sobre objetos que não estão relacionados com seu trabalho. Além disso, nem o dono do objeto, nem um de seus ascendentes participa da conversa. Esta situação pode levar membros a serem "intrometidos" e pode criar uma quebra de hierarquia.

  – Classificação: Comunicação e autoridade.

– Representação Formal:
```
Regra46(Membro)→
    pode_conversar(Membro,Objeto,OutroMembro),
    não(é_dono(Membro,Objeto),
    é_dono(Dono,Objeto),
    não(Dono == OutroMembro),
    não(é_ascendente(OutroMembro,Dono)).
```

A.1 Dicionário de Cláusulas

As variáveis que aparecem nas cláusulas indicam que tipo de elemento do grupo está sendo verificado. A variável "Participante" significa que o elemento pode tanto ser um membro, quanto uma comunidade.

Cláusulas básicas:

`é_compartilhado(Objeto)`: Verifica se Objeto é compartilhado.

`é_compartilhado_subhierarquia(Objeto,Sub-hierarquia)`: Verifica se Objeto é compartilhado por Sub-hierarquia.

`é_dono(Participante,Objeto)`: Verifica se Participante é dono do Objeto.

`é_privado(Objeto)`: Verifica se Objeto é privado.

`é_raiz_subhierarquia(Membro,Sub-hierarquia)`: Verifica se Membro é raiz da Sub-hierarquia.

`é_superior(Subordinado,Superior)`: Verifica se o membro Subordinado é diretamente subordinado ao membro Superior na hierarquia.

`está_embutida(Comunidade1,Comunidade2)`: Verifica se Comunidade1 está embutida em Comunidade2.

`existe_membro(Membro,Papel)`: Verifica se Membro é do tipo Papel.

`existe_objeto(Objeto,Classe)`: Verifica se Objeto é do tipo Classe.

`grau_de_estreiteza(Comunidade,Grau)`: Verifica se o grau de estreiteza[2] do modelo de colaboração da Comunidade é Grau.

`habilidades_condizentes(Membro1,ListaHabs1,Membro2, ListaHabs2)`: Verifica se o membro que é ascendente sobre o outro tem mais habilidades que o outro.

[2]O grau de estreiteza de um modelo de colaboração é um número inteiro fixo, e quanto mais estreita a colaboração entre os membros, maior o grau.

`habilidades_sobre_objeto(Membro,Objeto,ListaHabilidades)`: Gera a lista de habilidades que Membro possui sobre Objeto.

`membros_relacionados(Membro1,Membro2)`: Verifica se Membro1 e Membro2 estão hierarquicamente relacionados, ou seja, um é ascendente, descendente ou colega do outro.

`membros_diretamente_relacionados(Membro1,Membro2)`: Verifica se Membro1 e Membro2 estão diretamente relacionados na hierarquia, ou seja, um é ascendente ou descendente do outro.

`membros_intermediários(Membro1,Membro2,ListaMembros)`: Gera a lista de membros que estão entre Membro1 e Membro2 na hierarquia.

`pode_agir(Participante,Objeto)`: Verifica se Participante pode agir sobre Objeto.

`pode_falar(Informante,Objeto,Informado)`: Verifica se o participante Informante pode falar sobre Objeto ao participante Informado.

`pode_conversar(Informante,Objeto,Informado)`: Verifica se o participante Informante pode conversar sobre Objeto com o participante Informado.

`pode_mostrar(Expositor,Objeto,Observador)`: Verifica se o participante Expositor pode mostrar o Objeto ao participante Observador.

`pode_ver(Participante,Objeto)`: Verifica se Participante tem habilidade de ver Objeto.

`pertence_comunidade(Membro,Comunidade)`: Verifica se Membro pertence à Comunidade.

`pertence_várias_comunidades(Membro,Comunidades)`: Verifica se Membro pertence a várias Comunidades.

`sobrepõe_membros(Usuário,Membros)`: Verifica se Usuário é uma sobreposição de Membros.

`tem_membros_suficientes(ListaDeMembros)`: Verifica se a lista de membros tem mais de um membro.

`tem_modelo_coincidente(Comunidade)`: Verifica se Comunidade tem o modelo de colaboração coincidente.

`tem_modelo_de_encaixe(Comunidade)`: Verifica se Comunidade tem o modelo de colaboração de encaixe.

`tem_modelo_de_encaixe_nebuloso(Comunidade)`: Verifica se Comunidade tem o modelo de colaboração de encaixe nebuloso.

`tem_modelo_de_encaixe_rígido(Comunidade)`: Verifica se Comunidade tem o modelo de colaboração de encaixe rígido.

`tem_modelo_de_ilha(Comunidade)`: Verifica se Comunidade tem o modelo de colaboração de ilha.

`tem_modelo_de_ilha(ListaDeComunidades)`: Verifica se todas as Comunidades da lista tem o modelo de colaboração de ilha.

`tem_modelo_de_sobreposição(Comunidade)`: Verifica se Comunidade tem o modelo de colaboração de sobreposição.

`tem_objetos_de_tarefa(Participante,Objetos)`: Verifica se Participante possui Objetos de tarefa.

`tem_objetos_de_contexto(Participante,Objetos)`: Verifica se Participante possui Objetos de contexto.

