Um Modelo de Apoio à Expressão de Projetistas de Interfaces Multi-usuário

Raquel Oliveira Prates
Departamento de Informática – PUC-Rio
R. Marquês de São Vicente, 225
Gávea, Rio de Janeiro - RJ, 22453-900
raquel@les.inf.puc-rio.br
http://www.les.inf.puc-rio.br/~raquel

Clarisse Sieckenius de Souza
Departamento de Informática – PUC-Rio
R. Marquês de São Vicente, 225
Gávea, Rio de Janeiro - RJ, 22453-900
clarisse@inf.puc-rio.br
http://www.inf.puc-rio.br/~clarisse

RESUMO
A Engenharia Semiótica percebe a interface de uma aplicação computacional como sendo um ato de comunicação unilateral do *designer* para usuários. Em aplicações mono-usuário esta mensagem transmite ao usuário a interpretação do *designer* sobre o seu problema e a forma de interagir com a aplicação para resolvê-lo. Em interfaces multi-usuário esta mensagem é ainda mais complexa, uma vez que deve também transmitir aos usuários a organização e os modos de colaboração do grupo. A Engenharia Semiótica evidencia a necessidade de se fornecer ao projetista de interfaces ferramentas e ambientes que o auxiliem na expressão da sua mensagem aos usuários, ou seja, a interface. Neste trabalho, apresentamos um modelo de arquitetura que serve de base ao desenvolvimento de ferramentas e ambientes que apóiem o *designer* de interfaces multi-usuário no planejamento desta. Para isto, ele permite que o designer descreva o seu modelo conceitual do grupo, e lhe fornece indicadores qualitativos sobre esta descrição, sem no entanto restringir sua criatividade ou poder de decisão.

Palavras chave
Engenharia Semiótica, Interfaces multi-usuário, design de interfaces, design auxiliado por computador.

ABSTRACT
Semiotic Engineering perceives the interface of a computational application as a one-way communication act from designer to users. In single-user applications this message tells the users the designer's interpretation of their problem and how to interact with the application in order to solve it. In multi-user interfaces this message is even more complex, since it must also convey to users the organization of the group and collaboration means between its members. Semiotic Engineering brings forward the need for tools and environments that will support the planning of multi-user interfaces. In order to do so, they must allow the designers to describe their conceptual model of the group, provide them with qualitative indicators on this description, without, however, constraining their creativity or power to decide.

Keywords
Semiotic Engineering, Multi-user interfaces, Interface design, Computer-aided design.

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, 1994). 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 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, 1994). 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 & Roseman, 1998; Crow et al., 1997). Hoje há propostas de novos elementos de interface específicos para grupos (Hazemi & Macaulay, 1996; Greenberg & Roseman, 1998; Gutwin et al., 1996; Ackerman & Starr, 1996), diretrizes (Hamalainen et al., 1991; Mandviwalla & Olfman, 1994) e *frameworks* (Lim & Benbasat, 1991; Dewan & Choudary, 1992). 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 ainda o suficiente, uma vez que que não apóiam importantes passos do processo de *design*. Segundo Winograd (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 auxiliem na etapa de construção de interfaces, mas também aqueles que o auxiliem nas tomadas de decisões durante a etapa de planejamento destas interfaces.

Em Engenharia Semiótica, 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 outras mensagens, ela é considerada um artefato de meta-comunicação (de Souza, 1993). Ela responde 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, esta mensagem é mais complexa, uma vez que ela deve também deixar claro para os usuários como eles devem interagir entre si (Prates et al., 1997).

Usando a Engenharia Semiótica como base teórica, desenvolvemos uma forma de se apoiar o projetista de interfaces multi-usuário durante o processo de design da sua interface: o modelo de arquitetura de suporte ao *design* de interfaces multi-usuário. Este modelo permite que se desenvolva ferramentas ou ambientes que auxiliem o projetista durante o planejamento destas interfaces, mais especificamente na definição do grupo. As ferramentas ou ambientes são capazes de fornecer ao designer indicativos de qualidade sobre suas decisões, sem no entanto restringir a sua criatividade ou poder de decisão. Neste trabalho, apresentamos nosso modelo de arquitetura e descrevemos brevemente um protótipo dele derivado. Apresentamos sumariamente dois estudos de caso feitos usando o protótipo e os indícios que eles fornecem sobre as vantagens de se ter um ambiente baseado neste modelo de arquitetura.

