Facilitando o Desenvolvimento de Controle Complexo Usando Xchart

Fábio Nogueira de Lucena
Hans Kurt Edmund Liesenberg

Relatório Técnico IC–98-29

Agosto de 1998

Facilitando o Desenvolvimento de Controle Complexo Usando Xchart

Fábio Nogueira de Lucena* Hans Kurt Edmund Liesenberg†

Resumo

Um dos componentes de um sistema interativo cujo desenvolvimento apresenta maiores desafios é o gerenciador de diálogo. A especificação, o projeto, a implementação e a execução deste componente ainda são tarefas cujas ferramentas disponíveis praticamente não apresentam suporte, ao contrário do sucesso obtido por elas no desenvolvimento de outros componentes (a apresentação, por exemplo). Este trabalho descreve uma proposta para desenvolvimento de gerenciadores de diálogos composta pela linguagem Xchart e um ambiente de apoio. Xchart é uma linguagem visual e formal projetada para adequadamente capturar o controle complexo das funções de gerenciadores de diálogos, que podem estar organizados em uma arquitetura de software baseada em múltiplos agentes. Nesta proposta, controle complexo descrito através de diagramas executáveis eliminam a tarefa manual de gerar o código que o implementa e oferece suporte ao modelo de múltiplos agentes.

1 Introdução

A interface homem-computador (interface, por simplicidade) é o componente de um sistema interativo com o qual o usuário interage. Parte significativa dos custos de desenvolvimento de um sistema interativo é oriunda da definição e da confecção do volumoso e complexo código deste componente [21]. Ferramentas são comumente empregadas para reduzir tais custos. Entretanto, aquelas disponíveis não contemplam todos os tipos de interfaces e facilitam apenas a confecção de um subconjunto das funções de uma interface.

Interfaces complexas, ou seja, com comportamentos complexos e nas quais o usuário pode conduzir mais de uma tarefa concorrentemente, por exemplo, não são adequadamente consideradas pelas atuais ferramentas. Em tais interfaces, o gerenciador de diálogo é o componente que apresenta mais dificuldades de desenvolvimento. As ferramentas amplamente utilizadas têm, por outro lado, evitado tais dificuldades dando grande ênfase em outro componente: a apresentação.

Além das restrições funcionais das atuais ferramentas, os recursos oferecidos são de baixo nível e são usufruídos, na grande maioria dos casos, apenas por programadores especializados. Várias questões não são contempladas por tais ferramentas [23]. O presente trabalho

*Instituto de Informática (UFG), Caixa Postal 131, 74000-010 Goiânia/GO, fabio@inf.ufg.br
†Instituto de Computação (Unicamp), Caixa Postal 6175, 13381-970 Campinas/SP, hans@dcc.unicamp.br

concentra-se em uma destas questões: *apoiar o adequado desenvolvimento de gerenciadores de diálogo (especificação, organização e implementação).* O objetivo é conseguir um sucesso similar ao já obtido com as ferramentas disponíveis para desenvolver o componente de apresentação (reduzindo custos e produzindo interfaces melhores).

Proposta Xchart

O presente trabalho propõe a linguagem Xchart e um ambiente de apoio [15, 18] ao emprego dessa linguagem. A linguagem Xchart foi projetada para capturar a funcionalidade de gerenciadores de diálogos, sendo ela gráfica, formal e executável [16]. Através do ambiente, o usuário de Xchart confecciona e executa descrições nesta linguagem.

O projeto de interação de uma interface envolve a definição da aparência e do comportamento da interface. Parte deste projeto é pertinente ao gerenciador de diálogo. A linguagem Xchart é uma proposta para registrar essa parte do projeto.

Xchart é uma variante de Statechart [8], cujo emprego no domínio de interfaces é visto como promissor e conta com relatos de sucesso. Entretanto, uma análise desse emprego de Statechart, conforme originalmente definida, identificou mudanças que poderiam ser realizadas para melhor contemplar esse domínio [19]. Essas mudanças foram incorporadas em Xchart. Em particular, Statechart é muito utilizada teoricamente e carece de implementações que mostrem efetivamente como esta linguagem pode se integrar com os demais componentes de um sistema. As implicações do emprego de Statechart ainda é desconhecido ao longo do processo de desenvolvimento de um sistema. Apenas a especificação do controle, que é descrito isoladamente de outros componentes é contemplada. Neste sentido, Xchart permite capturar adequadamente gerenciadores de diálogos organizados em múltiplos e pequenos módulos (agentes) e oferece um protocolo de comunicação entre o controle descrito nesta linguagem (oferecido pelo sistema de execução de Xchart) e demais componentes de um sistema.