Provas intermediárias:

```
é_colega(Membro1,Membro2)→
    é_superior(Membro1,Superior),
    é_superior(Membro2,Superior).

é_descendente(Descendente,Ascendente)→
    é_superior(Ascendente,Descendente).
é_descendente(Descendente,Ascendente)→
    é_superior(Superior,Descendente),
    é_descendente(Superior,Ascendente).

membros_podem_conversar(Comunidade)→
    pertence_comunidade(Membro,Comunidade),
    pode_conversar(Membro,_,OutroMembro),
    pertence_comunidade(OutroMembro,Comunidade).

membros_podem_falar(Comunidade)→
    pertence_comunidade(Membro,Comunidade),
    pode_falar(Membro,_,OutroMembro),
    pertence_comunidade(OutroMembro,Comunidade).

membros_podem_mostrar(Comunidade)→
    pertence_comunidade(Membro,Comunidade),
    pode_mostrar(Membro,_,OutroMembro),
    pertence_comunidade(OutroMembro,Comunidade).
```

```
membros_têm_algum_contato(Membro1,Membro2)→
    pode_conversar(Membro1,_,Membro2).

membros_têm_algum_contato(Membro1,Membro2)→
    pode_falar(Membro1,_,Membro2).

membros_têm_algum_contato(Membro1,Membro2)→
    pode_falar(Membro2,_,Membro1).

membros_têm_algum_contato(Membro1,Membro2)→
    pode_mostrar(Membro1,_,Membro2).

membros_têm_algum_contato(Membro1,Membro2)→
    pode_mostrar(Membro2,_,Membro1).

membros_têm_algum_contato(Membro1,Membro2)→
    pode_ver(Membro1,Objeto)
    é_dono(Membro2,Objeto).

membros_têm_algum_contato(Membro1,Membro2)→
    pode_ver(Membro2,Objeto)
    é_dono(Membro1,Objeto).

membros_têm_contato(Comunidade)→
    membros_podem_conversar(Comunidade).

membros_têm_contato(Comunidade)→
    membros_podem_falar(Comunidade).

membros_têm_contato(Comunidade)→
    membros_podem_mostrar(Comunidade).

membros_têm_mesmas_habilidades(Comunidade,Objeto)→
    Para todos pares de membros da comunidade:
       habilidades_sobre_objeto(Membro1,Objeto,ListaHabs1),
       habilidades_sobre_objeto(Membro2,Objeto,ListaHabs2),
       ListaHabs1 == ListaHabs2.

membros_têm_habilidades_condizentes(Sub-hierarquia,Objeto)→
    membros_têm_mesmas_habilidades(Comunidade,Objeto).

membros_têm_habilidades_condizentes(Sub-hierarquia,Objeto)→
    Para todos pares de membros da sub-hierarquia:
       membros_relacionados(Membro1,Membro2),
       habilidades_sobre_objeto(Membro1,Objeto,ListaHabs1),
       habilidades_sobre_objeto(Membro2,Objeto,ListaHabs2),
       habilidades_condizentes(Membro1,ListaHabs1,
            Membro2,ListaHabs2).

não_é_porta_voz(Participante,Objeto)→
    não(é_dono(Participante,Objeto)).
```

não_é_porta_voz(Participante,Objeto)→
    é_dono(Participante,Objeto),
    é_compartilhado(Objeto).

par_tem_modelos_discrepantes(Comunidade1,Comunidade2)→
    grau_de_estreiteza(Comunidade1,Grau1),
    grau_de_estreiteza(Comunidade2,Grau2),
    |Grau1-Grau2| > 1.

pode_contar(Informante,Objeto,Informado)→
    pode_falar(Informante,Objeto,Informado).

pode_contar(Informante,Objeto,Informado)→
    pode_conversar(Informante,Objeto,Informado).

pode_informar(Informante,Objeto,Informado)→
    pode_falar(Informante,Objeto,Informado),
    não(é_dono(Informado,Objeto)).

pode_informar(Informante,Objeto,Informado)→
    pode_conversar(Informante,Objeto,Informado),
    não(é_dono(Informado,Objeto)).

pode_falar_ou_mostrar(Membro,Objeto)→
    pode_falar(Membro,Objeto,_).

pode_falar_ou_mostrar(Membro,Objeto)→
    pode_mostrar(Membro,Objeto,_).

pode_falar_ou_mostrar(ListaMembros,Objeto)→
    Para todo membro da Lista de Membros:
        pode_falar_ou_mostrar(Membro,Objeto).

possui_objetos(Participante,Objetos)→
    tem_objetos_de_tarefa(Participante,Objetos).

possui_objetos(Participante,Objetos)→
    tem_objetos_de_contexto(Participante,Objetos).

tem_algum_conhecimento(Participante,Objeto)→
    tem_conhecimento_direto(Participante,Objeto).

tem_algum_conhecimento(Participante,Objeto)→
    tem_conhecimento_indireto(Participante,Objeto).

tem_conhecimento_direto(Participante,Objeto)→
    pode_ver(Participante,Objeto).

tem_conhecimento_direto(Participante,Objeto)→
    pode_agir(Participante,Objeto).

tem_conhecimento_indireto(Participante,Objeto)→
    pode_mostrar(_,Objeto,Participante).

tem_conhecimento_indireto(Participante,Objeto)→
    pode_falar(_,Objeto,Participante).

tem_conhecimento_indireto(Participante,Objeto)→
    pode_conversar(Participante,Objeto,_).