Serão publicados trabalhos escritos em português ou em Inglês.

2. A ENGENHARIA SEMIÓTICA DE SISTEMAS MULTI-USUÁRIOS

Em abordagens semióticas (Nadin, 1988; Andersen et al., 1993; Jorna & 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 (de Souza, 1993; de Souza, 1996), em particular, a interface de um sistema é vista como sendo uma meta-comunicação entre *designer* e usuário, uma vez que a mensagem sendo enviada, a interface, é capaz de, ela mesma, trocar mensagens com o usuário. A mensagem é unilateral, uma vez que o usuário recebe a mensagem concluída e não pode dar continuidade ao processo de comunicação (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 segunda pergunta colocada pela Engenharia Semiótica, ou seja: como o usuário pode interagir com a aplicação para resolver seu(s) problema(s). O seu conteúdo é a sua funcionalidade, isto é, o que a aplicação é capaz de fazer, e a resposta para a primeira, ou seja: qual a interpretação do designer sobre o(s) problema(s) do usuário (Leite, 1998).

Quando um *designer* desenvolve um sistema multi-usuário, ele precisa comunicar ao grupo não apenas a sua interpretação sobre o(s) problema(s) que o grupo quer resolver e a forma de fazê-lo, mas também a organização do grupo e do trabalho entre seus membros. Assim, uma interface multi-usuário precisa comunicar aos membros do grupo os papéis que cada um deles pode assumir, a estrutura do grupo em que estes membros estão inseridos, as tarefas de cada um e como elas se relacionam, com que outros membros cada um pode comunicar-se e através de que linguagem e protocolos o fazem (Prates et al., 1997).

Ao interagir com a interface, o usuário forma uma idéia (seu interpretante) sobre a mensagem pretendida pelo *designer*. No entanto, a sua idéia adquirida sobre a mensagem e aquela pretendida pelo *designer*, dificilmente serão iguais. Contudo, para que a comunicação entre

designer e usuário seja de sucesso, é necessário que elas sejam consistentes. Em um grupo, diferentes membros podem assumir papéis e tarefas distintas e ter diferentes interfaces. Neste caso, a mensagem inicial que o *designer* está enviando a estes membros ou subgrupos de membros já é diferente e, assim, os usuários definitivamente atribuirão diferentes significados à interface. Não obstante, o significado atribuído por cada um dos membros 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, assim como com a mensagem pretendida pelo designer. Isto só é possível se, de início, as diferentes mensagens enviadas pelo projetista a diferentes membros forem consistentes entre si (Prates, 1998).

A Engenharia Semiótica adota uma posição complementar às atuais abordagens cognitivas para *design* de Interação Humano-Computador (IHC), à medida que ela enfatiza a expressão do *designer*, ao invés da interpretação do usuário[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 & 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.

Na próxima seção, apresentamos um modelo de arquitetura que permite que se apóie a expressão de projetistas de interfaces multi-usuário. Ambientes e ferramentas derivadas deste modelo dão suporte ao projetista durante o planejamento da sua mensagem aos membros de um grupo, permitindo que ele descreva esta mensagem e lhe fornecendo indicadores qualitativos sobre a mesma.

3. MODELO DE ARQUITETURA DE SUPORTE AO *DESIGN* DE INTERFACES MULTI-USUÁRIO

O projeto de interfaces multi-usuário pode ser visto como tendo duas etapas principais: a de planejamento e a de realização (Prates et al., 1998). Na etapa de planejamento o projetista define o que vai dizer, ou seja, o conteúdo de sua mensagem, enquanto que na de realização ele define como vai dizê-lo, isto é, a expressão da sua mensagem. O modelo de arquitetura apresentado nesta seção apóia o processo de *design top-down* destas interfaces. Para isto este modelo auxilia o projetista durante a etapa de planejamento da sua interface, mais especificamente na descrição do seu modelo conceitual de grupo[3]. Uma vez feita esta descrição, o modelo é capaz de fornecer ao projetista indicações sobre a etapa de realização.

Para que o modelo de arquitetura seja capaz de apoiar o projetista no planejamento da sua interface multi-usuário, ele é composto por uma linguagem de *design*[4], uma base de conhecimento e um cenógrafo. O componente responsável por fornecer indicações sobre a etapa de realização é o conselheiro de *widgets*. A Figura 1 mostra o modelo de arquitetura ao suporte de *design* de interfaces multi-usuário e o seu suporte ao processo de *design*. Em seguida descrevemos cada um dos componentes deste modelo.

