2
Fundamentação

Como dito anteriormente, o eXCeeD busca unir o apoio à reflexão oferecido por uma teoria de IHC, a EngSem, com a característica de agilidade de técnicas de prototipação de interfaces, o que torna este processo bem próximo dos métodos ágeis. Por esta razão, apresentamos uma explanação dos fundamentos práticos e teóricos que servem de base para a definição do eXCeeD. Veremos, no próximo capítulo, como os conceitos expostos serão utilizados na definição do eXCeeD.

2.1.
Métodos ágeis

Originadas principalmente da prática real dentro de empresas de desenvolvimento de software, nos anos noventa, várias metodologias propuseram o seguinte conjunto de idéias, adaptando e combinando velhos e novos conhecimentos:

"forte colaboração entre a equipe de programadores e as pessoas de negócio; comunicação face-a-face (mais eficiente do que documentação escrita); entrega freqüente de software que agregue valor; equipes com um bom relacionamento e auto-organizáveis; e formas de manter cuidadosamente o código e a equipe de maneira que modificações drásticas nos requisitos não sejam encaradas como uma crise"[4]

Estas metodologias buscam o equilíbrio entre a ausência e o excesso de processo, defendendo uma quantidade de processo suficiente para desenvolver e

[4] Agile Alliance. Disponível em <http://www.agilealliance.org/show/2>. Acesso em 12 abr. 2007.

manter sistemas computacionais interativos através de métodos ágeis de desenvolvimento de software. O movimento dos métodos ágeis é muito bem ilustrado pelo Manifesto para o Desenvolvimento Ágil de Software (Beck et al., 2001), que declara na seguinte mensagem os seus valores:

"Estamos descobrindo melhores maneiras de desenvolver software fazendo isto e também ajudando outros a fazê-lo. Através deste trabalho, passamos a dar mais valor:

*A **indivíduos e interações** do que a processos e ferramentas*

*A **software funcional** do que a documentação abrangente*

*A **colaboração com o cliente** do que a negociação de contratos*

*A **responder a mudanças** do que a seguir um plano*

Isto é, mesmo existindo valor nos itens da direita, nós valorizamos mais os itens da esquerda."

Mais próximos da realidade do processo de desenvolvimento de software, estes valores podem ser explicados conforme a seguir (Highsmith e Cockburn, 2001):

- **Indíviduos e interações** mais do que processos e ferramentas

  Softwares são feitos por pessoas. Desta maneira, as habilidades pessoais dos membros da equipe de desenvolvimento e a confiança na comunicação entre eles são características cruciais para o compartilhamento de informações e para o desenvolvimento do projeto;

- **Software funcional** mais do que documentação abrangente

  O mais importante no desenvolvimento de software é a sua implementação. Assim, esta atividade é a medida de produção de resultados e de conseqüente fornecimento de *feedback* por parte dos usuários. A pouca documentação é compensada pela comunicação

presencial entre os membros da equipe de desenvolvimento e os usuários;

- **Colaboração com o cliente** mais do que negociação de contratos

  A colaboração com o cliente indica que todos os envolvidos no processo de desenvolvimento do software, clientes, usuários e desenvolvedores, devem fazer parte de uma mesma equipe, se possível trabalhando todos em um mesmo ambiente. Aliar a utilização dos diferentes graus de conhecimentos e habilidades que estas pessoas possuem a um bom relacionamento entre elas, tem um impacto positivo nos resultados do sistema em construção. Vale ressaltar que este valor não nega a importância dos contratos. Estes são necessários, porém não são suficientes para o sucesso do produto;

- **Responder a mudanças** mais do que seguir um plano

  Desenvolver software seguindo um plano direciona a equipe de desenvolvimento a analisar este projeto e pensar nos possíveis problemas que possam ocorrer. No entanto, este desenvolvimento deve atender às mudanças que venham a ocorrer durante todo o processo, procurando adaptar-se as novas exigências que surjam com o decorrer do tempo.