tem_conhecimento_ou_contato(Membro,Objeto)→
    tem_algum_conhecimento(Membro,Objeto).

tem_conhecimento_ou_contato(Membro,Objeto)→
    é_dono(Comunidade[3],Objeto),
    Para todos os membros da comunidade que é dona do Objeto:
        membros_têm_algum_contato(Membro,MembroComunidade).

tem_habilidade(Participante,Objeto)→
    pode_mostrar(Participante,Objeto,_).

tem_habilidade(Participante,Objeto)→
    pode_ver(Participante,Objeto).

tem_habilidade(Participante,Objeto)→
    pode_falar(Participante,Objeto,_).

tem_habilidade(Participante,Objeto)→
    pode_conversar(Participante,Objeto,_).

tem_habilidade_ativa(Membro1,Objeto,Membro2))→
    pode_mostrar(Membro1,Objeto,Membro2).

tem_habilidade_ativa(Membro1,Objeto,Membro2))→
    pode_falar(Membro1,Objeto,Membro2).

tem_habilidade_ativa(Membro1,Objeto,Membro2))→
    pode_conversar(Membro1,Objeto,Membro2).

tem_modelo_menos_estreito(Comunidade1,Comunidade2)→
    grau_de_estreiteza(Comunidade1,Grau1),
    grau_de_estreiteza(Comunidade2,Grau2),
    Grau1 < Grau2.

têm_modelos_discrepantes(Participantes)→
    Para algum par de participantes:
        par_tem_modelos_discrepantes(Participante1,Participante2).

[3]Como objeto é compartilhado, ele deve pertencer a uma comunidade ou sub-hierarquia. A regra vale para ambos.

`todos_membros_podem_ver(ListaMembros,Objeto)`→

Para todo membro da lista:

`pode_ver(Membro,Objeto).`

`todos_membros_têm_algum_conhecimento(ListaMembros,Objeto)`→

Para todo membro da lista:

