Xchart: Um Modelo de Especificação e Implementação de Gerenciadores de Diálogo

Fábio Nogueira de Lucena

Tese de Doutorado

Instituto de Computação
Universidade Estadual de Campinas

Xchart: Um Modelo de Especificação e Implementação de Gerenciadores de Diálogo

Fábio Nogueira de Lucena[1]

19 de dezembro de 1997

Banca Examinadora:

- Hans Kurt Edmund Liesenberg (Orientador)
- Paulo César Masiero
  Instituto de Ciências Matemáticas de São Carlos (USP)
- Simão Sirineo Toscani
  Instituto de Informática (UFRGS)
- Luiz Eduardo Buzato
  Instituto de Computação (UNICAMP)
- Ricardo de Oliveira Anido
  Instituto de Computação (UNICAMP)
- Beatriz M. Daltrine (suplente)
  DCA/FEE (UNICAMP)
- Eliane Martins (suplente)
  Instituto de Computação (UNICAMP)

[1]**Este trabalho foi realizado com suporte financeiro do CNPq, da UFG e do projeto PROTEM Geotec do CNPq.**

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Lucena, Fábio Nogueira de

L963x Xchart: um modelo de especificação e implementação de gerenciadores de diálogo / Fábio Nogueira de Lucena -- Campinas, [S.P. : s.n.], 1998.

Orientador : Hans Kurt Edmund Liesenberg

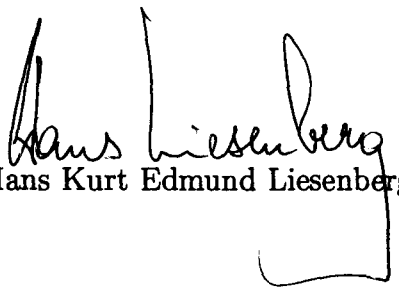
Dissertação (Doutorado) - Universidade Estadual de Campinas, Instituto de Computação .

1. Linguagem de programação . 2. Interfaces de usuário (Sistema de Computador) 3. Software - Desenvolvimento. 4. Software (Sistemas) I. Lisenberg, Hans Kurt Edmund . II. Universidade Estadual de Campinas. Instituto de Computação.

Xchart: Um Modelo de Especificação e Implementação de Gerenciadores de Diálogo

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Fábio Nogueira de Lucena e aprovada pela Banca Examinadora.

Campinas, 19 de dezembro de 1997.



Hans Kurt Edmund Liesenberg (Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Tese de Doutorado defendida e aprovada em 19 de dezembro de 1997 pela Banca Examinadora composta pelos Professores Doutores

Prof. Dr. Simão Sirineo Toscani

Prof. Dr. Paulo César Masiero

Prof. Dr. Luiz Eduardo Buzato

Prof. Dr. Ricaro de Oliveira Anido

Prof. Dr. Hans Kurt Emund Liesenberg

Agradecimentos

Essa tese não seria possível sem o enorme apoio que recebi. Forças de "origem desconhecida" foram fornecidas à medida que elas eram necessárias. Parte delas eu atribuo a Deus.

Minha família soube compreender minha ausência em muitas ocasiões. Meu pai, Raimundo Nogueira, e a minha mãe, Tereza Rodrigues não me deixaram em nenhum momento. Sem a colaboração deles, a realização desse trabalho teria sido muito difícil, senão impossível. As amizades por eles cultivadas também foram importantes.

O orientador Hans Liesenberg ensinou-me muito nesse período juntamente com a presença agradável da sua família: Maria Helena, Sophia e Vera Maria. Todos eles fortaleceram em mim o conceito de hospitalidade. Em momento algum foi-me negada a atenção requisitada. Sua prontidão e disponibilidade para discutir assuntos pertinentes a este trabalho foram importantes.

O Professor Luiz Eduardo Buzato e Ricardo Oliveira Anido ainda contribuíram com sugestões ao longo da realização desse trabalho. O Professor Buzato, em particular, através de discussões, estimulou a definição mais clara de alguns recursos da linguagem Xchart.

Aos colegas de mestrado e doutorado, meus agradecimentos. Eles são muitos. Em particular, Marcus Vinícius A. Andrade, Mário Massato Harada, Juliano Lopes Oliveira e Helena Cristina da Gama Leitão formam uma equipe de pessoas formidáveis. Aos alunos de graduação do IC/Unicamp (Mauro Nascimento Rezende e Tallys Hoover Yunes) meus agradecimentos pelas inúmeras horas de discussões acerca da implementação das ferramentas do ambiente Xchart. Aos alunos de mestrado Edilmar Lima, Luciana de Paula Brito, Osvaldo Severino Junior e, mais recentemente, Carlos Neves Junior, que continua trabalhando ativamente na implementação de Xchart.

A permanência em Campinas seria diferente sem a presença destes colegas. Em especial, Silvana Miquelim (namorada) soube me tratar de forma irrepreensível. Sua conduta, postura e determinação servem de exemplo. O seu carinho preencheu espaços e me mostrou que há saída para o mundo: o amor.

Prefácio

"A interface é freqüentemente o mais importante fator para determinar o sucesso ou falha de um sistema, além de ser um dos mais caros."
Baecker e Buxton[4, pág. 1]

O gerenciador de diálogo é um dos componentes mais complexos de uma interface e continua, em grande parte, sendo codificado manualmente em linguagens de programação convencionais. Isto contribui para elevar os custos de desenvolvimento e inibir alterações, que são indispensáveis ao desenvolvimento de interfaces.

Essa dissertação propõe a linguagem Xchart e um ambiente de apoio voltados para o desenvolvimento de gerenciadores de diálogo. Xchart possui recursos para capturar adequadamente as funções desses componentes de interfaces. Cada gerenciador é especificado através de um conjunto de diagramas. A semântica formal de Xchart permite que esses diagramas sejam automaticamente executados por um interpretador. Em tempo de execução, instâncias desses diagramas podem ser criadas dinamicamente e executadas concorrentemente, possivelmente em um sistema distribuído, apoiadas pelo sistema de execução de Xchart.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Developer-centered environments, which are typical of most systems, give ample support to using and managing widgets, organizing and arranging layouts, and testing prototype interfaces, but fall short of answering key questions.
Angel R. Puerta [110]

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 código desse componente. 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 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, entretanto, dado grande ênfase a outro componente: a apresentação.

O presente trabalho apresenta uma abordagem que visa reduzir as dificuldades de desenvolvimento de gerenciadores de diálogos de interfaces complexas. A abordagem inclui a especificação, a implementação e a execução deste componente com o objetivo de conseguir, para este componente, um sucesso similar ao já obtido com as ferramentas disponíveis para desenvolver o componente de apresentação.

1.1 Motivação

A qualidade de um sistema interativo não é exclusivamente determinada pelo código que implementa as funções desse sistema (ou componente de aplicação). A comunicação com o usuário, por exemplo, faz parte das responsabilidades da interface e também influi nessa qualidade. Embora não faltem argumentos para justificar a importância de interfaces [82], produzir "boas" interfaces não é uma tarefa simples. A inexistência de um método que assegure uma qualidade satisfatória exige um projeto iterativo, onde idéias são transformadas em protótipos e avaliadas até que um resultado satisfatório seja obtido [106]. A definição do comportamento e da aparência de uma interface não abrange todas as dificuldades. A implementação também possui vários desafios [98], alguns dos quais são focalizados pelas ferramentas existentes. Naturalmente, tornar mais fácil e rápida a produção do software de interfaces tem implicações sobre o projeto iterativo. Uma implementação de baixo custo estimula a experimentação de projetos alternativos.

Muitas das ferramentas concentram-se em aspectos visuais e estáticos de uma interface, por exemplo, leiaute de botões e rótulos de menus. Essas ferramentas permitem facilmente a construção de protótipos que combinam esses elementos. Contudo, tal facilidade de confecção não se verifica para outras funções. Pouco suporte é oferecido para lidar com os gráficos produzidos pelo sistema (editores de diagramas, por exemplo). Diálogos concorrentes podem ser úteis em várias interfaces e, para eles, praticamente nenhum suporte é fornecido pelas ferramentas existentes.

Além das restrições funcionais das ferramentas existentes, os recursos oferecidos são de baixo nível e são usufruídos, na grande maioria dos casos, apenas por programadores especializados. Conforme citação no início desse capítulo, há questões que não são resolvidas por tais ferramentas. O presente trabalho concentra-se em uma delas: apoiar o desenvolvimento adequado de gerenciadores de diálogo (especificação e implementação).

Nenhuma abordagem, num futuro que se possa vislumbrar, produzirá automaticamente todo o código de qualquer que seja a interface a partir de especificações de alto nível de abstração. Em conseqüência, pesquisas devem continuar produzindo abordagens que convirjam, cada vez mais, para a melhor forma de gerar tal código a partir de especificações. O objetivo é controlar os custos e produzir interfaces melhores.

1.2 Proposta

O presente trabalho propõe a linguagem Xchart e um ambiente de apoio [74, 84] ao emprego dessa linguagem. A linguagem Xchart foi projetada para capturar a funcionalidade de gerenciadores de diálogos, sendo ela gráfica, formal e executável [75]. Através do ambiente, o usuário de Xchart edita e executa descrições nessa linguagem.

O projeto de interação de uma interface envolve a definição da aparência e do comportamento da interface. Parte desse 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 [40], 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 [85]. Essas mudanças foram incorporadas em Xchart. Em particular, Statechart é muito utilizada teoricamente e carece de implementações que mostrem efetivamente como essa linguagem pode se integrar com os 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 interação. A tarefa manual de implementar o código correspondente a esse componente em linguagens de baixo nível, requerida em grande parte das atuais ferramentas, é substituída pela edição de diagramas, que são automaticamente convertidos em protótipos e rapidamente suscetíveis de serem testados. A complexidade da implementação desse componente não desaparece; ela é transferida para as ferramentas que dão apoio ao emprego de Xchart. Uma versão precursora do sistema de execução de Xchart proposto neste trabalho é apresentada em [81].

A linguagem e o ambiente de apoio Xchart favorecem a organização de um sistema interativo em pequenos módulos gerenciáveis. Esse interesse é fomentado pela carência, em muitas propostas, de mecanismos que auxiliem na organização do software resultante. Em sistemas grandes, uma organização adequada facilita a manutenção do sistema.

Evidências positivas do emprego dessa proposta foram obtidas através do desenvolvimento de vários gerenciadores de diálogos típicos em sistemas convencionais, e outros menos comuns. Em particular, o modelo sofisticado oferecido pela linguagem Xchart torna o seu sistema de execução relativamente complexo, que também provou ser suficientemente robusto e eficiente em relação aos testes realizados.

1.3 Contribuições

As principais contribuições do trabalho aqui apresentado surgem do interesse pela especificação e implementação de gerenciadores de diálogo de interfaces:

PROPOSTA ALTERNATIVA DE DESENVOLVIMENTO DE INTERFACES

O sistema de execução de Xchart contempla recursos para a implementação de gerenciadores de diálogos cujos sistemas interativos pertinentes apresentam concorrência entre a interface e a aplicação, além da possível concorrência no interior da própria interface. Tal tipo de suporte, entre outros, praticamente inexiste nas atuais ferramentas. O sistema de execução de Xchart ainda é uma das raras propostas para a implementação de linguagens baseadas em Statechart.

LINGUAGEM XCHART

A descrição da funcionalidade de gerenciadores de diálogo conta com mecanismos de especificação em Xchart não existentes em Statechart, enquanto outros sofreram mudanças (léxicas, sintáticas e semânticas). Xchart permite capturar adequadamente gerenciadores cujos sistemas subjacentes estão organizados em múltiplos e pequenos módulos. Uma das características mais relevantes de Xchart está no modelo de múltiplas instâncias. Esse e outros recursos tornam Xchart uma linguagem singular.

SEMÂNTICA FORMAL DE XCHART

A definição da semântica formal de Xchart apresenta desafios adicionais àqueles da semântica de Statechart. Este acréscimo é fruto dos novos recursos e do refinamento semântico de outros presentes em Statechart. Em particular, a serialização da execução das ações decorrentes de um micro-passo tornam-no mais "intuitivo"; em contrapartida, há um ônus adicional na definição formal da semântica.

INTEGRAÇÃO DA ABORDAGEM XCHART E ENGENHARIA DE SOFTWARE

Para tornar linguagens formais úteis é necessário prover mecanismos para conectá-las aos métodos de desenvolvimento de software comumente empregados. A abordagem Xchart visa preencher essa lacuna não claramente contemplada por linguagens correlatas. Por exemplo, o relacionamento entre descrição de controle e arquiteturas de software que implementam tal controle, praticamente ignorado na literatura, é enfatizado no presente trabalho. Não é suficiente diminuir a distância entre idéias de projeto e as implementações correspondentes. A forma dessa transformação também é importante. Xchart captura controle e oferece um modelo de implementação que integra tal controle às tecnologias existentes e geralmente utilizadas nos desenvolvimentos dos demais componentes.

1.4 Organização da Tese

O Capítulo 2 fornece os fundamentos da dissertação, trabalhos correlatos e apresenta a nova proposta para descrever gerenciadores de diálogo: Xchart. Aquele capítulo ainda caracteriza o emprego e o papel de Xchart no desenvolvimento de um sistema interativo. O Capítulo 3 define informalmente a linguagem Xchart e esclarece algumas diferenças entre esta linguagem e Statechart. O Capítulo 4 apresenta formalmente a linguagem Xchart. O Capítulo 5 apresenta o ambiente de apoio ao emprego dessa linguagem. O Capítulo 6 descreve estudos de casos do emprego da proposta aqui apresentada. Eles fornecem evidências da adequabilidade da abordagem Xchart e norteiam a identificação de trabalhos futuros, que são fornecidos no Capítulo 7, juntamente com os resultados obtidos e as considerações finais.

Capítulo 2

Software de Interfaces

Most commercial systems give far more support to widget construction than to interaction sequence composition, which is customarily done by programming. Thus, the most complex part of user-interface design has the least support.
Aeron Marcus & Andries van Dam [88]

Xchart é uma abordagem voltada para a confecção de parte do software de interfaces. Esse capítulo apresenta questões relevantes a esse software. Elas motivam e fornecem fundamentos que solidificam a **Proposta Xchart**, apresentada na Seção 2.5.

As funções de uma interface podem ser divididas em componentes lógicos (Seção 2.1), que favorecem a identificação daquelas que podem ser capturadas pela linguagem Xchart. Assim como Xchart, novas abordagens devem continuar surgindo em decorrência das limitações (Seção 2.2) das propostas atualmente disponíveis.

Muitos trabalhos correlatos (Seção 2.4) preocupam-se essencialmente com as dificuldades de desenvolvimento de um desses componentes lógicos, sem apresentar uma conexão clara entre ele e os demais módulos que implementam as outras funções de um sistema interativo. Integrar as funções descritas através de diagramas e executadas pelo ambiente Xchart (Capítulo 5) com o restante do código de um sistema exige uma análise cuidadosa de arquiteturas de software para tais sistemas (Seção 2.3). Sem essa análise, especificações em Xchart seriam implementadas de forma **ad hoc** pelo ambiente Xchart.

A linguagem Xchart, parte da abordagem proposta no corrente trabalho, é apresentada informal e detalhadamente no capítulo seguinte.

2.1 Introdução

O desenvolvimento de interfaces é uma atividade complexa que pode consumir mais de cinqüenta por cento do total de linhas de código de sistemas interativos [103]. Com o intuito de reduzir estes custos e aqueles advindos de manutenção, diversos mecanismos de suporte têm sido propostos para auxiliar a realização dessa atividade, entre os quais se encontram as arquiteturas de software (para suporte ao projeto de software) e as ferramentas de interface (para suporte à implementação). O processo de desenvolvimento de um sistema interativo é comentado na Seção 2.5.5 (pág. 38).

Um sistema interativo pode ser dividido em dois componentes principais: o componente computacional e o componente interativo (Figura 2-1). 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 exibe os resultados por esse produzidos. A ênfase deste trabalho está no desenvolvimento de interfaces. A separação entre esses componentes é comentada em detalhes na Seção 2.3.5.

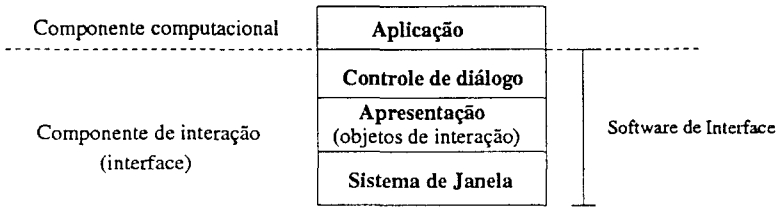


Figura 2-1: Organização lógica e funcional do software de um sistema interativo

A Figura 2-1 apresenta uma divisão lógica do código de um sistema interativo de acordo com as funções que executa. Essa organização facilita a identificação da parte do software de interfaces cujo desenvolvimento é enfatizado pelas ferramentas atualmente disponíveis, inclusive Xchart:

- 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). Existem dois componentes principais em sistemas de janelas: a *camada básica*, que implementa a funcionalidade do sistema, e o *gerente de janelas* (ou *window manager*), que cuida dos seus aspectos interativos. Em ambientes Unix, o sistema X WINDOWS é o padrão de fato para a camada básica, enquanto MOTIF WINDOW MANAGER e OPEN LOOK WINDOW MANAGER, por exemplo, são gerentes de janelas sobre esse sistema.

A camada básica oferece uma interface procedimental através da qual aplicações desenham gráficos em cada janela e recebem eventos do usuário. O gerente de janelas define o modelo de apresentação das janelas (por exemplo, se elas podem ser sobrepostas) e, também, o modo de interação com o usuário, que pode mover, modificar o tamanho ou destruir janelas, sob a coordenação do sistema de janelas.

Embora sejam úteis, 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.

  Callback é um mecanismo amplamente utilizado por toolkits para realizar a comunicação entre o software reutilizável de uma coleção de widgets e o restante do software de um sistema interativo. Um objeto de interação (*widget*) chama uma callback sempre que sua condição associada for satisfeita. Quando uma instância de um botão (widget) é criada, por exemplo, uma callback pode ser registrada com a instância. Nesse caso, sempre que o usuário clicar sobre o botão (condição), a callback é chamada. Uma interface pode compreender milhares de callbacks, o que torna a separação entre interface e aplicação mais difícil [103]. Embora existam mecanismos propostos com o intuito de reduzir o número de callbacks [96], este recurso fornece uma abstração de muito baixo nível para organizar o software de uma interface.

- Controle de diálogo

  O *controle 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 esses componentes. Se o usuário deseja salvar um arquivo ainda sem nome em um editor de texto, por exemplo, então esse componente é responsável por obter do usuário o nome do arquivo a ser salvo. Esse componente pode requisitar serviços oferecidos pela aplicação, em decorrência da ação do usuário sobre dispositivos de entrada. Um valor obtido pela aplicação como temperatura, por exemplo, é convertido, por esse componente, em um valor inteiro que determina a posição de um marcador em um termômetro

exibido pela apresentação ao usuário. Esse componente 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 os dados pertinentes são alterados.

As funções do controle de diálogo estão entre as mais complexas do software de uma interface e o apoio ao desenvolvimento delas tem apresentado progresso limitado [93, 101]. Xchart é uma proposta que enfatiza a descrição, a organização, a implementação e a execução desse componente de interfaces. Segundo Olsen [107], o controle de diálogo não é adequadamente descrito através de uma linguagem de programação convencional. Tal controle possui características peculiares para as quais construções de mais alto nível podem ser especificamente projetadas, em vez de utilizar as construções de linguagens de programação típicas. Estas últimas são suficientes, mas oferecem poucas abstrações.

É importante notar que esta organização apresenta apenas uma visão lógica das funções executadas por um sistema interativo. Ela não prescreve, por exemplo, os mecanismos utilizados para troca de informações entre os componentes nem apresenta considerações sobre o desempenho do sistema. Estas e outras decisões são tomadas durante a fase de projeto do software da interface. Nesta fase é identificada uma arquitetura de software para organizar o software do sistema. Xchart oferece, para implementar o software de um sistema interativo, clientes, portas e instâncias. Esses recursos são comentados nos Capítulos 3 e 5. O processo de desenvolvimento de um sistema interativo é discutido na Seção 2.5.5. A Seção seguinte apresenta dificuldades de ferramentas atualmente disponíveis para a implementação do software de interfaces.

2.2 Dificuldades com Propostas Existentes

During the eight years since the introduction of the Seeheim model the underlying technology has evolved into distributed, heterogeneous environments, which must be managed by the interface, and the requirements imposed by the application domains have become far more complicated. ... This model has to some extent been realized, but real progress has been achieved only in the area of the presentation component.

Morse e Reynolds [93]

Muitos trabalhos na área de interfaces têm suas atenções voltadas para o desenvolvimento do software de interfaces. Uma coletânea deles é apresentada na Seção 2.4. São vários os desafios de desenvolvimento desse software [98]. Em geral, cada trabalho tenta amenizar as dificuldades de um ou outro aspecto desse desenvolvimento e oferece ganhos

significativos de produtividade [99]. As propostas existentes, contudo, ainda apresentam dificuldades. As dificuldades que permanecem corroboram a ausência de consenso acerca de uma abordagem melhor [110, pág. 44] e a distância de uma "solução" para o problema do desenvolvimento de interfaces [97, pág. 2]. Algumas dificuldades:

- *Contemplam apenas a apresentação.*

  Grande parte das ferramentas existentes focalizam apenas o componente de apresentação de uma interface. Ao controle de diálogo, nessas abordagens, não é oferecido suporte. O código pertinente deve ser programado manualmente em uma linguagem de programação convencional. Essa ênfase pode ser constatada através dos progressos realizados nesse componente, ao contrário do que ocorreu com o controle de diálogo. Segundo Marcus e van Dam [88], a seqüência de interação, função do controle de diálogo, é o componente mais complexo de uma interface e o que recebe o menor suporte da maioria das ferramentas comerciais. Morse e Reynolds [93] afirmam que progresso real tem ocorrido apenas no componente de apresentação. Segundo [101]: "a programação de comportamento interativo tem sido a parte mais difícil da criação do software de interfaces, principalmente porque muitas ferramentas fornecem apenas uma fila de eventos de baixo nível que o programador deve interpretar e gerenciar". Desoi e Lively [27] ainda corroboram essas afirmações: "o maior obstáculo para prover uma abordagem que não exija o emprego de linguagens de baixo nível para o desenvolvimento de uma interface está na extrema dificuldade de especificar o comportamento de interfaces".

- *Recursos de baixo nível.*

  As ferramentas disponíveis oferecem recursos de baixo nível. Toolkits, por exemplo, embora comumente empregados [103], apresentam uma curva de aprendizado muito acentuada. Alguns autores acreditam que a construção de interfaces é desnecessariamente difícil devido ao baixo nível dos recursos atualmente oferecidos pelas ferramentas [133]. O apoio ao emprego de uma arquitetura de software, por exemplo, embora muito útil, é raramente um recurso fornecido por ferramentas (Seção 2.3).

- *Contemplam a construção de interfaces simples.*

  Klein é um dos responsáveis pelo desenvolvimento da ferramenta ALPHA, que oferece mecanismos para facilitar a separação entre interface e aplicação [65]. Em resposta a e-mail privado, Klein escreve:

  > "One area where we are finding Alpha has shortcomings is where the graphics on the screen is the whole point of the application (such as CAD

> and graphics editors). Where Alpha is best suited is where the graphics on the screen serve the application, but are not the ends to the means." [Jul/1996]

Szekely **et al.** [126] ressaltam as limitações das atuais ferramentas empregadas no desenvolvimento de interfaces. Em geral, as funções das ferramentas disponíveis estão restritas à criação de caixas de diálogo, edição de menus e leiaute de objetos de interação. Classes de aplicações como visualização, simulação e editores gráficos, por exemplo, não são consideradas. Em tais aplicações (i) dados manipulados têm estrutura complexa; (ii) são heterogêneos (música e desenho, por exemplo); (iii) a quantidade de dados varia dinamicamente (base de dados, outros programas ou dados confeccionados pelo usuário) e (iv) dados são alterados ao longo do tempo. As ferramentas, em geral, não permitem a especificação de objetos gráficos através de primitivas (como retângulos, círculos e linhas) cujas propriedades mudam em resposta à manipulação direta do usuário como, por exemplo, o redimensionamento desses objetos. Elas fornecem, em geral, recursos para a manipulação direta apenas de widgets predefinidos. A criação de tais objetos e comportamentos é realizada em linguagens de programação como C, por exemplo.

Kasik **et al.** [63] afirmam que uma ferramenta voltada para o desenvolvimento do software de uma interface deve ser capaz de lidar com especificações de diálogos complexas e que, infelizmente, muitas delas tratam apenas situações simples.

- *Diálogos multi-threads.*

  Concorrência na interface pode melhorar o desempenho dos usuários dessa interface ou ainda servir como ferramenta útil à organização do software de uma interface [52]. A ausência desse recurso, em conseqüência, pode ser um inconveniente de uma ferramenta [51]. Concorrência, entretanto, está entre os desafios da confecção de interfaces [98]. Atualmente é comum o usuário preencher campos em uma caixa de diálogo, interromper esta atividade, iniciar uma outra e só posteriormente retornar à caixa de diálogo e finalizar a atividade que estava em andamento. Processos da aplicação podem gerar dados que devem ser concorrentemente apresentados ao usuário. A execução de longas tarefas pode exigir concorrência entre a interface e a aplicação. Em conseqüência, a organização de interfaces em múltiplos threads de controle, executados concorrentemente, parece ser um requisito necessário, especialmente para interfaces futuras [37, 133].