Através destes valores, os métodos ágeis para o desenvolvimento de software enfatizam uma maior preocupação com os usuários e com as suas necessidades. Semelhante ao design participativo, eles trazem os usuários para o início do processo de desenvolvimento, quando as funcionalidades para o software sendo projetado ainda estão sendo definidas, e oferece a oportunidade de eles participarem de todo o processo informando quais as suas necessidades (Silva et al., 2006). A diferença entre os métodos ágeis e o design participativo, entretanto, está nas atividades em que os usuários estão envolvidos e no papel que eles exercem durante o processo. Nos métodos ágeis, no início do processo, eles são fornecedores de informação e depois avaliadores do sistema construído. Já no design participativo, eles têm uma participação mais ativa, não somente sendo

fornecedores de informação e avaliadores do que foi produzido, mas contribuindo diretamente com a construção do sistema.

Próximos dos métodos ágeis de desenvolvimento de software, o eXCeeD apóia-se nesses valores para definir os seus próprios. Apóia-se em particular no representante mais conhecido deste tipo de processo: o eXtreme Programming (XP) (Beck, 1999). Para tanto, o eXCeeD utiliza os conceitos envolvidos no XP em sua definição. A seguir, apresentaremos o XP, juntamente com seus os valores e práticas.

2.1.1. eXtreme programming (XP)

O XP é um processo de desenvolvimento de software ágil proposto para projetos que possuam uma equipe de dois a dez desenvolvedores e que estejam trabalhando com funcionalidades que ainda não estão muito bem definidas ou que mudam rapidamente em seu decorrer. Em seu livro, o autor do processo define o XP como *"uma maneira leve, eficiente, de baixo-risco, flexível, previsível, científica e divertida de desenvolver software"* (Beck, 1999 p. xvii).

O XP é pautado em quatro valores que estão intimamente relacionados com os valores apresentados anteriormente para os métodos ágeis. Estes valores são apresentados a seguir:

- **Comunicação**

  Durante o desenvolvimento de um software, problemas de comunicação podem trazer conseqüências desastrosas para o projeto. Por exemplo, suponhamos que um desenvolvedor possui uma decisão crítica a tomar, mas não sabe muito bem como proceder. O mais interessante que pode ocorrer é que ele procure conversar com uma pessoa na qual ele confie e que possa ajudá-lo na tomada daquela decisão. Se ele não conversar com a pessoa correta ou simplesmente decidir não conversar com alguém, é provável que a decisão tomada venha a não ser a mais adequada ao contexto do

projeto e ele só se dará conta do erro cometido mais adiante. Para que problemas deste tipo sejam minimizados ao máximo, o XP propõe algumas práticas, como programação aos pares e colaboração com cliente, que impõem a comunicação face-a-face entre todos os envolvidos no projeto;

- **Simplicidade**

  Com este valor, o XP acredita que é melhor adotar uma solução mais simples para o momento do que se perder tempo implementando uma outra que seja mais abrangente, mas que possivelmente não venha a ser utilizada depois. No futuro, caso seja necessário, esta solução mais simples adotada pode vir a ser modificada por outra que se adeque às novas exigências do software. Segundo Beck, os valores de comunicação e simplicidade estão intimamente relacionados, ou seja, quanto maior a comunicação entre os membros da equipe (clientes, usuários ou desenvolvedores), mais confiança se tem no que se deve ou não ser feito, optando sempre pelo mais simples;

- ***Feedback***

  A aplicação deste valor somente é possível através do valor da comunicação. No XP, o cliente participa ativamente de todo o desenvolvimento do sistema, tirando dúvidas dos desenvolvedores, decidindo quais as funcionalidades que o sistema deve possuir, quais delas devem ser implementadas em um primeiro momento e quais devem ser deixadas para depois e como o sistema deve ser testado, entre outras atividades. Todo o processo constitui-se em um aprendizado contínuo, onde a comunicação e a troca constante de conhecimento para a realização destas atividades atuam positivamente sobre a produtividade do projeto; e

- **Coragem**

  Este último valor indica que, para trabalhar com XP, é essencial ter coragem. É necessário coragem, por exemplo, para indicar aos

desenvolvedores quando há qualquer problema, para que ele possa ser solucionado e para modificar parte do código ou refazer determinada parte do sistema, corrigindo-o ou tornando-o mais simples sempre que isso se mostrar necessário.