`tem_algum_conhecimento(Membro,Objeto).`

Appendix B

Relatório Gerado pelo Protótipo para Teste do SERG

Neste apêndice mostramos extratos do relatório gerado pelo protótipo da implementação para o teste de modelagem da *Intranet* do SERG. A seleção foi feita para mostrar como o relatório descreve os diversos elementos do grupo definidos pelo *designer* e as relações entre eles.

```
************************************************************
REPORT FOR ERA #1
************************************************************

The roles that have been defined are:
Role: pesq jr - Description: participante do grupo que nao e
      doutor
Role: peao - Description: participante do grupo que trabalha
      no desenvolvimento de um projeto
Role: coord proj - Description: pesquisador que coordena um
      projeto
Role: coord geral - Description: pessoa que e responsavel pelo
      site e autoriza entrada de novos membros e responsavel
      pela manutencao do calendario geral
Role: aluno - Description: participantes do grupo que
      desenvolvem trabalho academico ligados a um programa
      academico
Role: orientador - Description: pesquisador doutor que orienta
      trabalho academico
Role: visitante - Description: pessoas que tem acesso
      exclusivamente de observacao ao site e que nao sao
      participantes do grupo
Role: pesq sr - Description: participante do grupo que e doutor

============================================================
```

```
The classes that have been defined are:
Class: trab publ - Description: trabalhos publicados
Class: trab des - Description: trabalhos em desenvolvimento
Class: agenda - Description: agenda de tarefas e eventos
Class: visao geral - Description: descricao geral de um projeto
Class: contrato - Description: contrato de um projeto
Class: modulos - Description: modulos de implementacao do
       sistema
Class: documentos - Description: modulos de documentacao do
       sistema
Class: info pessoal - Description: informacoes pessoais sobre
       os participantes do grupo
Class: biblioteca - Description: url referencias e livros
       virtuais
Class: info geral - Description: informacoes gerais para
       participantes do grupo
Class: acesso - Description: informa se objeto e compartilhado
       se tem alguem acessando e quem
Class: versao - Description: informa qual versao do trabalho e
       aquela
Class: estagio - Description: estagio de desenvolvimento do
       trabalho que uma pessoa se encontra
Class: presenca - Description: indica quem esta presente

================================================================

The hierarchy of the group modelled is shown below.
Notice that each member is indented in relation to its parent.
Hierarchy:
coord susema
  peao gilda
  peao cecilia
coord linx
  peao laura
    peao denise
  peao vio
  peao carm
orient ling
  al lg cecilia
orient es
  al flavio
  al gilda
  al cecilia
  al sergio
  al simone
```

```
  al deller
  al denise
  al jair
  al raquel
coord site
  visit
  pj flavio
  pj gilda
  pj cecilia
  pj sergio
  pj simone
  pj deller
  pj denise
  pj jair
  pj raquel
  ps vio
  ps carm
  ps laura
  ps isa
  ps css

================================================================

Next the members that have been declared are described, as well
as the objects they can act upon, their abilities, the
communities they belong to, their superior and their
subordinates.

Member: visit - Role: visitante - Type: population
 -> Superior: coord site
 -> visit has no subordinates.
 -> Communities that visit belongs to: [coord site com]
 -> visit does not own any objects.

...

Member: coord linx - Role: coord proj - Type: individual
 -> coord linx is at the top of the hierarchy of the group and
    therefore, has no superior.
 -> Subordinates: [peao laura, peao vio, peao carm]
 -> Communities that coord linx belongs to: [top community]
 -> coord linx can act upon the following objects:
      Private:[est linx, contrato linx, visao linx, ag proj linx]
      Shared :[ac dc2 linx, ac dc1 linx, vr st linx, ac sb4 linx,
               ac sb3 linx, ac sb2 linx, ac sb1 linx, ac st linx,
```

```
                sistema linx]
 -> Objects that belong to other members that coord linx can see:
      [doc2 linx, doc1 linx, subs4 linx, subs3 linx, subs2 linx,
       subs1 linx]
 -> Objects that coord linx can speak about: [subs1 linx,
      subs2 linx, subs3 linx, doc1 linx, doc2 linx]
 -> Objects that coord linx can have conversation about:
      [ag proj linx, visao linx, sistema linx, subs1 linx,
      subs2 linx, subs3 linx, doc1 linx, doc2 linx, est linx]
...

============================================================

Next the communities that have been declared (explicitly or
automatically) are described, as well as their superior (unless
it is a free-community, which may have members that belong to
different hierarchical levels), the objects they can act upon,
 and their abilities. Notice that when a community is able to
act upon, see, be shown, listen about or have a conversation
about an object, what it actually means is that its members
inherit that ability.

Community: subc pesq - Type: Subcommunity - Members: [pj flavio,
 pj gilda, pj cecilia, pj sergio, pj simone, pj deller,
 pj denise, pj jair, pj raquel, ps vio, ps carm, ps laura,
 ps isa, ps css]
 -> Superior: coord site
 -> subc pesq cannot act upon any objects.
 -> Objects that subc pesq may be shown during execution: [art1,
    art2, art4, art3]
 -> Objects that subc pesq can have conversation about:
    [tese raquel, tese jair, tese denise, tese deller,
    tese simone, tese sergio, tese flavio, prop gilda,
    qual cecilia, art1, art2, art4, art3]
...

============================================================

Next the objects that have been declared are described, as well
as their ownership status and the abilities members and
communities have on them.

Object: ac est cecilia - Class: acesso - Type: context
 -> Ownership Status: shared by community com orient cecilia
     Members that can act upon it are: [orient es, orient ling]
```

```
...

Object: est linx - Class: estagio - Type: context
 -> Ownership Status: private - Owner: [coord linx]
 -> Objects can be seen by:[peao laura com, coord linx com]
 -> Objects can be talked about by: [coord linx, coord linx com]

...

================================================================

Next the seeing, showing, speaking and talking abilities that
have been declared are described. It is presented who has the
ability in respect to which object and which other member and
the constraints that are applied on them.

Seeing ability:
Viewer: coord site com - Type: Community - Object: art es
 -> Ownership Status: private - Owner: [ps css]

...

Showing ability:
Shower: pj cecilia - Object: art3 - Potential Viewer:
        subc pesq - Type: Community
 -> Constraint: com concordancia de pj simone e pj sergio
 -> Ownership Status: shared by community subc art3
      Members that can act upon it are: [pj simone, pj sergio,
      pj cecilia]

...

Speaking ability:
Speaker: coord susema - Object: doc susema -
Listener: coord susema com - Type: Community
 -> Ownership Status: shared by community coord susema com
    Members that can act upon it are: [peao gilda, peao cecilia]

...

Talking ability:
Talker1: com orient cecilia - Type: Community -
Talker2: com orient cecilia - Type: Community -
Object: est cecilia
 -> Ownership Status: shared by community com orient cecilia
```

```
      Members that can act upon it are: [orient es, orient ling]

...

================================================================

Next are listed all the consistency rules that have been checked.
And those that have been broken, are stated followed by their
rationale and by each breaking instance. The breaking instances
are succeeded by the designers explanation for breaking it (in
the cases in which it has been given).

...

RULE 5: Community that has the island model as collaboration
model should not own any task objects.

EXPLANATION: When members work in the island model, it means that
their works are completely independent from each other and
therefore, they should not share any objects related to their
works.

CHECKED PARAMETERS: [Community]

   POTENTIAL INCONSISTENCY:
   Instance that broke the rule: [subc art5]

   EXPLANATION FOR INSTANCE: A tarefa (escrita de artigo) ja
                             esta encerrada).

...

================================================================

****************************************************************
REPORT FOR ERA #2
****************************************************************

The following explicit changes have been defined for this era.
Constraints may or may not have been defined for each one of
these changes.
-> Member pj deller role has become pesq ext.
-> Object tese denise class has become trab publ.
-> Object tese jair class has become trab publ.
-> Object tese raquel class has become trab publ.
```

-> Member pj denise role has become pesq sr.
-> Member pj jair role has become pesq sr.
-> Member pj raquel role has become pesq sr.
-> Object qual cecilia class has become trab publ.
-> Object prop gilda has changed its ownership status. Its status
   now is subcom and belongs to subc prop.
-> Object calendario has changed its ownership status. Its status
   now is public and belongs to all members.
-> Member peao gilda has changed his/her position in the
   hierarchy. His/Her new superior now is peao carm.
-> Member peao cecilia has changed his/her position in the
   hierarchy. His/Her new superior now is peao carm.
-> Object prop gilda class has become trab publ.

Changes to the structure of the model described in the previous era may have been defined. Next the model defined for this era is described.

==========

...

