Estadogramas no Desenvolvimento de Interfaces

Fábio Nogueira de Lucena (fabio@dcc.unicamp.br)
Hans Liesenberg (hans@dcc.unicamp.br)

DCC/IMECC/UNICAMP
Caixa Postal 6065
13081-970 Campinas/SP, Brasil

Relatório Técnico DCC–07/93

Abril de 1993

Estadogramas no Desenvolvimento de Interfaces

Fábio Nogueira de Lucena (fabio@dcc.unicamp.br)
Hans Liesenberg (hans@dcc.unicamp.br)

DCC/IMECC/UNICAMP
Caixa Postal 6065
13081-970 Campinas/SP, Brasil

Sumário

Enquanto é reconhecida a relevância de interfaces homem-computador em sistemas interativos, inclusive comercialmente, o software correspondente é difícil de ser construído. Várias ferramentas têm sido propostas para automatizar a geração de código de uma interface. Grande parte destas ferramentas são baseadas em estados e usam diagramas de transição de estados (DTEs) para especificação do controle de diálogo. Contudo, estes diagramas apresentam inconvenientes que os tornam inviáveis para a especificação de diálogos complexos.

Estadogramas (*statecharts*) apresentam indícios de serem adequados para descrever este tipo de comportamento. Estendem os DTEs e eliminam inconvenientes dos últimos. Embora seu emprego seja sugerido em vários trabalhos para a especificação de controle de diálogo, os resultados de experiências na utilização desta notação para este emprego particular (como ocorre em outras áreas) não são discutidos na literatura.

O uso dos estadogramas no desenvolvimento de uma interface real permitiu identificar mudanças que tornam estes diagramas mais apropriados para este emprego específico bem como limitações desta notação. Entre adaptações sugeridas encontram-se construções adicionais para permitir a especificação da apresentação

de uma interface. A observação do código gerado por uma ferramenta utilizada neste trabalho, que implementa estadogramas, ainda permitiu identificar elementos desejáveis quanto a estrutura do código a ser produzida. Este texto relata as mudanças sugeridas e limitações desta notação obtidos da análise deste desenvolvimento empírico. Ainda inclui comentários acerca de elementos desejáveis na estrutura do código a ser prodizida por uma implementação desta notação.

1 Introdução

Diante da relevância inegável do papel das interfaces homem-computador[1] (interfaces, por simplicidade) no sucesso de sistemas interativos, muitas pesquisas têm abordado a árdua tarefa de construí-las. Segundo Myers [15] o código de uma interface é volumoso, complexo e sua construção não é trivial. O intuito das pesquisas é reduzir os custos de desenvolvimento e melhorar a qualidade das interfaces produzidas. Grande parte dos trabalhos abordam o problema através do emprego de ferramentas.

As ferramentas visam fornecer suporte às etapas de desenvolvimento de interfaces: projeto, implementação, avaliação e manutenção.[2] Geralmente tem-se uma ferramenta orientada para cada atividade distinta de desenvolvimento. Quando as ferramentas estão integradas, o conjunto geralmente é denominado de UIMS (*User Interface Management System*), UIDS (*User Interface Development System*) ou ainda UIDE (*User Interface Development Environment*). Neste texto usa-se o termo "ferramentas" para caracterizar tais sistemas.

Uma abordagem comum na construção de ferramentas é criar uma técnica de descrição ou linguagem de alto nível (assistida pela ferramenta) para descrever a atividade de interesse. Existem várias técnicas para especificação e implementação de controle de interfaces, i.e., técnicas

[1] Neste trabalho interface homem-computador é a parte do software de um sistema interativo responsável por traduzir ações do usuário em ativações das funcionalidades do sistema, bem como permitir que os resultados da aplicação (funcionalidade) possam ser observados e coordenar esta interação.

[2] Detalhes sobre o ciclo de vida de uma interface podem ser obtidos em [1].

para descrição e implementação da sintaxe permitida das ações do usuário, das reações do computador e de como o diálogo[3] evolui ao longo do tempo. As técnicas, contudo, ainda apresentam inconvenientes. Myers em [14] comenta as mais comuns, por exemplo, os DTEs. Neste trabalho é focalizada uma extensão dos DTEs: os estadogramas.[4]

Estadogramas se enquadram no modelo de redes de transição conforme três modelos descritos e comentados por Green em [2]. Os outros dois modelos são *gramáticas* e *eventos*. Embora Green conclua que o modelo de eventos possua maior poder de expressão, cada um deles possui suas próprias vantagens e desvantagens [14]. Os problemas geralmente relacionam-se à inabilidade de representar alguns aspectos, à dificuldade de uso e, em muitos casos, nem todos os tipos de interfaces podem ser especificadas.