Especificações na linguagem Xchart são executadas por um interpretador. Não é exigida programação adicional em linguagem de baixo nível (por exemplo, C) ou geração de código passível de ser compilado—processo que inibe a imprescindível avaliação de alternativas para a realização do projeto de uma interface. A tarefa manual de implementar o código correspondente a este componente em linguagens de baixo nível, requerida em muitas das propostas existentes, é substituída pela edição de diagramas, que são automaticamente convertidos em protótipos e rapidamente suscetíveis de serem testados. A complexidade da implementação do gerenciador de diálogo e a conexão deste com os demais componentes não desaparece; ela é transferida para as ferramentas que dão apoio ao emprego de Xchart. Uma versão precursora destas ferramentas [17] deu origem à proposta corrente [18].

Organização do texto

A seção seguinte fornece uma apresentação de arquiteturas de software para sistemas interativos. A organização do software destes sistemas é importante para ilustrar como Xchart contempla as arquiteturas mais empregadas atualmente, ao contrário de outras abordagens, que geralmente não as consideram. A Seção 3 fornece uma breve apresentação da linguagem

Xchart, a Seção 4 descreve os principais itens pertinentes a semântica formal de Xchart. A Seção 5 descreve o ambiente Xchart e a Seção 6 fornece as considerações finais.

2 Software de Interfaces

Um sistema interativo pode ser dividido em dois componentes principais: o componente computacional e o componente interativo (Figura 1). O componente computacional (ou **aplicação**) define a funcionalidade do sistema, ou seja, funções que auxiliam os usuários na realização de suas tarefas. O componente de interação (ou **interface**) é o software responsável pelo diálogo entre o usuário e o sistema: identifica as intenções do usuário, converte-as em requisições para a aplicação e, no sentido inverso, exibe os resultados produzidos pelo componente computacional.

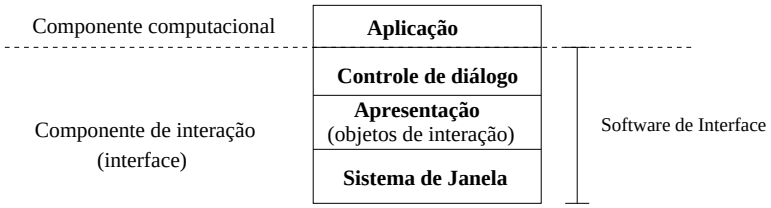

Figura 1: Organização lógica e funcional do software de um sistema interativo [function]

A Figura 1 apresenta uma divisão lógica do código de um sistema interativo de acordo com as funções que ele executa. Essa organização facilita a identificação da parte do software de interfaces cujo desenvolvimento é enfatizado pelas ferramentas atualmente disponíveis:

- Sistema de janelas

  O *sistema de janelas* interage diretamente com o sistema operacional do computador para permitir que a tela seja dividida em regiões distintas (as janelas). Em ambientes Unix, o sistema X WINDOWS é o padrão de fato, enquanto MOTIF WINDOW MANAGER e OPEN LOOK WINDOW MANAGER, por exemplo, são gerentes de janelas sobre este sistema. Os sistemas de janelas oferecem apenas serviços de criação e controle de eventos básicos (mudança de tamanho e de posição, por exemplo) sobre janelas. Os objetos de interação contidos em cada janela são geralmente fornecidos por *toolkits* que utilizam as facilidades e serviços do sistema de janelas subjacente.

- Apresentação (objetos de interação)

  Objetos de interação são elementos de interface que permitem a apresentação de informações manipuladas pela aplicação e recebem a entrada do usuário (em geral através de eventos gerados por teclado ou mouse). Exemplos típicos são botões, menus e campos de texto. Estes elementos, geralmente conhecidos como *widgets*, são fornecidos por toolkits como XVIEW e MOTIF.

- Gerenciador de diálogo

O *gerenciador de diálogo* é responsável pela sintaxe de interação com o usuário, pelo leiaute de objetos de interação e pela comunicação entre a apresentação e a aplicação, inclusive conversões entre dados manipulados por estes componentes. O gerenciador de diálogo ainda é responsável por manter a consistência entre várias apresentações de um mesmo dado. Por exemplo, um gráfico de barra e outro de pizza, representando os mesmos dados, devem ser devidamente atualizados quando estes dados são alterados.