Appendix C

Manual do Usuário do Protótipo

Model:

`new_model(ModelName)` Creates a new model. To do so, checks if the model name already exists or not.

`load_era(Era)` Loads the defined era of the current model.

`load_model(ModelName)` Loads a model from a File.

`write_report` Writes report of loaded model.

`save_model` Saves all database and definitions for the current era in a file.

Roles and Classes:

`list_roles` Lists all existent entries in db_role.

`role(Role,Description)` If entry (Role,Description) does not already exist in db_role, inserts it.

`delete_role(Role)` Deletes type Role from the database.

`clearall_roles` Deletes all roles from the database.

`list_classes` Lists all existent entries in db_class.

`class(Class,Description)` If entry (Class,Description) does not already exist in db_class, inserts it.

`delete_class(Class)` Deletes type Class from the database.

`clearall_classes` Deletes all classes from the database.

Members:

`list_members` Lists all existent entries in db_member.

list_member_type(Type) Lists all existent entries in db_member of a specific type.

list_individuals Lists all existent individuals in db_member.

list_subgroups Lists all existent subgroups in db_member.

list_populations Lists all existent populations in db_member.

list_role_instances(Role) Lists all instances of a Role in the database (db_member).

individual(Name,Role) Creates an instance of an individual.

population(Name,Role) Creates an instance of an individual.

subgroup(Name,Role,Qty) Creates an instance of an individual.

individuals(Num,Name,Role) Creates n instances of an individual.

subgroups(Num,Name,Role,Qty) Creates n instances of a subgroup.

populations(Num,Name,Role) Creates n instances of a population.

delete_member(Member) Deletes a member from the database (db_member). Member must be free, that is, at top level, without children or objects, in order to be deleted.

Objects:

list_objects Lists all existent entries in db_obj.

list_owners Lists all existent entries in db_owner.

list_owned_objs Lists all existent entries in db_owned.

list_shared Lists all the objects shared by members related by subtree or community. Entries in db_shared.

list_comm_owners Lists all community that own objects and the objects they own, that is, all existent entries in db_comm_owner.

list_hier_owners Lists all member that are subhierarcy roots and owners of objects shared by subhierarchy, that is, all existent entries in db_hier_owner.

object(Type,Name,Class) Creates an object Name, instance of Class.

objects(Num,Type,Name,Class) Creates n instances of an object.

private(Obj,Member) Defines Obj as being a private object of Member.

shared(Type,Obj,ListMembers) Defines Obj as being a shared object among ListMembers. Type can be free, composed, subcom or subhier.

shared(Type,Obj,SubTreeRoot,Levels) Creates a shared object in a hierarchical subtree of height Levels. Type can be free, composed, subcom or subhier.

public(Obj) Defines Obj as being a public object, that is, shared by all_members.

free_object(Obj) Frees an object from all members that own it.

delete_object(Obj) If object doesn't belong to anyone, it can be deleted.

free_owner(Owner) Frees the owner from any objects that he is an owner of. He can't be freed of public objects or shared objects that belong to communities or subhierarchies he is not the root of. Owner can be a member or a (sub)community.

delete_ownership(Owner,Obj) Deletes the ownership relation between Owner and Obj.

Hierarchy:

superior(Sup,Sub) Defines Sup as being superior (parent) to Sub.

subordinate(Sub,Sup) Defines Sub as being subordinate (child) to Sup.

free_member(Member) Frees member, by taking it out from hierarchy (actually to the top level). To be eligible for being freed, member must have no subordinates or objects.

hierarchy Prints the hierarchical tree (textually).

Communities:

list_communities Lists all existent communities and subcommunities.

list_belongs Lists all (sub)communities a member belongs to.

list_embedded Lists all (sub)communities a subcommunity is embedded in.

list_embedding Lists all subcommunities a (sub)community embeds.

subcommunity(Name,ListMembers) Creates subcommunity Name containing members ListMembers. A subcommunity is always embedded in its parent community.

free_community(Name,ListMembers) Creates free-community Name containing members ListMembers.

include_in_comm(NewMember,Comm) Includes a member in an existent subcommunity or free_community.

embed(SubCom,SupCom) Embeds one subcommunity into another. This allows the system to check the relation between embedded subcommunities and their collaboration model.

`delete_community(Subcomm)` Deletes a subcommunity or free_community. Communities cannot be deleted.

`remove_from_comm(Member,Subcomm)` Removes a member from a subcommunity or free_community.

`remove_from_all_subcomms(Member)` Removes a member from all subcommunities he belongs to.

`remove_from_all_freecomms(Member)` Removes a member from all subcommunities he belongs to.

`remove_from_def_comms(Member)` Removes a member from all communities that have been defined by the designer, that is, subcommunities and free-communities.

`delete_embed(SubCom,SupCom)` Allows to delete embedding relation between two subcommunities. A subcommunity is always embedded in a community.

Overlap:

`list_overlaps` Lists all overlaps and their Members.

`list_overlapped` Lists all overlapped members and the overlap they belong to.

`overlap(Name,ListMembers)` Creates an overlap Name of ListMembers, if it doesn't exist. If it already does, just includes the members (and warns designer). Overlap means that all members are facets of a same final user.

`delete_from_overlap(Member)` Removes a member from its overlap.

`delete_overlap(Name)` Removes overlap Name.

Collaboration Models:

`list_collabs` Lists all collaboration models for all communities.