Linguagem de Design

A linguagem de *design* possui dois componentes, um léxico e o outro semântico. O componente léxico é formado pelos construtores da linguagem, que representam as dimensões de caracterização de um grupo e têm por objetivo permitir ao projetista descrever o grupo e os seus modelos de colaboração e comunicação. O componente semântico, por sua vez, é formado por regras que verificam a descrição feita e identificam inconsistências em potencial presentes nela.

Os construtores da linguagem estruturam o espaço de solução existente, pois é através deles que o projetista deve expressar o modelo do grupo. Além disso, eles são independentes do domínio, uma vez que os valores que lhes podem ser atribuídos estão descritos de forma intensional ou extensional. A ligação com o domínio é feita pelo projetista no momento em que ele atribui um, dentre os valores possíveis, a cada um dos construtores. Para definir a organização do grupo, o projetista se utiliza de construtores que descrevem os papéis que podem ser assumidos pelos membros do grupo, os membros, propriamente ditos, e a hierarquia entre eles. As tarefas a serem executadas pelo grupo podem ser expressas em termos dos objetos sobre os quais cada membro pode agir. A interdependência entre as tarefas dos membros é definida explicitamente usando o construtor modelo de colaboração. Além disso, o projetista deve ainda determinar como os membros do grupo podem se comunicar, definindo, para isso, quem pode conversar ou falar (sem que o ouvinte possa responder) sobre o que e com quem, e quem pode ver o que.

---

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).

3 Para poder criar uma aplicação o designer deve criar um modelo conceitual desta aplicação. No caso de sistemas multi-usuário, o modelo conceitual de grupo faz parte deste modelo conceitual da aplicação (Prates, 1998).

4 Linguagem de design é toda aquela que permite ao designer expressar o seu design de algum artefato (Prates et al., 1998).

Os construtores descritos acima permitem ao *designer* fazer a descrição estática do grupo, ou seja, a descrição de características do grupo que não mudam no tempo, ou que existem inicialmente, e são chamados de construtores estáticos. Existem também os construtores relativos que permitem ao projetista descrever como as características do grupo mudam ao longo do tempo. Além das mudanças, os construtores permitem ainda que o *designer* defina as meta-mudanças, ou seja, o conjunto de mudanças que ele fornece aos membros do grupo para que eles mesmos modifiquem-no em tempo de execução. Desta forma, a linguagem de *design* permite ao projetista fazer a descrição estática e dinâmica do grupo.

Uma vez feita a descrição do modelo do grupo usando os construtores, o projetista pode então solicitar a verificação desta descrição pelas regras semânticas. As regras atribuem significado à combinação de valores definidas pelo projetista e identificam aquelas que aparentemente não fazem sentido. Como as regras se baseiam apenas nas combinações dos valores atribuídos aos construtores, isto é, na estrutura sintática da descrição feita pelo projetista, elas não levam em conta o domínio, e são consideradas separáveis de contexto.

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" faça perfeito sentido em um domínio específico. Assim, estas regras são descritivas e não prescritivas, o que significa que elas não retratam um julgamento absoluto de valor sobre as situações que identificam, e deixam 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 pode "passar por cima" dela, ou até mesmo desligar a regra que a identificou. Desta forma as regras não restringem a criatividade ou poder de decisão do *designer*.

Para ilustrar o funcionamento das regras, imagine-se que um projetista define que os membros do grupo devem trabalhar juntos em uma determinada tarefa. No entanto, ele não define que os membros podem conversar sobre os objetos da tarefa que eles compartilham. Esta situação viola uma das regras do modelo que determina que membros que trabalham juntos em uma tarefa devem poder conversar sobre os objetos que eles compartilham. Assim sendo, o projetista será informado desta inconsistência em potencial. Ele então avalia se a situação apontada é realmente uma inconsistência, e se for ele pode acertar a sua descrição do grupo. Se o projetista concluir que no seu domínio específico a situação faz sentido, ele pode então passar por cima da inconsistência.