  Atualmente, poucas ferramentas contemplam uma execução concorrente de partes de uma interface, mesmo para aplicações complexas, como aquela apresentada em [63]. Uma das ferramentas proeminentes, pelas várias tecnologias que emprega, é

AMULET [101]. Brad Myers, um dos responsáveis pela concepção desse sistema, escreve em resposta a um e-mail privado:

> "We do not have any particular support in Amulet to make it thread safe. ... It would be easy to "simulate" multiple threads by displaying multiple dialog boxes, and having constraints from the dialog boxes to the application code (and vice versa), and then whenever the user changed anything, the appropriate part of the application code will be notified, and vice versa. If the application was multi-threaded, then some locking might be needed to protect Amulet from re-entry. If you really need multiple threads and you discover some problems in Amulet with respect to multiple threads, we would be interested in fixing these problems so Amulet is better suited to multiple threads." [Sept, 1996]

- *Desempenho, confiabilidade, comunicação interface/aplicação, portabilidade.*

  Para o desenvolvimento de ferramentas futuras, desenvolvedores aconselham [103]: elas terão que obter desempenho aceitável e realizar a comunicação entre interface e aplicação. Isto inclui problemas com o uso de callbacks, portabilidade e recursos para facilitar a identificação de erros no software de uma interface. Muitas propostas apresentadas na literatura apresentam exigências quanto a plataforma, linguagens de programação, desempenho e outras bem diferentes do ambiente onde sistemas interativos reais são desenvolvidos. AMULET [101], por exemplo, explicitamente trata algumas dessas questões que não foram apropriadamente contempladas pelo seu predecessor (Garnet).

- *Especificação.*

  Grande parte das propostas atualmente em uso não contemplam um registro apropriado do projeto da interface (abordagens baseadas em modelos, pág. 30, visam tratar adequadamente essa questão). Em outras palavras, as propostas carecem de linguagens que registrem a interface antes do início da implementação ou construção de protótipos. Código é, freqüentemente, a única descrição precisa do próprio código disponível. Muitas propostas descrevem uma interface através de linguagens de baixo nível como C/C++ ou protótipos. O emprego de protótipos e descrições **ad hoc**, contudo, não é visto como meio adequado de registro de uma interface [113].

  Uma especificação da interface é o meio de comunicação entre projetistas, desenvolvedores, usuários e clientes. Se não há uma especificação disponível apropriada, então discussões acerca da funcionalidade da interface proposta podem ficar comprometidas. O emprego de métodos formais é uma alternativa para modelar e implementar sistemas interativos [46]. Métodos formais podem ser empregados não

apenas do ponto de vista de engenharia de software para interfaces [19, 39, 60], mas também do ponto de vista de outras disciplinas que contribuem para o progresso de interfaces (psicologia, por exemplo).

As atuais abordagens para o desenvolvimento do software de interfaces não apresentam limitações de forma infundada. As dificuldades de desenvolvimento de tal software não são exclusivas desse domínio. Verdadeiros desafios [98] terão que ser superados para eliminar tais dificuldades. As ferramentas de desenvolvimento ainda são muito difíceis de serem construídas [99], o que inibe experimentações. Por exemplo, muitas delas baseiam-se em protótipos anteriores desenvolvidos ao longo de vários anos [63, 65, 101].

2.3 Arquiteturas 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. Contudo, estabelecer uma arquitetura apropriada não é uma tarefa simples tampouco organizar o software de um sistema interativo segundo uma arquitetura predefinida. Programadores relatam dificuldades na organização desse software, o que torna a implementação correspondente mais difícil [98]. Segundo [102], mais pesquisas ainda são necessárias para identificar a melhor forma de organizar o código de um sistema interativo e como ferramentas podem dar suporte a essa tarefa.

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. Este é um problema grave, 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. Com isso, a facilidade de implementação passa a ser uma medida de qualidade das arquiteturas: o valor prático de uma arquitetura é inversamente proporcional à dificuldade de sua implementação (através de ferramentas de interface).

Por outro lado, se uma ferramenta não organiza adequadamente o código que gera, então os seus benefícios podem ser obscurecidos, especialmente para sistemas interativos de grande porte. Mesmo que uma ferramenta gere automaticamente código para realizar

uma parte significativa das funções de uma interface, sua utilidade será comprometida se ela não oferece uma estrutura apropriada para a confecção do código restante. Os problemas envolvidos incluem a identificação deste código restante e o seu relacionamento com aquele fornecido pela ferramenta. Em conseqüência, se a ferramenta não oferece suporte, então a reutilização e manutenção do código da interface passa a depender da disciplina e habilidade do projetista.

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 a diversos tipos de requisitos dos sistemas interativos, raras ferramentas oferecem suporte apropriado à implementação destas arquiteturas.

Além da abordagem **ad hoc**, obviamente indesejável, existem duas outras abordagens para definir arquiteturas de software de sistemas interativos. A abordagem *modular* trata cada uma das camadas apresentadas na Figura 2-1 como um módulo da arquitetura. A abordagem baseada em *agentes* divide as funções de cada camada entre um conjunto de pequenos elementos especializados e independentes (agentes) que cooperam entre si para realizar cada função. A definição de uma arquitetura pode ser norteada por vários critérios [6]. Independência de diálogo é um dos principais (Seção 2.3.5). O restante desta seção apresenta, sucintamente e de acordo com a abordagem que emprega, algumas das principais arquiteturas de software para interfaces WIMP (*Windows, Icons, Mice, Pointing*). Arquiteturas para interfaces non-WIMP são discutidas em [37].

2.3.1 Arquitetura ad hoc

Morse e Reynolds [93] 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**, callbacks podem tratar as ações dos usuários em uma arquitetura de software ilustrada na Figura 2-2. 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 essa organização promova a reutilização de fato do toolkit, independência entre a interface e a aplicação não se verifica.

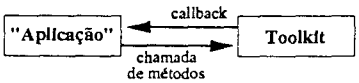


Figura 2-2: Arquitetura ad hoc para sistemas interativos

O emprego de toolkit/callbacks não significa isolamento entre interface e aplicação.

Tal separação também não é incompatível com o emprego desses recursos. A disciplina do projetista do software da interface é que permitirá essa separação ou não.

O módulo "Aplicação" não contém apenas código responsável pela funcionalidade do sistema. Detalhes de apresentação, por exemplo, a posição de barras de rolagem e a responsabilidade por manter a consistência entre widgets estariam, entre outros, implementados nesse mesmo módulo. O maior problema dessa organização 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.

2.3.2 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 2-1.

Seeheim

Uma das arquiteturas de interface mais conhecidas é a de Seeheim [35]. Ela divide uma interface em três módulos: Apresentação, Controle de Diálogo e Interface com a Aplicação (Figura 2-3).

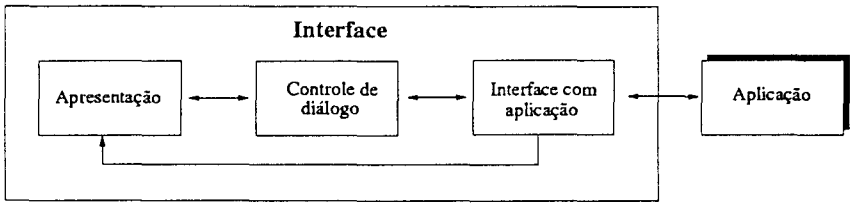


Figura 2-3: Modelo de Seeheim (adaptado de [35])

Nesta arquitetura, as funções dos componentes lógicos Sistema de janela e Objetos de interação (Figura 2-1) são realizadas pelo módulo de Apresentação. O componente lógico de Controle de diálogo é implementado pelo módulo Controle de diálogo e o módulo de interface com a aplicação é introduzido para realizar, explícita e exclusivamente, a comunicação entre a interface e a aplicação.

O módulo de apresentação permite que o usuário requisite a execução de tarefas e forneça dados para a aplicação. O resultado do processamento realizado pela aplicação também é exibido pelo módulo de apresentação.

As decisões do controle de diálogo são definidas de acordo com o *contexto da interação*. Por exemplo, em um editor de texto, a mesma ação de entrada de caracteres é interpretada diferentemente de acordo com o *modo* (ou *estado*) corrente: sobrescrever ou inserir. Em particular, quando não é necessária a intervenção do gerenciador de diálogo, por questões de eficiência, a apresentação pode receber dados diretamente da interface com a aplicação conforme a ligação entre esses componentes (Figura 2-3).

A aplicação atende requisições de serviços da interface e pode ainda obter dados do usuário. Essa comunicação é feita através do módulo de interface com a aplicação.

Há deficiências bem conhecidas associadas à arquitetura de Seeheim [123]. A complexidade dos três módulos em interfaces de grande porte e a dificuldade de prover retroalimentação semântica (*semantic feedback*) são algumas delas. A última refere-se à comunicação intensa entre apresentação e aplicação. Essa comunicação pode tornar o desempenho de um sistema interativo inaceitável devido ao gargalo estabelecido pelo fluxo obrigatório dos dados através dos módulos desse modelo.

Slinky/Arch

SLINKY é um metamodelo voltado para o atendimento de diferentes objetivos dentro de um conjunto de requisitos e funcionalidades de interface previamente fixados [6]. De acordo com um subconjunto de requisitos, uma arquitetura é especificamente derivada do metamodelo. O metamodelo compreende os seguintes módulos:

- *Domínio*: realiza as tarefas da aplicação;
- *Adaptador de domínio*: realiza tarefas pertinentes ao domínio, mas relacionadas à interface (por exemplo, relatar erros semânticos);
- *Controle de diálogo*: realiza as tarefas do componente controle de diálogo (pág. 9);
- *Apresentação*: fornece objetos de interação, independentes de toolkits específicos, para uso do controle de diálogo. Por exemplo, um "objeto seletor" pode ser implementado através de menus ou botões em diferentes toolkits;
- *Toolkit*.

O metamodelo permite a migração dessas atribuições entre seus módulos. As funções efetivamente desempenhadas por cada módulo são definidas de acordo com os objetivos prioritários do sistema interativo. Por exemplo, se um toolkit oferece objetos de interação suficientes para o desenvolvimento de um sistema interativo, então o módulo de apresentação pode ser mínimo, pois suas funções são desempenhadas pelo toolkit. Em outro caso, se desempenho é primordial, então a migração de parte das funções do adaptador

de domínio e do controle de diálogo para a apresentação pode ser suficiente, permitindo que *feedback* possa ser oferecido pela apresentação sem necessidade de comunicação com demais módulos. Naturalmente, essa abordagem compromete a independência entre interface e aplicação, comentada posteriormente.

A arquitetura ARCH (Figura 2-4) é um exemplo de instância do metamodelo SLINKY projetada com o intuito de minimizar e localizar os efeitos de mudanças nas tecnologias envolvidas na construção de interfaces [6]. De fato, o metamodelo foi criado como uma generalização das funções realizadas por cada módulo desta arquitetura. As mudanças podem ser conseqüência de alterações no hardware da interface, toolkit, requisitos da interface ou ainda na funcionalidade do sistema.

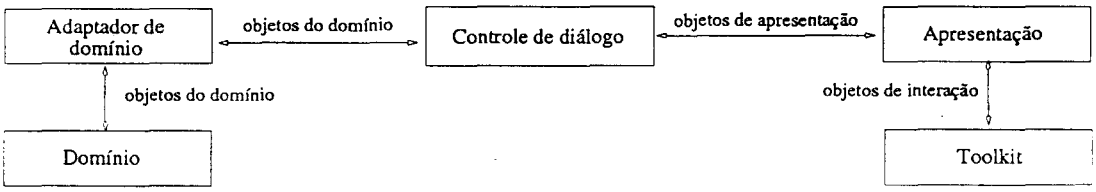


Figura 2-4: Arquitetura ARCH (adaptado de [6])

A organização de ARCH baseia-se na tecnologia disponível (toolkits e banco de dados, por exemplo) e enfatiza a independência entre módulos. Nessa arquitetura um toolkit pode ser substituído por outro sem interferir nos demais módulos. Porém, essa independência é obtida em detrimento de desempenho, já que o fluxo de informações percorre um longo caminho entre um toolkit e um domínio.

2.3.3 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 [5]. Agentes comunicam-se com outros agentes, inclusive o usuário.

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.

As principais vantagens desta abordagem são: (i) facilita o projeto iterativo (um agente define a unidade de modularidade e pode ser alterado sem interferir no comportamento dos demais agentes); (ii) garante compatibilidade com implementação distribuída ou paralela; e (iii) expressa, de modo natural, diálogo concorrente através da atribuição de diferentes agentes para tarefas realizadas concorrentemente por um usuário.

MVC (Model-View-Controller)

MVC [66] é uma das arquiteturas mais referenciadas no desenvolvimento de sistemas interativos. Ela divide uma interface em um conjunto de tríades de objetos pertencentes, respectivamente, às classes *model*, *view* e *controller* (Figura 2-5). Considerações pertinentes à aplicação (objeto model) são isoladas daquelas pertinentes à interface (objeto controller e objeto view). O código que trata da entrada do usuário é da responsabilidade de objetos controller. Objetos view são responsáveis pela saída do sistema. Essa separação é realizada através de um protocolo de comunicação bem definido que permite a troca de mensagens entre esses objetos. Objetos view implementam parte das funções de apresentação (Figura 2-1). Objetos controller implementam as demais funções de apresentação não contempladas por objetos view além das funções do controle de diálogo. Objetos model implementam a aplicação.

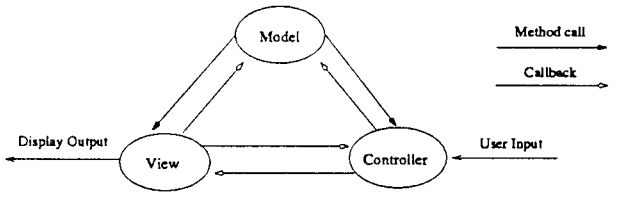


Figura 2-5: Arquitetura MVC

Um *objeto model* representa um modelo dos dados manipulados por uma aplicação (por exemplo, tabelas em uma base de dados). Esses dados podem ser alterados apenas através de métodos de acesso públicos. Se um dado é alterado, então um objeto model deve chamar o seu próprio método **changed**. Em conseqüência, a mensagem **update** é enviada para os objetos view associados ao objeto model. Em alguns casos, a responsabilidade de notificar mudanças é assumida por um objeto controller.

Um *objeto view* é uma apresentação de um objeto model. Podem existir várias apresentações (isto é, múltiplas visões) e, em conseqüência, vários objetos view podem estar associados a um dado objeto model. Por exemplo, dois objetos view podem apresentar valores de um objeto model através de um gráfico de barras e de um gráfico em forma de pizza.

Quando um objeto view é criado, ele envia uma mensagem para o objeto model ao qual ele será associado. Essa mensagem cria uma associação que determina a atualização do objeto view em decorrência de alterações no objeto model correspondente. A notificação destas alterações é realizada através do método **update** dos objetos view, chamado em conseqüência de uma ativação do método **changed** pelo objeto model. Um objeto view pode ainda se comunicar com o seu objeto model para recuperar valores alterados pelo último.

Objetos controller são responsáveis pela interpretação das ações dos usuários. Eles recebem as entradas dos usuários, decidem o que fazer e, quando é o caso, chamam métodos de objetos model. Por exemplo, quando o usuário pressiona um botão, o respectivo objeto controller chama o método pertinente do objeto model para tratar o evento do usuário, se for o caso. Um objeto controller também pode chamar métodos de um objeto view para solicitar a modificação da apresentação em função de mudanças no contexto da interface.

Se um usuário realiza, por exemplo, a seleção de um elemento gráfico, então um objeto controller precisa contactar o objeto view pertinente para identificar o elemento selecionado. Em outros casos, um objeto view confunde-se com um objeto controller. Por exemplo, é comum um objeto de interação apresentar um valor na tela e permitir que o usuário altere esse valor. No primeiro caso tem-se um objeto view que exibe um valor correspondente a alguma grandeza de um objeto model. No segundo, é necessário um objeto controller para interpretar as ações do usuário e, eventualmente, chamar um método do objeto model para alterar o valor dessa grandeza.

Essa divisão de tarefas isola um objeto model (aplicação) de aspectos de apresentação (view) e das ações dos usuários (controller). Contudo, ela é difícil de ser implementada. Shan [118] ressalta que a implementação geralmente resulta em um alto acoplamento entre os objetos, o que dificulta a reutilização e a manutenção de hierarquias. Além disso, um esforço substancial de aprendizado é necessário antes que um programador possa efetivamente usar MVC. Em [34] é apresentada uma proposta de implementação distribuída de MVC. Segundo os autores, tal proposta permite reduzir o tempo gasto em programação.

DV (Document/View)

Apenas as versões recentes do *framework* MFC (da Microsoft) dão suporte ao emprego da arquitetura Document/View (DV). As versões anteriores não ofereciam recursos específicos para a organização do software de uma interface.

Na arquitetura DV, os dados de uma aplicação são armazenados e manipulados através de um objeto *document*. Apresentações desses dados são fornecidas por um ou mais objetos *view*, que também interpretam as ações do usuário. Esses objetos separam a forma na qual os dados são armazenados e manipulados (document) da forma como são apresentados e a interface interage com os usuários (view). A Figura 2-6 apresenta um objeto document e um objeto view associado. Um objeto document implementa funções do componente lógico de aplicação, enquanto um objeto view implementa funções dos componentes de apresentação e controle de diálogo (Figura 2-1).

Objetos view traduzem as entradas do usuário em comandos que manipulam os dados do objeto document. Se os dados de um objeto document são alterados, então esse objeto sinaliza tais alterações através de chamada a método que envia uma mensagem para todos

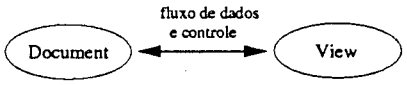


Figura 2-6: Document/View

os objetos view associados. Quando os objetos view recebem essa mensagem eles têm a possibilidade de refletir as mudanças ocorridas nos dados. Conforme observa Prosise, a interação entre esses objetos pode ser bastante complexa [109].

PAC (Presentation-Abstraction-Control)

A arquitetura PAC organiza um sistema em uma hierarquia de agentes [5], conforme a Figura 2-7 (parte esquerda). Um agente PAC possui três facetas: apresentação (*presentation*), abstração (*abstraction*) e controle (*control*). Qualquer que seja o nível hierárquico, a apresentação representa o comportamento perceptível do agente, a abstração implementa funções do domínio da aplicação de forma independente da apresentação e o controle une a abstração à apresentação.

A faceta controle ainda é responsável pela troca de informações entre a abstração e a apresentação bem como entre agentes PAC, suprindo uma deficiência da arquitetura MVC, que não possui um elemento explícito para comunicação entre agentes. As funções da faceta apresentação são similares àquelas do componente lógico de apresentação, as da faceta controle àquelas do componente de controle de diálogo e as da faceta abstração àquelas do componente de aplicação (Figura 2-1).

A hierarquia de agentes permite que o controle de um agente PAC gerencie, por exemplo, visões de uma mesma informação fornecidas por agentes "descendentes". A mudança em uma visão (fornecida por um agente) é sinalizada para o controle do agente ancestral que dá oportunidade para os demais agentes descendentes se atualizarem.

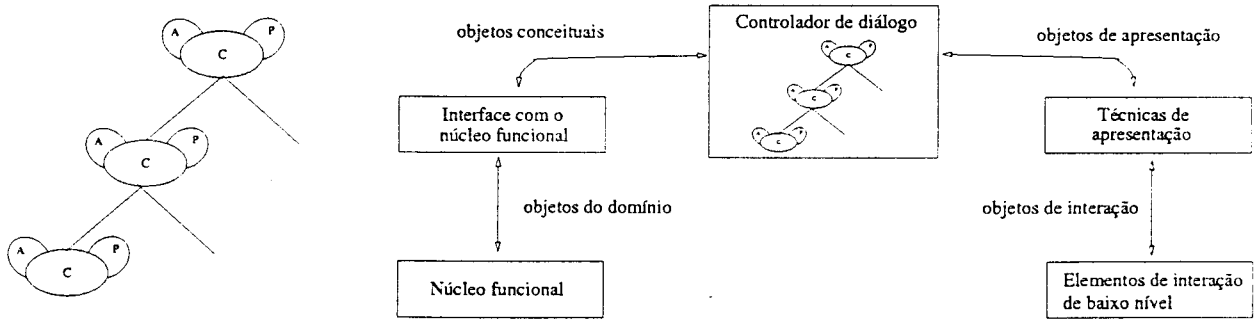


Figura 2-7: Arquiteturas PAC e PAC-AMODEUS

A arquitetura PAC-AMODEUS [115] é uma arquitetura híbrida (Figura 2-7, parte direi-

ta). Ela usa a arquitetura ARCH, vista anteriormente, como base para a divisão funcional do sistema. A modificação ocorre no gerenciador de diálogo, que é organizado em agentes, segundo o modelo PAC.

Rendezvous

A linguagem e arquitetura RENDEZVOUS foram propostos para simplificar a construção de aplicações *multimedia* distribuídas e, em particular, interfaces gráficas que empregam manipulação direta [53].

Nesta arquitetura a separação entre aplicação e interface é mantida por um objeto *link* que explicitamente mantém uma restrição (pág. 29) responsável pela comunicação entre um objeto *abstraction* (aplicação) e objetos *view* (interface), conforme mostra a Figura 2-8. Objetos view são organizados em hierarquias. O emprego de um objeto link como meio de comunicação entre aplicação e interface contrasta com propostas menos estruturadas e comumente empregadas, onde rotinas da aplicação são chamadas diretamente pelo gerenciador de diálogo.

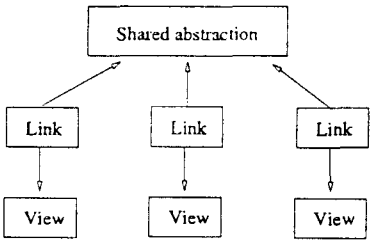


Figura 2-8: Arquitetura RENDEZVOUS

À semelhança da arquitetura DV, a arquitetura RENDEZVOUS combina as funções dos objetos controller e view (do modelo MVC) em um único objeto *view*. Um sistema interativo é dividido em um objeto abstraction e vários objetos view conectados por objetos link. Em uma aplicação convencional, um único objeto view é conectado a um único objeto abstraction através de um objeto link. Esta arquitetura faz uso de restrição (objeto link) para separar um objeto abstract de um objeto view sem necessidade de comunicação explícita entre eles. MVC, ao contrário, emprega chamadas de métodos e callbacks.

2.3.4 Comentários sobre Arquiteturas de Sistemas Interativos

A adoção de uma abordagem estruturada, em detrimento de uma abordagem ad hoc, não necessariamente elimina os problemas e requisitos conflitantes da organização do software de um sistema interativo. Por exemplo, uma solução de compromisso entre

desempenho e facilidade de manutenção é muitas vezes necessária: pode-se prejudicar o desempenho quando módulos de software adicionais são criados para isolar tecnologias sujeitas a mudanças; por outro lado, a ausência de tais componentes pode melhorar o desempenho ao mesmo tempo que compromete a manutenção.

Estas decisões, baseadas em necessidades e objetivos distintos, tornam inviável a identificação de uma única arquitetura de software capaz de expressar todas as decisões dos projetistas e que seja adequada a qualquer tipo de sistema. Em conseqüência, diversas arquiteturas têm sido propostas. Reutilização de código, desempenho e simplicidade são alguns dos objetivos que norteiam o projeto dessas arquiteturas [6]. A diferença entre elas é estabelecida, basicamente, pelo subconjunto de objetivos enfatizado.

A forma de troca de informações entre módulos e agentes não é prescrita pelas arquiteturas comentadas. Por exemplo, as arquiteturas ARCH e SLINKY não fixam previamente memória compartilhada como mecanismo de troca de informação entre os vários módulos dessas arquiteturas. Arquiteturas de software são modelos que guiam o processo de organização de um software sem impor restrições ou requisitos desnecessários. Por exemplo, embora o conceito de agente PAC possa ser capturado por um objeto em uma linguagem orientada a objetos, esse paradigma de implementação não é definido antecipadamente pela arquitetura, tornando essa decisão suscetível de ser influenciada por fatores cuja relevância pode variar de um sistema interativo para outro.

Na arquitetura MVC há uma grande interação entre objetos view e controller. Algumas arquiteturas (DV, por exemplo) reduzem esta sobrecarga de comunicação combinando as função desses objetos em apenas um objeto view. Uma tendência similar ocorre na arquitetura RENDEZVOUS.

Muitos widgets fornecidos por toolkits modernos combinam as funções desempenhadas por objetos controller e view. O *framework* MFC (Microsoft), por exemplo, oferece widgets que desempenham, simultaneamente, as funções de um objeto view e controller. Em conseqüência, o emprego de tal ferramenta impede a separação entre estes objetos conforme definido pela arquitetura MVC, por exemplo.

2.3.5 Acoplamento entre Interface e Aplicação

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 tenham repercussão limitada no outro. Embora essa separação seja idealmente desejada do ponto de vista de engenharia de software, a divisão de funções entre esses componentes envolve várias considerações [30, 49]. Uma breve e elucidativa discussão sobre o assunto pode ser obtida em [140].

A divisão de um sistema em interface e aplicação é amplamente defendida pelos benefícios inerentes. A existência de dificuldades de desenvolvimento e manutenção de grandes sistemas que não apresentam essa separação é um consenso. Para algumas ferramentas essa divisão é o principal objetivo [65]. Embora desejável, esse recurso comum às arquiteturas de software (divisão) contribui com as dificuldades de implementação de uma arquitetura [8, 59, 96, 104, 130, 142].

Essa separação produz componentes isolados que precisam comunicar entre si e, em particular, sistemas interativos modernos exigem grande fluxo de informação entre eles para prover *feedback* adequado para as ações dos usuários. A fronteira entre as responsabilidades desses componentes também pode não ser facilmente identificada e, em alguns casos, a transferência de funções entre eles pode se transformar em necessidade [6, 48, 140]. Tal fluxo pode requerer um desempenho incompatível com a divisão estabelecida, dificultando ou, em alguns casos, impedindo o seu emprego. Por exemplo, se o usuário arrasta um objeto sob o mouse (como uma peça do jogo de xadrez sobre um tabuleiro), então é necessária uma comunicação freqüente para permitir resposta adequada da interface, que colore de forma diferente as posições válidas e aquelas inválidas sobre as quais o cursor passa. Se parte da semântica da aplicação (posições legais para uma dada peça e cenário) é transferida para a interface, então parte do fluxo entre esses componentes é eliminada.

A separação entre interface e aplicação pode ser estimulada com o emprego de Xchart. A Seção 6.6 (pág. 193) fornece um exemplo.