`tightness` Lists all collaboration model and their degrees of tightness.

`collab_model(Comm,Collab)` Changes community Comm's collaboration model. If it were the island_default it is ok, if it had already another one defined, it changes but tells designer what was done.

Communicative Abilities:

`list_sees` Lists all the objects that can be seen and by whom, and the constraints that apply on it.

list_seen Lists all the objects that can be seen and the list of viewers that can see each object.

list_viewers Lists all the viewers tha can see an object and the list of objects each one can see.

list_seeing Lists all database related to seeing.

see(Viewer,Obj) Allow member to see an object without constraints.

see(Viewer,Obj,Constraint) Allow member to see an object, but constraints apply.

constrain_see(Viewer,Obj,NewConstraint) Allows to include or change the constraint to an ability to see.

delete_see(Viewer,Obj) Deletes Viewer's ability to see Object.

delete_seeing_obj(Obj) Deletes all Viewers' ability to see Object.

delete_seeing(Participant) Deletes all Participant's (member or community) abilities to see any objects.

list_shows Lists which members can show, which objects to whom and the constraints that apply to their ability to show.

list_showers Lists which members can show which objects.

list_shown Lists objects that are allowed to be shown by members, and who can show them.

list_pot_viewers Lists which members are potential viewers of which objects.

list_showing Lists all database related to showing

show(Shower,Obj,Viewer) Creates the ability of Shower deciding (or not) to show Obj to the list of potential viewers. Constraints may apply on this ability.

show(Shower,Obj,Viewer,Constraint) Creates the ability of Shower deciding (or not) to show Obj to the list of potential viewers, but constraints apply.

constrain_show(Shower,Obj,Viewer,NewConstraint) Allows to include or change the constraint to an ability to show. Don't need to enter the list of viewers, since there can only be an entry for each pair (Shower,Obj).

delete_show(Shower,Obj,Viewer) Deletes a Shower's ability to show an Object to a Viewer.

delete_showing_obj(Obj) Deltes all Showers' abilities to show Object to Potential Viewers.

delete_showing(Member) Deletes Shower's abilities to show any object to any Potential Viewers.

delete_being_shown(Participant) Deletes Participant's (member or community) abilities to being shown (being a Potential Viewer) any object.

list_speaks Lists all talks that can go on between a speaker and a listener.

list_talks Lists all conversations that can go on about an object.

list_talkers Lists all members that can speak about or have a conversation about an object.

list_listeners Lists all members that can listen someone talk about an object.

list_talked_objs Lists all objects that can be talked about, and what kind of talk members can have about it (talk or conversation).

list_listened_objs List for each object, everyone that can listen someone else talk about it.

list_speaking Lists all database related to speaking.

list_talking Lists all database related to having conversations.

speak(Speaker,Obj,Listener) Creates the ability of Speaker talking to Listener (who cannot answer) about an object, without any constraints.

speak(Speaker,Obj,Listener,Constraint) Creates the ability of Speaker talking to Listener (who cannot answer) about an object, but constraints apply.

constrain_speak(Speaker,Obj,Listener,NewConstraint) Allows to include or change the constraint to an ability to speak.

talk(Talker1,Talker2,Obj) Creates the ability of Talker1 and Talker2 having a conversation about an Object, without constraints.

talk(Talker1,Talker2,Obj,Constraint) Creates the ability of a group of members of talking about an Object among themselves, and constraints apply.

constrain_talk(Talker1,Talker2,Obj,NewConstraint) Allows to include or change the constraint to an ability to talk.

delete_speak(Speaker,Obj,Listener) Deletes Speaker's ability to speak to Listener about Object.

delete_talk(Talker1,Talker2,Obj) Deletes Talkers' ability to have a conversation about Object. Type checks are done just to be able to give the correct error message.

delete_speaking_obj(Obj) Delete all speakers' abilities of speaking about Object to listeners.

delete_speaking(Member) Deletes Speaker's abilities to speak about any object to any listener.

`delete_listening(Participant)` Deletes all participant's (member or community) abilities to listen about any object.

`delete_talking_obj(Obj)` Deletes all talkers' abilities of having a conversation about Object.

`delete_talking(Participant)` Deletes all participant's (member or community) abilities to have a conversation.

`delete_abilities_obj(Obj)` Deletes all abilities that members or (sub)communities have over an object.

`delete_abilities(Participant)` Delets all participant's (member or community) abilities.

Changes in time:

`list_changes` Lists all changes that have been defined for all eras.

`new_era` Defines a new era in the model.

`change_member_role(Member,NewRole,Constraint)` Allows a member to change roles in the current era.

`change_obj_class(Obj,NewClass,Constraint)` Allows a member to change roles in the current era.

`change_member_type(Member,NewType,NewQty,Constraint)` Allows a member to change types (valid types are individual, subgroup or population) in the current era. NewQty only has to be defined when NewType is subgroup.

`change_ownership(Object,NewStatus,NewOwner,Levels,Constraint)` Allows change on object's ownership. Levels only have to be defined when the object's NewStatus is subhier. As for NewOwner, it should not be defined when object's NewStatus is public.

`change_superior(Member,NewSuperior,Constraint)` Allows designer to change a Member's superior in a new era.

Rules Checking:

`check_rule(Rule)` Checks one specific rule requested by designer.

`check_pi_rules` Checks consistency for potential inconsistency rules.