Todos estes quatro valores, **comunicação**, **simplicidade**, ***feedback*** e **coragem**, são apoiados por um último: **respeito**. É necessário respeito por todos os participantes do desenvolvimento do projeto para compartilhar e discutir opiniões e para tomar as melhores decisões, bem como por cada participante individualmente, confiando que ele pode fazer o trabalho da maneira mais simples e com a máxima coragem que possui.

Aliado aos valores apresentados anteriormente, Beck aponta mais um conjunto de doze práticas que regem o XP. Estas práticas são apresentadas a seguir:

- **Jogo do planejamento**

  O projeto deve ter um planejamento inicial mínimo, mas que pode ser totalmente adaptável a mudanças. Este planejamento deve conter, por exemplo, estimativas de tempo de implementação das funcionalidades e da data de entrega de cada uma delas. Ademais, uma das maiores preocupações do XP como um processo de desenvolvimento é manter os clientes satisfeitos. Assim, qualquer modificação feita neste planejamento deve ser realizada sob sua presença e aprovação;

- **Releases curtos**

  O planejamento inicial ajuda a definir como o sistema será construído e entregue. O XP propõe que o sistema seja entregue em pequenas porções, cada qual contendo um conjunto de funcionalidades implementadas conforme o grau de importância definida pelo cliente. Considerando que, antes da adição de um novo conjunto de funcionalidades, o sistema funcionava corretamente, à medida que as novas funcionalidades vão sendo acrescentadas é

necessário que todo o sistema seja testado novamente de maneira a garantir a sua integridade;

- **Metáfora**

  Uma única estória deve guiar todo o desenvolvimento do software, indicando através de uma metáfora como o sistema funciona. Através desta metáfora todos podem entender os elementos básicos e relações existentes no projeto. Para tanto, um conjunto de palavras associadas a tal metáfora deve ser utilizado durante todo o decorrer do projeto;

- **Design simples**

  Deve-se manter o projeto o mais simples possível, pensando-se apenas nas soluções necessárias para o momento e, desta maneira, implementando-se a opção mais simples que funcionar. Com o passar do tempo, no decorrer do projeto, pode-se ir simplificando o projeto continuamente;

- **Testes**

  Para cada funcionalidade implementada, um conjunto de testes deve ser escrito com o intuito de verificar a integridade do que foi implementado. Os testes devem ser feitos de maneira exaustiva por funcionalidade antes que ela seja integrada ao restante do sistema. Depois, o sistema por completo também deve ser testado com o intuito de verificar se ele ainda continua funcionando corretamente com a nova funcionalidade que foi acrescentada;

- ***Refactoring***

  Sempre que se percebe que se pode melhorar ou simplificar o projeto em determinada parte, deve-se fazê-lo. No entanto, as alterações podem introduzir erros indesejáveis. Assim, depois que o *refactoring* é realizado, todos os testes devem ser executados novamente para que os possíveis erros possam ser detectados e solucionados;

- **Programação em pares**

  Toda a implementação do sistema deve ser realizada por duplas de programadores. Esta prática favorece a comunicação entre os membros da equipe de desenvolvimento e a discussão de idéias, um dos valores do XP. Além disso, os erros são menos freqüentes pois é possível que enquanto um componente do par esteja programando, o outro supervisione o código sendo produzido, apontando os erros;

- **Código coletivo**

  Esta prática indica que, em um projeto seguindo o XP, todos devem ter conhecimento de todo o código produzido e também liberdade para modificá-lo quando julgarem necessário, sempre com o intuito de melhorar o projeto, tornando-o mais simples;

- **Integração contínua**

  A integração das diferentes unidades do sistema deve ser feita freqüentemente, sempre após a execução do seu conjunto de testes. Esta prática garante que os conflitos entre as diferentes partes e o custo de integração sejam mínimos;

- **Semana de quarenta horas**

  A semana de trabalho de toda a equipe não deve ter uma carga horária maior do que quarenta horas, para evitar que os desenvolvedores se desgastem e permitir que eles estejam sempre aptos a executar as outras práticas;

- **Cliente presente**[5]

[5] Sabemos que *clientes* e *usuários* possuem papéis diferentes durante o processo de desenvolvimento de software. Contudo, Beck (1999) parece não fazer distinção entre eles, uma vez que denomina tal prática como **cliente presente** e afirma em sua definição que *usuários* é que devem colaborar durante o desenvolvimento do sistema.