As funções do gerenciador de diálogo estão entre as mais complexas do software de uma interface e o apoio ao desenvolvimento delas apresentou progresso limitado [20, 22]. Xchart é uma proposta que enfatiza o desenvolvimento deste componente de interfaces.

Esta organização apresenta apenas uma visão lógica das funções executadas por um sistema interativo. Para todo sistema, tais funções são dispostas segundo uma arquitetura de software. Uma *arquitetura de software* é um modelo abstrato que estabelece a organização do software de um sistema. Uma arquitetura identifica componentes, divide funções e estabelece um protocolo de comunicação entre eles. Estas decisões têm implicações diretas nos custos de criação e manutenção do software resultante. Reduzir tais custos faz parte dos objetivos de arquiteturas. Programadores, contudo, relatam dificuldades em organizar o software de interfaces [21].

Ferramentas de interface, abrangendo *toolkits*, *interface builders* e UIMSs, visam dar suporte à implementação do software de uma interface. No entanto, estas ferramentas não permitem, em geral, um mapeamento simples e direto das decisões tomadas para o projeto arquitetônico escolhido para a interface (não existem abstrações que podem ser utilizadas). Este é um problema, especialmente para sistemas interativos grandes e complexos, onde a definição de uma arquitetura apropriada é fundamental para possibilitar a evolução e a manutenção do sistema. Esta ausência de suporte à implementação de arquiteturas de software é resultante da distância semântica existente entre os modelos abstratos das arquiteturas e os paradigmas de implementação adotados pelas ferramentas de interface.

Atualmente existem várias propostas de arquiteturas de software e um número ainda maior de ferramentas de interface. No entanto, enquanto diferentes arquiteturas se mostram adequadas para atender aos diversos tipos de requisitos dos sistemas interativos, raras ferramentas oferecem suporte apropriado à implementação destas arquiteturas. Ao contrário do que ocorre com Statechart, Xchart é apresentada para capturar gerenciadores de diálogos complexos considerando adequadamente, conforme arquiteturas baseadas em agentes, a posterior implementação destes gerenciadores.

Existem duas abordagens, além da abordagem *ad hoc*, obviamente indesejável, para definir a arquitetura de software de sistemas interativos. A abordagem *modular* trata cada uma das camadas apresentadas na Figura 1 como um módulo da arquitetura. A abordagem baseada em *agentes* divide as funções de cada camada entre um conjunto de elementos especializados e independentes (agentes) que cooperam entre si para realizar cada função. A diferença entre elas é estabelecida, basicamente, pelo subconjunto de objetivos atendidos por cada arquitetura.

Arquitetura ad hoc

Morse e Reynolds [20] afirmam que programadores estão enfrentando os mesmos problemas de 15 anos atrás no desenvolvimento de sistemas grandes: aos usuários de toolkits são fornecidas muitas informações que, se não organizadas e gerenciadas adequadamente, tornam o software de uma interface inviável. Por exemplo, organizando o código de forma **ad hoc**, funções podem tratar as ações dos usuários em uma arquitetura de software ilustrada na Figura 2. Nessa arquitetura o código da interface e da aplicação residem em um mesmo módulo ("Aplicação"). Os detalhes de baixo nível da interface, fornecidos por um toolkit, são isolados em um outro módulo (Toolkit). Embora esta organização promova a reutilização de fato do toolkit, independência entre a interface e a aplicação, muito importante para o desenvolvimento iterativo e posterior implementação, não se verifica.

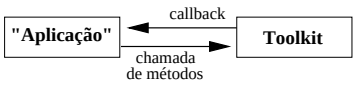

Figura 2: Arquitetura ad hoc para sistemas interativos [messy]

O maior problema da organização **ad hoc** é a alta coesão de um módulo que implementa funções distintas. Uma organização mais estruturada identificaria pelo menos dois componentes que substituiriam a "Aplicação": interface e aplicação, cujas funções seriam aquelas definidas no início desta seção.

Uma dos pontos mais discutidos acerca de arquiteturas de software de sistemas interativos é geralmente referenciado como **independência de diálogo**. Esse conceito refere-se à separação entre o código da interface e da aplicação. Essa separação permite que mudanças em um tenha repercussão limitada no outro. Embora esta separação seja desejada do ponto de vista de engenharia de software, a divisão de funções entre estes componentes envolve várias considerações. Esta separação, por exemplo, contribui com as dificuldades de implementação de uma arquitetura [6]. A separação entre interface e aplicação pode ser estimulada com o emprego de Xchart [5].