`check_bi_rules` Checks consistency for beware inconsistency rules.

`check_dd_rules` Checks consistency for dangling declaration rules.

`check_fm_rules` Checks consistency for feedback warning rules.

`check_ci_rules` Checks consistency for potential inconsistency rules.

`check_all_rules` Checks all consistency rules.

`clearall_db` Clears all databases.

References

Ackerman, M. and Starr, B. (1996). Social activity indicators for groupware. *IEEE Computer*, June:37–42.

ADDLabs (*s.d.*). Internet Location: http://www.inf.puc-rio.br/ addlabs/index.html.

Adler, P. S. and Winograd, T. A. (1992). *Usability: Turning Technologies into Tools*. Oxford University Press, Inc.

Andersen, P. B., Holmqvist, B., and Jensen, J. F. (1993). *The Computer as Medium*. Cambridge University Press.

AulaNet (s.d.). Internet Location: http://aulanet.les.inf.puc-rio.br/aulanet/index.html.

Barbosa, S. D. J., da Cunha, C. K. V., and da Silva, S. R. P. (1998). Knowledge and communication perspectives in extensible applications. In *IHC'98*. (To appear in.).

Bayle, E., Bellamy, R., Casaday, G., and Erickson, T. (1998). Putting it all together: Towards a pattern language for interaction design - A CHI 97 workshop. *SIG***CHI** *Bulletin*, 30(1):20–26.

Bleser, T. W. and Sibert, J. (1990). Toto: A tool for selecting interaction techniques. In *Proceedings of User Interface Software and Technology (UIST'90)*, pages 135–142.

Carroll, J. M. and McKendree, J. (1987). Interface design issues for advice-giving expert systems. *Communications of the ACM*, 30(1):14–31.

Crow, D., Parsowith, S., and Wise, G. B. (1997). The evolution of cscw: Past, present and future development. *SIG***CHI** *Bulletin*, 29(2):20–26.

de Souza, C. S. (1993). The semiotic engineering of user interface languages. *International Journal of Man-Machine Studies*, 39:753–773.

de Souza, C. S. (1996). The semiotic engineering of concreteness and abstractness: From user interface languages to end user programming languages. *Dagstuhl Seminar Report #135 on Informatics and Semiotics*, page 11, February. Dagstuhl, Germany. Extended version published as monograph, available as MCC08/96 25 pages.

de Souza, C. S. (1997). Supporting end-user programming with explanatory discourse. In *Proceedings of the 1997 International Conference On Intelligent Systems and Semiotics (ISAS'97)*, volume 39, pages 461–466.

de Souza, C. S. (1998). Leading users from interaction into programming: The teaching-centered approach of semiotic engineering. *Subimited to Interacting wiht Computers*.

de Souza, C. S., Prates, R. O., and Varejão, F. M. (1997). Multmodel communication between software designers and software users. In *III Workshop em Sistemas Multimidia e Hipermidia no Brasil*, pages 93–105, São Paulo, UFSCar.

Dewan, P. and Choudary, R. (1992). A high-level and flexible framework for implementing multiuser user interfaces. *ACM Transactions on Information Systems*, 10(4):345–380.

Dix, A., Finlay, J., Abowd, G., and Beale, R. (1993). *Human-Computer Interaction*. Prentice-Hall International.

Eco, U. (1976). *A Theory of Semiotics*. Indiana University Press.

Eco, U. and Sebeok, T. (1983). *The sign of three*. Indiana University Press, Bloomongton, Indiana.

Ellis, C., Gibbs, S., and Rein, G. (1991). Groupware: Some issues and experiences. *Comunications of the ACM*, 34(1):39–58.

Ellis, C. and Wainer, J. (1994). A conceptual model of groupware. *ACM*, pages 79–88.

Garcia, A. C. and Vivácqua, A. (1996). The use of active documents to assist conflict mitigation in concurrent engineering. In *3^{rd} Conference on Concurrent Engineering, Research and Applications (CE'96)*.

Gelernter, D. and Jagannathan, S. (1990). *Programming Linsguistics*. MIT Press.

Greenberg, S. (1997). Human factors and web development. chapter Collaborative Interfaces for the Web, pages 241–254. LEA Press.

Greenberg, S., Gutwin, C., and Cockburn, A. (1996a). Awereness through fisheye views in relaxed-WYSIWIS groupware. In *Proceedings of Graphics Interface*, pages 28–38, Toronto, Canada.

Greenberg, S., Gutwin, C., and Roseman, M. (1996b). Semantic telepointers for groupware. In *Proceedings of OzCHI'96 Sixth Australian Conference on Computer-Human Interaction*, pages 28–38, Hamilton, New Zealand.

Greenberg, S. and Roseman, M. (1998). Computer supprted cooperative work. Trends in Software Series, chapter Groupware Toolkits for Synchronous Work. John Wiley & Sons Ltd.

GroupLab (s.d.). http://www.cpsc.ucalgary.ca/projects/grouplab/.

Grudin, J. (1994a). Computer-supported cooperative work: History and focus. *IEEE Computer*, May:19–26.

Grudin, J. (1994b). Groupware and social dynamics: Eight challenges for developers. *CACM*, 37(1):93–58.

Gutwin, C., Greenberg, S., and Roseman, M. (1996). People and computer XI. chapter Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation, pages 281–298. Springer-Verlag.