Entre as notações que fazem parte de redes de transição, os estadogramas gozam de posição confortável reconhecida em vários trabalhos [1, 21]. Isto deve-se, sobretudo, às extensões sobre os DTEs que eliminam inconvenientes destes últimos enquanto retêm a natureza gráfica [3, 4, 21]. Convém ressaltar que DTE é a base das notações de redes de transição. Ainda é interessante destacar a inexistência de consenso acerca de uma linguagem para especificação de controle de diálogo. Apesar dos indícios favoráveis, tem-se pouco conhecimento sobre este emprego em particular. Neste sentido, este trabalho assemelha-se ao realizado por Wasserman [20] para os DTEs. Naquele trabalho Wasserman estende os DTEs para adequá-los à especificação de interfaces. Da mesma forma são propostas, neste texto, algumas mudanças nos estadogramas. Entretanto, neste trabalho estamos ressaltando principalmente os resultados de um desenvolvimento real que envolveu sua especificação em estadogramas seguida da implementação de uma interface não trivial.

Os resultados visam subsidiar a construção de ferramentas de apoio

[3] Comunicação existente entre o ser humano e o sistema de computação — é a troca de símbolos e ações entre o usuário e o computador.

[4] Diagramas propostos para descrição do comportamento de sistemas reativos. Estendem os diagramas de transição de estados com concorrência, comunicação (*broadcast*) e hierarquia.

à construção de interfaces[5] que venha considerar estadogramas como sua linguagem central. As dificuldades relatadas registram inconvenientes a serem eliminados e extensões a serem realizadas para tornar propício o emprego desta notação. Especializações desta notação não é um fato novo. Johnson **et al.** [12], por exemplo, ressaltam a importância dos formalismos visuais como estadogramas e exemplificam especializações desta notação.

Este texto assume que o leitor possui conhecimentos básicos sobre estadogramas. Está além do escopo apresentar uma descrição completa dos estadogramas e exemplos do seu uso. Uma descrição exaustiva é fornecida em [3]. A semântica formal pode ser obtida em [6] e [16]. Uma descrição didática e informal pode ser encontrada em [9]. Entre ambientes que empregam estadogramas podem ser citados Statemaster [21], Statemate [5] e Reacto [13].

Abaixo comenta-se sucintamente o modelo de Seeheim (muito utilizado no âmbito de interfaces). Na seção seguinte segue um breve comentário acerca da interface desenvolvida para servir de "bancada" para experiências com os estadogramas. Na seção 3 são descritos os resultados do emprego de estadogramas no desenvolvimento desta interface e algumas mudanças sugeridas. Por último, as considerações finais encerram este trabalho.

Modelo de Seeheim

Nosso trabalho baseia-se no modelo de Seeheim. O modelo de Seeheim [2] (fig. 1) permite identificar vários componentes que conceitualmente compõem uma interface. A componente de apresentação é responsável por produzir a saída para o usuário e receber as entradas fornecidas. O controle de diálogo é o foco deste trabalho bem como a sua descrição na forma de estadogramas. Este componente coordena a seq:uência da in-

[5] Hartson e Hix [7] comentam várias ferramentas desenvolvidas para a construção de interfaces.

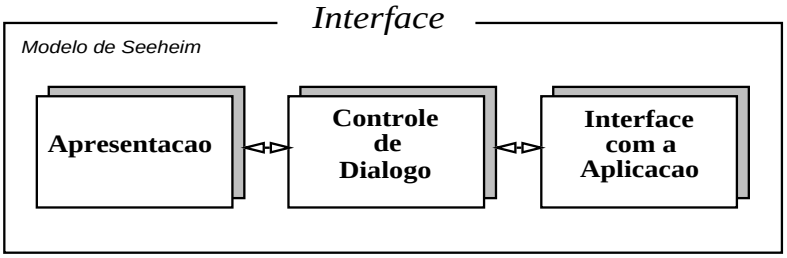


Figura 1: Um modelo de arquitetura de interface.

teração do usuário com o computador (equivale à sintaxe da interação).[6] O componente mais a direita corresponde à semântica da interface. Engloba a visão que a interface tem da aplicação e vice-versa.