2.4 Ferramentas de Apoio

A linguagem Xchart possui construções projetadas para a descrição do controle de diálogo de interfaces. Tais construções, contudo, podem também ser utilizadas para a descrição do controle de um domínio maior: sistemas reativos. Em conseqüência, as propostas aqui comentadas estão divididas entre aquelas voltadas para o desenvolvimento de interfaces e aquelas voltadas para a confecção de comportamentos complexos em geral. Elas fornecem um panorama geral de trabalhos correlatos. Tais propostas são denominadas, neste texto, *ferramentas*. Uma ferramenta, portanto, designa genericamente toda e qualquer tecnologia utilizada para auxiliar o desenvolvimento do software de interfaces.

2.4.1 Ferramentas de Desenvolvimento de Interfaces

Em decorrência do grande número de ferramentas de desenvolvimento de interfaces, apenas algumas das mais populares são comentadas nessa seção. Em [94], por exemplo, são listadas mais de 120 ferramentas que fazem uso de um variado conjunto de tecnologias, o que torna a avaliação e classificação delas uma tarefa cuidadosa [132]. Uma classifi-

cação e análise de ferramentas pode ser obtida em [101]. A classificação aqui apresentada enfatiza a funcionalidade da interface cujo desenvolvimento é apoiado por ferramentas. Houve uma tendência em considerar questões pertinentes ao controle de diálogo, pois esse componente é o responsável pelas funções que Xchart visa descrever. O trabalho em [101], em contrapartida, comenta ferramentas sem a ênfase aqui presente. Ferramentas também são discutidas em [48, 100].

Diagramas de Transição entre Estados (DTEs)

Em engenharia de software, Diagrama de Transição entre Estados, ou DTE, é uma das raras notações "amplamente" conhecidas e utilizadas [137]. Entre as propostas de desenvolvimento de interfaces, DTE é, provavelmente, o modelo mais conhecido. DTEs são empregados neste texto para caracterizar os diagramas convencionais, compostos apenas por estados e transições entre estados e inclusive suas variantes mais próximas: máquinas de Mealy, Moore [18], RTN (*Recursive Transition Networks*) e ATN (*Augmented Transition Networks*). As duas últimas são comentadas em [36].

REACTION HANDLER é uma das primeiras ferramentas conhecidas para o desenvolvimento de interfaces [105]. Nessa proposta, DTEs são utilizados para capturar o controle de diálogo. Desde então, DTEs têm sido empregados continuamente com essa finalidade. Em [10], os DTEs são utilizados para implementar o controle reativo existente em aplicações MS-WINDOWS. Um uso similar é apresentado em [112]. Em [136], os DTEs são estendidos com recursos de apresentação para descrever interfaces. McDaniel **et al.** [90] mostram benefícios do emprego de DTEs para a análise da interface de um sistema complexo. Lee [68, pág. 72] defende o emprego de DTEs para representar o relacionamento dinâmico entre objetos de interação onde os eventos são produzidos por ações dos usuários.

Thimbleby e Witten [129] conduziram um estudo envolvendo um forno microondas onde foi necessário descrever a programação do relógio do aparelho, cuja especificação estimada em DTEs conteria acima de 10000 estados. Contudo, o maior inconveniente dos DTEs, o grande crescimento do número de estados com o aumento da complexidade a ser capturada, não impede o seu emprego mesmo em desenvolvimentos reais e recentes. A definição da interface do sistema ATC [39], 200k linhas de código, empregou DTEs para descrever parte do controle de diálogo. Em [124] é apresentado um sistema que permite consultas remotas a dados referenciados geograficamente cuja interface foi projetada utilizando-se DTEs. Em [47] são comentados 5 desenvolvimentos de longa duração (mais de 2 anos) onde DTEs foram empregados nos projetos das interfaces.

Entre as críticas aos DTEs, Myers [102] sustenta que "muitos dos sistemas interativos são *mode-free*" ou ainda que interfaces que fazem uso de manipulação direta não apresentam modos [95]. Nesses casos o usuário teria uma grande variedade de opções de ação,

o que exigiria uma grande quantidade de transições saindo de cada estado. Jacob [61], contudo, afirma que apesar da aparência, interfaces que fazem uso de manipulação direta apresentam o diálogo conduzido por modos e ainda mostra como identificá-los. Myers também cita as características de interfaces para as quais DTEs são úteis para descrevê-las: (i) interfaces com grande número de modos, (ii) fluxo de controle global (por exemplo, um comando provoca a execução de uma caixa de diálogo da qual uma outra pode ser aberta e assim por diante) ou (iii) detalhes de baixo nível da operação de objetos de interação. DeSoi e Lively [27] acrescentam que DTEs fornecem uma abordagem gráfica para conectar a interface à aplicação em qualquer nível da especificação, desde detalhes de baixo nível a objetos de interação de alto nível.

Conforme [139], o conceito de modo pode ser um recurso utilizado para melhorar a qualidade de uma interface. Hix e Hartson [54, pág. 213] também afirmam que muitas interfaces contêm muitos estados, que são vistos pelos usuários como modos e sugerem o emprego de DTEs para capturá-los. Puerta [110] afirma que muitos modelos de diálogo têm mostrado evidências de sucesso e, entre eles, estão os DTEs. A recente metodologia de projeto de interface apresentada em [111] também emprega os DTEs, que são utilizados para capturar o comportamento interativo de objetos. O amplo emprego de DTEs e notações derivadas (Statecharts, por exemplo), neste contexto, pode ser compreendido pela dificuldade de descrever tais sistemas sem a noção de estado [91].

Statechart

Statechart [40] estende os DTEs eliminando inconvenientes dessa linguagem (o crescimento do número de estados é um deles) sem perder suas características gráficas (linguagem visual). Statechart ainda é formal. Statechart acrescenta concorrência aos DTEs, o que permite um crescimento mais controlado do número de estados à medida que a complexidade do controle especificado aumenta. Ou seja, o principal argumento desfavorável ao emprego de DTEs não se aplica a Statechart. Segundo Harel [41], formalismos visuais apropriados podem ter um grande impacto positivo no trabalho de especialistas em computação.

No presente texto, variante de Statechart, ou simplesmente variante, é toda e qualquer linguagem que apresenta recursos herdados de Statechart, conforme definida em [40], [45] e [44]. É interessante observar, contudo, que tais referências apresentam versões distintas de Statechart (pág. 110).

O emprego de Statechart para descrever interfaces é sugerido em vários trabalhos [5, pág. 166], [40, 120]. Em [17] uma variante de Statechart é utilizada para descrever o comportamento de objetos de interação. JASMINUM [16] é uma proposta para descrição de interfaces que faz uso de ADVcharts (variante de Statechart). Ferramentas baseadas em Statechart para o desenvolvimento do software de interfaces são descritas em [116] e

[138]. Statechart também pode ser utilizada na descrição de manipulação direta conforme [134]. Em [85] são apresentados os resultados obtidos do emprego de Statechart neste domínio.

O interesse por Statechart não se restringe à descrição de interfaces (um subconjunto dos sistemas reativos). STATEMATE [43] e BETTERSTATE [2] são ferramentas comerciais construídas em torno de Statechart para o desenvolvimento de sistemas com comportamentos complexos em geral. UML [22] é uma coleção de técnicas voltadas para a especificação, visualização, construção e documentação de software apoiada por um grande consórcio de empresas. A definição de UML inclui o que de melhor existe nas metodologias de desenvolvimento de software, conforme sustentam os autores. Segundo UML, a descrição de comportamento é feita através de uma variante de Statechart. Modelos para o desenvolvimento de sistemas orientados a objetos deveriam ser, do ponto de vista de comportamento, expressivos e rigorosos bem como intuitivos e bem estruturados. Ainda deveriam ser suficientemente precisos para produzir modelos executáveis, conforme sustentado em [42], que apresenta uma variante de Statecharts para desempenhar esse papel.

Beeck [7] compara 20 variantes de Statechart. Em [31] é descrita uma técnica para implementação dessa linguagem baseada no paradigma de objetos. Glinz [33] apresenta uma variante para descrever cenários com o propósito de tornar a linguagem resultante mais oportuna que Statechart neste domínio. A geração de código a partir de descrições em Statechart é contemplada em [119]. Em [81] é descrita uma máquina abstrata que executa diagramas em Statechart. Davis [23] mostra evidências favoráveis ao emprego de Statechart para descrever comportamento reativo. OMT é uma metodologia que faz uso de uma variante de Statechart para descrever o modelo dinâmico de objetos [114].

Apresentação e Manipulação Automática de Dados

Muitas ferramentas fornecem suporte à implementação da apresentação de uma interface através de programação. Ou seja, cada detalhe de um dado da aplicação a ser apresentado a um usuário deve ser previamente analisado para que um projeto correspondente seja definido. Alternativamente, a apresentação de dados pode ser automaticamente gerada a partir de uma definição deles.

A tradução de dados da aplicação em representações gráficas é empregada em [87]. No sentido inverso, essa tradução também pode ser estendida para contemplar manipulação direta, ou seja, o usuário manipula representações gráficas com conseqüente implicação sobre os dados da aplicação subjacente. Por exemplo, se um objeto gráfico é removido, então os dados pertinentes da aplicação também são removidos. TRIP2 [89] é uma ferramenta que oferece suporte à tradução bidirecional. Uma limitação comum de abordagens similares está no conjunto restrito de regras que governa a tradução automática. Bons

resultados, contudo, são atingidos apenas em domínios muito especializados. Os recursos fornecidos por essa abordagem compreendem apenas parte da funcionalidade de uma interface. A sintaxe de interação, por exemplo, não é contemplada.

Geração Automática da Interface

Muito esforço pode ser necessário para produzir uma versão inicial de uma interface. A ferramenta UOFA* converte automaticamente uma descrição de alto nível da interface (fácil de ser produzida) em uma versão inicial da apresentação e do controle de diálogo da interface [122]. Posteriormente esses componentes são iterativamente refinados pelo projetista da interface, a partir da versão inicial cujos detalhes foram automaticamente fornecidos pela ferramenta. Embora uma primeira versão seja obtida a um custo baixo, o refinamento dessa versão provavelmente é inevitável para um grande número de interfaces além de contemplar adequadamente apenas um subconjunto reduzido de interfaces.

Construtores de Interfaces

Construtores de Interfaces (*Interface Builders*) estendem toolkits com recursos que permitem o projetista de uma interface interativamente definir caixas de diálogos, menus e outros componentes básicos de uma interface. Objetos de interação são posicionados através do uso de mouse e código pertinente ao projeto interativamente fornecido pode ser automaticamente gerado. GILT é um exemplo [102].

Construtores de interfaces enfatizam parcialmente as funções de apresentação de uma interface. Por exemplo, a área de desenho de um editor gráfico não é apoiada por esse tipo de ferramenta. Os aspectos desenvolvidos com o apoio desse tipo de ferramenta ainda são estáticos. Por exemplo, não há como especificar que determinada opção de menu torna-se desabilitada ou habilitada conforme valor de determinada variável.

Tratadores de Eventos

A linguagem ERL (*event-response language*) permite descrever a sintaxe de interação de uma interface através de respostas do sistema às ações de um usuário [52]. ERL possui o mesmo poder de expressão de DTEs e permite a descrição de diálogos concorrentes de forma compacta através de regras: se uma condição é satisfeita, então uma ação preestabelecida é executada. Sassafras [51] é um sistema de execução para ERL. Sassafras fornece suporte à execução de diálogos concorrentes, à comunicação entre componentes de um sistema interativo e à sincronização entre eles.

ALGAE [32] é uma linguagem projetada para capturar diálogos concorrentes através da definição de tratadores de eventos com sintaxe baseada na linguagem C. Tratadores de eventos residem nos componentes de uma interface, organizada segundo a arquitetura

de Seeheim e cuja comunicação entre seus componentes é provida pela troca de eventos entre eles. Eventos podem ser gerados tanto pela aplicação quanto pela interface.

Embora Sassafras ofereça mecanismos sofisticados para a implementação de diálogos concorrentes, uma descrição em ERL, ALGAE ou em outras linguagens baseadas em eventos apresenta inconvenientes. Myers [102] cita que é difícil criar código correto, pois o fluxo de controle não é localizado e pequenas mudanças podem afetar diferentes partes de um programa além de ser difícil para o projetista compreender o código de sistemas grandes.

Uma comparação entre o modelo oferecido por tratadores de eventos, DTEs e gramáticas é apresentado em [36].

Restrições

Várias áreas da computação beneficiam-se do conceito de restrição [50]. Uma *restrição* é uma relação entre valores. Sempre que um valor é alterado, aqueles dependentes são devidamente recalculados para manter a validade das restrições. Embora restrições produzam vários benefícios no domínio de interfaces [12], elas ainda são apoiadas por um reduzido número de ferramentas. O principal emprego de restrições neste domínio é manter o leiaute de objetos da aplicação na tela do computador. Outros empregos incluem a manutenção do mapeamento entre objetos da aplicação e objetos gráficos, manutenção de múltiplas visões de um mesmo objeto da aplicação e animação.

Tais funções, contudo, fornecem apenas parte da funcionalidade de uma interface. A seqüência (sintaxe) de interação, por exemplo, pode ser descrita mais apropriadamente com DTEs. A carência de abstrações (notações) para especificar uma interface usando restrições e a complexidade de manutenção das restrições quando são em grande número, por exemplo, estão entre as dificuldades dessa abordagem [27], além da demanda computacional exigida para satisfazer grandes redes de restrições [56, 143]. AMULET, comentado abaixo, é uma das ferramentas que melhor explora essa tecnologia. Em AMULET, restrições são descritas em código C++ através de um modelo de difícil aprendizado e emprego. A carência de abstrações de especificação reflete-se na inexistência de abordagem que permita fragmentar grandes redes de restrições em pequenos módulos.

Amulet

A maioria das ferramentas disponíveis oferece uma coleção de widgets que lida com aspectos de apresentação de uma interface. Em alguns sistemas interativos, contudo, parte significativa do código de uma interface é pertinente à manipulação de elementos gráficos para os quais não existem widgets disponíveis. Por exemplo: editores gráficos, ferramentas CAD e outros. Nestes casos, tal código é programado diretamente através da interface de programação do sistema de janelas em uso, sem suporte de alto nível. AMULET [101], ao

contrário, fornece várias abstrações para implementar esse tipo de código: objetos gráficos (atualização automática); interactors (manipulam interações bem comportadas); objetos de comando (manipulam operações de edição); restrições (mantêm relacionamentos entre objetos); gestos (reconhecimento) e animações (tipo de restrição especial).

Os principais atrativos de AMULET estão no acúmulo, em uma única ferramenta, de vários conceitos e recursos de programação correspondentes para acelerar o desenvolvimento do software de interfaces: restrições e interactors estão entre os principais. Restrições foram comentadas na seção anterior. Interactors encapsulam comportamentos em objetos, que são "conectados" a objetos de apresentação fornecendo-lhes reação a ação de um usuário. Interactors são apropriados se os comportamentos desejados estão definidos ou podem ser compostos de outros mais simples. A dificuldade de Interactors reside no suporte apenas a comportamentos "aceitos" e comumente empregados. Ou seja, manipulações típicas de mouse/teclado: arrastar um objeto, selecionar um elemento dentre uma lista e outros. Não são indicados, portanto, para estabelecer a seqüência de interação com o usuário.

Os benefícios de AMULET não possuem custo zero. O sofisticado modelo oferecido exige um programador especializado. A extensa documentação, por exemplo, fornece indícios acerca dos custos necessários para dominar o emprego dos mecanismos oferecidos.

Alpha

ALPHA [65] é apresentada como uma ferramenta voltada principalmente para auxiliar a separação entre interface e aplicação. Sistemas interativos desenvolvidos com o apoio de ALPHA são organizados em uma coleção de agentes. O elemento de ligação entre esses agentes é fornecido por um gerenciador de diálogo que centraliza todo o comportamento da interface descrito na linguagem Slang. Slang é uma linguagem de programação similar a C. Um programa escrito em Slang é denominado de diálogo. Cada agente dá origem a um processo. A comunicação entre agentes é realizada necessariamente via um processo que executa o gerenciador de diálogo. Toda a comunicação faz uso de TCP/IP, ou seja, agentes podem estar distribuídos sobre nós de uma rede.

Abordagens Baseadas em Modelos

Abordagens baseadas em modelos (*model-based*) usam uma base de conhecimento central para armazenar a descrição de todos os aspectos do projeto de uma interface. A descrição é conhecida por um modelo e contém informações acerca das tarefas que os usuários irão realizar, dados da aplicação, os comandos que os usuários podem realizar, a apresentação e o comportamento da interface além das características dos usuários. Interfaces são desenvolvidas via ferramentas utilizadas em tempo de projeto para construir e refinar

modelos. Interfaces são automaticamente produzidas dos modelos confeccionados.

Em vez de programar uma interface usando um toolkit ou abordagens que fazem uso de recursos de baixo nível, desenvolvedores produzem uma descrição da interface em linguagens de especificação de alto nível [125]. Por exemplo, embora restrição seja um recurso poderoso, atualmente ele pode ser usufruído através de programação de baixo nível, enquanto linguagens de mais alto nível poderiam facilitar o seu emprego [27]. Abordagens baseadas em domínio visam permitir que especialistas em um domínio e que não são programadores possam desenvolver sistemas complexos.

Esse objetivo também foi parcialmente perseguido por UIMSs (*User Interface Management Systems*) [101]. Muitos UIMSs empregavam DTEs para descrever o diálogo, enquanto outras funções eram desenvolvidas em outras linguagens. Embora abordagens baseadas em modelos sejam mais sofisticadas do que os UIMSs predecessores, elas ainda não são populares. Os desenvolvedores de software empregam, na maioria, construtores de interfaces, toolkits e linguagens de programação para construir interfaces. Abordagens baseadas em modelos são, contudo, promissoras, pois da mesma forma que linguagens como C substituíram, praticamente, o emprego de linguagens de máquina, acredita-se que elas possam substituir as abordagens que empregam recursos de baixo nível atualmente empregadas.

Existe um variado conjunto de ferramentas baseadas em modelos. Contudo, elas ainda não atingiram uma maturidade satisfatória. Em [127] são apresentadas algumas dificuldades: (i) as linguagens existentes não são expressivas o suficiente; (ii) desempenho não satisfatório (interpretação) e (iii) são difíceis de serem utilizadas, comparando-se com construtores de interfaces atualmente disponíveis (desenvolvedores não conhecem as linguagens de especificação ou não desejam aprendê-las). Essa abordagem é especialmente atrativa enquanto fornece uma alternativa promissora para as ferramentas atualmente existentes. MOBI-D [110] é uma recente ferramenta baseada em modelos.

2.4.2 Ferramentas de Desenvolvimento de Controle

À semelhança das abordagens para desenvolvimento de interfaces, existe um coleção considerável de propostas para a descrição de controle complexo. Em [70] são comparados os resultados do emprego prático de 18 abordagens, inclusive Statechart, no desenvolvimento de um problema comum. Em [7] são comparadas 20 variantes de Statechart. Halbawachs [38] comenta em detalhe aspectos pertinentes à implementação de várias abordagens, Statechart e outras, voltadas para o desenvolvimento de sistemas reativos. Tais referências apresentam uma extensa lista de outros trabalhos e uma abrangente visão do desenvolvimento de sistemas reativos.

Ao contrário dessas referências, essa seção tem o propósito de mostrar evidências

favoráveis ao emprego de Statechart para descrever sistemas reativos, um superconjunto de gerenciadores de diálogo, em vez de exaustivamente discutir o desenvolvimento de tais sistemas.

RS

A reação de um sistema reativo descrito na linguagem RS [131] é organizada em regras: $C \Rightarrow A$. C é uma condição de disparo e A é a ação executada se a regra é executada. RS foi projetada com a intenção de facilitar a descrição do controle de sistemas reativos e a implementação eficiente desse controle. No contexto do desenvolvimento de interfaces essa proposta pode ser vista como um tratador de eventos (comentada em seção anterior).

O emprego de regras também é adotado em [14] para a descrição do comportamento de sistemas distribuídos. Contudo, esta proposta elimina um inconveniente de sistemas baseados em regras: a dificuldade de interpretação de um grande número de regras. O usuário manipula descrições em uma versão simplificada de *Statechart*, que posteriormente são convertidas em regras. O trabalho em [14], contudo, enfatiza tolerância a falhas em sistemas distribuídos—objetivo além dos interesses imediatos de interfaces em geral.

ROOM

ROOM [117] compreende a linguagem ROOMCHARTS e ferramentas de suporte que tentam eliminar dificuldades com metodologias de propósito geral para o desenvolvimento de sistemas de tempo real. ROOMCHARTS herda vários recursos de Statechart para descrever o comportamento de tais sistemas.

Modelo Dinâmico de Objetos

OBJECTCHART [21] e OMT [114] são abordagens que empregam variantes de Statechart para descrever o comportamento dinâmico de objetos. A inexistência de um modelo formal é o motivo de insucesso de algumas metodologias, conforme [42], que apresenta uma outra abordagem baseada em Statechart. O cuidado com a semântica formal, contudo, não é suficiente. A integração entre o comportamento descrito em Statechart e os demais aspectos de um sistema também é importante.

Outras Abordagens

No início desta seção foram apresentados trabalhos contendo comentários sobre várias linguagens empregadas no desenvolvimento de sistemas reativos em geral. Uma investigação dessas linguagens, contudo, está além do escopo do presente trabalho.

O controle de diálogo também pode ser descrito na linguagem em forma de texto CSP [1]. Esse emprego está fundamentado na semântica formal e executável de CSP. Essas características, embora positivas, não são suficientes. Statechart, por exemplo, é uma linguagem formal e executável, além de visual.

2.5 Xchart: Uma Nova Proposta

> *The challenge of formal methods (precise notations and mathematical models) in interactive system design and human computer interaction is to produce a precise framework in which the role and scope of these models may be clearly understood.*
> Harrison e Thimbleby [46, pág. 2]

Dentre as ferramentas para o desenvolvimento do software de interfaces, não se espera que uma única abordagem se sobressaia [27]. Muitas possuem propósitos distintos que contemplam funções variadas e que coexistem em uma mesma interface. Em conseqüência, algumas abordagens atualmente disponíveis continuarão em uso no horizonte que se pode vislumbrar.

O tipo de interface ainda pode induzir o emprego de determinada abordagem. Green [36], por exemplo, afirma que alguns tipos de interfaces são muito mais simples de serem descritas através de DTEs do que através de abordagens baseadas em tratadores de eventos e gramáticas. Para descrever o controle de diálogo de uma interface há evidências favoráveis ao emprego de DTEs (pág. 25) e, mais recentemente, da extensão mais conhecida de DTEs: Statechart (pág. 26). O interesse demonstrado na literatura e a existência de ferramentas, inclusive comerciais, que fazem uso dessas abordagens, tornam notórias essas evidências.

Essas evidências não impediram, contudo, o surgimento de variantes de Statechart, ou seja, linguagens com recursos herdados de Statechart mas que apresentam diferenças em relação à versão original. Statechart foi proposta para descrever sistemas reativos. O foco em um subdomínio dos sistemas reativos, a aplicação em outros domínios, as questões em aberto levantadas em [40] ou mesmo a maturação natural de toda e qualquer linguagem fomenta o surgimento dessas variantes. Um ponto comum entre as variantes são "ajustes", em alguns de seus recursos, com o objetivo de adequadamente capturar o subdomínio de sistemas reativos em consideração. Hong **et al** [55] afirmam:

> Though the graphical syntax of statecharts has been defined, there remain concepts that are still ambiguous or need to be improved.

Harel afirma que a sintaxe e a semântica de Statechart não são definitivas, adaptações devem ser efetivadas para acomodar sutilezas dessa linguagem [40]. Em resposta, um rico

conjunto de variantes de Statechart existe atualmente. A existência dessas variantes mostram que a dupla Statechart e STATEMATE [43] (versão "oficial" da linguagem e ambiente de apoio) não é vista como definitiva em projeto e implementação de código baseado em DTEs. Uma afirmação semelhante é sustentada em [144].

O emprego de Statechart no domínio específico de interfaces, embora estimulado por alguns trabalhos e efetivamente utilizada com sucesso (Seção 2.4.1), não faz uso de recursos que podem aumentar a expressividade e eliminar dificuldades desta linguagem neste contexto [85]. A análise do emprego de Statechart no domínio de interfaces e as mudanças sugeridas em [85] deram origem à linguagem Xchart. As idéias iniciais tanto da linguagem Xchart quanto do ambiente de apoio originaram-se a partir de [72]. Posturas similares foram empregadas em outros trabalhos que definiram várias variantes de Statechart e, conforme [41], resultados promissores têm sido obtidos.

Novamente, Statechart foi projetada para descrever sistemas reativos, um superconjunto de gerenciadores de diálogos. A ênfase em uma grande classe de sistemas não estimula a presença de recursos adequados a um subconjunto dessa classe. Por exemplo, recursos úteis à descrição de interfaces baseados em DTEs, diagramas recursivos [26], foram identificados bem antes do surgimento de Statechart, embora esta linguagem não os incorpore.

Xchart propõe modificações para estender ainda mais o sucesso parcial de Statechart neste domínio, conforme proclamado em alguns trabalhos. Diferenças entre Xchart e Statechart são apresentadas na Seção 3.11 (pág. 110). Xchart surge

- das limitações atualmente existentes nas ferramentas disponíveis (Seção 2.2). Pesquisas devem continuar produzindo alternativas e melhorando as propostas atuais—esse é um dos raros consensos neste domínio. No futuro adiante, ninguém vislumbra nem mesmo a existência de um possível "nível de solução"; e

- das evidências favoráveis ao emprego de linguagens precursoras, inclusive Statechart, para descrever interfaces. Xchart é uma proposta fundamentada em experiências anteriores com Statechart neste domínio, o que permitiu identificar recursos desejados para Xchart.

Uma das dificuldades comuns entre muitos trabalhos é a ausência de um exemplo claro do emprego da proposta em questão. Nem sempre é nítido o tipo de ajuda fornecida, como ela se insere no desenvolvimento de todo um software e se integra com as técnicas empregadas nesse desenvolvimento. O restante dessa seção elucida essas e outras considerações para a linguagem Xchart.

2.5.1 Fundamentos

XCHART É UMA ABORDAGEM BASEADA EM MODELOS.