Arquiteturas Modulares

Nestas arquiteturas as tarefas de um sistema interativo são distribuídas entre um reduzido número de grandes módulos de software. Cada módulo é responsável pelo conjunto de funções necessárias à execução de determinada tarefa. As arquiteturas modulares assemelham-se à divisão lógica apresentada na Figura 1. Seeheim [7] e SLINKY [3] estão entre as principais arquiteturas modulares.

Arquiteturas Baseadas em Agentes

Um *agente* é um sistema de processamento de informação completo, que inclui receptores e transmissores de eventos, uma memória para manter um estado e um processador que ciclicamente processa eventos de entrada [2]. Agentes comunicam-se com outros agentes.

Em arquiteturas baseadas em agentes, um sistema é organizado em um conjunto de agentes especializados que descentralizam as execuções das funções de um sistema interativo e cooperam entre si para realizá-las. Estas arquiteturas visam produzir comportamento complexo através da cooperação entre unidades computacionais simples e independentes. MVC [14], PAC [2] e RENDEZVOUS [12] estão entre as principais arquiteturas baseadas em agentes.

3 Linguagem Xchart

Nesta seção são apresentados os principais recursos de Xchart. Não há espaço suficiente para exemplificar cada um destes recursos. Elucidar os benefícios de Xchart exige (a) um exemplo cujo controle é complexo (o que dificulta a explicação, além de compararações com a precursora Statechart) e (b) de mais espaço que o disponível. Em conseqüência, sugere-se os exemplos comentados em [5] para tal finalidade. Ainda assume-se familiaridade do leitor com detalhes semânticos de Statechart. Sem estes, a contribuição dos itens abaixo não pode ser adequadamente dimensionada, embora apresentem diferenças significativas em relação à linguagem precursora (Statechart).

XCHART, INSTÂNCIA, CONFIGURAÇÃO, ESTADO E HIERARQUIA DE ESTADOS

A linguagem Xchart oferece vários recursos que são combinados formando um Xchart (diagrama). Um Xchart não é sensível à ocorrência de eventos, não produz nenhum resultado nem pode ser animado. Uma *instância* de um Xchart pode ser animada, reage a estímulos externos, produz controle. Um Xchart é um molde estático a partir do qual instâncias podem ser criadas e animadas. Uma instância é referenciada através do seu identificador. Cada identificador identifica uma única instância. Configuração é um instantâneo de uma instância—refere-se ao conjunto de estados ativos, valores de variáveis e outras informações que caracterizam uma instância em um dado instante de tempo.

Um *estado* representa uma situação conceitual. Todos os recursos da linguagem Xchart necessariamente estão associados a um estado. Um estado pode ser refinado em um conjunto de estados. Essa relação estabelece uma hierarquia entre estados.

AÇÃO, ATIVIDADE, CONTROLE, PORTA E REAÇÃO

Ação é o recurso utilizado para alterar o valor de uma variável, gerar eventos primitivos e gerenciar atividades, entre outras. Descrições em Xchart são responsáveis pelo controle de um sistema, enquanto as demais funcionalidades são fornecidas por *atividades*. A especificação e o desenvolvimento de atividades não são contemplados pela linguagem Xchart. O objeto passível de ser descrito nesta linguagem é o *controle* de um sistema.

Um Xchart não produz controle. Xchart é um modelo através do qual controle pode ser capturado. O resultado é uma especificação de alto nível, que pode ser executada, reproduzindo o controle modelado. O controle produzido torna-se visível além das fronteiras de uma instância através de *portas*. Uma porta é o destino da reação de uma instância. Uma *reação* é o controle produzido por uma instância na presença de um estímulo externo e conforme a configuração dessa instância.

Regras e transições

Gatilhos, regras, estados e transições entre estados são combinados em um Xchart, a partir do qual uma instância pode ser criada e interagir com o ambiente externo. Regras e transições são mecanismos por excelência para capturar relacionamentos do tipo causa-efeito. Regras, ao contrário de transições, não provocam a alteração do conjunto de estados ativos. A causa é definida por um gatilho e o efeito por ações. Controle é produzido se e somente se regras e/ou transições são executadas.

Prioridade entre transições

Há várias possibilidades de não-determinismo em um Xchart. Por exemplo, quando duas transições estão associadas a estados concorrentes, elas podem ser executadas concorrentemente. Neste caso, o não-determinismo se reflete nas seqüências possíveis de reações. Em outros casos, duas ou mais transições podem formar um conjunto onde apenas uma delas pode ser executada através de uma seleção arbitrária. Esta seleção pode ser determinística se níveis de prioridade forem adequadamente atribuídos a transições. Quando tal conjunto for identificado, apenas a transição que possuir o maior número de prioridade será executada.