2 A interface para experimentação

Foi desenvolvida uma interface para um sistema de composição musical a partir de requisitos estabelecidos em outro trabalho. Uma característica peculiar do sistema é a utilização de árvores para representar composições. Estas árvores substituem, portanto, as partituras convencionais. O objetivo do sistema é auxiliar o compositor no processo de composição musical. Está além do escopo deste trabalho descrever detalhadamente os aspectos de controle da interface desenvolvida, bem como a aplicação subjacente. Abaixo seguem as informações estritamente relevantes para este texto.

A comunicação entre o sistema e a interface dá-se através de um protocolo. Isto conduz a um acoplamento fraco entre o núcleo do sistema e a interface. A interface permite a coexistência de vários estilos de interação, inclusive manipulação direta. O desenvolvimento da mesma seguiu tanto quanto possível isolado do desenvolvimento da aplicação.

[6] Convém ressaltar que estadograma é uma notação para descrever fluxos de controle orientados a evento.

Buscou-se a independência entre estes componentes do sistema interativo, ou seja, independência de diálogo. A interface engloba um editor de árvores, que podem ser editadas com o uso de um mouse. Vários comandos podem ser disparados clicando-se sobre ícones ou selecionando opções de menu. Ainda fornece uma linguagem de comandos para usuários especialistas. A interface é abrangente e permitiu uma avaliação preliminar do uso de estadogramas na descrição do seu comportamento.

3 Adequação de estadogramas no contexto de interfaces

Esta seção relata várias sugestões de mudanças e limitações[7] na capacidade de expressão dos estadogramas identificados durante o desenvolvimento de uma interface real. Dos itens abaixo alguns incluem sugestões para eliminar as dificuldades encontradas. Algumas dessas sugestões foram derivadas de facilidades e construções encontradas em sistemas de apoio à construção de interfaces com abordagens distintas da adotada neste trabalho.

[7] As limitações detectadas devem-se, principalmente, ao fato de que estadogramas foram originalmente concebidos para descrever uma *classe* de sistemas conhecidos por sistemas reativos. Interfaces são sistemas reativos, embora com suas peculiaridades e características próprias, que nem sempre são encontradas em sistemas reativos de uma maneira geral. Como esta notação foi proposta para descrever conceitos comuns aos sistemas reativos, é natural que estes casos particulares não sejam previstos na notação original.

3.1 Modo[8]

Podemos identificar duas dificuldades relacionadas ao conceito de modo:

- Conforme Myers [14], uma das dificuldades com interfaces que usam manipulação direta é a inexistência de modos. Myers afirma que o usuário pode fornecer qualquer comando praticamente a qualquer momento. É importante ressaltar que estadogramas modelam o comportamento como um conjunto de estados (modos) e transições entre eles. Logo, como especificar tal interface se a noção de modos (estados) inexiste?

  Felizmente Jacob [11] afirma que, apesar da aparência, manipulação direta é altamente modal! Ainda mostra a estrutura de co-rotinas existente em interfaces que usam este estilo de interação e como identificar estados nestas interfaces. Um objeto de interação na linguagem definida por Jacob pode ser vista na figura 2. Nota-se que a seção **TOKENS** (fig. 2) nada mais é que a relação entre eventos lógicos e físicos, mapeamento de responsabilidade do componente de apresentação (modelo de Seeheim) visto anteriormente.

  Estadogramas fornecem mecanismos apenas para a descrição de sintaxe. Uma deficiência óbvia relaciona-se ao aspectos léxicos (apresentação) de uma interface. Não há como fornecer um mapeamento entre eventos físicos e lógicos, nem como exibir um simples caractere, por exemplo. Aspectos léxicos, contudo, podem coexistir isoladamente de sintaxe [18].

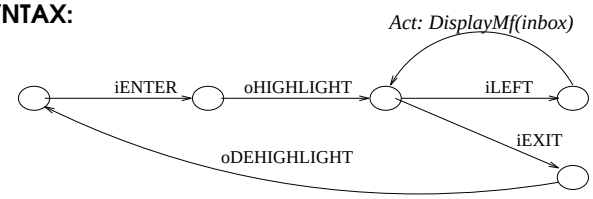
- Outra questão relativa a modo diz respeito ao conceito de estado modelado em estadograma. Por exemplo, o estado **User Protocol**, fig. 3 (i), contém um subestado **User Checker**. Este subestado, no entanto, modela um processo: verificar a existência do

[8] **Modo** é o estado ou coleção de estados nos quais apenas um subconjunto de todas as possíveis tarefas de interação com o usuário podem ser realizadas, i.e., em modos distintos têm-se diferentes tarefas que podem ser realizadas. O uso de modo ainda permite que seja dado significado diferente para uma mesma entrada, conforme o modo corrente.