Os usuários devem cooperar fortemente com a equipe de desenvolvimento durante todo o processo de desenvolvimento de software. São eles que definem, juntamente com a equipe de desenvolvimento, em que ordem as funcionalidades definidas para o sistema deverão ser implementadas, contribuem na elaboração dos testes necessários para o sistema e também fornecem *feedback* constante sobre o sistema que está sendo construído; e

- **Código padronizado**

  Como o código é implementado por diferentes pessoas, a utilização de regras de codificação facilita o entendimento e a manutenção do código por todos. Esta prática garante que outras práticas, como o *refactoring* e código coletivo sejam executadas sem maiores dificuldades.

Além destas práticas, Teles (2004) aponta mais uma: a prática das **reuniões em pé**. Realizadas diariamente, no início do expediente, nestas reuniões se discute a execução do trabalho no dia anterior, identificando e solucionando os problemas encontrados, e como dar seguimento ao trabalho no dia corrente. Ao final da reunião, um planejamento diário é decidido, indicando que funcionalidades deverão ser implementadas naquele dia e quem será o responsável por cada uma delas. Desta maneira, por permitirem avaliar o progresso do projeto diariamente, estas reuniões revelam-se de extrema importância para o seu sucesso.

No XP, uma funcionalidade é descrita em uma "estória"[6] (ou cartão de estória). Uma estória contém a descrição textual sumarizada de algo que os usuários desejam para um sistema sendo projetado, em no máximo três sentenças (Wells, 1999). Escritas pelos analistas ou pelos próprios usuários (ou pelos dois, conjuntamente) em linguagem natural, as estórias apresentam os requisitos do sistema sendo projetado de maneira abstrata. São elaboradas em substituição ao

[6] Vale observar que, no XP, estórias não correspondem a cenários, tal como definido por (Carroll, 2000): *"Cenários são estórias sobre pessoas e suas atividades"*. Normalmente, elas apresentam apenas a descrição de uma funcionalidade do sistema requisitada pelo usuário.

documento de especificação de requisitos dos métodos tradicionais de desenvolvimento de software. As estórias são utilizadas em três momentos no XP: na estimativa de tempo para implementação das funcionalidades do sistema, na implementação propriamente dita e, por último, na realização de testes do sistema (Beck, 1999). Cada estória é uma pequena parte do sistema e, quando combinadas, as estórias fazem (ou espera-se que façam) um todo lógico e consistente. Essa característica do XP é possível graças principalmente às práticas **programação em pares**, **cliente presente**, **jogo do planejamento**, ***refactoring***, **código coletivo** e **integração contínua**. A prática programação em pares reforça a comunicação e o trabalho em equipe. Já as práticas cliente presente e jogo do planejamento garantem que as decisões não são tomadas individualmente, mas pela equipe de desenvolvimento juntamente com os clientes. As práticas do ***refactoring*** e do **código coletivo** asseguram que qualquer eventual incoerência de alguma das partes do software poderá ser corrigida. Além disso, esta correção poderá ser realizada por qualquer membro da equipe de desenvolvimento, uma vez que o poder de modificar o código é concedido a todos. A prática do código coletivo ainda assegura que os membros da equipe de desenvolvimento tenham a visão global do sistema, uma vez que ela afirma que todos devem ter um conhecimento geral do que está sendo produzido. Por último, a **integração contínua** garante que as diferentes partes de código construídas estão sendo combinadas freqüentemente e os possíveis problemas decorrentes desta junção serão detectados rapidamente e corrigidos de maneira a garantir a consistência do código implementado de uma maneira geral.