Alguns dos inconvenientes das atuais ferramentas, descritos na Seção 2.2, são objetivos que propostas baseadas em modelos visam atingir (pág. 30). A maioria das ferramentas existentes fornecem recursos de baixo nível através, principalmente, de programação. Uma alternativa é empregar linguagens de alto nível para capturar o projeto de interação de uma interface e, a partir das descrições obtidas, automaticamente gerar o complexo código de uma interface. Para capturar tal projeto é necessário um conjunto de linguagens. Em particular, Xchart é uma delas que visa capturar o controle de diálogo. Esse emprego visa obter objetivos similares àqueles do emprego de Statechart para representar programas em Occam ou C, por exemplo. Se não existem benefícios claros na representação em alto nível (Statechart) de construções naquelas linguagens de programação, então os algoritmos descritos em [119] e [81], respectivamente, teriam pouca utilidade.

Embora a filosofia de abordagens baseadas em modelos seja simples e promissora, existem várias dificuldades nesse caminho [127]: (i) escopo restrito das linguagens (Xchart, por exemplo, captura apenas parte das funções de uma interface); (ii) baixo desempenho (o Capítulo 5 descreve o sofisticado sistema de execução de Xchart. Esse sistema foi projetado para interpretar descrições nessa linguagem de forma eficiente.) e (iii) difícil de usar (empregar Xchart é uma espécie de programação que exige conhecimento da elaborada semântica de Xchart. Em muitos casos, projetistas não possuem esse conhecimento ou não estão dispostos a adquiri-lo).

Essas dificuldades são compatíveis com as várias observações feitas, nesse capítulo, acerca dos desafios de se capturar e desenvolver o controle de diálogo de uma interface. Xchart não é apresentada como uma solução para estas dificuldades. Xchart é uma proposta que pretende dar um passo adiante de abordagens baseadas em DTEs nesse domínio. De certa forma, isso significa mostrar como as diferenças entre Statechart e Xchart favorecem essa última linguagem. Evidências nesse sentido são fornecidas por todo esse texto e, em particular, no Capítulo 6.

XCHART É FORMAL.

O emprego de linguagens formais para a especificação de interfaces não é recente [60]. O crescimento da complexidade dos sistemas e a possibilidade de erros sutis, inclusive fatais, que podem ser atribuídos à interface, tornam promissor o emprego de linguagens formais. O emprego de métodos formais não garante o sucesso de um sistema, mas pode revelar ambigüidades, inconsistências e ausência de elementos que, de outra forma, poderiam não ser detectados [20]. Nesse sentido, Xchart preenche um requisito não atendido por muitas outras abordagens, que fazem uso

de protótipos e outras técnicas não formais para especificar interfaces. MOBI-D [110] é uma proposta recente baseada em modelos que emprega linguagens formais na definição do projeto de interação de interfaces. O emprego de abordagens formais, contudo, ainda possui mais perguntas do que respostas [113]. Xchart tenta responder algumas dessas perguntas como indicado nas próximas seções.

XCHART É VISUAL.

Embora formal, especificações em Xchart não compreendem obscuras e volumosas descrições em forma de texto, ao contrário, suas construções incluem símbolos gráficos (retângulos com cantos arredondados, arestas e outros). O emprego de recursos visuais não garante a legibilidade de uma linguagem, embora construções cuidadosamente elaboradas possam produzir benefícios. De forma análoga, acredita-se que elementos gráficos possam ser alternativas mais simples de serem empregadas do que as construções oferecidas por linguagens de programação como C++.

XCHART É EXECUTÁVEL.

Esta característica permite a construção rápida de protótipos, a simulação de especificações [15] ou ainda a implementação dessas especificações [81]. Xchart, contudo, não apenas oferece uma linguagem executável, mas conecta essa linguagem às ferramentas comumente empregadas no desenvolvimento de software. O Capítulo 5 fornece detalhes.

2.5.2 Origem do Nome

Por curiosidade, a 22ª letra do alfabeto grego (**X** e χ) é identificada em Inglês pela palavra CHI, que coincide com o acrônimo de *Computer-Human Interface*. Um diagrama ou *chart* na linguagem proposta no presente texto é utilizada no domínio de interfaces e, em conseqüência,

$$\text{XCHART} = \text{interface (X)} + \text{diagrama (CHART)}$$

2.5.3 Função

Do ponto de vista funcional, Xchart é uma linguagem para a descrição das funções do controle de diálogo de uma interface. Ou seja, capturar as funções de apresentação e de aplicação estão além dos propósitos dessa linguagem. Embora Xchart possa ser utilizada para descrever o controle existente em uma aplicação, esse emprego não é considerado nesse trabalho.

Do ponto de vista de arquitetura de software, Xchart contempla arquiteturas baseadas no modelo de agentes (Seção 2.3.3). Em outras palavras, o controle de diálogo e o sistema

interativo podem estar organizados em múltiplos agentes. Nesse caso, as funções do gerenciador de diálogo estão divididas entre vários agentes, que também possuem outras funções (apresentação, por exemplo). O Sistema de Execução de Xchart (Capítulo 5) assume algumas responsabilidades: comunicação entre agentes para fins de controle e entre estes e os demais componentes de um sistema interativo. O sistema de execução é flexível o suficiente para auxiliar a implementação das arquiteturas baseadas em múltiplos agentes PAC e MVC usando ferramentas convencionais (toolkits) para a implementação da apresentação da interface. Dessa forma, o projetista de software trabalha em um nível de abstração mais elevado, sem se preocupar com os mecanismos para a implementação de tais arquiteturas, pois eles são fornecidos através de construções de alto nível, o que não exige o emprego de programação convencional. Segundo Bass e Coutaz [5], é mais útil e também mais difícil construir uma ferramenta de apoio ao emprego de uma arquitetura do que a definição da arquitetura em questão.

2.5.4 Domínio

A linguagem Xchart possui recursos elaborados para capturar gerenciadores de diálogo sofisticados. Interfaces relativamente simples onde, por exemplo, não é empregada concorrência ou grande parte da interface pode ser desenvolvida através do emprego de um construtor de interfaces, não caracterizam adequadamente o domínio de aplicação de Xchart. O domínio de emprego de Xchart pode ser caracterizado pelos tipos de interfaces abaixo:

- Interfaces com grande número de modos e/ou cuja seqüência de interação seja sofisticada. Por exemplo, as interfaces de sistemas de editoração eletrônica, CAD e outros apresentam comportamentos complexos, que requerem uma notação adequada para descrevê-los. As deficiências do emprego de linguagens de programação para descrever tais comportamentos são bem conhecidas [107].

- Interfaces concorrentes. O usuário pode conduzir várias tarefas simultaneamente de forma independente das operações realizadas pela aplicação. Ou seja, concorrência pode existir apenas na interface (entre elementos da interface) ou ainda entre a interface e a aplicação. Isso permite, por exemplo, interação contínua com o usuário enquanto concorrentemente a aplicação é executada.

- Gráficos da aplicação. O comportamento de elementos gráficos pertinentes a aplicação pode ser descrito em Xchart. Em geral, construtores de interfaces apresentam mecanismos limitados para manipular os gráficos da aplicação. Em um editor gráfico, por exemplo, pode-se estabelecer, em Xchart, uma conexão entre uma reta e um retângulo de forma que, se o usuário arrasta um desses elementos, o outro

acompanha os movimentos do mouse. Essa é uma associação típica que pode ser descrita em Xchart.

Nos casos acima, Xchart não fornece recursos para o desenvolvimento de toda a interface ou sistema interativo pertinente. Xchart fornece suporte ao desenvolvimento dos gerenciadores de diálogos de tais interfaces.

2.5.5 Processo

O projeto de interfaces é um processo iterativo. Continuamente usuários e avaliações realimentam o projeto até que um resultado satisfatório seja obtido. As iterações podem melhorar substancialmente a qualidade do projeto resultante [106]. Esse processo é diferente daqueles provenientes da engenharia de software onde, basicamente, um projeto é realizado, implementado e posteriormente avaliado. O primeiro focaliza o usuário, enquanto os últimos o sistema.

Os modelos de processo de projeto de interfaces enfatizam a constante avaliação e criação de protótipos. Estas atividades também existem em modelos da Engenharia de Software como o modelo Espiral [11]. Contudo, os modelos da Engenharia de Software, inclusive o modelo Espiral, foram projetados com o objetivo principal de desenvolver as funções de um sistema (componente de aplicação). Para identificar o melhor nome para um rótulo de um menu, por exemplo, podem ser necessárias várias iterações e avaliações com os usuários. Essas iterações podem ser realizadas em poucos minutos. A noção de um ciclo no modelo Espiral é, contudo, bem mais sofisticada do que cada uma dessas iterações necessita.

A inadequação dos modelos de engenharia de software estimulou a comunidade envolvida com interfaces a confeccionar modelos mais apropriados para tratar o desenvolvimento de interfaces. O modelo Estrela [47] é um dos mais conceituados, segundo a comunidade de interfaces, para o desenvolvimento de sistemas interativos.

Está além dos propósitos desse trabalho investigar como os resultados produzidos por tais comunidades podem ser combinados para beneficiar o desenvolvimento de sistemas interativos. Atualmente, elas tratam paralelamente, conforme definições, métodos e ferramentas próprios o mesmo problema, sem considerar a existência de pontos em comum. Essa não é uma questão recente [28, 135] e recebe extensivo tratamento em [128]. Contudo, perspectivas diferentes persistem assim como a necessidade de um processo de desenvolvimento que integre o desenvolvimento da interface e da aplicação atendendo o que prescrevem ambas as comunidades [13].

No presente trabalho, em vez de identificar precisamente o *instante* em que a abordagem Xchart é empregada, é identificada a *tarefa*, necessária ao desenvolvimento de interfaces, para a qual Xchart é proposta para capturar um de seus produtos: a definição

do gerenciador de diálogo. Dessa forma, evita-se mais considerações acerca de modelos sem prejudicar a caracterização de Xchart.

Hix e Hartson [54] apresentam um conjunto de atividades necessárias ao desenvolvimento de um sistema interativo sem, contudo, estabelecer conexões temporais entre elas. A Figura 2-9 apresenta tais atividades. Aquelas acima da linha tracejada tratam do desenvolvimento da aplicação de um sistema interativo. Aquelas abaixo da linha tracejada focalizam o desenvolvimento da interface. As conexões refletem a troca de informações entre atividades, o que é diferente da relação temporal entre as atividades conectadas.

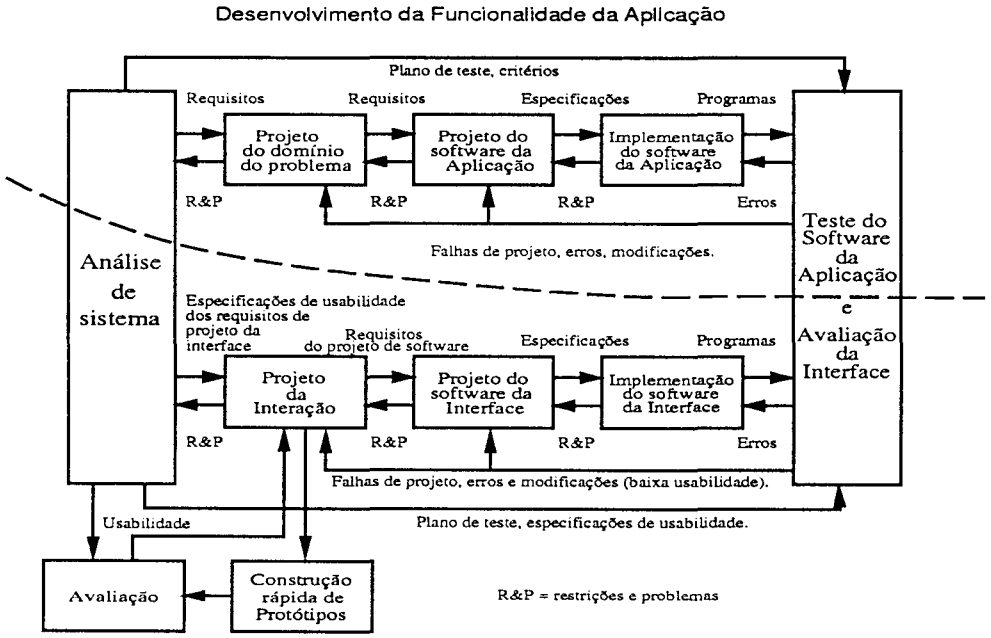


Figura 2-9: Modelo de desenvolvimento de sistemas interativos (adaptado de [54])

Uma descrição extensiva desse modelo de atividades é apresentada em [54]. Duas das atividades apresentadas, contudo, merecem atenção especial: o projeto da interação e o projeto do software da interface. Essas duas atividades são extraídas desse modelo e exibidas na Figura 2-10. O projeto da interação faz parte do domínio do comportamento, ou seja, preocupa-se com as ações dos usuários, feedback, aparência da tela, tarefas realizadas pelos usuários, leiaute de tela e estilos de interação. O projeto do software da interface envolve questões do projeto de software como algoritmos, estruturas de dados, arquitetura de software, toolkits e outras.

Conforme a Figura 2-10, de um lado tem-se o domínio do comportamento, onde as representações envolvidas abstraem-se de questões de software. Elas focalizam a análise de tarefas, análise funcional, alocação de tarefas e modelagem do usuário. A ênfase está no

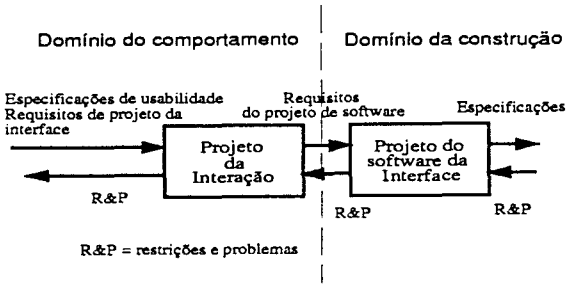


Figura 2-10: Projeto da Interação × Projeto do Software da Interface.

usuário. Do outro lado, domínio de construção, as representações envolvidas dão suporte às atividades de um projetista e responsável pela implementação do software da interface. A ênfase está no código que implementa a interação. Representações desses domínios têm propósitos distintos. Aquelas empregadas para capturar o projeto da interação serão, em um instante posterior do desenvolvimento de uma interface, convertidas em representações empregadas no domínio de construção. Em [54] é sugerido o emprego da linguagem UAN (*User Action Notation*) para capturar parte do projeto da interação de uma interface. Neste trabalho, a linguagem Xchart é proposta para capturar parte do projeto do software da interface (controle de diálogo).

Ao longo do desenvolvimento de uma interface, o projeto da interação terá que ser convertido, em algum instante, em representações apropriadas ao projeto do software da interface em questão. Xchart será utilizada nesse instante para capturar parte do projeto da interação do ponto de vista do projetista do software da interface.

Esse processo é compatível com abordagens baseadas em modelos. Por exemplo, o modelo de diálogo segundo MOBI-D [110] é parcialmente obtido do modelo de tarefas anteriormente produzido. Xchart é uma proposta de modelo de diálogo, em vez da inexistência desse modelo que, na grande parte dos casos, são programados em uma linguagem de programação.

Capítulo 3

Linguagem Xchart

The complexity of software is an essential property, not an accidental one. Hence, descriptions of a software entity that abstract away its complexity often abstract away its essence.
Fred Brooks Jr, IEEE Computer, 10-19, 1987

Os objetivos, fundamentos e motivações da linguagem Xchart foram apresentados no capítulo anterior. Este capítulo apresenta informalmente Xchart como uma linguagem de alto nível para a especificação de controle complexo. Os recursos de Xchart podem ser combinados de várias formas, algumas combinações válidas, outras não; algumas simples, outras sofisticadas; há ainda aquelas de comportamento sutil, onde uma análise cuidadosa faz-se necessária para obter uma interpretação correta. Em casos não-triviais, a semântica formal pode ser imprescindível (Capítulo seguinte).

Xchart herda recursos de Statechart. Apesar das semelhanças léxicas e sintáticas, as construções acrescentadas, as alterações e o modelo de controle (Seção 3.1) oferecido por Xchart tornam essa linguagem, em muitos aspectos, bem diferente de Statechart. Algumas diferenças são apresentadas na Seção 3.11. Ao longo de todo o Capítulo presente capítulo são fornecidos exemplos para ilustrar os conceitos apresentados. Os exemplos não têm o objetivo de servir a algum outro propósito. Estudos de casos empregando Xchart, no domínio de interfaces, são fornecidos no Capítulo 6. Os principais conceitos da linguagem Xchart são introduzidos na Seção 3.2.

3.1 Especificação de Controle na Linguagem Xchart

Segundo o Dicionário Aurélio, Editora Nova Fronteira, 1988, *controle* é a

> fiscalização exercida sobre as atividades de pessoas, órgãos, departamentos, ou sobre produtos, etc., para que tais atividades, ou produtos, não se desviem das normas preestabelecidas.

Comportamento é a

> maneira de se comportar, procedimento, conduta; conjunto de atitudes e reações do indivíduo em face do meio social.

Controle e comportamento são termos comumente empregados, nesse texto, para caracterizar a reação de um software a estímulos externos e internos. Tal software pode, por exemplo, gerenciar a operação de máquinas de acordo com dados colhidos por sensores e decidir que o sistema de ventilação deve ser ligado em conseqüência da temperatura atingida. Esse tipo de reação é comum em sistemas reativos [40], que consistem de sistemas de comportamento complexo onde a manutenção de relações do tipo causa/efeito, dependentes de um contexto, predominam sobre as demais funções.

O controle/comportamento desses sistemas pode ser capturado através de estruturas de controle convencionais presentes em linguagens como C, Pascal e, conseqüentemente, por linguagens de máquina. Contudo, parece existir um consenso, na literatura, acerca da inadequação das estruturas oferecidas por tais linguagens para capturar o controle de sistemas reativos. Isto se torna notório através da existência de linguagens cujo propósito principal é capturar esse controle. A linguagem Statechart é uma das linguagens proeminentes empregadas com essa finalidade. A linguagem Xchart é uma nova proposta com objetivos similares. Da mesma forma que a linguagem Statechart estende os DTEs, a linguagem Xchart altera a semântica de alguns recursos "herdados" da linguagem Statechart e acrescenta outros com o intuito de melhor contemplar um subconjunto da categoria de sistemas reativos: gerenciadores de diálogo de interfaces homem-computador [85].

O emprego da linguagem Xchart pressupõe um isolamento lógico do controle de um sistema. Se tal isolamento é difícil de ser obtido, então o controle do sistema não é adequadamente descrito na linguagem Xchart. A linguagem Xchart não sugere uma organização em um único módulo, que pode ser indesejável, de todo o controle de um sistema e muito menos exige que uma plataforma específica seja utilizada para viabilizar o seu emprego.

Sistema é todo e qualquer artefato cujo controle será descrito na linguagem Xchart. Entretanto, *software* é o elemento que melhor ilustra o tipo de artefato para o qual a linguagem Xchart foi desenvolvida (inclusive um ambiente de apoio). Todo sistema contém

um Subsistema Reativo, que é responsável pela produção do controle descrito na linguagem Xchart. A Figura 3-1 mostra essa organização lógica. Os "demais subsistemas" são formados por clientes e portas. *Clientes* são os principais fornecedores de "entradas" para instâncias de diagramas Xchart, enquanto *portas* acumulam as "saídas" produzidas por instâncias e consumidas por clientes (instância é definida a seguir). As funções dos clientes estão além dos propósitos da linguagem Xchart e devem ser desenvolvidas (especificadas, por exemplo) com o apoio de ferramentas pertinentes. Descrições detalhadas desses termos são fornecidos no Capítulo 5.

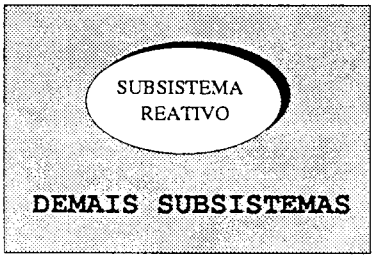


Figura 3-1: Modelo lógico de organização de um sistema

O controle ou subsistema reativo de um sistema, descrito na linguagem Xchart, é composto por um conjunto de diagramas que servem de base para a formação de instâncias. Um *diagrama definido na linguagem Xchart*, ou simplesmente um *Xchart*, é a menor unidade utilizada para representar controle. Um Xchart é composto de vários elementos: estados, eventos, regras, transições e outros.

A partir de cada Xchart pode ser criado um número arbitrário de instâncias, cada uma com identidade própria, distinta das demais. Uma instância, um Xchart em execução, é capaz de reproduzir o controle descrito no Xchart a partir do qual foi criada. Um Xchart é uma entidade estática, enquanto uma instância é a animação de um Xchart. A Figura 3-2 ilustra a relação entre Xcharts e instâncias.

Um sistema desenvolvido com o apoio da linguagem Xchart, visto na Figura 3-1, é refinado na Figura 3-3. A função do subsistema reativo é realizada por um conjunto de instâncias. Uma instância pode se comunicar com o *ambiente externo*, ou seja, com outras instâncias em execução no subsistema reativo ou com os demais subsistemas do sistema através de clientes e portas. Clientes geram *estímulos externos*. São estímulos porque desencadeiam uma reação na instância que os recebe. São externos porque são gerados além das "fronteiras" de uma instância.

O controle reproduzido pelo subsistema reativo torna-se visível aos demais subsistemas através de *portas*. Uma instância reage a um estímulo externo e deposita a sua reação (controle) em uma porta. Os clientes "consomem" as reações depositadas em portas. A comunicação entre instâncias, ou seja, no interior do subsistema reativo, não é realizada

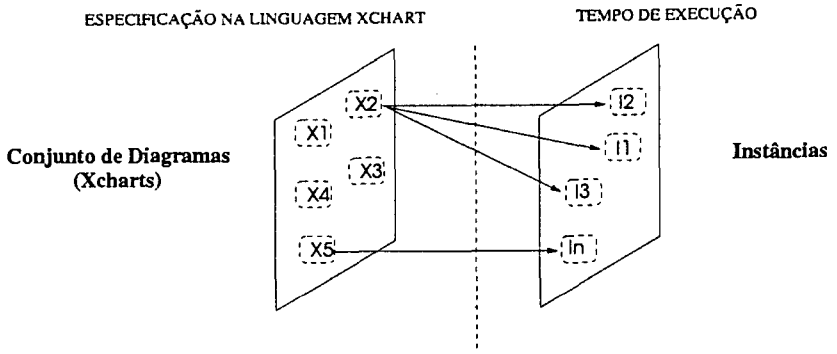


Figura 3-2: Relação entre Xcharts e instâncias em tempo de execução. Para cada diagrama em Xchart podem existir uma ou mais instâncias em execução pertinentes.

através de portas, mas através de estímulos externos trocados entre elas.

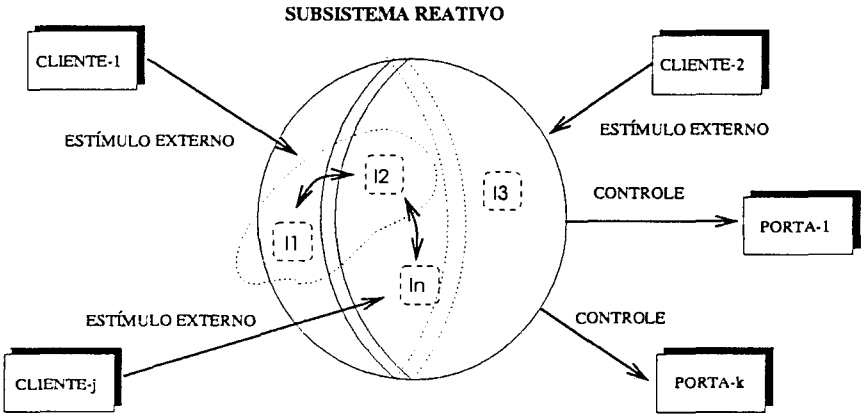


Figura 3-3: Modelo lógico refinado da organização de um sistema em Xchart. Clientes sinalizam a ocorrência de estímulos externos para um subconjunto das instâncias do subsistema reativo, que deposita o controle produzido em portas.

A comunicação realizada entre uma instância e o ambiente externo é assíncrona e não assume a existência de um único relógio compartilhado entre eles. Em conseqüência, um estímulo externo sinalizado para todas as instâncias de um subsistema reativo não será, necessariamente, tratado no mesmo instante por todas as instâncias ou na ordem cronológica em que os estímulos são gerados.

3.2 Visão Geral da Linguagem Xchart

O primeiro exemplo de um Xchart, mostrado na Figura 3-5, pág. 48, tem o intuito de fornecer uma visão geral dos principais recursos da linguagem e introduzir algumas

noções que serão melhor explicadas no restante deste texto. As noções estão organizadas nos seguintes tópicos:

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 uma entidade estática a partir do qual instâncias podem ser criadas e animadas. Uma instância é referenciada através do seu identificador único. 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 conceitualmente uma situação (Seção 3.3.1). Todos os recursos da linguagem Xchart estão necessariamente associados a um estado. Um estado pode ser refinado em um conjunto de subestados. Essa relação estabelece uma hierarquia entre estados (Seção 3.3.2).

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 (Seção 3.3.10). A Figura 3-5, na pág. 48, contém várias ações: **raise(f)**, que gera o evento primitivo **f**; $\mathbf{x := x + 1}$, que acrescenta 1 (**1**) ao valor de **x**; **raise(g)**, que gera o evento primitivo **g**; **sync(a)**, que dispara a execução síncrona da atividade **a**; **raise(h)**, que gera o evento primitivo **h** e $\mathbf{x := -x}$, que altera o sinal do valor armazenado na variável **x**. O subsistema reativo é responsável pelo controle de um sistema, enquanto as demais funcionalidades de um sistema são fornecidas por *atividades* (Seção 3.3.12). A especificação e o desenvolvimento de atividades não são contemplados pela linguagem Xchart. O objeto passível de ser descrito nessa 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* (Seção 3.3.19). 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.

EVENTOS PRIMITIVOS, EVENTOS, CONDIÇÕES E GATILHOS

Eventos primitivos são instâncias de tipos de eventos primitivos, que podem ser organizados na forma de uma hierarquia (Seção 3.3.5). Evento é uma construção mais elaborada que compreende, inclusive, os eventos primitivos (Seção 3.3.7). Por exemplo,

quando uma atividade **A** é iniciada, o evento **started(A)** é automaticamente gerado, quando uma variável **v** muda de valor o evento **ch(v)** também é gerado. **started(A)** e **ch(v)** são alguns exemplos de eventos, enquanto evento primitivo designa exclusivamente uma instância de um tipo de evento primitivo. Um evento também pode ser gerado por um relógio.