INTERACTION_OBJECT MessageFileDisplayButton IS
IVARS:
position := { 100,200,64,24 }; --i.e., coordinates of screen rectangle
METHODS:
Draw() { DrawTextButton(position,"Display"); }
TOKENS:

iLEFT	{ click left mouse button }
iENTER	{ locator moves inside rectangle given by position }
iEXIT	{ locator moves outside rectangle given by position }
oHIGHLIGHT	{ invert video of rectangle given by position }
oDEHIGHLIGHT	{ same as oHIGHLIGHT }

SYNTAX:



end INTERACTION_OBJECT;

Figura 2: Objeto de interação [11].

usuário. O evento *Illegal Username* pode ser visto naturalmente como o retorno de uma função `NameChecker()` que recebe como entrada *Username*. Neste caso um estado foi utilizado para modelar um processo invisível ao usuário. A figura 3 (*ii*) é uma modelagem alternativa e mais substancial que a anterior para o usuário, pois reflete o que o usuário percebe da interação. A última modelagem é consistente com as propriedades consideradas desejáveis em uma linguagem para especificação de interfaces segundo [10]. Ambas as alternativas são possíveis, contudo, só uma definição mais rigorosa de estado poderá conduzir a especificações mais homogêneas.

3.2 Reutilização de comportamentos

Embora a especificação em estadogramas possa seguir *bottom-up* ou *top-down* e paralela à especificação de outros aspectos do sistema (p.ex.,

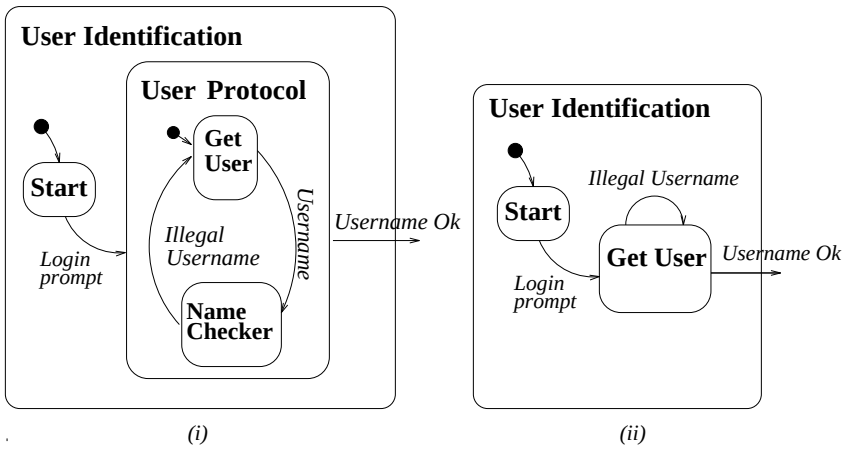


Figura 3: Modo em estadogramas.

funcional) e por pessoas distintas, é preciso que estes recursos não sejam exclusivos de uma especificação particular.

O tratamento de forma monolítica de um estadograma impede a existência de bibliotecas de comportamentos que possam ser reutilizados no desenvolvimento de outras especificações.

Usando-se uma linguagem de programação convencional pode-se criar rotinas e colocá-las em uma biblioteca para uso posterior, sem a necessidade de se ter acesso ao código destas rotinas ou refazê-las. Da mesma forma é preciso estabelecer um meio de reutilizar comportamentos sem a necessidade de termos acesso a sua especificação, mas apenas à "interface" de um estado. Estes comportamentos reutilizáveis podem corresponder à sintaxe de objetos de interação [1] comumente encontrados em sistemas interativos. Por exemplo, poder-se-ia ter diálogos para procurar por arquivos, alterar parâmetros de impressão, selecionar cores e assim por diante. Todos estes comportamentos têm objetivos bem definidos e podem ser aplicáveis em várias situações.