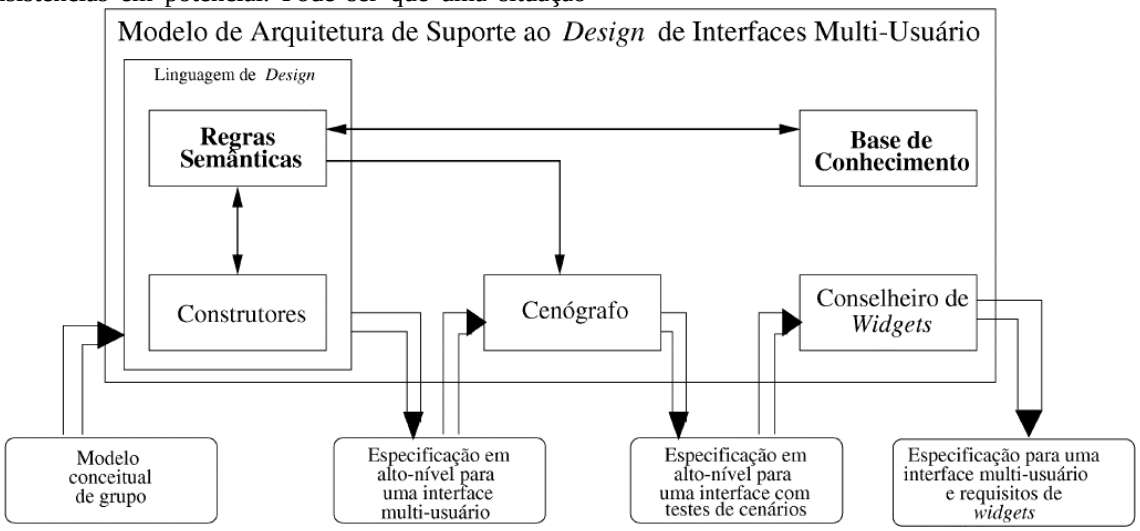

Figura 1 - Modelo de arquitetura de suporte ao design de interfaces multi-usuário e seu suporte ao processo de design.

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 o 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.

Unindo as explicações do modelo para as regras e as do projetista para as suas decisões que não estão de acordo com as regras, obtemos a lógica do modelo de grupo sendo especificado. Afinal, as regras explicam as situações que

são consideradas consistentes, e as explicações do projetista dão conta das que são consideradas inconsistentes. Assim, ao final da especificação, o projetista obtém não apenas uma descrição do seu modelo, mas também uma explicação para ele, contendo as justificativas das suas decisões.

No exemplo descrito anteriormente, quando o sistema identificasse a inconsistência em potencial, ele também forneceria ao projetista a explicação da regra violada. No caso acima, ele explicaria que se os membros trabalham juntos, compartilhando um ou mais objetos, e não podem conversar sobre eles, então eles não podem coordenar ou planejar suas ações sobre eles, tornando o seu trabalho mais difícil. Neste momento, se o projetista concluir que no seu domínio específico este não é o caso, por exemplo as ações de cada membro sobre os objetos são independentes, ou os membros estão em um mesmo local e podem conversar pessoalmente, fora do sistema, ele pode então entrar a devida explicação do seu caso na base de conhecimento.

Cenógrafo

O objetivo do cenógrafo é permitir ao projetista verificar se o conjunto de meta-mudanças sendo fornecido aos membros do grupo não os permite introduzir inconsistências na descrição do grupo. Para isto, o projetista interage com o cenógrafo e a partir de um cenário concreto – um estado do grupo no qual um ou mais membros podem definir mudanças – gera cenários (Carroll, 1995) possíveis de serem criados pelos membros, chamados cenários típicos. As regras semânticas fazem então a verificação dos cenários típicos criados e informam o *designer* quando alguma inconsistência é encontrada. Cabe ao *designer*, então, a decisão de se é necessário alterar o conjunto de meta-mudanças sendo fornecido aos membros, ou se no domínio da aplicação aquela inconsistência faz sentido.

Conselheiro de *Widgets*

Enquanto a linguagem de *design*, a base de conhecimento e o cenógrafo permitem que se apóie o projetista na etapa de planejamento de uma interface multi-usuário, o conselheiro de *widgets* dá um primeiro passo na direção da etapa de realização desta interface. 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. Note-se que o conselheiro de *widgets* não define os elementos de interface a serem usados, mas gera uma série de requisitos para aqueles que devem ser escolhidos e sugestões sobre eles. Isto porque, neste ponto do processo de *design*, as definições estão ainda em alto-nível e uma escolha definitiva de *widgets* seria precipitada. Além disso, o conjunto de elementos de interface que podem ser usados em uma interface multi-usuário é ainda um conjunto aberto, uma vez que à medida que novas aplicações de *groupware* são desenvolvidas, novas informações se mostram necessárias, e *widgets* são criados ou adaptados (Gutwin et al., 1995). Assim, não é possível sistematizar esta escolha de *widgets*, pelo menos por enquanto.