Condições são expressões lógicas. Em um instante qualquer da execução de uma instância, uma condição assume ou o valor verdadeiro ou o valor falso. A Figura 3-5 na pág. 48 contém uma única condição não nula: $\mathbf{x > 0}$. *Gatilho* é a combinação de um evento e uma condição (Seção 3.3.9). Na parte inferior do Xchart na Figura 3-5, o evento **e and f and g and h** é combinado com a condição $\mathbf{x > 0}$ formando um gatilho. O gatilho é habilitado apenas se o evento ocorre e a condição é verdadeira. Gatilho habilitado é uma condição necessária para que uma regra ou transição possa ser executada.

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 (Seção 3.3.13) e transições (Seção 3.3.14) são mecanismos por excelência para capturar relacionamentos do tipo causa-efeito. 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.

CONCORRÊNCIA

Quando mais de uma operação pode estar em execução em um mesmo instante de tempo ou quando conceitualmente é apropriado tratá-las concorrentemente pode-se utilizar estados concorrentes para capturar o controle concorrente dessas operações. Isto é, um conjunto de estados pode estar ativo em um dado instante de tempo, cada um deles podendo reagir de forma distinta a uma mesma entrada ou cooperar com os demais.

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. Nesse 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 (Seção 3.3.17). Quando tal conjunto for identificado, apenas a transição que possuir a maior prioridade será executada.

ESTÍMULO EXTERNO

Todas as reações de uma instância são desencadeadas por estímulos produzidos externamente a essa instância. Um *estímulo externo* é um conjunto de eventos ocorridos além das fronteiras da instância que o recebe. Um estímulo externo pode ser sinalizado para um subconjunto das instâncias do subsistema reativo. Cada instância recebe a sua própria seqüência de estímulos externos ao longo de sua existência. Para toda reação há, direta ou indiretamente, um estímulo externo causador. A Figura 3-4 sugere que não há controle (saída) se não existe estímulo externo (entrada). Após tratado, um estímulo é descartado ou consumido. Cada estímulo externo provoca uma reação, possivelmente nula. Uma reação é a execução de um passo.

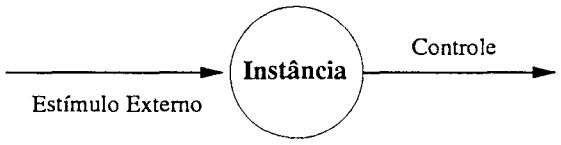


Figura 3-4: Visão funcional de uma instância

PASSO E MICRO-PASSO

Quando um estímulo externo é recebido por uma instância, ele é tratado de acordo com a configuração da instância. Esse tratamento pode identificar um conjunto de regras e/ou transições cujos gatilhos estão habilitados. Em algumas situações, nem todos os elementos desse conjunto podem ser executados. Transições excludentes (Seção 3.3.17), por exemplo, podem identificar um subconjunto de regras a ser executado denominado de *micro-passo*. Executar um micro-passo significa executar cada um dos elementos desse subconjunto. Uma seqüência de micro-passos forma um *passo*. Executar um passo significa executar cada um dos micro-passos da seqüência pertinente. Cada micro-passo dessa 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 3-5 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 dessa 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. O conjunto obtido não é um "novo" estímulo externo — é o mesmo estímulo externo, que provocou o início do passo, acrescido de um evento primitivo. Um segundo micro-passo é identificado conforme a configuração obtida pela execução da transição comentada acima. O segundo micro-passo é composto apenas da transição do estado **C** para o **D**. A execução do segundo micro-passo envolve a execução das ações **raise(g)** e **sync(a)**. A primeira delas gera um evento primitivo, que é tratado de forma análoga àquele gerado anteriormente. A outra ação dispara a execução da atividade **a**. A instância é "congelada" (Seção 3.6) enquanto a execução não é finalizada. O evento primitivo **g** 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, após o fim de uma nova execução da atividade **a**, 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 três primeiros micro-passos. A condição $\mathbf{x} > \mathbf{0}$ 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 $\mathbf{x} := -\mathbf{x}$. Após a execução dessa 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 dessas 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.

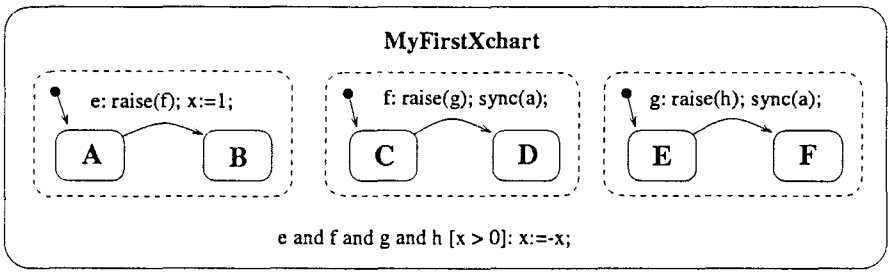


Figura 3-5: Primeiro Xchart

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 transições mutuamente excludentes fazem parte desse conjunto, então tais casos são eliminados (Seção 3.3.17). 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 ao subsistema reativo apenas quando ele termina. Nesse instante a instância passa a tratar o próximo estímulo externo, ou aguarda por um, caso sua fila esteja vazia. Durante esse processo de reação de uma instância, os demais subsistemas 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. Detalhes desse processo são apresentados na Seção 3.9.

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.

FILA DE ESTÍMULOS EXTERNOS E DURAÇÃO DE UM ESTÍMULO EXTERNO

Se a execução de uma instância é suficientemente rápida, então a reação a um estímulo externo termina antes da ocorrência do próximo estímulo externo. Algumas reações, contudo, podem durar por períodos consideráveis da mesma forma que um conjunto numeroso de estímulos externos pode ser enviado a uma instância em um intervalo relativamente curto de tempo. Nesses casos, onde a vazão de estímulos externos é maior que a taxa de tratamento da instância, eles são armazenados em uma fila—eles não são fundidos. À medida que estímulos são "consumidos", aqueles que aguardam vão sendo tratados conforme a ordem de chegada (o primeiro que chega é o primeiro a ser tratado). Cada instância possui a sua própria fila.

Quando um estímulo externo é retirado da fila para ser tratado, a existência dele ainda se prolonga até o fim do passo que ele provoca. Quando o passo termina, o estímulo é "consumido" — já foi removido da fila e não pode provocar mais nenhuma reação. Nem todos os estímulos externos na fila de uma instância são necessariamente tratados. Quando um estímulo provoca o fim da execução de uma instância, os estímulos que

aguardam na fila da instância são descartados. A fila de uma instância se comporta conforme o *status* da instância.

STATUS DE INSTÂNCIA

Desde o instante em que é criada até o fim da execução de uma instância, ela alterna entre três status ditados por ações executadas nesse intervalo. Após a criação, uma instância é dita *ativa*. Ao tratar um estímulo externo que provoca a execução de uma ação **call**, por exemplo, a instância se torna *inoperante* até que essa ação seja finalizada, quando ela se torna ativa novamente. Quando a instância é finalizada, então ela se torna *inativa*. Uma instância inativa não muda mais o seu status. Por outro lado, o status de uma instância pode alternar entre inoperante e ativo. A Seção 3.6 fornece detalhes.

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. Em conseqüência, um estímulo externo desencadeia um processo finito, que identifica um número finito de regras e/ou transições a serem executadas em um determinado passo. Um estado não pode ser ativado e posteriormente desativado em um mesmo passo, o que também corrobora a natureza finita de um passo. Essas restrições impedem, naturalmente, a existência de ciclos onde estados seriam ativados e desativados infinitamente, ou ainda a execução infinita de regras em um estado sempre ativo.

EXCEÇÕES

A linguagem Xchart não possui recursos explícitos para tratamento de exceções. Por exemplo, não é discutido nenhum mecanismo para detectar e tratar situações de overflow/underflow em variáveis de controle. Do ponto de vista do responsável por uma especificação tudo funciona conforme a definição dos recursos da linguagem Xchart, sem casos excepcionais. Naturalmente, o emprego de software e hardware convencionais é incompatível com tal abstração. O sistema de execução da linguagem Xchart, máquina abstrata que executa Xcharts, trata várias situações extraordinárias. Detalhes são fornecidos no Capítulo 5.

TEMPO

Do ponto de vista de uma instância, o tempo real contínuo é dividido em intervalos definidos por instantes de tempo (Seção 3.9). Por exemplo, a Figura 3-6 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 desse instante, a instância permanece com a configuração obtida até a ocorrência de um novo estímulo externo.

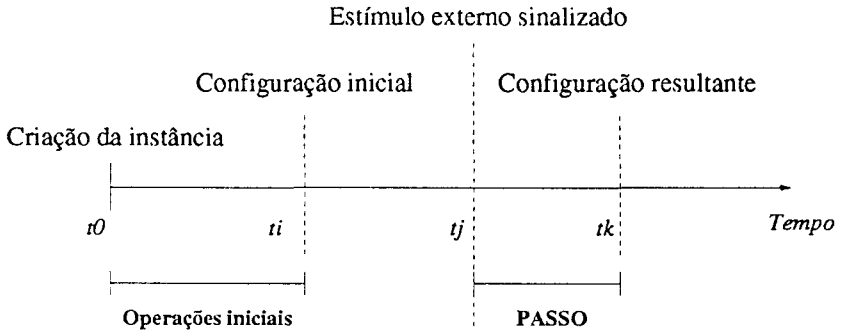


Figura 3-6: Instantes típicos da execução de uma instância

3.3 Recursos Básicos

As subseções seguintes definem as construções oferecidas por Xchart para descrever controle. Essa seção não tem o intuito de justificar a existência dessas construções. O Capítulo 6 fornece indícios positivos do emprego de Xchart e, em particular, de suas construções.

3.3.1 Estado

Um *estado* captura conceitualmente uma situação, contexto ou modo. Por exemplo, um editor de texto pode estar no modo inserir ou sobrescrever; uma opção de menu pode estar disponível ou não em determinado instante; uma exceção pode conduzir um sistema de um estado de operação normal a um estado correspondente a uma circunstância especial e assim por diante. Cada uma dessas situações pode ser capturada por um estado. Um estado também pode assumir outra conotação adequada a um domínio de aplicação. Todo

estado possui um atributo: em um determinado instante, ele está ou não *ativo*. Um estado é representado por um retângulo com cantos arredondados em cujo interior reside seu identificador.

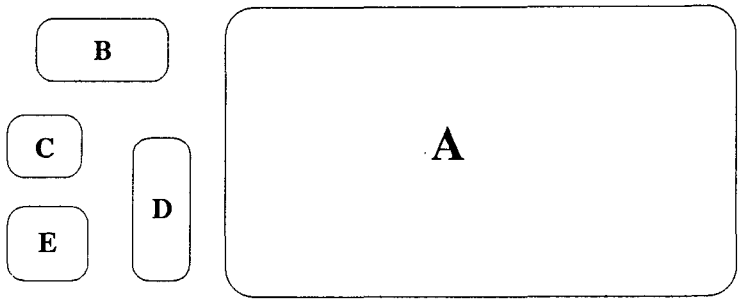


Figura 3-7: Exemplos de representação de estados

A Figura 3-7 ilustra vários estados identificados por **A**, **B**, **C**, **D** e **E**. Um estado não possui necessariamente um identificador. Contudo, só é possível fazer referência a um estado, em algumas funções, comentadas adiante, se ele possuir um identificador.

Estado fictício é a denominação de um círculo (•) utilizado como origem ou destino de transições (Seção 3.3.14). Conforme o nome sugere, não se trata de um estado "convencional". Estados podem estar dispostos em uma hierarquia.

3.3.2 Hierarquia de Estados e Diagrama na Linguagem Xchart

Um estado pode ser decomposto em um ou mais estados, que são denominados *subestados*. Um subestado também pode ser decomposto em outros subestados e assim sucessivamente. Essa decomposição forma uma *hierarquia*, que é um mecanismo útil para controlar a complexidade de uma especificação, organizando-a em níveis distintos de abstração. Por exemplo, a Figura 3-8 organiza os estados da Figura 3-7 em uma hierarquia. Os estados **B**, **C**, **D** e **E** são os subestados imediatos do estado **A**. O estado **A** é o estado mais externo da hierarquia, denominado *raiz*.

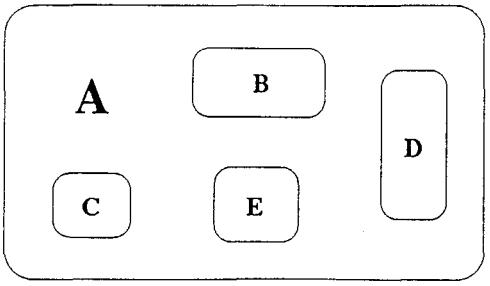


Figura 3-8: Hierarquia de estados

Subestados de um estado s são ditos *descendentes* de s, que é denominado *ancestral* destes subestados (descendentes). Esta relação é transitiva, ou seja, para uma hierarquia qualquer, a raiz é o ancestral de todos os demais estados, que são seus descendentes. Para cada estado, exceto a raiz, há um único e bem definido ancestral direto. Exceto quando dito o contrário, subestados de um estado referem-se apenas aqueles diretamente descendentes sem o emprego dessa relação transitiva.

Um diagrama na linguagem Xchart compreende, necessariamente, uma hierarquia de estados. O identificador da raiz de uma hierarquia também identifica o diagrama na linguagem Xchart ao qual pertence. Em conseqüência, a Figura 3-8 também representa um diagrama na linguagem Xchart identificado por **A**.

Alguns Xcharts podem necessitar dos recursos ∇ e Δ para controlar o volume de detalhes em um determinado nível de uma hierarquia. No Xchart **S** da Figura 3-9, por exemplo, o estado **Z** é descrito apenas parcialmente, o que é indicado pelo símbolo ∇. Nesse caso, **Z** não é descrito na hierarquia cuja raiz é o estado **S**. A descrição do estado **Z** não é fornecida nessa hierarquia. **Z** é descrito no lado direito dessa Figura. A hierarquia cuja raiz é o estado **Z** não constitui, contudo, um Xchart. O símbolo Δ informa que **Z** é apenas o refinamento de um estado que apenas é referenciado em uma outra hierarquia.

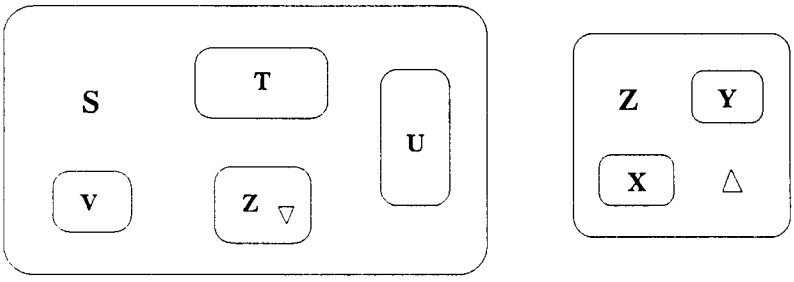


Figura 3-9: Descrição disjunta de uma hierarquia

3.3.3 Tipos de Refinamento de um Estado

Um estado é *básico* se e somente se ele não possui nenhum subestado. Na Figura 3-10, os estados **Basic**, **A**, **C**, **D** e **E** são básicos. O estado **B** pode ser básico, pois a sua definição não é fornecida na hierarquia mostrada. Se um estado não é básico, então o seu tipo é *exclusivo* ou *concorrente*. Em ambos os casos, o estado pertinente possui subestados cuja ativação depende do tipo desse estado.

Se um estado exclusivo está ativo, então no máximo um de seus subestados também está. Em particular, se nenhum dos subestados de um estado exclusivo ativo está ativo (Seção 3.3.14), então o estado exclusivo é dito "temporariamente básico". Diferentemente, quando um estado concorrente está ativo, todos os seus subestados necessariamente

também estão ativos.

O tipo de um estado não básico é identificado pelo contorno dos seus subestados. Se os subestados de um estado possuem contornos contínuos, então o estado é exclusivo. Se os subestados possuem contornos tracejados, então o estado é concorrente. Na Figura 3-10, o estado **Exclusive** é do tipo exclusivo, enquanto o estado **Concurrent** é do tipo concorrente. Os demais são básicos, exceto **B**, cujo tipo não pode ser identificado pela figura. Uma hierarquia de estados separada refina o estado **B**.

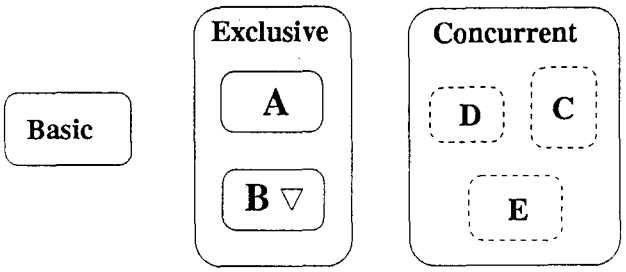


Figura 3-10: Tipos de refinamento de estado

3.3.4 Variáveis de Controle

Uma variável de controle é um repositório que armazena um valor inteiro ou lógico, conforme o seu tipo. Uma variável de controle, ou simplesmente variável, pode ser local ou global em relação a um Xchart. Cada variável local a um Xchart possui o seu próprio repositório para cada instância desse Xchart. Por outro lado, há um único repositório, compartilhado entre todas as instâncias de um subsistema reativo, para cada variável global. O valor inicial de uma variável pode ser fornecido na definição da variável. Se não é fornecido um valor, então é assumido o valor **0** para variáveis inteiras e o valor verdadeiro (**TRUE**) para variáveis lógicas.

A definição de variáveis de controle locais é feita no interior da raiz do Xchart em questão. A Figura 3-11, por exemplo, descreve o Xchart **Root** contendo a definição de suas variáveis inteiras (**a**, **b**, **c**) e lógicas (**d**, **e**, **f**). Cada instância desse Xchart terá o seu próprio repositório para cada uma dessas variáveis inteiras e lógicas. Esse repositório não é compartilhado entre instâncias. A mudança de valor de uma dessas variáveis em uma instância não se propaga ou afeta a mesma variável em outras instâncias derivadas de um mesmo Xchart. Os valores iniciais das variáveis inteiras, nesse exemplo, são: **10** (**a**), **0** (**b**) e **21** (**c**). Os valores iniciais das variáveis lógicas são falso (**FALSE**) para a variável **d** e **TRUE** para as demais (**e** e **f**).

A Figura 3-12 apresenta parte de uma especificação em Xchart contendo os Xcharts **A** e **B**, além da definição das variáveis de controle globais **a** e **b** (inteiras). Quando a

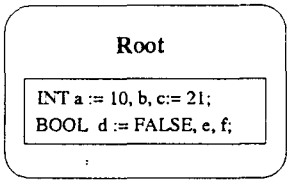


Figura 3-11: Definição de variáveis de controle locais

primeira instância desta especificação for criada, os valores iniciais de **a** e **b** serão iguais a **0**. Posteriormente, todas as instâncias de **A** e **B** consultarão e atualizarão o mesmo conteúdo de cada uma dessas variáveis. Se uma instância de **A**, por exemplo, altera o valor de **a**, então o novo valor será aquele obtido por toda e qualquer instância que consulte o valor de **a**.

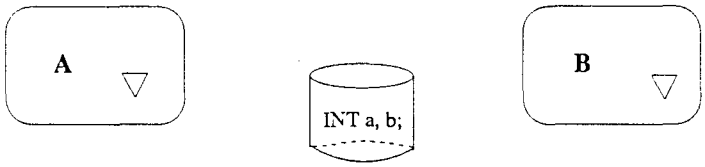


Figura 3-12: Definição de variáveis de controle globais

3.3.5 Eventos Primitivos, Tipos e Hierarquia

Tipo de evento primitivo, instância de um tipo de evento primitivo e evento são três elementos distintos. Por simplicidade, evento primitivo é sinônimo de instância de tipo de evento primitivo. Evento é definido na Seção 3.3.7. Tipo de evento primitivo, evento primitivo e condições nas quais um tipo de evento primitivo *ocorre* são descritos nessa seção.

Uma instância de um tipo de evento primitivo, ou simplesmente evento primitivo, é a abstração empregada para modelar um acontecimento. Abrir a porta de uma geladeira e ligar um automóvel são exemplos de situações que podem ser modeladas como eventos primitivos. Ambos poderiam ser instâncias do tipo de evento primitivo **Manusear**, por exemplo. Tipos de eventos primitivos podem estar organizados em uma hierarquia. A Figura 3-14 ilustra uma hierarquia de tipos de eventos primitivos.

Eventos primitivos são instâncias de tipos de eventos primitivos, que são criadas pelo ambiente externo ou por instâncias de Xcharts. Em contrapartida, tipos de eventos primitivos fazem parte de um conjunto de informações invariável durante a execução de uma instância assim como a hierarquia de estados de um diagrama.

Conceitualmente um evento primitivo representa um acontecimento instantâneo, não persistente, ocorrido em um instante particular. Contudo, a ocorrência de um evento

primitivo é registrada através de um estímulo externo, que persiste enquanto não tratado pela instância para a qual é sinalizado. A Figura 3-13 exibe instantes de tempo característicos do ciclo de vida de um evento primitivo em relação a uma particular instância. Nessa figura um evento primitivo é gerado no instante **t0**. A geração desse evento primitivo, contudo, não é instantaneamente sinalizada para a instância pertinente. Primeiro ele é acumulado em um estímulo externo, possivelmente com outros eventos primitivos. Após a confecção do estímulo externo, ele pode ser sinalizado no instante **t1**, posterior a **t0**. As reações provocadas pelo estímulo criado não são instantaneamente obtidas e executadas (Seção 3.9). Apenas no instante **t2** é iniciado o tratamento desse estímulo. O tratamento é realizado em um período não nulo de tempo — esse período pode perdurar. O instante **t3** registra o fim do estímulo criado e, em decorrência, o fim do evento primitivo gerado no instante **t0**. O fim de um estímulo externo também coincide com o fim das conseqüências (efeitos) por ele causadas em uma dada instância.

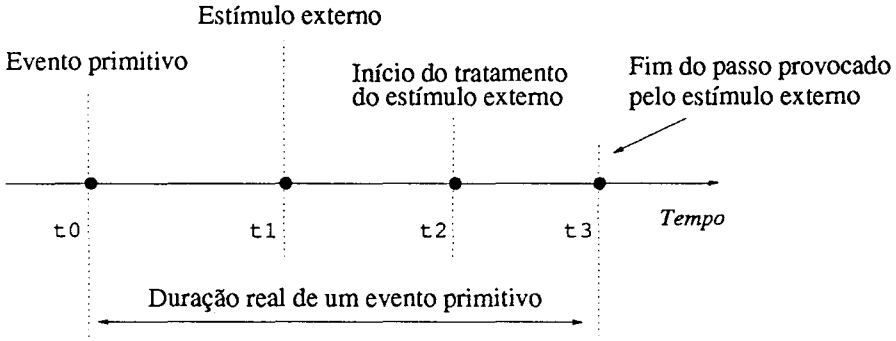


Figura 3-13: Instantes típicos associados a evento primitivo, evento e estímulo externo

Vários fatores influenciam a extensão do intervalo entre **t0** e **t3** (Figura 3-13). Em particular, o intervalo entre **t2** e **t3** é descrito em detalhes na Seção 3.9. Essa figura ainda ilustra a duração de um estímulo externo e, em conseqüência, a duração de eventos. Independentemente do evento **tr(x > 0)**, por exemplo, ter sido gerado durante a execução de um passo ou ser fornecido como parte de um estímulo externo, o fim do passo também marca o fim da existência dos eventos pertinentes. Ou seja, o evento exemplificado **tr(x > 0)** persiste apenas até o fim do passo assim como todos os demais.

Para estabelecer a reação de uma instância é imprescindível identificar se um determinado tipo de evento primitivo ocorre ou não. Em geral, o gatilho de regras/transições está condicionado à ocorrência de determinados tipos de eventos primitivos. Por exemplo, o gatilho **f and g** exige a ocorrência dos tipos de evento primitivo **f** e **g** para ser habilitado. Gatilhos são vistos em detalhes na Seção 3.3.9. Em conseqüência, é preciso definir as condições nas quais um determinado tipo de evento primitivo *ocorre*.

Um tipo de evento primitivo **tipo** ocorre se e somente se existe um evento primitivo

evp gerado a partir do tipo de evento primitivo **tevp** tal que:

(i) **tevp** é o próprio **tipo** ou

(ii) **tipo** é ancestral de **tevp** na hierarquia de tipos de evento primitivo.

Se nenhuma das condições acima é satisfeita, então diz-se que o tipo **tevp** não ocorre dada a ocorrência do evento primitivo **evp**. Durante a execução de uma instância, quando um estímulo externo é tratado, todos os eventos primitivos do estímulo externo são consultados no processo de identificação dos tipos de eventos primitivos que ocorrem na presença desse conjunto de eventos primitivos. Se nenhum evento primitivo desse conjunto provoca a ocorrência de um dado tipo, então o estímulo externo não provoca a ocorrência do tipo considerado. Por exemplo, para a hierarquia apresentada na Figura 3-14, se um estímulo externo contendo apenas um evento primitivo gerado a partir do tipo de evento primitivo **b** é gerado, então os tipos de eventos primitivos **b**, **letter**, **keyboard** e **event** ocorrem. Por outro lado, os tipos **a**, **number** e **mouse** não ocorrem. Durante a execução de um passo eventos primitivos podem ser gerados e, nesse caso, um tipo de evento primitivo ocorre se uma das condições acima é satisfeita para o evento primitivo que desencadeou a execução do passo ou para aqueles eventos primitivos gerados em micro-passos anteriores do passo em questão.