Passo e micro-passo

Quando um estímulo externo é recebido por uma instância, ele é tratado de acordo com a configuração da instância. Este tratamento pode identificar um conjunto de regras e/ou transições, cujos gatilhos estão habilitados, para execução (micro-passo). Executar um micro-passo significa executar cada um dos elementos deste conjunto. Uma seqüência de micro-passos forma um *passo*. Executar um passo significa executar cada um dos micro-passos da seqüência pertinente. Cada micro-passo desta seqüência é conseqüência da execução do anterior. O primeiro micro-passo é conseqüência imediata do estímulo externo. A execução desse micro-passo pode provocar a execução de um segundo micro-passo e assim por diante. Essa reação em cadeia é finita (comentado adiante). Em conseqüência, do ponto de vista de uma instância, os "efeitos" de um estímulo externo são finitos.

Se os estados **A**, **C** e **E** de uma instância do Xchart da Figura 3 estão ativos, então um estímulo externo formado por um único evento primitivo do tipo **e** provoca um primeiro micro-passo formado por uma única transição, aquela do estado **A** para o estado **B**. Para a configuração fornecida e o estímulo externo exemplificado, nenhuma outra transição ou regra possui um gatilho habilitado. A execução desta transição gera o evento primitivo **f** através da execução da ação **raise(f)**, além de atualizar o valor de **x** para **1**. Esse evento primitivo é então acumulado aos eventos do estímulo externo inicialmente fornecido à instância. O conjunto resultante realimenta a instância. Um segundo micro-passo é identificado conforme a configuração obtida pela execução da transição comentada acima. O segundo micro-passo é composto apenas pela transição do estado **C** para o **D**. O evento primitivo **g**, gerado pelo segundo micro-passo, provoca um terceiro micro-passo após finalizada a atividade **a**: transição de **E** para **F**. A execução da ação **raise** dessa transição, neste cenário, é suficiente para provocar a execução de um quarto micro-passo, formado pela única regra do Xchart. O evento **e and f and g and h** ocorre porque o

evento primitivo do tipo **e** faz parte do estímulo externo, que é acrescido dos eventos primitivos dos tipos **f**, **g** e **h**, gerados nos micro-passos anteriores. A condição $\mathbf{x > 0}$ também é verdadeira ao fim do terceiro micro-passo. O primeiro micro-passo atribui o valor **1** à **x**. A execução do quarto micro-passo provoca a execução da ação $\mathbf{x := -x}$. Após a execução desta ação, a instância torna disponível os efeitos causados pelo passo que acaba de ser executado (propaga os eventos **f**, **g** e **h** para outras instâncias, se for o caso, e libera o novo valor de **x**). O fim destas operações marca o início do tratamento do próximo estímulo externo, se existe algum aguardando na fila de estímulos. Se não existe nenhum estímulo, então a instância aguarda pela ocorrência de um.

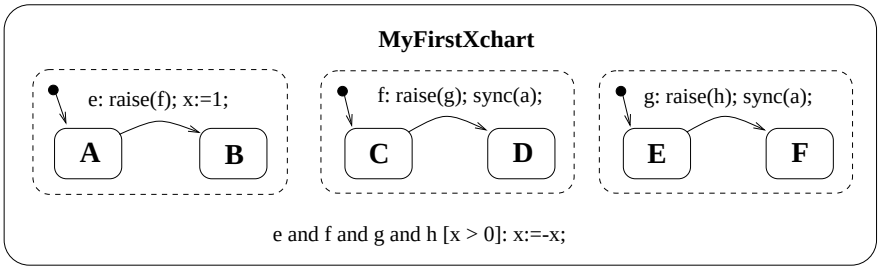

Figura 3: Primeiro Xchart [primeiro]

Modelo de execução

Durante toda a existência de uma instância, ela aguarda por estímulos externos, sem os quais ela não produz controle. Ao tratar um estímulo externo, uma instância identifica todas as regras e/ou transições habilitados pelo estímulo. Se há casos de exclusão mútua entre as transições, então eles são eliminados. Identificado este conjunto (primeiro micro-passo), passa-se a executar a ação correspondente a cada um de seus elementos. Algumas das ações podem ser executadas concorrentemente ou não, conforme a especificação. Tais execuções podem gerar eventos primitivos, alterar o valor de variáveis bem como causar outras alterações. Tais alterações podem habilitar outros gatilhos cujas regras e/ou transições correspondentes serão executadas em um segundo micro-passo e assim por diante. Quando um micro-passo não mais habilitar gatilhos, ou regras e/ou transições não mais puderem ser executadas, então o passo é finalizado. As conseqüências de um passo tornam-se visíveis aos demais componentes do sistema apenas quando ele termina. Neste instante a instância passa a tratar o próximo estímulo externo, ou aguarda por um, caso sua fila esteja vazia. Durante este processo de reação de uma instância, os demais componentes de um sistema podem ser executados concorrentemente. Ações que manipulam atividades, por exemplo, são depositadas em portas que podem ser consultadas enquanto a instância trata o estímulo externo.