3.3 Alteração dinâmica do comportamento

O desenvolvimento interativo de protótipos é uma técnica amplamente empregada na construção de interfaces [15]. Para dar suporte a esta atividade cíclica, sistemas de apoio à construção de interfaces devem permitir que o comportamento seja alterado dinamicamente e que os resultados sejam refletidos imediatamente. Usando-se estadogramas é preciso estabelecer uma semântica para alterações realizadas. Por exemplo, ao simular uma especificação o que ocorre se removermos um estado corrente?

Construção de protótipos não é a única aplicação da mudança dinâmica de uma especificação. Tratadores de eventos[9] podem ser criados e destruídos dinamicamente. Várias instâncias de um mesmo tratador podem estar ativas simultaneamente. Para os estadogramas ter-se-ia que permitir a existência de várias "instâncias" de um estado. Neste sentido uma especificação em estadogramas seria análoga a um tipo em uma linguagem de programação convencional.

Qualquer editor que permite a edição simultânea de vários arquivos, por exemplo, mostra a necessidade destes recursos para que possa ser especificado apropriadamente. Parece mais natural imaginar que se tem múltiplas instâncias da especificação de um editor. Operações como *cut* e *paste*, por exemplo, entre arquivos em edição seriam descritas como troca de eventos entre instâncias de um mesmo estado.

3.4 *Undo*

Sistemas interativos geralmente fornecem uma maneira do usuário reparar uma operação disparada por engano ou erro. Isto estimula o uso do sistema! *Undo* é um recurso freq:uentemente utilizado e os estadogramas devem possuir uma construção para facilitar a sua especificação.

Parece razoável que após um *undo* os estadogramas retornem à última configuração em que estiveram antes do disparo mais recente de uma

[9] São elementos básicos do modelo de eventos que respondem à ocorrência de determinado evento. Veja [2] para mais detalhes.

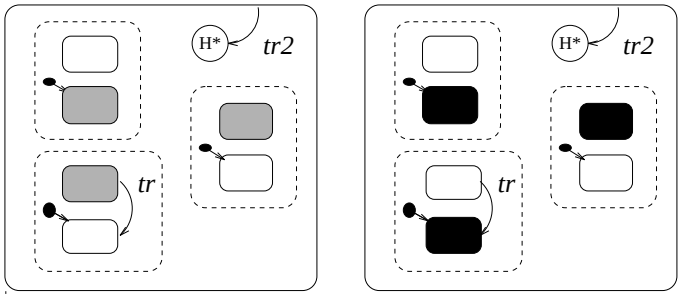


Figura 4: Uma proposta para *undo* (undo(tr) ≡ tr2 + ações específicas).

transição cujo efeito deseja-se desfazer.

Um *undo* geralmente envolve ações específicas a serem realizadas tanto pela aplicação (semântica) quanto pelo componente de apresentação (léxica). A saída (resposta) dos estadogramas é sempre na forma de ações. Portanto, esta é a única maneira de como os estadogramas se comunicam com o exterior ou o modificam. Em conseq:uência, uma maneira natural de se observar esta situação é permitir que sejam especificadas ações de *undo* para cada transição. Assim, sempre que um *undo* ocorrer, as ações necessárias serão executadas e o estadograma se estabiliza na última configuração obtida.

Em cinza (fig. 4) tem-se a configuração dos estadogramas antes da transição *tr*. Em preto tem-se a próxima configuração como conseq:uência da transição *tr*. Após a entrada em todos os estados em preto o sistema se estabiliza. Neste instante, se ocorrer um *undo* a próxima configuração volta a ser a cinza. Esta configuração é a mesma configuração obtida pela transição *tr2* devido ao *history* H^* (exceto se houver um cancelamento de *history* nos níveis inferiores) juntamente com as ações especificadas exatamente para estas situações.

3.5 Escopo de eventos no disparo de transições

A figura 5 registra parte de uma especificação em estadogramas. Supondo uma configuração em determinado instante como sendo ativos os

estados **A**, **B**, **C** e **D** e que o usuário requisite informações de ajuda (ou seja, gera o evento *F1* possivelmente através da tecla F1) causando a transição do estado **D** para o estado **Help**. Geralmente isto equivale a uma caixa de diálogo que exibe informações sobre determinado comando. Para prosseguir com a operação do sistema (retornar ao estado anterior e retirar a caixa de diálogo de ajuda da tela) é necessário que o usuário saia do estado de ajuda. Na especificação isto é feito gerando-se o evento ESC. Contudo, observe-se a transição rotulada com o evento ESC do estado **B** para **End**. Estas transições são conflitantes, pois ora uma deve ser seguida e ora outra, mas deveriam ser mantidas por causa do princípio de *consistência*.[10]

Neste caso particular temos o fim da ajuda e a finalização do sistema provocados pelo mesmo evento ESC. Da mesma forma espera-se que toda e qualquer caixa de diálogo possa ser removida com a tecla ESC, bem como menus e outros objetos de interação.