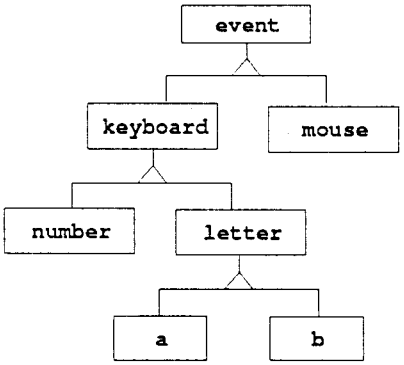


Figura 3-14: Hierarquia de tipos de eventos primitivos

3.3.6 Dados Acoplados a Eventos Primitivos

Um evento primitivo pode carregar dados consigo. Em conseqüência, ele pode ser visto como um veículo de transporte de informações entre instâncias ou entre instâncias e o ambiente externo. Qualquer que seja o caso, instâncias de Xcharts não manipulam os

dados associados a um evento primitivo. Esses dados são utilizados exclusivamente por atividades.

Quando um evento primitivo é criado, um construtor do tipo de evento primitivo pertinente é executado. Esse construtor fornece os dados que acompanharão o evento primitivo criado onde quer que ele seja visível. Quando uma instância de Xchart trata um estímulo externo, todos os dados associados aos eventos primitivos pertencentes ao estímulo externo tornam-se disponíveis. Dessa forma, o resultado funcional da execução da reação de uma instância inclui os dados associados aos eventos primitivos de um estímulo externo considerado.

3.3.7 Evento

Evento primitivo é uma abstração que modela um acontecimento. Evento ("evento não-primitivo") é uma construção mais sofisticada, elaborada a partir de uma possível combinação de vários elementos. Ao tratar um estímulo externo, uma das principais tarefas da reação ao estímulo é identificar gatilhos habilitados, cujos eventos necessariamente ocorrem. Em grande parte dos casos, um evento *ocorre* se ele faz parte do estímulo externo em questão. Esta Seção descreve as demais condições que estabelecem se um evento ocorre ou não. O ciclo de vida de um evento (duração) é análogo ao de um evento primitivo comentado na Seção 3.3.5. Os elementos que podem compor um evento e as condições nas quais se diz que eles ocorrem são definidos abaixo:

EVENTO NULO

O evento nulo é aquele que ocorre na presença de todo e qualquer estímulo externo.

TIPO DE EVENTO PRIMITIVO

Um tipo de evento primitivo também representa um evento. Ao contrário, um evento primitivo não representa um evento. Ou seja, um evento primitivo não pode ser parte de um gatilho, mas o seu tipo, a partir do qual foi gerado, é um legítimo representante de um evento. A ocorrência ou não de um tipo de evento primitivo é definida na Seção 3.3.5.

EVENTOS ASSOCIADOS A CONDIÇÕES

tr(c) e **fs(c)** são eventos que ocorrem sempre que o valor da condição **c** é alterado, respectivamente, de falso para verdadeiro e de verdadeiro para falso. O valor da condição pode, por exemplo, ser alterado pela execução de uma ação em um micro-passo. No micro-passo seguinte o evento torna-se visível. Exemplos:

- **tr(x + y = z)** ocorre sempre que esta comparação se torna verdadeira. Para isto é necessário que: (i) o valor de pelo menos uma das variáveis envolvidas (**x**, **y** ou

z) seja alterado; (ii) $\mathbf{x + y = z}$ seja falsa antes desta alteração e (iii) sua avaliação após a alteração deve produzir o resultado verdadeiro.

- $\mathbf{fs(x < 4)}$ é um evento que ocorre sempre que **x** passa de um valor inferior a 4 para um valor superior ou igual a 4. Neste instante, a condição $\mathbf{x < 4}$, que era verdadeira, torna-se falsa, provocando a ocorrência deste evento.
- Conforme a definição, $\mathbf{tr(1 = 1)}$ e $\mathbf{fs(1 > 2)}$ nunca ocorrem.

EVENTO ASSOCIADO A EXPRESSÕES

Se **v** é uma expressão numérica ou lógica, então o evento $\mathbf{ch(v)}$ ocorre sempre que o valor da expressão **v** é alterado. Por exemplo, se o valor de **v** é **k** e a execução de alguma ação altera este valor para um valor diferente de **k**, então $\mathbf{ch(v)}$ é gerado. Exemplos:

- $\mathbf{ch(x + y)}$ é gerado sempre que o valor desta soma é alterado. Naturalmente, pelo menos o valor de uma das variáveis deve ser alterado. Ainda pode ocorrer uma alteração em ambos os valores sem que esse evento seja gerado, ou seja, a soma permanece constante.
- $\mathbf{ch((3 + y) <= 2)}$ é o evento gerado sempre que esta condição é falsa e se torna verdadeira ou vice-versa.
- Conforme a definição, $\mathbf{ch(x - x)}$ e $\mathbf{ch(1 > 0)}$ são eventos que jamais ocorrem.

EVENTOS GERADOS POR ATIVAÇÃO/DESATIVAÇÃO DE ESTADO

Dado um estado **s** de uma instância **i** de um Xchart, os eventos $\mathbf{ex(s, i)}$ e $\mathbf{en(s, i)}$ são gerados sempre que **s** é desativado e ativado, respectivamente. Por simplicidade, $\mathbf{en(s)}$ e $\mathbf{ex(s)}$ referem-se ao estado **s** da instância na qual a ocorrência desses eventos é verificada. O identificador **i**, conforme comentado anteriormente, identifica uma única instância.

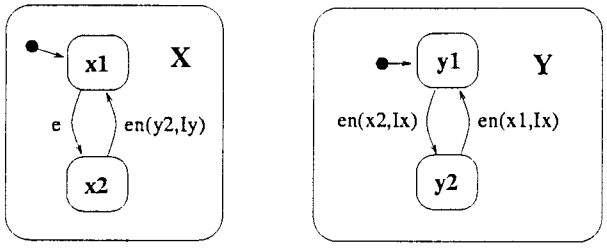


Figura 3-15: Eventos gerados por ativação/desativação de estado

Na Figura 3-15 são exibidos dois Xcharts: **X** e **Y**. Seja **Ix** uma instância de **X** e **Iy** uma instância de **Y** com os estados básicos **x1** e **y1** ativos, respectivamente, em

um determinado instante da execução dessas instâncias. Nessa configuração, se um estímulo externo contendo o evento **e** é tratado pela instância **Ix**, então a transição de **x1** para **x2** é executada. Essa transição desativa o estado **x1** e ativa o estado **x2**. Em conseqüência, os eventos **ex(x1, Ix)** e **en(x2, Ix)** são gerados. Eles formam um estímulo externo que provoca, na instância **Iy**, a transição do estado **y1** para **y2**. A desativação de **y1** e a ativação de **y2** provocam a geração dos eventos **ex(y1, Iy)** e **en(y2, Iy)**. Analogamente ao caso anterior, eles formam um estímulo externo que provoca, na instância **Ix**, a transição de **x2** para **x1** e assim por diante. As instâncias se "estabilizam" quando os estados básicos **x1** e **y1** tornarem-se ativos.

EVENTOS ASSOCIADOS A ATIVIDADES

Se **a** é uma atividade, então os seguintes eventos ocorrem nas condições apresentadas:

- **started(a)**
  Esse evento ocorre quando a atividade **a** é iniciada. Esse evento é necessariamente provocado por um elemento de ação **start(a, p)** ou **sync(a, p)** para uma porta **p** qualquer. Ações são apresentadas na Seção 3.3.10.
- **stopped(a)**
  Esse evento ocorre quando a atividade **a** é finalizada pela execução do elemento de ação **stop(a, p)** ou ainda pelo término normal da atividade para uma porta **p** qualquer.

À semelhança das condições **active** e **hanging** (pág. 63), as formas **started(a, p)** e **stopped(a, p)** não existem. Os eventos correspondentes são gerados independentemente da porta em que a atividade em questão é iniciada ou finalizada.

EVENTOS TEMPORAIS

A cada estado está associado um cronômetro. Para cada estado ativado, durante um passo, o cronômetro pertinente inicia sua contagem imediatamente após o fim do passo. O tempo contado por esse cronômetro é utilizado para gerar os eventos **after** e **every**, associados a intervalos de tempo em que um estado permanece ativo. Esses eventos fazem parte, necessariamente, de regras associadas aos estados ativados ou de transições que partem desses estados. Quando o estado é desativado, o cronômetro é zerado e sua operação é interrompida. Há ainda o evento temporal **at**, que é gerado por um relógio real.

Eventos temporais podem ser gerados apenas pelo relógio do sistema ou cronômetros associados a estados. Eles não são tratados imediatamente no instante em que são gerados. Quando gerado, um evento temporal é encapsulado em um estímulo externo

e sinalizado para a instância pertinente, ou seja, pode perdurar em uma fila antes de ser tratado.

- **after**
  Na Figura 3-16 (a), após transcorridos 2 segundos de permanência no estado **A**, o cronômetro desse estado gera o evento **after(2s)**, que forma um estímulo externo enviado à instância em questão. Ao tratar tal estímulo, a transição rotulada com **after(2s)** é executada, pois o evento **after(2s)** ocorre e o estado **A** está ativo. Dois segundos após a ativação do estado **B**, a transição para o estado **A** é executada em decorrência de reação similar.

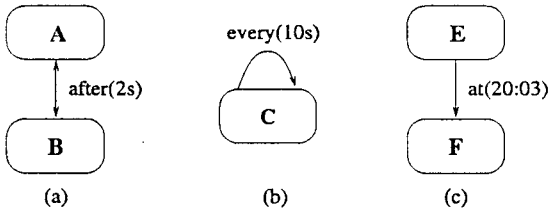


Figura 3-16: Eventos temporais

- **every**
  Na Figura 3-16 (b), após a ativação do estado **C**, um estímulo externo contendo o evento **every(10s)** será gerado a cada 10 segundos, enquanto o estado **C** permanecer ativo.
- **at**
  O evento **at** ocorre quando um instante de tempo específico é atingido, conforme o relógio real do contexto de execução da instância em questão. Na Figura 3-16 (c), o evento **at(20 : 03)** forma um estímulo externo que é sinalizado para a instância pertinente no instante especificado. Ao tratar tal estímulo, a transição de **E** para **F** é executada, caso o estado **E** esteja ativo.

EVENTOS COMPOSTOS

Eventos podem ser combinados logicamente formando outros eventos. Os operadores lógicos empregados são: **or**, **and** e **not**. Em particular, **not** permite negar a ocorrência de um evento. Exemplos:

- **not(e)** é o evento que ocorre se e somente se qualquer outro evento diferente do evento cujo tipo é **e** ocorrer. Se nenhum evento ocorre, então **not(e)** também não ocorre. O evento **e** não pode ser o evento nulo.
- **(e and not(f)) or (not(e) and f)** é o evento que ocorre se e somente se ou o evento **e** ou o evento **f** ocorre, de forma exclusiva.

3.3.8 Condição

Condição é uma expressão lógica cujo valor é falso ou verdadeiro em um instante qualquer. Abaixo são exibidas as condições válidas em Xchart:

CONDIÇÃO NULA

Condição que sempre é satisfeita, isto é, seu valor é sempre verdadeiro.

CONDIÇÃO ASSOCIADA A ESTADO

A condição **in(s)** é verdadeira se e somente se o estado **s** da instância em questão está ativo. Tal condição pode estar associada a um estado de uma instância qualquer. Por exemplo, **in(s,j)** é uma condição verdadeira quando o estado **s** na instância **j** estiver ativo.

EVENTO NÃO OCORRIDO NO PASSO CORRENTE (REAÇÃO EM CADEIA)

ny(e) é uma condição verdadeira se e somente se o evento **e** não é parte do estímulo que provocou o início do passo em execução nem foi gerado até o micro-passo em que é avaliada.

Por exemplo, se uma instância do Xchart exibido na Figura 3-17 está com os estados básicos **A**, **B** e **C** ativos, então um estímulo externo contendo o evento **e** provoca a seguinte reação em cadeia: no primeiro micro-passo ocorre a transição de **A** para o **D**. Em conseqüência, o evento **g** é gerado e provoca, no segundo micro-passo, a execução da transição de **B** para **E**. Essa transição gera o evento **h**, que poderia provocar a execução da transição de **C** para **F** no terceiro micro-passo. Contudo, a condição **ny(e)** é falsa na fase de avaliação do terceiro micro-passo. A condição **ny(e)** é falsa porque o evento **e** é parte do estímulo externo e, até aquele momento ele ocorreu. Analogamente, as condições **ny(g)** e **ny(h)** também seriam avaliadas como falsas. Por outro lado, se o estímulo externo contivesse apenas o evento **g**, então uma reação em cadeia de tamanho dois seria iniciada. No primeiro micro-passo seria executada a transição de **B** para **E** e no segundo aquela de **C** para **F**, pois o evento **h** foi gerado no micro-passo anterior e a condição **ny(e)** é verdadeira.

CONDIÇÕES ASSOCIADAS A ATIVIDADES

O status de execução de uma atividade pode ser consultado através das condições **active** e **hanging**. O valor destas condições está intrinsecamente relacionado às ações que controlam atividades (Seção 3.3.10) e ao ciclo de vida de uma atividade (Seção 3.3.12).

- **active(a)**
  Essa condição é verdadeira se e somente se a atividade **a** estiver ativa (em execução). Ou seja, em algum instante do passado a ação **start(a,p)** ou **sync(a,p)**

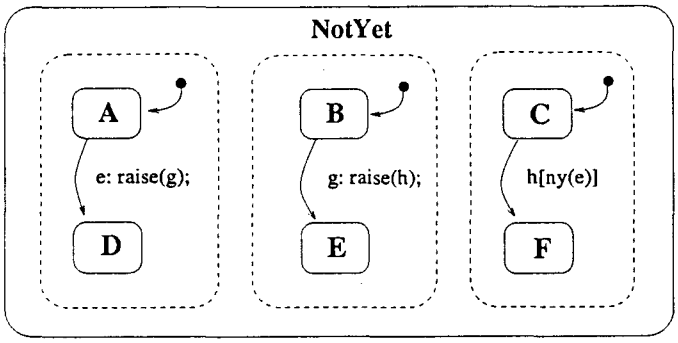


Figura 3-17: Condição **ny** em uma reação em cadeia

foi executada para uma porta **p** e, no instante em que essa condição é avaliada, a atividade **a** está em execução.

- **hanging(a)**
  Essa condição é verdadeira se e somente se a atividade **a** estiver suspensa, ou seja, em algum instante do passado a ação **start(a, p)** ou **sync(a, p)** foi executada e, posteriormente, a ação **suspend(a, p)**.

Não existem as formas **active(a, p)** ou **hanging(a, p)**, pois o status de uma atividade não depende de uma porta.

CONDIÇÕES ENVOLVENDO EXPRESSÕES NUMÉRICAS

Os operadores de igualdade (=), diferença (<>), maior (>), menor (<), maior ou igual (>=) e menor ou igual (<=) podem estabelecer uma relação entre duas expressões numéricas. A relação resultante é uma condição. Por exemplo, **2 < 3** é uma condição sempre verdadeira. Em contrapartida, **x = y** só é verdadeira quando os valores das variáveis numéricas **x** e **y** são iguais. Os demais operadores, assim como os exemplificados, têm a semântica convencional. Naturalmente, uma variável lógica pode ser utilizada onde uma condição é esperada.

CONDIÇÕES COMPOSTAS

À semelhança de eventos, condições também podem ser combinadas conforme abaixo:

- **c1 and c2** é uma condição verdadeira, se e somente se **c1** e **c2** são verdadeiras.
- **c1 or c2** é verdadeira se e somente se pelo menos uma das condições é verdadeira.
- **not(c)** é verdadeira se e somente se **c** é uma condição falsa.

3.3.9 Gatilho

Um gatilho **e**[**c**] é a combinação do evento **e** e da condição **c**. Um gatilho **e**[**c**] é dito *habilitado* se e somente se o evento **e** ocorre e o valor da condição **c** é verdadeiro. Se uma regra ou transição é executada em um dado micro-passo, então necessariamente o seu gatilho estava habilitado na fase de avaliação desse micro-passo (detalhes na Seção 3.9). Durante a execução de um micro-passo, não necessariamente os gatilhos habilitados das regras e transições do micro-passo permanecem habilitados. Por exemplo, o gatilho da regra $\mathbf{e}[\mathbf{x} = \mathbf{2}] : \mathbf{x} := \mathbf{3}$ não permanece habilitado durante a execução do micro-passo que contém essa regra, pois a execução da ação $\mathbf{x} := \mathbf{3}$ torna a condição $\mathbf{x} = \mathbf{2}$ falsa. Alguns gatilhos são ditos especiais:

GATILHO NULO
: O gatilho nulo é formado pelo evento e pela condição nulos. Ou seja, ele é habilitado sempre que qualquer evento ocorre. Um gatilho nulo pode ser simulado através de gatilhos convencionais como $[\mathbf{1} = \mathbf{1}]$, onde o evento é nulo e o valor da condição é sempre verdadeiro.

GATILHO ENTRY
: O gatilho **entry**[**c**] sempre está associado a uma regra no interior de um estado. Se tal gatilho está habilitado, então a condição **c** é verdadeira e o estado que contém o gatilho será ativado durante a execução do micro-passo em questão. O processo de execução de um micro-passo é descrito em detalhes na Seção 3.9.

GATILHO EXIT
: O gatilho **exit**[**c**] assemelha-se ao gatilho anterior. Neste caso, contudo, o estado que contém a regra deve estar sendo desativado em vez de ativado como no caso anterior.

Gatilhos são exemplificados por todo o texto. Em particular, as Seções 3.3.13 e 3.3.14 fornecem vários exemplos, como aqueles da Figura 3-27 na pág. 77.

3.3.10 Ação

A saída produzida pela execução de uma instância é obtida conforme a configuração da instância e um estímulo externo (entrada). Um estímulo externo pode habilitar um ou mais gatilhos que provocam, possivelmente, a execução de regras (Seção 3.3.13) e/ou transições (Seção 3.3.14). Para cada regra ou transição a ser executada tem-se uma ação correspondente, que deve ser executada. Se se trata de uma transição, então outras ações decorrentes da desativação/ativação de estados serão, possivelmente, executadas. *Ação* é o mecanismo empregado para gerar controle. Sem ações não há como expressar o controle

produzido por uma instância. Toda e qualquer reação de uma instância de um Xchart é expressa através de ações, que alteram o valor de variáveis de controle ou manipulam atividades. Um evento que terá repercussão em outras instâncias e a criação de uma instância de um Xchart também são exemplos de ações. Em uma relação de causa/efeito, um gatilho está para a causa assim como uma ação está para o efeito.

Uma ação pode ser composta por vários elementos de ação, listados abaixo. A execução de alguns desses elementos pode tornar uma instância inoperante (Seção 3.6). Geralmente um elemento de ação também é referenciado simplesmente como uma ação. Quase todos os elementos de ação são indivisíveis, ou seja, a execução de tais elementos não pode ser interrompida — não há nenhum estado intermediário entre o início e o fim da execução. Em conseqüência, a execução concorrente de $\mathbf{x := x + 1}$ e $\mathbf{x := x + 2}$ sempre adicionará o valor **3** ao valor de **x**. Os elementos de ação pertinentes a uma iteração não são indivisíveis. Contudo, qualquer seqüência de elementos de ação é tratada como indivisível se o modificador **atomic** é empregado (pág. 71). Os elementos de ação são apresentados a seguir:

AÇÃO NULA

A ação nula não produz controle. Por exemplo, na Figura 3-19, pág. 67, a transição do estado **K** para o **L** está rotulada com gatilho e ação nulos.

ATRIBUIÇÃO

Em Xchart são possíveis dois tipos de atribuição:

- $\mathbf{v := ExpNum}$, onde **v** é uma variável numérica e **ExpNum** é uma expressão cuja avaliação produz um valor numérico. Por exemplo, $\mathbf{v := x - (2 * y)}$ é uma atribuição válida, onde **v**, **x** e **y** são variáveis de controle inteiras.
- $\mathbf{c :=}$ **ExpLog**, onde **c** é uma variável lógica e **ExpLog** é uma expressão cuja avaliação produz um valor lógico. Por exemplo, $\mathbf{c := ((2 + x) < z)}$ **and b** é uma atribuição válida, onde **c** e **b** são variáveis lógicas e as demais inteiras.

GERAÇÃO DE EVENTOS PRIMITIVOS

raise é a construção utilizada para gerar um evento primitivo (instância de um tipo de evento primitivo). Se o evento primitivo deve ser visível apenas na instância do Xchart que executa esta ação, então ele é acumulado aos eventos primitivos ocorridos desde o início do passo e o conjunto resultante é utilizado na identificação do próximo micro-passo. Esse conjunto resultante é denominado estímulo externo acumulado (Seção 3.9). Se o evento primitivo é visível em outras instâncias, então ele será acumulado àqueles visíveis em tais instâncias, gerados desde o início do passo e, ao fim do passo, o conjunto resultante é enviado às instâncias pertinentes. Todos os eventos gerados por

uma instância em determinado passo só são visíveis a outras instâncias após o fim do passo em questão. A Seção 3.9 comenta minuciosamente esse processo. A visibilidade ou escopo de um evento primitivo é indicada de acordo com as formas de chamada da função **raise**:

- **raise(e)**
  Gera um evento primitivo do tipo **e** visível apenas na instância que executa esta ação.
- **raise(e, inst)**
  Gera um evento primitivo do tipo **e** a ser sinalizado para a instância **inst**.
- **raise(e, all)**
  Gera um evento primitivo do tipo **e** visível em todas as instâncias do sistema descrito em Xchart.

Todas as ações acima geram um evento primitivo a partir do tipo de evento primitivo fornecido. Cada tipo de evento primitivo possui um construtor que é executado quando qualquer uma dessas ações é executada. Tal construtor é uma atividade responsável por estabelecer os valores dos possíveis dados acoplados à instância que é criada. Se mais de uma instância de um mesmo tipo de evento primitivo são criadas, em um mesmo passo, então apenas os dados acoplados ao evento primitivo pelo último construtor prevalecerão. Por exemplo, se duas ou mais ações do tipo **raise(e)** são executadas em um passo, então os dados acoplados ao evento primitivo pela segunda ação substituirão aqueles acoplados pela ação anterior.

ESPERA POR TIPOS DE EVENTOS PRIMITIVOS

A instância que executa a ação **wait** torna-se inoperante (Seção 3.6) e permanece nesse estado até que o tipo de evento primitivo especificado ocorra. Variantes:

- **wait(e)**
  Torna a instância que executa essa ação inoperante até que o tipo **e** ocorra. Só a ocorrência deste tipo de evento primitivo torna a instância ativa novamente. Por exemplo, na Figura 3-18, a execução da transição final conduz a instância do Xchart **A** ao estado inoperante até que ocorra o evento **f**. Quando esse tipo ocorre a instância é finalizada.
- **wait(e, k)**
  Torna a instância que executa essa ação inoperante até que uma das duas situações seguintes se caracterize: (i) o evento **e** ocorra ou (ii) o intervalo de **k** unidades de tempo tenha transcorrido desde o início da execução desta ação. Ou seja, a espera possui um limite definido. A Figura 3-18 exemplifica esta ação. A regra

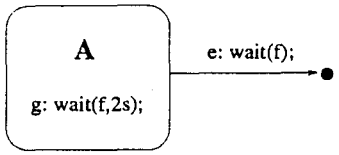


Figura 3-18: Espera por ocorrência de evento (**wait**)

g : wait(f, 2s), quando executada, faz com que a instância permaneça inoperante por no máximo **2** segundos. Se o tipo **f** ocorre antes de expirar o prazo de 2 segundos, a instância torna-se ativa nesse instante.

CONTROLE DE ATIVIDADES

A execução de atividades (Seção 3.3.12) não é da responsabilidade da execução de uma instância. Uma instância apenas determina quando uma atividade deve ser iniciada, interrompida e assim por diante. Os elementos de ação descritos abaixo controlam a execução de atividades. Em particular, se nenhuma porta (Seção 3.3.19) é fornecida, então o controle pertinente é sinalizado para a porta padrão da instância que executa o elemento de ação pertinente.

- **sync(a)**
  A instância que executa o elemento de ação **sync(a)** dá início à execução da atividade **a** e permanece inoperante até que ela finalize. Este controle é sinalizado para a porta padrão da instância. O elemento **sync(a, p)** provoca o início da execução da atividade **a** na porta **p**.

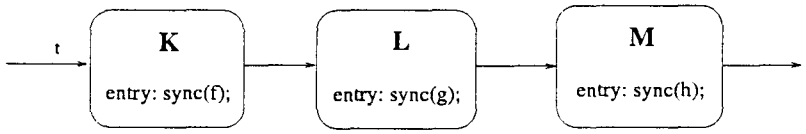


Figura 3-19: Execução síncrona entre atividade e instância

Na Figura 3-19 é mostrado parte de um Xchart. Quando uma instância desse Xchart ativar o estado **K**, a regra **entry : sync(f)** é executada. Em conseqüência, a atividade **f** é executada de forma síncrona (**sync(f)**). A instância permanece inoperante, aguardando pelo fim da execução da atividade **f**. Quando a atividade termina, a instância torna-se ativa e qualquer estímulo externo provoca a transição de **K** para **L**, conforme o rótulo nulo da transição entre esses estados. Ao ativar o estado **L** a instância torna-se novamente inoperante, pois a ação **sync(g)** é executada. O término da execução de **g** torna a instância ativa, o que a torna suscetível de ativar o estado **M** com qualquer que seja o estímulo externo tratado.

Novamente a instância torna-se inoperante enquanto durar a execução da atividade **h**. Durante a execução da atividade **f** (assim como **g** e **h**), que pode perdurar, a configuração da instância não é conduzida ao estado **L** (e **M**, respectivamente), caso estímulos externos ocorram no período em que a instância aguarda pelo término dessas atividades. A Seção 3.6 fornece detalhes.

- **start(a)**
  Ao contrário de **sync**, a instância que executa **start** permanece ativa. A execução da atividade e da instância prosseguem independentemente. Essa ação dispara a execução da atividade **a**, que é executada de forma assíncrona. O elemento **start(a, p)** provoca o mesmo efeito na porta **p**, em vez da porta padrão no caso anterior.