Gutwin, C., Stark, G., and Greenberg, S. (1995). Supporting workspace awareness in educational groupware. In *Proceedings of Computer Supported Collaborative Learning (CSCL'95)*, pages 147–156.

Hamalainen, M., Holsapple, C., Suh, Y., and Whinston, A. (1991). User interface design principles for team support systems. In *24th Annual Hawaii International Conference on System Sciences*, volume III, pages 461–470.

Hartson, H. R. (1998). Human-computer interaction: Interdisciplinary roots and trends. *The Journal of Systems and Software*, 43:103–118.

Hazemi, R. and Macaulay, L. (1996). Requirements for graphical user interface development environments for groupware. *Interacting with Computers*, 8(1):69–88.

Hewitt, B. and Nigel Gilbert, G. (1993). CSCW in practice: An introduction and case studies. chapter Groupware Interfaces. Springer-Verlag.

Jackobson, R. (1970). Linguística e comunicação. chapter 4, pages 73–87. Cultrix, São Paulo.

Jorna, R. and Van Heusden, B. (1996). Semiotics of the user interface. *Semiotica*, 109(3/4):237–250.

Kim, W. C. and Foley, J. D. (1990). Don: User interface presentation design assistant. In *Proceedings of User Interface Software and Technology (UIST'90)*, pages 10–20.

Kit, L. I. (*s.d.*). Internet Location: http://hcitel.uwaterloo.ca.

Laufer, C. and Fuks, H. (1995). ACCORD – conversation clichés for cooperation. In *Proceedings of International Workshop on the Design of Cooperative Systems (COOP'95)*, pages 351–369, France.

Leite, J. (1998). Modelos e formalismos para engenharia semiótica de interfaces de usuário. Tese de Doutorado, Departamento de Informática, PUC-Rio.

Lim, F. J. and Benbasat, I. (1991). A communication based framework for group interfaces in computer-supported collaboration. In *24th Annual Hawaii International Conference on System Sciences*, volume III, pages 610–620.

Mandviwalla, M. and Olfman, L. (1994). What do groups need? a proposed set of generic groupware requirements. *ACM Transactions on Computer-Humam Interaction*, 1(3):245–268.

Mark, W. (1986). User centered system design. chapter Knowledge-based Interface Design, pages 219–238. Lawrence Erlbaum, Hillsdale, NJ.

Martins, I. (1998). Um instrumento de análise semiótica para linguagens visuais de interfaces. Tese de Doutorado, Departamento de Informática, PUC-Rio.

Nadin, M. (1988). Interface design: A semiotic paradigm. *Semiotica*, 69(3/4):269–302.

Nake, F. (1994). Human-computer interation — signs and signals interfacing. *Languages of Design*, 2:193–205.

Norman, D. A. (1986). User centered system design. chapter Cognitive Engineering, pages 31–61. Lawrence Erlbaum, Hillsdale, NJ.

Paris, C., Swartout, W., and Mann, W. (1991). *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Kluwer Academics.

Peirce, C. S. (1931–1958). *Collected Papers*. Harvard University Press.

Pereira, F. C. N. and Warren, D. H. D. (1980). Definite clause grammars for language analysis — a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13(3):231–278.

Prates, R. O., de Souza, C. S., and Garcia, A. C. B. (1997). A semiotic framework for multi-user interfaces. *SIG***CHI** *Bulletin*, 29(2):28–39.

Prates, R. O., Leite, J., and de Souza, C. S. (1998). Semiotically based user interfaces design languages. In *Primeiro Workshopp em ICH*.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., and Carey, T. (1994). *Human Computer-Interaction*. Addison Wesley.

Roseman, M. and Greenberg, S. (1996a). Building real-time groupware with GroupKit, a goupware toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1):66–106.

Roseman, M. and Greenberg, S. (1996b). Teamrooms: Network places for collaboration. In *Computer Supported Cooperative Work (CSCW'96)*, pages 325–333.

Salvador, T., Scholtz, J., and Larson, J. (1996). The denver model for groupware design. *SIG***CHI** *Bulletin*, 28(1):52–58.

Santos, A. (1995). *Multimedia and Groupware for Editing*. Springer.

Searle, J. (1979). Expression and meaning. chapter 1, pages 1–29. Cambridge University Press, Cambridge U.K.

SERG (s.d.). Internet Location: http://www.inf.puc-rio.br/ sergweb/index.html
Intranet Location: http://www.les.inf.puc-rio.br.bf/ sergweb.

Smith, G. and Rodden, T. (1995). SOL: A shared object toolkit for cooperative interfaces. *International Journal on Human-Computer Studies*, 42:207–234.

Stefik, M., Bobrow, D. G., Foster, G., Lanning, S., and Tatar, D. (1987). WYSIWIS revised: Early experiences with multiuser interfaces. *ACM Transactions on Office Information Systems*, 5(2):147–167.

TeamWave Software Ltd. (s.d.). http://www.teamwave.com.

Wiecha, C. and Boies, S. (1990). Generating user interfaces: Principles and use of its style rules. In *Proceedings of User Interface Software and Technology (UIST'90)*, pages 21–30.

Winograd, T. (1996). *Bringing Design to Software*. Addison Wesley.

Winograd, T. and Flores, F. (1987). *Understanding Computers and Cognition, A New Fundation for Design*. Addison-Wesley.

Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606.