Causalidade

Cada passo é dividido, conforme visto anteriormente, em uma seqüência bem definida de micro-passos. Cada micro-passo é, necessariamente, conseqüência da execução do micro-passo imediatamente anterior ou se se trata do primeiro micro-passo, então é conseqüência do estímulo externo. Ou seja, o micro-passo formado pela regra $\mathbf{e[x > 0] : sync(a)}$

pode ser executado imediatamente após outro composto pela regra $\mathbf{f[x < 0] : x := 1}$. Nesse caso, em um mesmo passo, foram executadas regras cujos gatilhos não podem pertencer a um mesmo micro-passo. As condições $\mathbf{x < 0}$ e $\mathbf{x > 0}$ não podem ser verdadeiras durante a avaliação de um único micro-passo, pois a configuração de uma instância não se altera durante esse processo.

Passo é finito

Uma transição ou regra é executada no máximo uma vez por passo. Ou seja, se uma transição ou regra é executada em um dado passo, então só poderá ser executada novamente a partir do próximo passo. Esta restrição impede, naturalmente, a existência de ciclos onde estados seriam ativados e desativados infinitamente, ou ainda a execução infinita de regras em um estado sempre ativo.

Tempo

Do ponto de vista de uma instância, o tempo real contínuo é dividido em intervalos definidos por instantes de tempo. Por exemplo, a Figura 4 exibe o instante **ti** e os sucessores **tj** e **tk**. Os instantes **ti** e **tk** estão relacionados às configurações assumidas pela instância em resposta à criação da instância e ao passo executado em decorrência de um estímulo externo recebido no instante **tj** por tal instância. No instante **tk** é obtida a configuração resultante após a execução de um passo dada a configuração anterior no instante **ti**. O passo é iniciado após um estímulo externo ser sinalizado no instante **tj**. Nesse caso, a figura sugere que a fila da instância estava vazia após a configuração inicial ser obtida no instante **ti**.

Desde o instante em que uma instância é criada até aquele em que uma transição final é executada, a "vida" da instância é marcada por um conjunto de elementos discretos como **ti** e **tk**. Os intervalos refletem a espera por estímulos externos (quando a fila está vazia) e a duração de um passo. Quando um estímulo é recebido, inicia-se o processo de identificação e execução do passo provocado pelo estímulo. Apenas ao fim da execução do passo, em um instante de tempo posterior ao início do tratamento do estímulo externo, uma nova configuração é obtida para a instância. A partir deste instante, a instância permanece com a configuração obtida até a ocorrência de um novo estímulo externo.

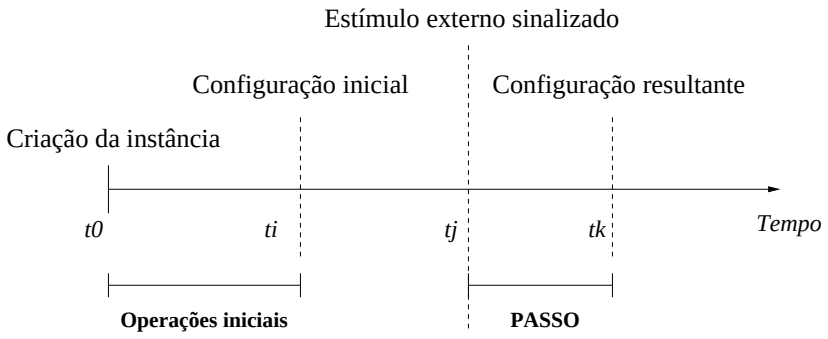

Figura 4: Instantes típicos da execução de uma instância [tempo]

4 Semântica Formal de Xchart

Os engenhosos mecanismos de Statechart tornam a definição semântica dessa linguagem uma tarefa não-trivial e suscetível a interpretações errôneas. O trabalho em [10], por exemplo, é parcialmente motivado por confusões acerca da semântica de Statechart. Uma das primeiras propostas para a semântica formal de *Statechart* foi apresentada em [8]. Tal proposta não inclui definições de mecanismos importantes de Statechart, por exemplo, *history-star* e função de configuração. Algumas foram incluídas na mais recente versão [10]. Entre as semânticas disponíveis e aquela aqui apresentada, nota-se um empenho em relação ao rigor e completitude da última.