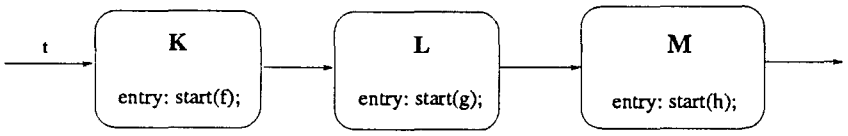


Figura 3-20: Execução assíncrona entre atividade e instância

A Figura 3-20 apresenta a versão assíncrona da descrição da Figura 3-19. A ativação do estado **K** não torna a instância em questão inoperante. Ou seja, a ocorrência de qualquer estímulo externo, quando o estado **K** estiver ativo, mesmo que a atividade **f** ainda não tenha sido completamente executada, provoca a desativação do estado **K** seguida da ativação do estado **L** (execução da transição de rótulo nulo entre esses estados). De forma análoga, qualquer estímulo externo provoca a transição de **L** para **M**, se o estado **L** está ativo, mesmo que a atividade **g** ainda esteja em execução. Esse comportamento também é válido para o estado **M**, que é desativado com qualquer estímulo externo mesmo antes do término da execução de **h**, executada como conseqüência da ativação de **M**. A animação da instância prossegue independentemente da execução dessas atividades, que não são "interrompidas" ou "finalizadas" quando os estados **K**, **L** e **M** são desativados. Para interromper/finalizar uma atividade deve ser empregada a ação **suspend** ou **stop**.

- **stop(a)**
  Finaliza a atividade **a** — a execução de **a** é definitivamente interrompida através da porta padrão da instância. Alternativamente, **stop(a, p)** produz efeito similar na porta **p**.

- **stop(all)**
  Similar à ação **stop**, contudo, nesse caso, todas as atividades serão finalizadas

através da porta padrão da instância que executa esta ação. Em particular, **stop(all, p)** interrompe todas as atividades através da porta **p**.

- **suspend(a)**
  Esta ação torna a atividade **a** suspensa, ou seja, sua execução é temporariamente interrompida. Enquanto suspensa, a condição **hanging** é verdadeira. Esse elemento sinaliza o controle pertinente para a porta padrão. Se esse controle deve ser sinalizado para uma porta **p** específica, então deve-se usar a forma **suspend(a, p)**.
- **resume(a)**
  Esta ação retoma a execução da atividade **a**, suspensa pela ação **suspend(a)**. A ação **resume(a, p)** produz o mesmo efeito através da porta **p**, em vez da porta padrão.

CRIANDO INSTÂNCIAS DE XCHARTS

Existem duas ações para criar instâncias de Xcharts. A diferença entre elas está no status da instância que cria uma outra instância. A ação **call** torna inoperante a instância que a executa. Em contrapartida, a ação **run** cria uma instância que é executada concorrentemente com a instância que executa essa ação.

- **call(xchart, id)**
  Cria uma instância do Xchart **xchart**. A instância é identificada pelo argumento **id**. Durante a "existência" da instância **id**, a instância que executou a ação **call** permanece inoperante.

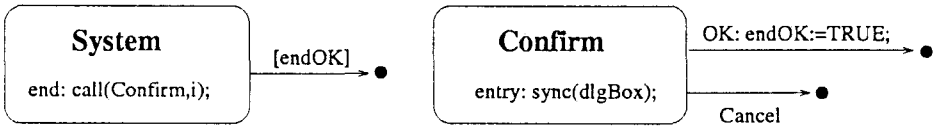


Figura 3-21: Execução síncrona de instância (**call**)

A Figura 3-21 descreve uma situação típica em sistemas interativos, onde um usuário deseja interromper a execução de um programa. (Três transições finais são apresentadas. Esse tipo de transição é descrito na Seção 3.3.14.) O evento **end**, que representa essa intenção do usuário é sinalizado para o estado **System** que modela o comportamento principal do sistema interativo. Em conseqüência, uma instância **i** do Xchart **Confirm** é criada. A instância que executa esta ação permanece inoperante até que a instância criada seja finalizada. A instância **i** descreve o comportamento de uma caixa de diálogo com a pergunta típica "Realmente deseja sair?" e os botões rotulados com "Sim" e "Não". A atividade **dlgBox** é responsável por exibir tal caixa de diálogo e gerar o evento **OK** ou

Cancel, conforme a ação do usuário sobre um dos botões. Qualquer um destes eventos provoca o fim da instância criada, o que automaticamente torna a instância de **System** ativa. Se o evento gerado é **OK**, então a variável lógica **endOK** torna-se verdadeira e, neste caso, ao fim da execução de **call**, a instância de **System** executa a transição rotulada com **[endOK]** e finaliza a sua execução.

- **run(xchart, id)**
  Cria uma instância identificada por **id**, do Xchart **xchart**. A instância que executa esta ação não se torna inoperante enquanto a instância criada é executada. Ao contrário, ambas permanecem ativas após a execução dessa ação.

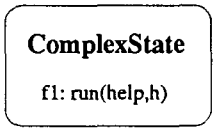


Figura 3-22: Execução assíncrona de instância (**run**)

A Figura 3-22 exibe o Xchart **ComplexState** contendo a regra **f1 : run(help, h)**. Se essa regra é executada para uma instância desse Xchart, então uma instância **h** do Xchart **help** é criada. A instância de **ComplexState** que executa essa regra não se torna inoperante, ela permanece ativa. Após a execução dessa ação a instância criada é executada concorrentemente com aquela que criou a instância.

LIMPANDO HISTÓRIA

O destino de uma transição pode ser um símbolo de história (Seção 3.3.18). Um símbolo de história sempre está associado a um único estado **s** e determina o subestado de **s** mais recentemente ativado. A ação **clear(h)** torna nulo o conteúdo do atributo do símbolo de história **h**. Ou seja, após a execução de uma ação **clear(h)**, a próxima ativação do estado associado ao **h**, pelo símbolo de história **h**, comportar-se-á como uma ativação por transição inicial, pois o atributo de **h** não contém nenhum informação. O atributo de um símbolo de história é devidamente atualizado toda vez que o estado associado ao símbolo torna-se ativo.

Na Figura 3-23, por exemplo, tem-se a ativação do estado **C** pelo símbolo de história **H1**, que é o destino da transição cuja origem é o estado **E**. Em um cenário onde o estado **B** está ativo e um estímulo externo contendo um evento primitivo do tipo **e** é sinalizado, a ação **clear(H1)** é executada e o estado **E** é ativado. Um novo estímulo externo de mesmo conteúdo provoca a ativação do estado **C** seguida da ativação do estado **A**, obtido pela transição inicial. Embora **B** seja o último subestado de **C** que esteve ativo, a ação **clear(H1)** elimina essa informação do atributo de **H1**. A ativação do estado **C**, contudo, faz com que o atributo de **H1** seja devidamente atualizado.

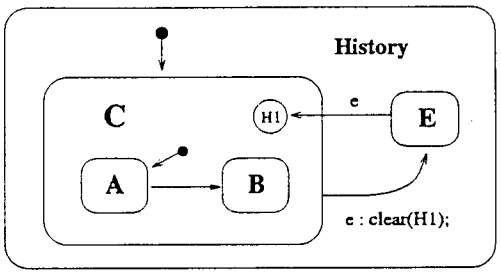


Figura 3-23: Limpando o conteúdo de um atributo de história (**clear**)

Nesse caso, o atributo indica que o estado **A** é o último subestado de **C** que esteve ativo. Naturalmente, se a transição do estado **A** para o estado **B** é executada, então o atributo é atualizado, informando que o estado **B** é o subestado mais recentemente ativo do estado **C**.

ATOMICIDADE

Todos os elementos de ação vistos até agora são indivisíveis, ou seja, a execução de cada um deles não pode ser interrompida, não existe nenhum estado intermediário entre o início e o fim da execução desses elementos. Atribuições, **call** e **clear** estão entre esses elementos, por exemplo. Os elementos seguintes, contudo, não são indivisíveis, assim como uma seqüência de elementos de ação, quaisquer que sejam eles, também não apresenta essa propriedade. O elemento de ação **atomic** é utilizado para assegurar que elementos de ações não indivisíveis comportem-se como indivisíveis. Esse recurso também pode ser aplicado à uma seqüência de elementos, tornando a seqüência indivisível. Dessa forma, enquanto $\mathbf{x := x + 1}$ é equivalente à **atomic** $\{\ \mathbf{x := x + 1}\ \}$, a seqüência $\mathbf{x := x + 1; x := x * 2}$ produz o valor $\mathbf{(x + 1) * 2}$ apenas em condições particulares. Na presença de concorrência, esse valor final pode ser assegurado através da construção

atomic $\{\ \mathbf{x := x + 1;\ x := x * 2;}\ \}$

Quando esse elemento de ação **atomic** é executado, sabe-se que, necessariamente, o valor final de **x** será $\mathbf{(x + 1) * 2}$, pois essa seqüência é indivisível.

A Figura 3-24 exibe o Xchart **Atomicity** e a variável de controle local **x**. Após a criação de uma instância de **Atomicity**, os estados **A** e **C** tornam-se ativos. Se um estímulo externo contendo o evento primitivo do tipo **e** é sinalizado para essa instância, então as transições de **A** para **B** e de **C** para **D** serão executadas. O valor final de **x** pode ser **12** ou **16**, conforme a ordem de execução das atribuições. Se atribuições não fossem indivisíveis, então os valores **4** e **8** também poderiam ser obtidos.

Abaixo sequem as seqüências possíveis equivalentes aos resultados compatíveis com uma execução concorrente dessas ações:

$\mathbf{x := x + 2;\ x := x + 2;\ x := x * 2;\ x := x * 2;}$ ou
$\mathbf{x := x + 2;\ x := x * 2;\ x := x + 2;\ x := x * 2;}$

Se a única seqüência desejada é a última, então o modificador **atomic** pode ser empregado para evitar o *interleaving* indesejável. Por exemplo,

$\mathbf{atomic\ \{\ x := x + 2;\ x := x * 2;\ \};}$

é tratada como uma única ação. Ao executar essa sentença, necessariamente o valor final de **x** é dado por $\mathbf{(x + 2) * 2}$. No Xchart exemplificado, o emprego desse modificador para as ações das transições de **A** para **B** e de **C** para **D** produzirá, deterministicamente, o valor final **12**. Se **atomic** é empregado em apenas uma das transições, então o valor **16** também pode ser produzido para **x**.

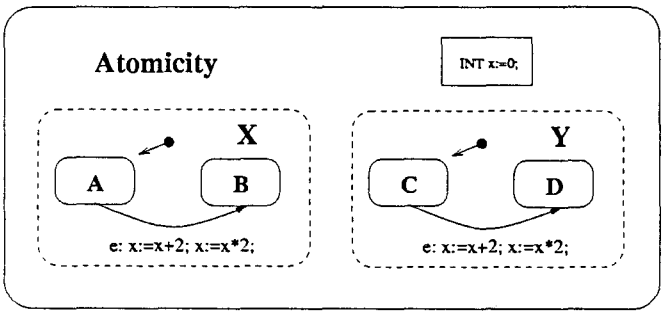


Figura 3-24: Atomicidade de elementos de ação

O elemento de ação **atomic** não torna possível a ocorrência de *deadlock*, pois a única dependência entre tais elementos é estritamente temporal (há uma ordem de execução entre elas).

REPETIÇÃO

Uma seqüência qualquer de elementos de ações pode ser executada enquanto uma determinada condição for verdadeira ou ainda um número preestabelecido de vezes. As seqüências estabelecidas pelos elementos **while**, **do** e **for** não são indivisíveis. Se tal propriedade é desejada, então o modificador **atomic** deve ser empregado. As estruturas abaixo permitem a especificação de operações infinitas, que devem ser evitadas. Os efeitos de um passo só são visíveis ao subsistema reativo ao fim do mesmo. Dessa forma, se uma ação não tem fim, então o restante do subsistema reativo permanece "congelado" a espera do fim do passo da instância que executou tal ação.

- **while**
  A ação **while** executa uma seqüência de elementos de ação enquanto determinada condição for verdadeira. Por exemplo, a execução de

  while (cond) { a_1; a_2; ...; a_n; };

  inicia-se com a avaliação da condição **cond**. Se verdadeira, então as ações $a_1, ..., a_n$ são executadas. Ao fim da execução dessa seqüência tal condição é novamente avaliada e, se verdadeira, a seqüência é executada novamente. Esse processo se repete até que a condição seja falsa.

- **do**
  A execução do **while** é dependente da condição **cond**, que sempre é avaliada antes do início da execução da seqüência de elementos de ação pertinente. A execução do **do**, em contrapartida, envolve a avaliação da condição após a execução da seqüência pertinente. Ou seja, primeiro é executada a seqüência de elementos de ação e posteriormente a condição é avaliada. Se verdadeira, o processo se repete. Esse elemento de ação termina sua execução quando a condição é avaliada em falsa. Por exemplo,

  do { a_1; a_2; ...; a_n; } while (cond);

  executa a seqüência de ações e, ao término da execução de **an**, a condição é avaliada. Se verdadeira e enquanto for verdadeira, a execução da seqüência é repetida.

- **for**
  A forma geral de um elemento de ação **for** é dada por

  for (atr1; cond; atr2) { a_1; a_2; ...; a_n; }

  A execução desse elemento envolve a execução da atribuição **atr1** após a qual a condição **cond** é avaliada. Se verdadeira, a seqüência de elementos de ação é executada seguida da execução da atribuição **atr2**. Após a execução de **atr2** a condição é novamente avaliada. Se verdadeira e enquanto verdadeira, a seqüência de elementos de ação seguida de **atr2** é executada. Por exemplo, a execução da ação

  for ($x := 1$; $x < 11$; $x := x + 1$) { a_1; a_2; a_3; };

  faz com que as ações a_1, a_2 e a_3 sejam executadas 10 vezes se as ações a_1, a_2 e a_3 não alteram o valor de **x**. Neste caso a condição $x < 11$ torna-se falsa apenas quando a ação $x := x + 1$ for executada 10 vezes já que a variável **x** recebe o valor **1** no início da execução do **for**.

DECISÃO

Uma seqüência de elementos de ação pode ou não ser executada de acordo com a avaliação de uma condição. A ação

if (cond) { a_1; a_2; ...; a_n; } else { a_{n+1}; a_{n+2}, ...; a_m };

é executada conforme a avaliação da condição **cond**. Se o valor obtido é verdadeiro, então a execução do elemento **if** é finalizada com a execução das ações a_1, a_2, ..., a_n. Caso contrário, os elementos a_{n+1}, a_{n+2}, ..., a_m são executados e a execução do **if** é finalizada.

3.3.11 Modificador step

Quando se tem acesso ao valor de uma variável ou expressão, lógica ou numérica, o valor obtido é o valor corrente ou atual. Ou seja, aquele disponível no instante em que a consulta é efetivada, seja durante um passo ou mesmo durante a execução de um micro-passo (Seção 3.9). Alternativamente, pode-se desejar obter o valor de uma variável ou de outros elementos da configuração de uma instância disponíveis no início de um passo, mesmo que ações de micro-passos anteriores tenham alterado o valor corrente desses elementos. Para obter tal valor é utilizado o modificador **step**.

O modificador **step** pode ser utilizado em atribuições presentes em ações (Seção 3.3.10) para obter o valor de variáveis e expressões ou ainda em combinação com condições: aquelas associadas a estados (**in**, por exemplo) e aquelas associadas a atividades (como **active**).

Na Figura 3-25, por exemplo, se uma instância do Xchart **Guard** está com os estados básicos **A**, **B** e **C** ativos e o valor de **x** é **3**, então um estímulo externo contendo um evento primitivo do tipo **e** provoca a transição do estado **A** para **D**. Em conseqüência, a transição de **B** para **E** é executada, pois embora o valor de **x** seja **2** no instante em que a condição é avaliada, o valor consultado através da condição é aquele disponível no início do passo, ou seja, o valor **3**. Essa última transição ainda gera um evento primitivo do tipo **h**, que provoca a execução da transição de **C** para **F**, pois o tipo **h** ocorre e a condição $x = 2$ é verdadeira.

Outro exemplo: a expressão **step(in(a))** é verdadeira apenas se o estado **a** estava ativo no início do passo onde é avaliada. Se um dado estado **s**, por exemplo, é concomitantemente a origem e o destino de uma dada transição, então para verificar se essa transição foi executada ou não, pode-se usar o gatilho abaixo:

tr(not(in(s))) [in(s) and step(in(s))]

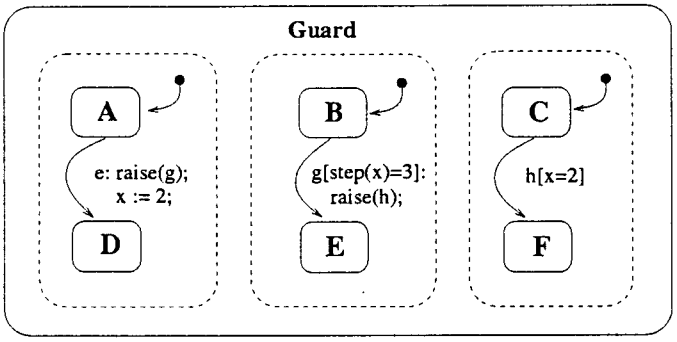


Figura 3-25: Obtendo o valor no início de passo de variável (step)

Esse gatilho é verdadeiro se e somente se: (a) o estado s estava ativo no início do passo p em questão; (b) o estado s está ativo, no passo p, no instante em que o gatilho é avaliado e (c) o estado s foi desativado no intervalo compreendido entre o início do passo p e o instante em que o gatilho é avaliado.

3.3.12 Atividade

Atividades são procedimentos/computações que fornecem a funcionalidade do sistema cujo controle é modelado em Xchart. Atividades podem ser implementadas como procedimentos, rotinas, métodos ou funções em linguagens de programação convencionais. A execução de uma atividade dura por um período não nulo, como imprimir relatórios, fechar arquivos, soar uma campainha, calcular o valor de $(-b \pm \sqrt{b^2 - 4ac})/2a$. Atividades realizam operações além dos propósitos para os quais a linguagem Xchart foi projetada.

A execução de atividades não é da responsabilidade do subsistema reativo, como indicado na Figura 3-3, na pág. 44. A execução de especificações na linguagem Xchart fornece o controle que indica quando uma atividade deve ser iniciada, finalizada e assim por diante.

Embora a execução de atividades esteja além dos interesses de Xchart, o subsistema reativo oferece alguns serviços que podem ser utilizados por atividades. Por exemplo, eventos primitivos podem possuir dados acoplados aos mesmos. Esses dados podem ser estabelecidos por uma atividade e recuperados por uma outra (o subsistema reativo é responsável pelo transporte dos dados associados a um evento). Atividades ainda podem gerar eventos primitivos, criar instâncias de Xcharts e manipular variáveis de controle globais. Tais serviços são acessíveis através da Interface de Programação de Xchart (IPX).

O ciclo de vida de uma atividade é exibido na Figura 3-26. Do ponto de vista do subsistema reativo, uma atividade pode estar *inerte*, *ativa*, ou *suspensa*. Inicialmente, toda e qualquer atividade está inerte. É necessária a execução da ação **start(a)** ou

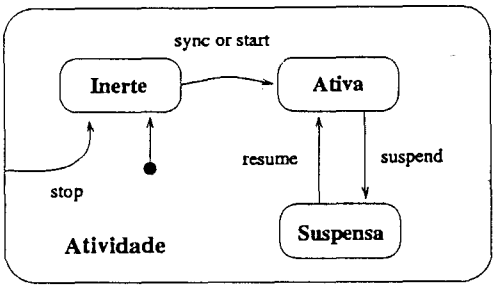


Figura 3-26: Ciclo de vida de uma atividade

sync(a) para que uma atividade **a** se torne ativa. Se uma atividade **a** está ativa, então ela se torna inerte com a execução da ação **stop(a)**, ou suspensa, com a execução de **suspend(a)**.

3.3.13 Regra

Uma regra **g : a** é a união do gatilho **g** e da ação **a**, que é executada quando a regra é executada. A ação **a** é uma seqüência de elementos de ação. Regras rotulam transições ou são especificadas no interior de estados. Nesse texto, quando se faz referência ao termo regra, entende-se regra associada a estado. No outro caso o termo regra é explicitamente conectado à transição da qual é parte. Uma regra é executada no máximo uma vez por passo. Diz-se que uma regra está habilitada se o seu gatilho está habilitado. Uma regra é exemplificada abaixo:

e1 and e2 [(v = 8) and in(s, i10)] : start(report);

Esta regra está habilitada quando os tipos de evento **e1** e **e2** ocorrem e o valor corrente da expressão **v** é **8** e o estado **s**, na instância **i10**, está ativo. Uma regra habilitada não necessariamente é executada. É necessário que o estado ao qual pertence esteja ativo para que ela possa ser executada. Se esse for o caso, então a regra é dita relevante. Uma regra rotulando uma transição possui a mesma sintaxe, mas a sua semântica de execução envolve outras considerações apresentadas na Seção 3.3.14. Se uma regra é executada, a sua ação é executada. Na regra acima, a ação **start** inicia a execução da atividade **report**.

Na Figura 3-27 são fornecidos exemplos de regras. Quando uma instância do Xchart **MsgErro** é criada, o estado **MsgErro** é ativado. Em conseqüência, a execução da atividade **ShowMsg** é iniciada, pois o gatilho está habilitado nesse instante. Quando uma instância desse Xchart é finalizada, a ação **stop(ShowMsg)** é executada. Se o tipo de evento **keyboard** ocorre, dado um estímulo externo sinalizado para uma instância de

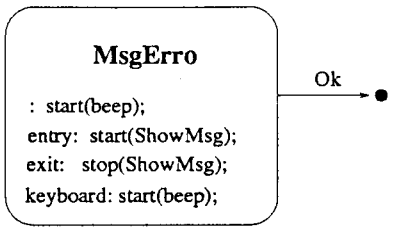


Figura 3-27: Exemplos de regras

MsgErro, então a ação **beep** é executada duas vezes. Uma delas é decorrente da regra cujo gatilho é nulo. A outra da regra onde o gatilho é **keyboard**.

Ordem de Execução de Regras

Cada estado ativo de uma instância pode conter uma ou mais regras a serem executadas em um mesmo micro-passo. Nesse caso, a hierarquia e o tipo dos estados determinarão a ordem da execução de suas ações ou se elas serão executadas concorrentemente. Tal ordem de execução apenas oferece um nível a mais de controle sobre a execução de regras. Se ela é irrelevante para a especificação, então o conteúdo dessa seção pode ser desconsiderado. As regras associadas a um estado s são executadas antes daquelas associadas a um descendente de s. Se não existe essa relação entre dois estados que possuem regras a serem executadas, então as ações das regras associadas a esses estados são executadas concorrentemente.

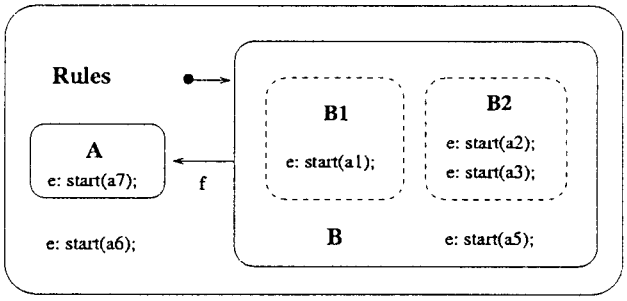


Figura 3-28: Ordem de execução de regras

Por exemplo, uma instância do Xchart **Rules** (Figura 3-28) com os estados básicos **B1** e **B2** ativos, ao tratar um estímulo externo contendo um evento do tipo **e**, executará várias regras, cujas ações produzirão um efeito similar ao da execução de uma das seqüências abaixo:

a6; a5; a1; a2; a3;
a6; a5; a1; a3; a2;

a6; a5; a2; a1; a3;
a6; a5; a2; a3; a1;
a6; a5; a3; a1; a2;
a6; a5; a3; a2; a1;

Pela hierarquia de estados do Xchart **Rules**, sempre a ação **a6** será executada antes da ação **a5**, que será seguida pelas demais. Regras associadas a um mesmo estado não são executadas concorrentemente. Uma particular ordem de execução é estabelecida em tempo de execução quando a especificação não determinar uma relação de ordem total. Por exemplo, se as regras do estado **B2** possuem prioridades distintas, então a de maior prioridade será executada antes da outra. Se a regra **e : start(a2)** possui maior prioridade, então as seqüências possíveis seriam reduzidas para:

a6; a5; a1; a2; a3;
a6; a5; a2; a1; a3;
a6; a5; a2; a3; a1;

3.3.14 Transição

Em geral, uma *transição* estabelece uma relação entre estados. Esse é o único mecanismo através do qual estados podem ser ativados/desativados. Toda transição possui uma origem (conjunto de estados ou estado fictício), um rótulo (gatilho e ação), um destino (conjunto de estados e/ou símbolos de história, ou ainda um estado fictício) e um número de prioridade. As várias combinações possíveis dos tipos de elementos que podem compor a origem e o destino identificam os tipos de transição.

Para ser executada, é necessário, mas não suficiente, que o conjunto origem de estados da transição esteja ativo e a regra que a rotula esteja habilitada. (Um conjunto de transições que satisfaz tais condições pode conter transições excludentes e, nesse caso, nem todas poderão ser executadas. A Seção 3.3.17 fornece detalhes.) Se o conjunto origem de estados está ativo, então diz-se que a transição é relevante. Transição habilitada é aquela cuja regra que a rotula está habilitada. A execução de uma transição significa:

(i) desativar os estados origem e executar as conseqüências de tais desativações;

(ii) executar a ação da regra que rotula a transição e

(iii) ativar os estados destino e executar as ações decorrentes de tais ativações.

As operações acima são executadas na ordem em que aparecem. Ou seja, não há concorrência entre as ações decorrentes da desativação e ativação de estados de uma dada transição. Os tipos de transição são definidos abaixo:

A transição rotulada com a regra **t1** é dita convencional, na Figura 3-29, ou seja, estabelece uma relação entre estados ou entre estados e símbolos de história. O estado de origem **A** deve estar ativo e a regra, identificada por **t1**, habilitada para que a transição possa ser executada. Em conseqüência, o estado **A** torna-se inativo, a ação da regra é executada e o estado **B** torna-se ativo, nesta ordem.