Para eliminar este problema pode-se associar um escopo a eventos. Caso surja uma situação conflitante poder-se-ia estabelecer que seria disparada a transição que tivesse origem no estado ativo de nível hierárquico mais baixo. No caso particular a transição com origem no estado **Help** e não a transição com origem no estado **B** seria disparada.

Outras situações interessantes podem ser obtidas com este conceito. Pode-se definir, por exemplo, estados cujo único objetivo é restringir o efeito de alguns eventos, i.e., inibi-los.

3.6 Transição "complemento" e pseudo transição

A necessidade de especificar todas as interações possíveis usando transições entre estados constitui um dos inconvenientes dos estadogramas. Uma forma de amenizar este problema é fornecer uma espécie de transição "complemento" que sempre é percorrida quando a entrada do usuário não corresponder a nenhum evento que afete a configuração corrente.

[10] O projeto de interfaces ainda não conta com "regras de ouro," contudo, alguns princípios [17] são consensuais como consistência que busca dar um tratamento homogêneo para atividades afins.

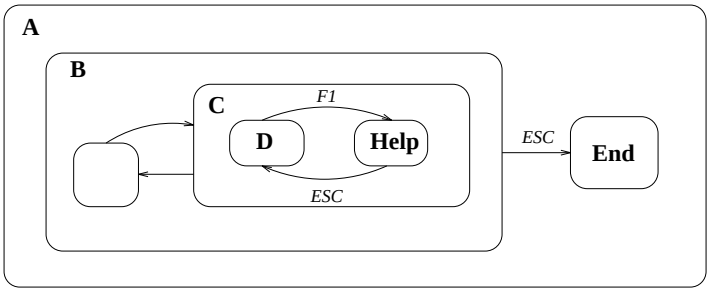


Figura 5: Definindo contexto para eventos.

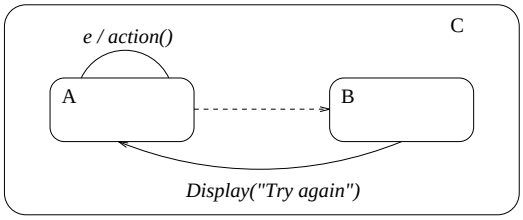


Figura 6: Pseudo e “complemento” transições.

A figura 6 exemplifica uma transição "complemento." A transição tracejada é deste tipo. Sempre que o estado **A** estiver ativo e não possuir transição para tratar um evento gerado, percorre-se a transição tracejada conduzindo ao estado **B**.

Há casos onde não se deseja uma mudança de contexto mas espera-se que determinado evento seja tratado. Por exemplo, uma operação como "redesenhe" geralmente não provoca uma mudança de estado. Simplesmente espera-se que determinado desenho seja refeito. Nos estadogramas originais não há como registrar este fato sem provocar transições entre estados. Na figura 6 o arco não orientado ligando o estado **A** a si próprio permite refletir esta situação. Neste caso, sempre que o estado **A** estiver ativo e ocorrer o evento e, a ação `action()` é executada sem provocar mudança de contexto, i.e., sem uma conseq:uente desativação e ativação de **A**.

4 Considerações finais

Este artigo teceu observações sobre a notação dos estadogramas para a especificação e implementação de diálogo conforme propostos originalmente. Parte delas decorrem do confronto entre o que a literatura considera necessário e o que fornecido pelos estadogramas como linguagem de especificação, e as demais pelas dificuldades com o desenvolvimento de uma interface real.

As observações permitem concluir que novos mecanismos precisam ser acrescidos bem como se faz necessárias algumas mudanças na notação para o seu uso adequado para especificação de diálogo. Construções como *undo* e acréscimo de abstrações de recursos léxicos são alguns exemplos. Do ponto de vista de implementação viu-se que a operação concorrente e assíncrona dos componentes de um sistema interativo permite maior flexibilidade, já que evita estrutura rígida da arquitetura mestre/escravo típica das arquiteturas que se utilizam de funções *callback*. Algumas das dificuldades identificadas foram seguidas por propostas na tentativa de eliminá-las. Entretanto, espera-se que venham a ser analisadas com mais profundidade em trabalho futuro, pois este em particular restringiu-se a

observar os estadogramas como propostos originalmente.