4. PROTÓTIPO

A partir do modelo de arquitetura apresentado na seção anterior pode-se derivar diferentes ferramentas e ambientes que dêem apoio ao projetista durante a etapa de planejamento de interfaces multi-usuário. Com o objetivo de mostrar que a implementação de tal ferramenta ou ambiente é possível implementamos um protótipo, que é uma instância simplificada do modelo proposto.

O protótipo está implementado em PROLOG e contém a parte estática da linguagem de *design*, ou seja, os construtores estáticos e as regras semânticas que verificam suas combinações, e a base de conhecimento. Para utilizar o protótipo, o projetista 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, onde os comandos permitem ao projetista descrever o grupo através da atribuição de valores aos construtores e requisitar a verificação da sua descrição. Além disso, existem comandos que mostram ao projetista as características já definidas por ele e as relações entre elas. A base de conhecimento atua quando o projetista 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 projetista 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.

A seguir, descrevemos os dois estudos de caso que fizemos usando este protótipo. O objetivos destes estudos de caso era obtermos indícios sobre a aplicabilidade do modelo proposto, ou seja, sua utilidade (Fischer, 1998) para projetistas de interfaces multi-usuário.

5. ESTUDOS DE CASOS

Usando o protótipo foram feitos dois estudos de caso, cujos objetivos eram verificar que na prática é possível fazer a modelagem de um grupo usando o modelo de arquitetura proposto e obter indícios das melhorias na qualidade do modelo do grupo que poderiam advir do uso do modelo de arquitetura. O primeiro estudo de caso consiste da modelagem do *website* da *Intranet* do SERG (*Semiotic Engineering Research Group*). A modelagem foi feita a partir “do zero”, uma vez que não existia ainda uma

modelagem da interação do grupo. O segundo estudo de caso feito foi a remodelagem do modelo de grupo do sistema Qualitas, para o qual já tinha sido feita uma modelagem usando técnicas típicas e genéricas[5] de Engenharia de *Software*. Em seguida, apresentamos cada um dos estudos de caso feitos e as indicações sobre o modelo de arquitetura e o protótipo que obtivemos com cada um deles.

Caso 1: *Intranet* do SERG

O objetivo deste teste era 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 em Engenharia Semiótica da PUC-Rio. No entanto, 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.

Dados os problemas causados pela participação espontânea e voluntária dos membros do SERG, o objetivo do *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. Como resultado deste teste, obtivemos observações que se referem tanto ao modelo de arquitetura, quanto ao protótipo.

Um ponto de nosso interesse neste teste era observar a adequação da linguagem de *design* proposta (Prates, 1998), tanto de seus construtores, quanto do seu conjunto de regras semânticas. Em relação aos construtores, o participante foi capaz de especificar todos os aspectos desejados do seu modelo conceitual do grupo, no entanto, durante este processo, ele identificou que em alguns momentos para atingir a especificação desejada, ele deveria dividi-la em um grande número de passos. Assim, o participante sugeriu alguns acréscimos aos construtores da linguagem que tinham como objetivo aproximar a linguagem de *design* aos objetivos de expressão do *designer*. Cabe aqui ressaltar que os acréscimos sugeridos não indicaram a necessidade de que novos aspectos de grupos fossem incluídos à linguagem, mas sim da inclusão de novos construtores e valores que representassem uma combinação daqueles já existentes.

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

Em relação ao conjunto de regras semânticas, conforme o esperado, a verificação de cada regra teve um dos três resultados:

(a) A especificação "obedecia" à regra e nenhuma inconsistência foi encontrada.

(b) A inconsistência foi encontrada e era relevante.

(c) A inconsistência foi encontrada, mas não era relevante.