Depois de apresentarmos brevemente a fundamentação prática do eXCeeD, passamos à sua fundamentação teórica: a EngSem. Em seguida, na mesma seção, falaremos sobre duas ferramentas epistêmicas também baseadas nesta teoria: a MoLIC e as expressões para sistemas de ajuda online.

2.2.
Engenharia Semiótica

A EngSem é uma teoria que caracteriza a interação humano-computador como uma comunicação humana mediada por computador (de Souza, 2005). Desta definição, temos que a interação é estritamente uma conversa entre o designer e o usuário que ocorre através da interface de um software e ambos são, portanto, interlocutores de um mesmo processo comunicativo. Nesta conversa, o designer envia ao usuário uma mensagem unidirecional, de uma única vez, consistindo em um conteúdo completo e imutável, codificado através da interface do software. Por meio desta mensagem, o designer apresenta uma possível solução para as necessidades e as preferências dos usuários identificadas por ele durante o projeto da interface, apresentando o seu ponto de vista no momento da concepção do software. Enunciada em primeira pessoa, a mensagem pode ser parafraseada no texto a seguir:

"Eis a minha interpretação de quem você (usuário) é, o que aprendi que você tem de fazer, preferencialmente de que forma, e por quê. Eis, portanto, o sistema que conseqüentemente concebi para você, o qual você pode ou deve usar assim, a fim de realizar uma série de objetivos associados com esta (minha) visão." (ibid p. 84)

Segundo a EngSem, a interação humano-computador também é um caso particular de metacomunicação mediada por computador. A metacomunicação é uma comunicação sobre a comunicação. Uma das características importantes da metacomunicação é a maneira como ela fornece aos usuários a chave para interpretar a própria comunicação. Através de algumas indicações fornecidas na própria comunicação, o emissor de uma mensagem qualquer pode conduzir a interpretação do receptor desta mensagem por determinados caminhos conforme desejar. Sob o ponto de vista da EngSem, os softwares são artefatos de metacomunicação, ou seja, artefatos que comunicam uma mensagem sobre a sua própria comunicação, e o designer tem o papel de conduzir a comunicação com o usuário através da interface do sistema da maneira como desejar. Uma vez que esta mensagem foi enviada pelo designer, ela é capaz de trocar novas mensagens

com o usuário, indicando como ela própria deve ser interpretada para alcançar os objetivos do usuário identificados durante o projeto do sistema.

Contudo, quando o usuário estiver interagindo com a interface do software, a comunicação designer-usuário não ocorrerá diretamente, pois o designer não estará presente no momento em que esta interação ocorre. Em tempo de interação, a comunicação será feita por intermédio do preposto do designer, um agente comunicante que fala ao usuário em nome do designer. O preposto do designer pode se comunicar de maneira implícita, através de elementos de interface, tais como rótulos ou campos de texto, ou de maneira explícita, através da ajuda do software ou de um agente inteligente, como por exemplo o assistente do Microsoft® Office. Desta maneira, a comunicação designer-usuário projetada em tempo de design vai acontecer sobre a comunicação usuário-preposto em tempo de interação.

Ao definir a interação como uma conversa entre o designer e o usuário, a EngSem leva em consideração a seguinte afirmação de (Eco, 1980): *"todo processo de comunicação entre seres humanos pressupõe um sistema de significação como condição necessária"*. Estabelecido por convenções culturais e sociais, um sistema de significação possui um código que relaciona conteúdos a determinadas expressões. Para que o processo de comunicação ocorra, é necessário que um ou mais sistemas de significação sejam utilizados por emissores e interpretadores de signos para uma variedade de objetivos. A definição de signo utilizada na EngSem é a da semiótica de Peirce, onde um signo é definido como *"algo que signifique alguma coisa para alguém de alguma maneira ou capacidade"* (Peirce, 1931-1958). Esta definição destaca a interpretação como um componente fundamental na semiótica e, por conseguinte, na teoria da EngSem (de Souza, 2005). Signos na interface de um sistema podem ser, por exemplo, a figura de impressora no botão de imprimir documento ou o componente de seleção da fonte no Microsoft® Office Word. Quando o designer está projetando a interface de um software, ele define um ou mais sistemas de significação nos quais codificará a sua mensagem, estabelecendo quais os signos que farão parte daquela interface. Depois, quando o usuário está interagindo com o software, ele decodifica a mensagem enviada, interpretando o signos definidos na interface e se comunicando com o preposto do designer por meio do sistema de

significação instituído. Desta maneira, podemos dizer que o processo de significação envolve a produção de signos pelos designers e a interpretação de signos pelos usuários.