A definição formal de Xchart herda as dificuldades semânticas de Statechart sobre as quais muitos trabalhos têm sido realizados [4, 13] e trata outras introduzidas pelos novos recursos e alterações em algumas das construções de Statechart. Além do interesse teórico, tais dificuldades devem ser eliminadas se uma ferramenta que executa Xchart deve ser implementada. Xchart não é apresentada apenas como uma linguagem para descrição do controle de gerenciadores de diálogo, onde especificações são manualmente convertidas em código em etapas posteriores de desenvolvimento. A proposta apresentada inclui a especificação e a implementação de gerenciadores de diálogo a partir de descrições em Xchart. Ainda convém ressaltar que a implementação dá suporte à organização do software em múltiplos agentes.

Entre as dificuldades adicionais da semântica de Xchart, por exemplo, o emprego de consistência local torna a semântica mais difícil de ser definida, embora mais "intuitiva" [4, 13]. Xchart vai além de consistência local: dado um conjunto de ações a serem executadas, um subconjunto delas pode fazer parte de uma seqüência onde cada ação deve ser executada na ordem prescrita pela seqüência. Ao executar uma ação dessa seqüência, a configuração da instância é alterada e, possivelmente, influi na execução das ações seguintes. Nem Statechart ou uma de suas variantes conhecidas, exceto Xchart, oferece esta causalidade (considerada por muitos um elemento que torna mais clara uma linguagem reativa). Ou seja, a execução de um micro-passo está associada a uma seqüência de micro-configurações denominadas, em Xchart, de transientes. Xchart portanto, emprega configurações, que são obtidas conforme o tratamento de estímulos externos, micro-configurações que são resultantes da execução de micro-passos e micro-configurações transientes, obtidas entre a execução de ações de um micro-passo. Dessa forma, $\mathbf{x} := \mathbf{1}; \mathbf{x} := \mathbf{x} + \mathbf{1}$, por exemplo, necessariamente deposita o valor **2** em **x**, ao contrário das demais variantes de Statechart.

5 Ambiente Xchart

Ambiente Xchart denota um conjunto de ferramentas desenvolvidas para apoiar o emprego da linguagem Xchart. O usuário de Xchart é beneficiado pelo emprego dessas ferramentas. Elas contemplam a edição, geração de informações utilizadas em tempo de execução e software responsável pela execução de instâncias de Xcharts.

O sistema de execução de Xchart apóia, em tempo de execução, o emprego da linguagem Xchart. O suporte à noção de cliente, porta, instâncias e outros, é fornecido por esse software, que é uma implementação da semântica de Xchart. As propostas existentes para

softwares similares são pouco conhecidas. Statemate [11] e BetterState [1] são sistemas comerciais atualmente construídos sobre Statechart. Produtos de domínio público com propósitos similares são desconhecidos. Acredita-se que a dificuldade de implementação dessas ferramentas justifiquem esse cenário. Neste sentido, o Ambiente Xchart também pode estimular a experimentação com Xchart, uma vez que estas ferramentas comerciais custam em torno de alguns milhares de dólares. A grande ênfase destas ferramentas está em recursos típicos de Statechart e, portanto, nenhuma oferece, por exemplo, a possibilidade de execução distribuída de instâncias. Uma noção do interesse pela qualidade das ferramentas que compõem o Ambiente Xchart pode ser medida pelos vários trabalhos já desenvolvidos com o editor de Xchart.[1]

Entre os softwares que compõem o ambiente Xchart está o protocolo de comunicação entre instâncias de Xcharts e programas que usufruem do controle gerado pela reação destas instâncias. Propostas baseadas em Statecharts não consideram tal questão, o que compromete a etapa de implementação.

6 Considerações Finais

Evidências positivas do emprego dessa proposta foram obtidas através do desenvolvimento de vários gerenciadores de diálogos típicos em sistemas convencionais além de outros menos comuns [5]. A semântica formal apresenta propostas para questões ainda em discussão e foi de fato implementada através das ferramentas do ambiente Xchart. O modelo sofisticado oferecido por Xchart exige um sistema de execução relativamente complexo, que também provou ser suficientemente robusto e eficiente em relação aos testes realizados. Este sistema também é uma proposta singular de implementação de linguagens derivadas de Statechart.