A relevância das inconsistências encontradas foram julgadas pelo participante com base nos seus objetivos de modelagem, que eram claramente dependentes do domínio, e na explicação das regras oferecidas pelo protótipo. Em todos os casos de inconsistências relevantes foram duas as possíveis causas: (1) o participante esqueceu de definir algum aspecto do grupo, embora o tivesse previsto e (2) a situação inconsistente tinha sido gerada por "efeitos colaterais" de descrição e não tinha sido prevista pelo projetista. Em ambos os casos, inconsistências relevantes causaram modificações da descrição pelo projetista. Com as inconsistências relevantes alcançamos o nosso objetivo ao propor as regras, que oferecem um indício das vantagens que um sistema de apoio baseado no modelo de arquitetura pode oferecer a projetistas de interfaces multi-usuário.

Embora o ponto principal deste trabalho não seja o protótipo implementado, mas sim o modelo de arquitetura em que este é baseado, é interessante apresentar algumas das observações relacionadas a ele. Afinal, elas fornecem *insights* tanto sobre as ferramentas e ambientes a serem derivados do modelo, quanto sobre limitações que a implementação pode impor ao modelo. 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 do grupo e as relações estabelecidas. Assim, ela requer grande esforço do projetista 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 projetista. Por exemplo, quando o projetista 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 projetista (de Souza et al., 1997). Os demais pontos levantados sobre o protótipo eram referentes à implementação dos construtores da linguagem de *design* e da função de verificação das regras (Prates, 1998).

Caso 2: Qualitas

O sistema Qualitas (TeCGraf, 1998) tem como objetivo permitir aos funcionários de uma empresa manter a consistência e qualidade (conforme normas ISO) da avaliação de requisições de clientes, bem como da criação

e cumprimento de contratos que atendem a estas requisições. Diferentemente do *site* da *Intranet* do SERG, o sistema Qualitas ao ser modelado em nosso protótipo 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 do nosso modelo, 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.

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. Porém, estes dois requisitos 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. Entretanto, 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, devem existir também aqueles que dão conta da descrição dinâmica.

Decidiu-se continuar o teste apesar deste obstáculo, escolhendo-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 de grupo sendo definido em uma ferramenta que implementasse todo o modelo de arquitetura.

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

Após especificado o modelo conceitual do grupo do 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 pelo *designer*. Ao explicar estas situações o participante definiu a lógica das decisões de projeto tomadas na modelagem do sistema, que até então não tinham sido explicitamente documentadas.

A verificação da descrição do modelo conceitual do grupo deste participante teve duas contribuições significantes na especificação e modelagem. A primeira delas aconteceu quando o sistema apontou uma inconsistência em potencial na interação entre dois membros do grupo. O projetista não considerou a situação uma inconsistência. No entanto, ao explicar a decisão que levava a esta situação ele se deu conta que membros de um determinado papel na verdade sempre assumiam dois papéis distintos dentro do grupo. O 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* o projetista resolveu modificar a sua especificação do grupo e descrever estes dois papéis separadamente, ao invés de fornecer uma explicação sobre a situação.

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 o participante se deu conta de que não estava muito certo de como explicar esta decisão de projeto. 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 o participante pudesse imaginar porque esta habilidade poderia ser desejável, ele também conjeturava sobre os problemas que ela poderia trazer. No caso desta decisão de projeto 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 trazer, uma vez que no mundo real a concretização desta habilidade exigia um certo esforço por parte do membro do grupo, enquanto que no mundo virtual este custo passaria a ser praticamente nulo.

Esta inconsistência em potencial identificada não levou o participante a alterar a sua descrição do grupo, mas a explicá-la. O participante forneceu como explicação os motivos pelos quais ele imaginava que esta situação poderia ser desejável, mas que deveriam ser confirmados com os usuários, e também os problemas que ele antevia que poderiam decorrer dela e que não tinham sido considerados. Note-se que esta informação, uma vez documentada pelo projetista, poderia ser usada tanto por

ele quanto por outros membros da equipe de *design* do Qualitas.

Este exemplo vivenciado pelo 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 pelo 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).

6. CONCLUSÃO

Neste trabalho apresentamos um modelo de arquitetura desenvolvido conforme a proposta da Engenharia Semiótica, uma vez que ele serve de base para o desenvolvimento de ferramentas e ambientes que dêem apoio à expressão de projetistas de interfaces multi-usuário. Para auxiliar o projetista durante a etapa de planejamento da sua interface, este modelo lhe oferece uma linguagem de *design*, uma base de conhecimento, um cenógrafo e um conselheiro de *widgets*. A linguagem de *design* é separável de contexto e permite que o projetista descreva aspectos estáticos e dinâmicos do grupo. Além disso, ela atribui significado aos valores conferidos aos construtores e às suas combinações, apontando ao projetista inconsistências em potencial. A base de conhecimento complementa as informações fornecidas pelo conjunto de regras, explicando como as dimensões descritas pelos construtores estão relacionadas e como o valor atribuído a uma pode afetar a outra. Ademais, 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 projetista diretrizes para e sugestões sobre a escolha de *widgets* com base na descrição do grupo especificada.