Ao iniciar a construção de um artefato de metacomunicação em IHC, o designer deverá ter também outras preocupações em mente. Na EngSem, o espaço de design é estruturado segundo o modelo de comunicação proposto por (Jakobson, 1960), que envolve seis elementos relacionados: o emissor, o receptor, o contexto, o canal, o código e a mensagem. Ao realizar um projeto de IHC, o designer deverá então conhecer (de Souza, 2005 p. 87): (i) quem é o usuário ao qual ele enviará a sua mensagem, segundo a sua interpretação; (ii) qual o contexto no qual a metacomunicação designer-usuário ocorrerá; (iii) qual o código que deverá ser utilizado durante a comunicação; (iv) quais os canais disponíveis para que a metacomunicação designer-usuário ocorra e como eles poderão ser utilizados; e (v) qual a mensagem que ele (designer) deseja transmitir aos usuários. Seguindo este modelo, a EngSem define um artefato computacional como o canal pelo qual a mensagem codificada pelo designer é enviada aos usuários e por onde a troca de mensagens decorrentes da comunicação usuário-preposto ocorre. O modelo para o espaço de design definido pela EngSem é apresentado na Figura 2.

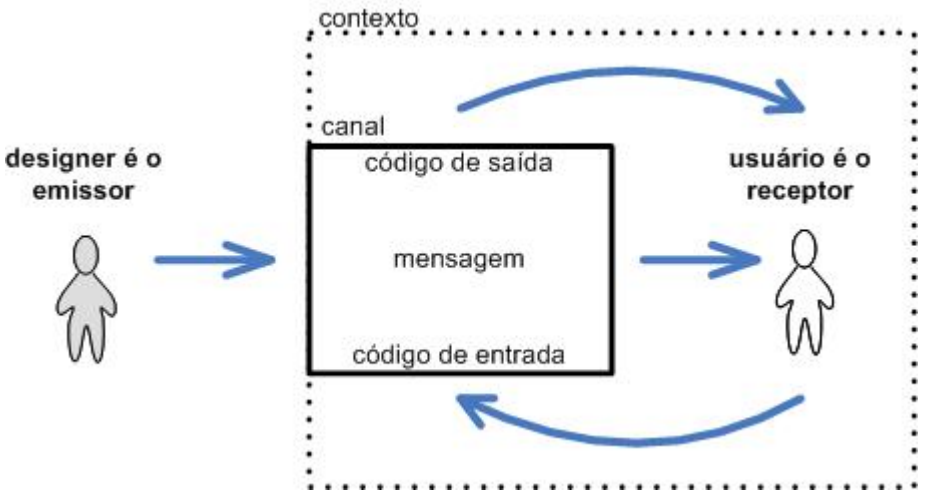

Figura 2 – Espaço de design segundo a EngSem.

Para auxiliar a reflexão do designer durante o processo de construção de um artefato de metacomunicação em IHC, a EngSem oferece algumas ferramentas,

denominadas de ferramentas epistêmicas (de Souza, 2005). Em seu livro, de Souza define uma ferramenta epistêmica como *"one that is not used to yield directly the answer to the problem, but to increase the problem-solver's understanding of the problem itself and the implications it brings about"* (ibid., p. 33). Conseqüentemente, o uso de tais ferramentas auxilia o designer no processo de reflexão envolvido no projeto da interação, possibilitando a elaboração e avaliação de diferentes soluções candidatas antes que opte pela solução que julgue ser a mais adequada ao problema em questão. A seguir apresentaremos duas ferramentas epistêmicas fundamentadas na EngSem, o sistema de ajuda segundo a EngSem e a linguagem para a modelagem da interação MoLIC.