Embora muitas propostas de variantes de Statechart tenham surgido, acredita-se que a proposta aqui apresentada possa ser empregada de forma mais efetiva por vários motivos. Instâncias de diagramas executadas concorrentemente sobre nós de uma rede oferece uma abstração não disponível em outrs propostas, nem mesmo na mais recente "atualização" de Statechart [9]. Xchart não é útil apenas para especificar, mas também oferece recursos imprescindíveis às demais fases de desenvolvimento de um sistema (implementação e execução).

Referências

[1] M. Ali. A Better Way to State It. *IEEE Computer*, 28(12):72–74, December 1995.

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

[3] Len Bass, Ross Faneuf, Reed Little, Niels Mayer, Bob Pellegrino, Scott Reed, Robert Seacord, Sylvia Sheppard, and Martha R. Szczur. A Metamodel for the Runtime Architecture of an Interactive System. *SIGCHI Bulletin*, 24(1):32–37, January 1992.

[1] http://www.dcc.unicamp.br/proj-xchart/editor/

[4] M. Beeck. A Comparison of Statecharts Variants. *LNCS*, 863:128–148, 1994.

[5] Fábio Nogueira de Lucena. *Xchart: Um Modelo de Especifica,c ao e Implementa,c ao de Gerenciadores de Diálogo*. PhD thesis, Universidade Estadual de Campinas, Instituto de Computa,c ao, Caixa Postal 6176, Campinas/SP, dezembro 1997.

[6] Ernest A. Edmonds, editor. *The Separable User Interface*. Academic Press, 1992.

[7] Mark Green. Report on Dialogue Specification Tools. In Günther E. Pfaff, editor, *User Interface Management Systems*, pages 9–20. Springer-Verlag, 1985.

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

[9] D. Harel and Eran Gery. Executable Object Modeling with Statecharts. *IEEE Computer*, 30(7):31–42, July 1997.

[10] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, Oct 1996.

[11] 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.

[12] Ralph D. Hill, Tom Brinck, Steven L. Rohall, John F. Patterson, and Wayne Wilner. The Rendezvous Architecture and Language for Constructing Multiuser Applications. *Transactions on Computer-Human Interaction*, 1(2):81–125, June 1994.

[13] C. Huizing and W. P. de Roever. Introduction to Design Choices in the Semantics of Statecharts. *Information Processing Letters*, 37:205–213, 1991.

[14] G.E. Krasner and S. T. Pope. A Cookbook for Using the MVC User Interface Paradigm in Smalltalk. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.

[15] Fábio N. Lucena, Luiz Eduardo Buzato, and Hans K.E. Liesenberg. Xchart: Um Sistema de Gerenciamento de Interfaces Homem-Computador. *Workshop de Sistemas Hipermidia (WoSH'96/SBRC'96)*, May 1996. Fortaleza/CE.

[16] Fábio N. Lucena, Mário Massato Harada, and Hans K.E. Liesenberg. Operational Semantics of Extended Statecharts (XCHARTS). *3rd Workshop on Logic, Language, Information and Computation (WoLLIC'96)*, 1996. Salvador/BA, May 8-10.

[17] Fábio N. Lucena and Hans K. E. Liesenberg. A Statechart Engine to Support Implementations of Complex Behaviour. *XXI Semish*, pages 177–191, 1994. Caxambu/MG, Brazil.

[18] Fábio N. Lucena, Hans K. E. Liesenberg, and Luiz E. Buzato. Xchart-Based Complex Dialogue Development. In *Simpósio Nipo-Brasileiro de Ciência e Tecnologia*, pages 387–396, Campos do Jordão/SP, August 1995.

[19] Fábio N. Lucena and Hans K.E. Liesenberg. Reflections on Using Statecharts to Capture User Interface Behaviour. *Proceedings of XIV Int. Conf. of the Chilean CSS*, October 1994.

[20] Alan Morse and George Reynolds. Overcome Current Growth Limits in User Interface Development. *Communications of the ACM*, 36(4):73–81, April 1993.

[21] Brad A. Myers. Challenges of HCI Design and Implementation. *Interactions*, 1(1):73–83, 1994.

[22] Brad A. Myers. UIMSs, Toolkits, Interface Builders. *Handbook of User Interface Design*, 1997. Jacob Nielsen, editor. To appear.

[23] A. R. Puerta. A Model-Based Interface Development Environment. *IEEE Software*, 14(4):40–47, August 1997.