O modelo de arquitetura permite que se forneça ao projetista indicativos de qualidade sobre a sua descrição do seu modelo conceitual de grupo, através das inconsistências em potencial e suas explicações. É importante ressaltar que o modelo de arquitetura não faz um julgamento de valor sobre a descrição do designer e o deixa como responsável pelo seu modelo. Assim, o modelo, propriamente dito, não gera melhorias de qualidade, mas auxilia o projetista a fazê-lo, uma vez que aponta e explica situações "inconsistentes" e "efeitos colaterais" das decisões tomadas.

Para avaliar a viabilidade da implementação de ferramentas ou ambientes baseados no modelo de arquitetura, implementamos um protótipo que é uma instância simplificada deste modelo. Usando este protótipo fizemos dois breves estudos de caso, que indicaram como cada um dos componentes do modelo de arquitetura, a menos do conselheiro de *widgets*, auxilia o projetista e contribui para o *design* de um modelo de grupo mais consistente e coeso. Conforme foi visto, o protótipo foi capaz de apontar para o projetista problemas que teriam passado despercebidos nesta etapa e que provavelmente apareceriam, ou mais adiante no processo de *design*, ou apenas após a entrega da aplicação, durante o seu uso. Desta forma, o protótipo, junto a outros ambientes e ferramentas derivados do modelo de arquitetura, pode ser considerado um instrumento de avaliação formativa, isto é, avaliação feita anteriormente à implementação do sistema e que influencia o seu desenvolvimento (Preece et al., 1994). Este tipo de avaliação permite a redução tanto do custo financeiro do sistema, uma vez que permite a correção de erros no início do processo de *design*, quanto de uso, na medida que permite que *designers* ofereçam um sistema de melhor qualidade para os usuários (Hartson, 1998), diminuindo a oportunidade de os usuários não entenderem a mensagem sendo passada.

Uma ferramenta desenvolvida a partir do nosso modelo de arquitetura pode ser definida como sendo um sistema especialista em *design* de interfaces multi-usuário. Na verdade, o sistema é 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 & Foley, 1990; Wiecha & Boies, 1990; Bleser & Sibert, 1990), nenhum deles é para interfaces multi-usuário ou para a etapa de planejamento da interface.

Além disso, um importante ponto de divergência entre os ambientes derivados do modelo de arquitetura e os demais é que enquanto os primeiros aconselham, os outros agem. Isto é, os sistemas derivados do nosso modelo de arquitetura apontam ao *designer* inconsistências em potencial e fornecem explicações sobre elas, ajudando-o então a pesar as conseqüências e implicações de suas decisões. Os outros sistemas citados acima, por sua vez, tomam controle do *design* e, baseados em preferências e prioridades definidas pelo *designer*, tomam decisões por ele.

A diferença é fundamental, uma vez que ela caracteriza os sistemas apresentados como incompatíveis com a proposta da Engenharia Semiótica. Afinal, ao tomar decisões de *design* no lugar do 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*.

7. PRÓXIMOS PASSOS

Embora os breves estudos de caso apresentados neste trabalho nos forneçam alguns indícios da validade do modelo de arquitetura e sua aplicabilidade, eles são apenas uma avaliação preliminar deste modelo e das ferramentas derivadas dele. Assim, pretendemos continuar trabalhando na direção de uma avaliação extensiva. Nos testes feitos tivemos indicadores de como os construtores e regras dinâmicas da linguagem de *design* e o cenógrafo poderiam auxiliar o projetista. Portanto, o primeiro passo na direção de uma avaliação extensiva envolve incluir estes componentes no protótipo.

Além disso, testamos apenas o suporte do modelo de arquitetura à etapa de planejamento da interface e nenhum teste foi realizado para se obter indícios sobre seu suporte inicial à etapa de realização. Para tanto, teríamos que implementar também o conselheiro de *widgets* e avaliar o suporte que ele se propõe a fornecer ao projetista.