2.2.1. Sistemas de ajuda

Com base na teoria da EngSem, Silveira (2002) apresentou uma proposta para o projeto e construção do sistema de ajuda. Nesta proposta, a autora defende que *"a melhor alternativa que o preposto tem de realizar a metacomunicação é o sistema de ajuda"* (Silveira, 2002 p. 58), pois através dele o designer pode enviar diretamente ao usuário a sua mensagem, apresentando de maneira explícita todas as decisões que foram tomadas por ele durante o projeto do sistema, bem como a justificativa para estas decisões.

Para que a comunicação designer-usuário através do sistema de ajuda ocorra, é proposto o conjunto de expressões de comunicabilidade apresentado na Tabela 1. Derivadas do método de avaliação de comunicabilidade (de Souza, 2005) e das dúvidas mais freqüentes dos usuários durante a interação com um sistema (Sellen e Nicol, 1990; Baecker et al., 1995 *apud* Silveira, 2002), estas expressões representam possíveis falas do usuário durante sua interação, em momentos nos quais a transmissão da mensagem do designer através do seu preposto não foi bem sucedida, ou seja, em um momento em que ocorreu uma falha na comunicação entre o designer e o usuário. As respostas a estas expressões, por sua vez, representam as falas do preposto do designer. Desta

maneira, utilizando as expressões do sistema de ajuda, o usuário pode acessá-lo sempre que necessário para encontrar as respostas aos problemas encontrados.

Tabela 1 – Expressões do sistema de ajuda (reproduzida de (Silveira, 2002 p. 46-48)).

Expressões do sistema de Ajuda	***Falha comunicativa Associada***
A quem isto afeta? De quem isto depende? Quem pode fazer isto?	O usuário quer saber se, executando determinada tarefa, a quem (determinados papéis de usuário) ele vai afetar ou de quem esta tarefa dependerá para sua realização. No outro caso (“Quem pode fazer isto?”), ele pode querer saber quais papéis estão habilitados a realizar a tarefa em questão.
Como faço isto?	O usuário não sabe como executar determinada tarefa.
E agora?	O usuário não sabe o que fazer como próximo passo na interação ou o usuário não sabe nem determinar a tarefa que necessita realizar.
Epa!	O usuário efetuou uma ação e/ou tarefa indesejada e quer desfazê-la.
Existe outra maneira de fazer isto?	O usuário sabe como realizar a tarefa em questão, mas deseja saber se existem outras possibilidades de caminhos que levem ao mesmo resultado.
O que aconteceu?	O usuário executa determinada ação (acreditando ser a correta para o que deseja realizar) e a resposta esperada não ocorre (ou não obtém a resposta desejada ou não obtém resposta alguma). Ele não consegue entender o aconteceu.

Expressões do sistema de Ajuda	***Falha comunicativa associada***
O que é isto?	O usuário não compreende determinado elemento encontrado na interface.
Onde está?	O usuário sabe o que quer fazer, mas não consegue encontrar o elemento correspondente na interface.
Onde eu estava?	O usuário quer saber "onde ele estava", ou seja, qual a tarefa na qual ele estava trabalhando anteriormente. Ele quer saber seus passos anteriores para entender o estado em que está no momento.
Para que serve isto?	O usuário quer saber a utilidade da tarefa em questão.
Por que devo fazer isto?	O usuário quer saber por que deve fazer determinada tarefa.
Por que não funciona?	O usuário executa determinada ação que ele acredita ser a necessária no momento, e não obtém a resposta desejada. Ele tenta a mesma opção mais de uma vez, porque está convencido de estar fazendo a coisa certa.
Socorro!	O usuário quer um maior detalhamento dos dados de ajuda.

Em seu trabalho, Silveira propõe que o conteúdo da mensagem que será enviada através do sistema de ajuda deve ser capturado gradativamente durante todo o projeto do sistema, evitando a perda do *rationale* das decisões tomadas pelo designer. Para tanto, ela sugere que sejam utilizados diversos modelos de design de IHC, preenchidos em períodos distintos do projeto do sistema. Segundo ela, estes modelos representam uma parte do conhecimento apresentado pelo

sistema de ajuda, e a sua construção e utilização facilita a coleta de informações requeridas para a elaboração do sistema de ajuda. Além de serem utilizadas para a construção do sistema de ajuda de determinada aplicação, estas expressões e informações obtidas são utilizadas também para guiar a discussão da equipe de designers durante todo o processo (Silveira, 2002).