Novas extensões estão sendo detalhadas e visam contemplar construções para a especificação de diálogos multi-usuários em sistemas distribuídos. Uma das diretrizes deste trabalho é preservar a natureza gráfica da notação.

Referências

[1] Len Bass and Joëlle Coutaz. *Developing Software for the User Interface*. SEI Series in Software Engineering. Addison-Wesley Publishing Company, Inc., 1991.

[2] Mark Green. A Survey of Three Dialogue Models. *ACM Transactions on Graphics*, 5(3):244–275, July 1986.

[3] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[4] D. Harel. On Visual Formalism. *Communications of the ACM*, 31(5):514–530, May 1988.

[5] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and A. Shtul-Trauring. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering*, April 1990.

[6] D. Harel, A. Pnueli, J.P. Schmidt, and R. Sherman. On the Formal Semantics of Statecharts. *Proceedings of 2nd. IEEE Symposium on Logic in Computer Science*, pages 54–64, 1987.

[7] H. Rex Hartson and Deborah Hix. Human-Computer Interface Development: Concepts and Systems for its Management. *ACM Computing Surveys*, 21(1):5–92, March 1989.

[8] Rex Hartson. User-Interface Management Control and Communication. *IEEE Software*, pages 62–70, January 1989.

[9] i Logix Inc. STATEMATE: Semantics of statecharts, August 1989. Burlington, MA.

[10] Robert J. K. Jacob. Using Formal Specifications in the Design of a Human-Computer Interface. *Communications of the ACM*, 26(4):259–264, April 1983.

[11] Robert J. K. Jacob. A Specification Language for Direct-Manipulation User Interfaces. *ACM Transactions on Graphics*, 5(4):283–317, October 1986.

[12] Jeff A. Johnson, Bonnie A. Nardi, Craig L. Zarmer, and James R. Miller. Ace: Building Interactive Graphical Applications. *Communications of the ACM*, 36(4):41–55, April 1993.

[13] Limei Gilham, Allen Goldberg, and T. C. Wang. Toward Reliable Reactive Systems. *ACM SIGSOFT Engineering Notes*, 14(3):68–74, May 1989.

[14] Brad A. Myers. User-Interface Tools: Introduction and Survey. *IEEE Software*, pages 15–23, January 1989.

[15] Brad A. Myers and Mary Beth Rosson. Survey on User Interface Programming. In *Human Factors in Computing Systems CHI'92 Conference Proceedings*, pages 195–202, Monterey, California, May 1992.

[16] A. Pnueli and M. Shalev. What is in a Step: On the Semantics of Statecharts. In *Lecture Notes in Computer Science*, pages 244–264. Springer-Verlag, 1991. Number 526.

[17] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley Publishing Company, Inc., 1987.

[18] Gurminder Singh. *Automating the Lexical and Syntactic Desigh of Graphical User Interfaces.* PhD thesis, University of Alberta, Edmonton, Alberta, 1989.

[19] H. W. Six and J. Voss. User Interface Development: Problems and Experiences. *Lecture Notes in Computer Science*, 555:306–319, June 1991.

[20] Anthony I. Wasserman. Extending State Transition Diagrams for the Specification of Human-Computer Interaction. *IEEE Transactions on Software Engineering*, SE-11(8):699–713, August 1985.

[21] Pierre D. Wellner. Statemaster: A UIMS based on Statecharts for Prototyping and Target Implementation. In *Human Factors in Computing Systems, Proceedings SIGCHI'89*, pages 177–182, Austin,TX, April 1989.

Relatórios Técnicos – 1992

01/92 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

02/92 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

03/92 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

04/92 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

05/92 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

06/92 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

07/92 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

08/92 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

09/92 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

10/92 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

11/92 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

12/92 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

01/93 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

02/93 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

03/93 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

04/93 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

05/93 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

06/93 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

07/93 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

08/93 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

09/93 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

10/93 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

11/93 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

12/93 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

13/93 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

14/93 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

15/93 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

16/93 **$\mathcal{LL}$ – An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

17/93 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

18/93 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

`reltec@dcc.unicamp.br`