Uma vez completo o protótipo, podemos ter indicadores sobre todos os aspectos propostos pelo modelo de arquitetura e a aplicabilidade de cada um deles (Prates, 1998). No entanto, ter um protótipo completo não é suficiente para avaliar extensivamente o modelo de arquitetura, uma vez que o próprio determina a usabilidade do modelo de arquitetura no qual ele se baseia. Então, para que possamos obter resultados conclusivos é necessário transformar este protótipo em uma ferramenta bem-acabada, que possa ser usada com eficiência. Ou seja, antes de ser usada para testes do modelo, a própria ferramenta tem que ser submetida a testes de usabilidade.

AGRADECIMENTOS

Agradecemos em especial aos participantes dos testes do SERG e do Qualitas pelo seu tempo, disposição e valiosa contribuição para a avaliação do trabalho apresentado. Agradecemos também aos membros do SERG pelas discussões e comentários que enriqueceram este trabalho. Finalmente, as autoras são gratas ao CNPq pelo apoio dado ao desenvolvimento das pesquisas apresentadas neste trabalho.

REFERÊNCIAS

Ackerman, M. and Starr, B. (1996). Social Activity Indicators for Groupware, IEEE Computer, 37-42.

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

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

Bleser, T. and Sibert, J. (1990). Toto: A Tool for Selecting Interaction Techniques. In Proceedings of User Interface Software and Technology (UIST'90), 135-142.

Carroll, J. (1995). Scenario-Based Design. John Wiley & Sons.

Crow, D., S. Parsowith and Wise, G. B.(1997). The Evolution of CSCW: Past, Present and Future Development. SIGCHI 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, 11. Extended version published as monograph available as MCC08/96, PUC-Rio.

de Souza, C. S., Prates, R. O. and Varejão, F. M. (1997). Multimodel Communication Between Software Designers and Software Users. Anais do III Workshop em Sistemas Multimidia e Hipermidia no Brasil, 93-105.

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.

Ellis, C., Gibbs, S. and Rein, G. (1991). Groupware: Some Issues and Experiences. Comunications of the ACM, 34, 1, 39-58.

Fischer, G. (1998). Beyond 'Couch Potatoes': From Consumers to Designers". In Proceedings of The 5th Asia Pacific Computer Human Interaction Conference, 2-9.

Greenberg, S. and Roseman, M. (1998). Computer Supprted Cooperative Work. In Groupware Toolkits for Synchronous Work, Ed. M. Beandouin-Lafon, John Wiley & Sons Ltd.

Grudin, J. (1994). Groupware and Social Dynamics: Eight Challenges for Developers. Comunications of the ACM, 37, 1, 93-58.

Gutwin, C., Stark, G. and Greenberg, S. (1995). Supporting Workspace Awareness in Educational Groupware. In Proceedings of Computer Supported Collaborative Learning (CSCL'95), 147-156.

Gutwin, C., Greenberg, S. and Roseman, M. (1996). Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation. People and Computer XI, Eds. R. J. Sasse and A. Connibgham and R. Winder, Springer-Verlag, 281-298.

Hamalainen, M., Holsapple, C., Suh, Y. and Whinston, A. (1991). User interface design principles for team support systems. In Proceedings of the 24th Annual Hawaii International Conference on System Sciences, III, 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.

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), 10-20.

Leite, J. C. (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 Proceedings of 24th Annual Hawaii International Conference on System Sciences, III, 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.

Nadin, M.(1988). Interface Design: A Semiotic Paradigm. Semiotica, 69, 3/4, 269-302.

Prates, R. O., de Souza, C. S. and Garcia, A. C. B.(1997). A Semiotic Framework for Multi-User Interfaces. SIGCHI Bulletin, 29, 2, 28-39.

Prates, R. O. (1998). A Engenharia Semiótica de Linguagens de Interfaces Multi-usuário. Tese de doutorado, Departamento de Informática, PUC-Rio.

Prates, R. O., Leite, J. C. and de Souza, C. S. (1998). Semiotically Based User Interfaces Design Languages. Atas do I Workshop sobre Fatores Humanos em Sistemas Computacionais (IHC'98), 18-27.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. and Carey, T. (1994). Human-Computer Interaction, Addison Wesley.

TeCGraf, (1998). Qualitas: Sistema de Informação. Relatório de atividades, Petrobrás/TeCGraf – PUC-Rio.

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), 21-30.

Winograd, T. (1996). Bringing Design to Software, Addison Wesley.