2.2.2. MoLIC

Com intuito de apoiar a modelagem da interação como uma conversa, como definido na teoria da EngSem, foi proposta a linguagem MoLIC (Paula, 2003; Silva, 2005), acrônimo para *Modeling Language for Interaction as Conversation*. Assim como o sistema de ajuda, este modelo de interação constitui-se em uma ferramenta epistêmica, apoiando as reflexões do designer acerca do processo de interação humano-computador em desenvolvimento, permitindo que todas as prováveis conversas que ocorrerão entre os designers e os usuários durante a interação com o sistema sejam modeladas.

A MoLIC é composta por quatro artefatos inter-relacionados (Silva, 2005, p. 51; Silva e Barbosa, 2007): (1) um diagrama de metas, (2) um esquema conceitual de signos e (3) um diagrama de interação complementado por uma (4) especificação textual. O diagrama de metas fornece uma organização hierárquica de todas as metas que o usuário deverá alcançar com o sistema sendo projetado. O esquema conceitual de signos propõe a definição e organização dos conceitos que estão envolvidos no domínio do sistema, incluindo artefatos ou informações que fazem parte das atividades realizadas pelo usuário. O diagrama de interação, por sua vez, permite representar de maneira simples a conversa que se dará entre o usuário e o preposto do designer para que o primeiro alcance as metas identificadas. Finalmente, a especificação textual possibilita um maior detalhamento das conversas que foram definidas no diagrama de interação.

Silva propôs, na segunda edição da MoLIC, que um processo para o projeto da interação humano-computador que utilize este modelo poderá ser realizado em duas etapas: uma etapa de estruturação da conversa seguida de uma etapa de

detalhamento da conversa, cada uma delas com foco e nível de abstração distintos. Na primeira etapa do projeto da interação proposto, o designer é levado a refletir sobre as seguintes questões de projeto: todas as possíveis conversas usuário-preposto, a recuperação de *breakdowns*, consistência do discurso interativo (padrões de interação) e caminhos de interação alternativos. Através destas questões, a modelagem é proposta em um nível de abstração capaz de fornecer a visão global da interação do sistema, tal como será percebida por um usuário. Nessa etapa, utiliza-se a MoLIC de uma maneira mais simplificada do que o poder de expressão da linguagem completa permite, fornecendo como resultado apenas o diagrama de interação ainda em uma versão reduzida. Esta visão global é de extrema importância para a construção de um discurso coeso e consistente do preposto por diferentes membros de uma equipe de projeto da interação (Paula et al., 2005).

É na segunda etapa, de detalhamento da conversa usuário-preposto modelada anteriormente, que o designer obtém uma visão detalhada da interação. Nesta etapa, são especificados os diálogos usuário-preposto, e são tratadas questões como a estruturação dos diálogos travados em determinado momento, a ocorrência de restrições no rumo da conversa em função dos diálogos definidos, a definição e estruturação dos signos envolvidos nos diálogos e a classificação dos mecanismos de reparo de *breakdowns*. Desta maneira, o designer deverá estar preocupado com a ordem em que os diálogos ocorrerão em determinada conversa ou com a dependência de execução de determinado conjunto de diálogos a partir de outro, por exemplo. Durante o detalhamento da conversa, o designer complementa o esquema conceitual de signos iniciado na primeira etapa e produz a especificação textual, onde armazena os detalhes da modelagem da conversa usuário-preposto. Posteriormente, os resultados obtidos nesta etapa poderão ser utilizados como insumo para outras fases do projeto da interação e para a implementação da interface.

No Capítulo 4, veremos como as expressões de comunicabilidade propostas no sistema de ajuda e a linguagem de modelagem de interação MoLIC serão utilizados para a definição do processo proposto nesta dissertação.