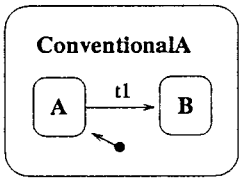


Figura 3-29: Transição convencional 1:1

Uma transição convencional pode ter como origem e/ou destino um conjunto de estados. Na Figura 3-30, caso os estados **C** e **D** de uma instância do Xchart **ConventionalB** estejam ativos e a regra **t2** esteja habilitada, então esta transição é executada e os estados **E** e **F** são ativados. No exemplo, o destino da transição **t2** não inclui explicitamente todos os estados que devem ser ativados, pois **F** também é ativado e não faz parte do destino por ser subestado de estado concorrente. Ainda convém ressaltar que tanto a origem quanto o destino podem conter estados em níveis hierárquicos distintos.

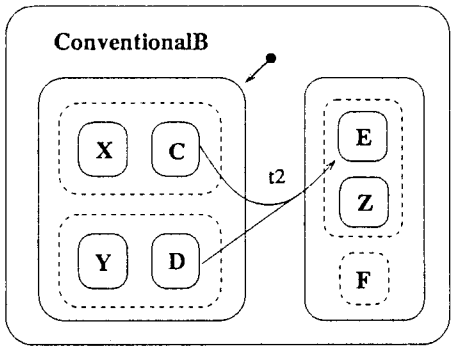


Figura 3-30: Transição convencional N:1

O destino de uma transição pode conter símbolos de história. No exemplo da Figura 3-31, a transição rotulada com **t9** parte do estado **M** e tem como destino o símbolo de história **H**. A ativação dos subestados de **N** não ocorre via transições iniciais, mas de acordo com o passado recente da ativação de **N**. Símbolos de história são explorados na Seção 3.3.18.

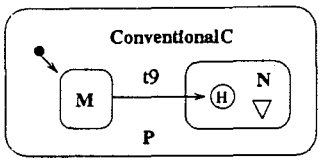


Figura 3-31: Transição cujo destino é símbolo de história

Na Figura 3-32, as transições **t10** e **t11** são ditas *internas*, e as transições **t12** e **t13** são ditas *externas*. Uma transição interna, quando executada, não provoca a desativação do estado de origem, ou seja, as transições **t10** e **t11** não provocam a desativação do estado **X**. Em contrapartida, a transição **t12** desativa todos os descendentes ativos de **X**. A transição **t13** provoca a desativação e posterior ativação do estado **W**. Transições externas não são permitidas partindo da raiz de um Xchart ou entre subestados de um estado concorrente. Naturalmente, se a transição externa é uma transição final, então ela é uma transição válida. Transições internas podem existir para todo e qualquer estado.

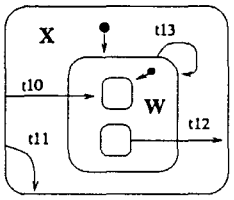


Figura 3-32: Transições externas e internas

Transições iniciais (•⟶) distinguem-se das demais por possuírem um estado fictício como origem •. Quando um estado exclusivo é ativado, o processo de ativação possivelmente prossegue por um de seus subestados. A identificação desse subestado é feita através do destino de uma transição inicial. No exemplo da Figura 3-33, quando uma instância de **Initial** é criada, o estado **Initial** é ativado. Esse estado é exclusivo, ou seja, o processo de ativação prossegue com a possível ativação de um de seus subestados. Nesse caso, ou a transição inicial **t5** ou a **t6** será executada. Se o gatilho de **t6** é o único habilitado, então o estado **I** é ativado após a ativação do estado **J**. Se o gatilho de **t5** é o único habilitado, então **J** é ativado. Nesse último caso, o processo é novamente aplicado ao estado **J** e assim por diante, até que um estado básico seja atingido (**J** pode permanecer temporariamente básico ou o estado **H** é ativado). Transições iniciais estão associadas ao processo de ativação de estados, comentado na Seção 3.3.16. Se **t5** e **t6** estão habilitados, então tem-se um caso de exclusão mútua entre essas transições (Seção 3.3.17). Se nenhuma delas está habilitada, então o estado **Initial** permanece temporariamente básico.

Uma transição final é utilizada para finalizar a execução de uma instância. Ou seja,

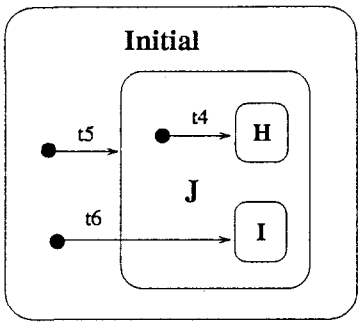


Figura 3-33: Transições iniciais

a execução de tal transição torna uma instância inativa. As transições finais (—•) empregam um estado fictício como destino. No exemplo da Figura 3-34, se uma instância de **L** executa a transição rotulada com **t7**, então ela é finalizada. A transição rotulada com **t8** também é final — ela parte do estado **K** e tem como destino um estado fictício. Se ela for executada, então a instância pertinente também é finalizada. O destino de uma transição final sempre está além da borda do estado mais externo, ao contrário das transições pseudo-finais.

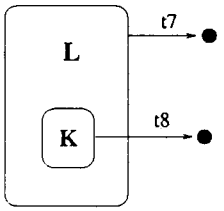


Figura 3-34: Transições finais

As transições rotuladas com **t1**, **t2**, **t3** e **t4**, na Figura 3-35, são ditas *pseudo-finais*. São transições cujo destino é um estado fictício associado a um estado exclusivo. Quando uma transição pseudo-final é executada, o estado **s** que contém o estado fictício permanece temporariamente básico pelo menos até o final do passo em questão. Um posterior estímulo externo é necessário para ativar um dos subestados de **s**, necessariamente através de uma transição inicial. A transição **t1**, por exemplo, quando executada, desativa todos os subestados de **Pseudo – finais**, que permanece temporariamente básico até que o próximo estímulo externo seja tratado, quando a transição inicial para **C** é executada. A transição **t4** tem efeito similar. A transição **t2** desativa os subestados de X, que permanece temporariamente básico até o próximo estímulo externo. A transição **t3** ativa o estado **X**, que permanece temporariamente básico até a ocorrência de um estímulo externo que habilite a transição inicial para o estado **B**.

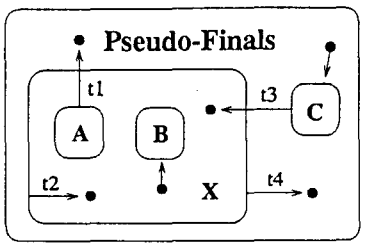


Figura 3-35: Transições pseudo-finais

Exemplos

A Figura 3-36 ilustra dez transições. Três delas diferem das demais por possuírem um estado fictício como origem ou destino. As duas rotuladas com **e** e [c] são transições iniciais. A terceira é uma transição final rotulada com o tipo de evento primitivo **e3**.

As transições iniciais indicam que, possivelmente, ou o estado **C** ou o estado **B** será ativado após a criação de uma instância de **A**. As transições iniciais também são utilizadas quando a transição rotulada com **e4** for executada. O estado **C** será ativado, no momento da criação da instância, se a regra [c] estiver habilitada, o que só é possível caso a condição c seja verdadeira. Se esta condição não é verdadeira neste instante, então o estado **C** não pode ser ativado. Por outro lado, o estado **B** só é ativado caso o tipo **e** ocorra. Ou seja, se a condição **c** não for verdadeira, então um subestado de **A** só será ativado com a ocorrência de um estímulo externo. Mesmo com a ocorrência de um estímulo externo, se nenhuma dessas situações se verifica, o estado **A** permanece temporariamente básico, ou seja, nenhum dos seus subestados é ativado até que o tipo **e** ocorra ou que a condição **c** torne-se verdadeira. Um estímulo externo contendo **tr(c)** e um evento primitivo do tipo **e** é suficiente para habilitar ambas as transições iniciais. Nesse caso, elas são ditas excludentes e uma delas é selecionada, conforme o processo descrito na Seção 3.3.17.

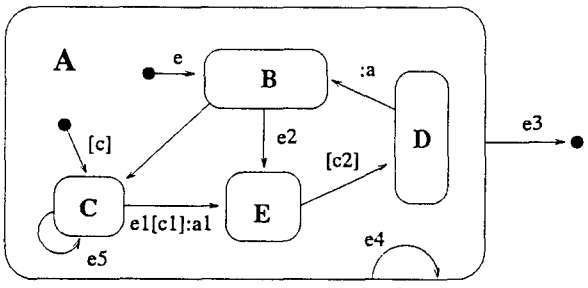


Figura 3-36: Exemplos de transições

A única transição final indica que qualquer instância do Xchart **A** será finalizada quando um estímulo externo provocar a ocorrência do tipo **e3**. Tal estímulo é suficiente, pois o estado raiz de toda e qualquer instância sempre está ativo.

A transição do estado **C** para o estado **E** é rotulada pela regra **e1[c1] : a1**. Quando o estado **C** estiver ativo, ocorrer o tipo **e1** e a condição **c1** for verdadeira, a transição para o estado **E** ocorre, a ação **a1** é executada e o estado **E** é ativado.

A transição do estado **B** para o estado **C** está rotulada por uma regra nula (evento, condição e ação nulos). Se **B** está ativo, qualquer estímulo externo habilita esta transição. Se um estímulo externo contém o evento **e2**, por exemplo, as duas transições partindo do estado **B** ficam habilitadas. Estas transições são excludentes (Seção 3.3.17): apenas uma é executada. Se o estímulo externo não contém um evento que provoca a ocorrência do tipo **e2**, então a transição para o estado **C** é executada. Na presença de prioridade entre transições, os tipos **e3** e **e4** também poderiam impedir a execução da transição de **B** para **C**.

A transição de **E** para **D** é executada se a condição **c2** é verdadeira quando qualquer estímulo externo, exceto se contiver **e3** e/ou **e4**, for tratado.

A transição do estado **D** para o estado **B** está rotulada com um evento e condição nulos. Apenas a ação não é nula. Quando o estado **D** estiver ativo, qualquer estímulo externo provoca a transição para **B** e a conseqüente execução da ação **a**, desde que o estímulo não contenha eventos primitivos dos tipos **e3** e/ou **e4**. Neste último caso, se as transições rotuladas com tais tipos possuírem maior prioridade do que a transição de **D** para **B**, então esta última não será executada.

A transição rotulada com **e4** é uma transição interna. Se ela é executada, então todos os subestados ativos de **A** são desativados. O estado **A** não é desativado. Em contrapartida, aquela rotulada com **e5** provoca a desativação do estado **C** seguida da sua ativação.

O Xchart **X** (Figura 3-37) exibe algumas transições sintaticamente válidas e outras não permitidas. A transição externa associada ao estado **B** não é permitida porque desativa o estado **B** enquanto o estado **A** permanece ativo (todos os subestados de um estado concorrente ou todos estão ativos ou não estão ativos). Não são permitidas transições entre estados concorrentes. A transição do estado **A** para o estado **X** desativa todos os subestados de **X** e inicia o processo de ativação desse estado. Essa transição é similar àquela interna associada ao estado **X**. A transição de **A** para **B** e aquela de **(A, B)** para **(A, B)** não são permitidas, pois a origem e o destino possuem estados concorrentes. A transição externa associada ao estado **X** não é permitida. A raiz de um Xchart só pode ser desativada pela execução de uma transição final.

3.3.15 Conseqüências de uma Transição

A Figura 3-38 exibe dois Xcharts. Em uma instância do Xchart **AB**, quando a transição **t1** é executada, o estado origem **A** é desativado, a ação da regra que rotula a transição

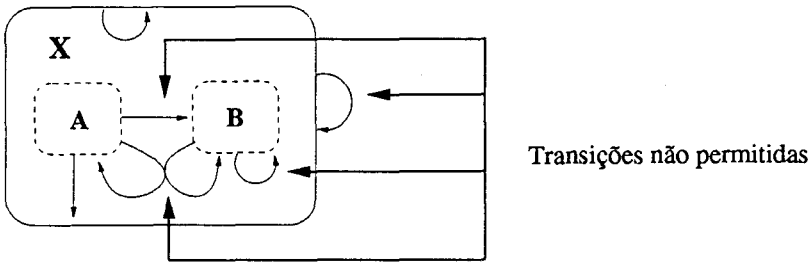


Figura 3-37: Algumas transições válidas e outras não permitidas

é executada e o estado **B** é ativado. O processo de ativação e desativação de um estado é comentado na Seção 3.3.16. Grande parte das transições vistas até agora são similares à transição **t1**. Entretanto, a execução da transição rotulada pela regra **t2**, no Xchart **CDEF**, envolve mais tarefas do que aquelas descritas no caso anterior. Não é suficiente desativar a origem **D** e ativar o destino **E**. Os estados **C**, **D** e **CD** devem ser desativados e a ação da regra que rotula a transição executada, seguida da ativação de **EF** e dos seus subestados **E** e **F**. Para toda e qualquer transição é necessário identificar um conjunto de estados que serão desativados e outro que será ativado. O conceito de Ancestral Comum Mais Próximo ou ACMP, por simplicidade, é empregado com essa finalidade.

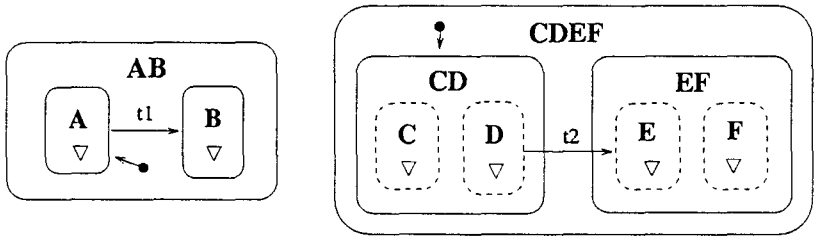


Figura 3-38: ACMP de transição: **t1**, estado **AB**; **t2**, estado **CDEF**

O ACMP de uma transição não está definido para transições iniciais e finais. Nesses casos, os conjuntos de estados a serem ativados e desativados são identificados sem exigir a noção de um ACMP. A definição do ACMP de uma transição **t** é estabelecida sobre o conjunto formado pela união da origem e destino da transição. A origem contém apenas estados, mas o destino ainda pode conter estados fictícios e/ou símbolos de história. Cada símbolo de história é substituído pelo estado que contém o símbolo. Analogamente, cada estado fictício é substituído pelo estado que o contém. Dessa forma, o conjunto união é composto apenas por estados. Definido o conjunto união **U** para uma transição **t**, o ACMP de **t** é o estado que satisfaz as seguintes exigências:

I Se um dos estados de **U** é ancestral de todos os demais, então ele é o ACMP.

II Se **U** contém um único estado **s**, então dois casos são possíveis:

(a) Se **t** é externa, então o ACMP é o ancestral direto de **s**.

(b) Se **t** é interna, então o ACMP é o próprio estado **s**.

III Se o ACMP não pode ser identificado pelas condições anteriores, então o ACMP é o estado que satisfaz as seguintes restrições:

(a) O ACMP é um ancestral de todos os estados do conjunto **U**.

(b) Se um conjunto não unitário de estados satisfaz essa condição, então o ACMP é o elemento que é descendente de todos os demais desse conjunto, ou seja, ele é o "mais próximo".

Dada uma transição, a sua execução inclui a desativação de todos os subestados do seu ACMP — naturalmente, um subestado só pode ser desativado se estiver ativo. Todos os estados que compõem o destino da transição serão ativados. Se a transição é pseudo-final, então os subestados do destino não são ativados. Ainda podem existir estados que são ancestrais daqueles que compõem o destino e são descendentes do ACMP da transição. Esses estados, se existirem, também são ativados. Por exemplo, na Figura 3-38, o estado **CD** do Xchart **CDEF** é ancestral da origem da transição **t2** e descendente do ACMP, que é o estado **CDEF**. O processo descrito na Seção 3.3.16 estabelece as operações que são realizadas para desativar e ativar um estado qualquer.

A Figura 3-39 exibe o Xchart **X**, que possui várias transições. O ACMP da transição **t1** é o estado **N**, aplicação da norma I (descrita acima). A transição **t2** tem vários estados como origem, o destino inclui o estado **L** e o símbolo de história **H1**, associado ao estado **H**. O ACMP de **t2** é o estado **X** (norma III). **J** é o ACMP de **t3** (norma III) e **K** é o ACMP de **t4** (norma III) e de **t5** (norma I). As transições **t6** e **t7** têm como origem e destino um único estado, contudo, o ACMP da transição externa **t6** é **X** (norma IIa), enquanto o ACMP da transição interna **t7** é o estado **P** (norma IIb). A transição **t6** desativa o estado **P**, pois é um dos descendentes ativos de **X** quando essa transição é executada. Em contrapartida, a transição **t7** não desativa o estado **P**. A transição **t10** possui como ACMP o estado **X** (norma IIb).

As transições não convencionais, **t8** e **t9**, não possuem ACMP. A execução da **t8** não implica a desativação de nenhum estado, ao contrário da **t9**, que não ativa nenhum estado e desativa o estado **X**. Desativar a raiz significa finalizar a instância pertinente, ou seja, a transição **t9** é final.

3.3.16 Processo de Ativação e Desativação de Estados

Quando uma instância de um Xchart é criada, quando é destruída ou ainda quando uma transição é executada, alguns estados são ativados enquanto outros são desativados. Por

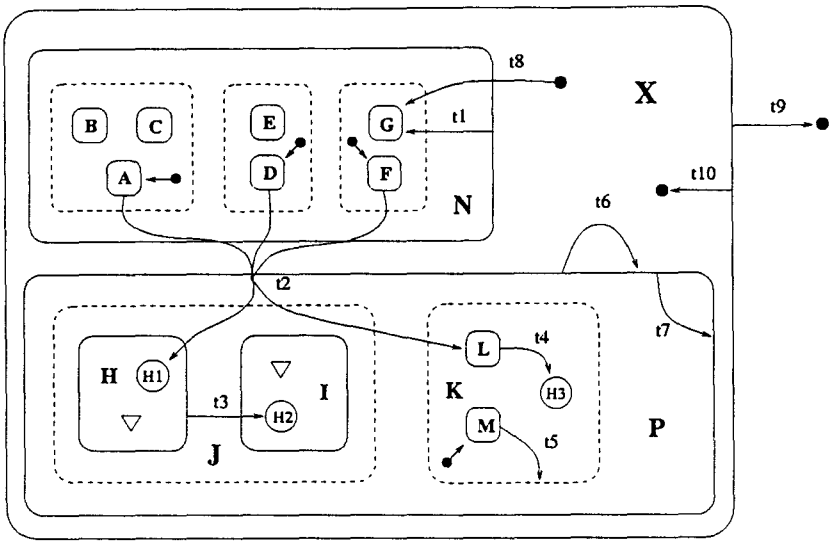


Figura 3-39: Mais exemplos de transições

exemplo, na criação de uma instância, o estado mais externo é ativado; na destruição, o mesmo estado é desativado. A execução de uma transição estabelece um conjunto de estados que devem ser desativados e outro daqueles que devem ser ativados, conforme Seção 3.3.14. Essa seção define as conseqüências da ativação e desativação de um estado.

Exceto quando estados concorrentes estão envolvidos, há uma seqüência única em que desativações e ativações ocorrem. Essa ordem permite identificar a ordem em que regras associadas à ativação (**entry**) e desativação (**exit**) de estados serão executadas. Se um estado concorrente deve ser desativado/ativado, então mais de uma seqüência é possível.

A desativação/ativação de todo e qualquer estado é realizada conforme as normas abaixo. Nada é afirmado acerca de estados ativos simultaneamente, ou seja, subestados de estados concorrentes. As execuções das regras associadas a esses estados são realizadas concorrentemente. Concorrência é explorada na Seção 3.4.

- Um estado é desativado somente após a desativação de seus subestados.
- Um estado é ativado somente após a ativação de seus ancestrais.
- Se um estado exclusivo **s** é ativado, então uma das transições iniciais associadas a esse estado pode conduzir à ativação de um de seus subestados. Se nenhuma transição inicial está habilitada, então o estado **s** permanece temporariamente básico.
- Se um estado concorrente é ativado, então todos os seus subestados são ativados concorrentemente. Cada subestado, contudo, segue estas normas de ativação.
- Se um estado básico é ativado, então o processo de ativação desse estado é finalizado.

A Figura 3-40 exibe o Xchart **X**. Quando uma instância desse Xchart é criada, o estado **X** é ativado. **X** é exclusivo, ou seja, no máximo um de seus subestados deve ser ativado quando ele for ativado. O subestado a ser ativado é necessariamente identificado por uma transição inicial que, nesse caso, conduz ao estado **D**. Contudo, **D** só é ativado quando todos os seus ancestrais forem ativados, conforme as normas fornecidas acima, que estabelecem uma seqüência bem definida de ativações: os estados **X**, **H**, **G**, **F** e **D**, nessa ordem.

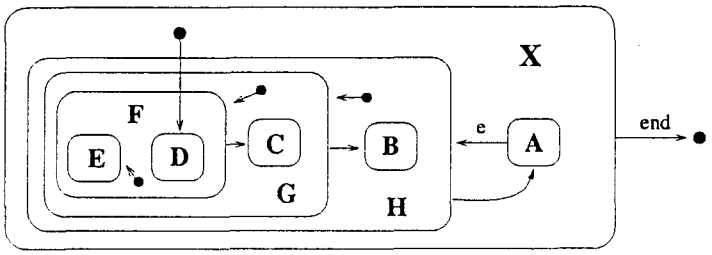


Figura 3-40: Ativação seqüencial de estados

No cenário onde o estado **A** de uma instância de **X** está ativo e um estímulo externo, que provoca a ocorrência do tipo **e**, é tratado, a transição de **A** para **H** é executada. O ACMP dessa transição é o estado **X**. Ou seja, a sua execução provoca a desativação de todos os subestados ativos de **X** seguida da ativação de **H**. O estado **A** é básico, ou seja, desativá-lo não implica a desativação de subestados. Em contrapartida, **H** é um estado exclusivo e pode exigir a ativação de um de seus subestados, conforme transições iniciais. Nesse caso, a ativação de **H** provoca a ativação do estado **G**. O processo repete-se recursivamente para o estado **G** e seus descendentes até que o estado básico **E** seja ativado, o que caracteriza o fim do processo de ativação do estado **H**. Após a ativação do estado **E**, se um estímulo externo que provoca a ocorrência do tipo de evento primitivo **end** é tratado, então a instância é finalizada (o estado raiz é desativado). Desativar **X** significa desativar **E**, **F**, **G**, **H** e, por último, **X**, nesta ordem.

Não é possível identificar uma única seqüência quando pelo menos um estado concorrente está envolvido, pois os seus subestados são executados concorrentemente. Por exemplo, no processo de criação de uma instância do Xchart exibido na Figura 3-41, primeiro o estado **Y** é ativado. Em conseqüência, ambos os subestados **F** e **A** são ativados. Contudo, não se pode afirmar que o subestado **F**, por exemplo, é ativado antes do estado **A** ou vice-versa — ambas as alternativas são igualmente prováveis. Analogamente, não se pode afirmar que o estado **E** é ativado antes do estado **B**. Esse comportamento não contraria as normas estabelecidas anteriormente, que continuam válidas. Por exemplo, o estado **E** é necessariamente ativado após **F** assim como o **B** segue a ativação do **A**.

A desativação do estado **Y** também é não-determinística. Se a transição final é execu-

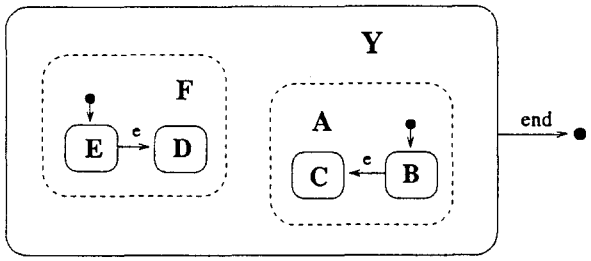


Figura 3-41: Ativação concorrente de estados

tada para uma instância desse Xchart, o estado **Y** é desativado. Essa desativação provoca a desativação dos estados **F** e **A**. Contudo, não se pode estabelecer qual estado que será desativado antes do outro. Por exemplo, se os estados básicos ativos são **E** e **B**, então qualquer uma das seqüências abaixo é igualmente provável e válida para a desativação do estado **Y**:

- **E, F, B, A, Y**
- **E, B, F, A, Y**
- **E, B, A, F, Y**
- **B, A, E, F, Y**
- **B, E, A, F, Y**
- **B, E, F, A, Y**

3.3.17 Transições Excludentes e Prioridade entre Transições

No processo de identificação de um micro-passo não é suficiente identificar as regras habilitadas para serem executadas (Seção 3.9). Duas ou mais transições podem ser mutuamente *excludentes*, ou seja, a execução de uma delas impede a execução das demais. Por exemplo, duas transições que partem de um mesmo estado, conforme Figura 3-42, não podem ser executadas em um mesmo passo. Ou a transição que ativa o estado **X**, ou aquela que ativa **Y** será executada, exclusivamente. A execução de uma delas é incompatível com a execução da outra. O destino de uma transição sempre é ativado e, nesse caso particular, **X** e **Y** jamais estarão ativos simultaneamente. Se em um dado micro-passo essas transições estão habilitadas, então apenas uma delas é selecionada para execução. Nesta seção são apresentados os casos onde transições são ditas excludentes. Prioridade entre transições é um mecanismo empregado especialmente para auxiliar essa seleção.

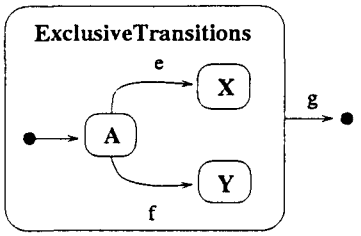


Figura 3-42: Transições excludentes: caso trivial

Os casos de transições excludentes são variações daqueles apresentados na Figura 3-43. Se o estado **A** de uma instância do Xchart **V** está ativo, um estímulo externo que provoca a ocorrência do tipo **e** habilita as duas transições para o estado **B**. Apenas uma delas será executada, pois não é possível desativar o estado **A** duas vezes, ou seja, ou a atividade **a** ou a **b** será executada. Analogamente, se o estado **A** de uma instância de **Y** está ativo e ocorre o tipo **e**, então o estado **T** é desativado por apenas uma das transições habilitadas. O caso apresentado no Xchart **H** é similar, pois a execução das duas transições rotuladas com o tipo de evento **e** exigem a desativação consecutiva de um mesmo estado, o que não é possível.

No Xchart **X** são apresentadas transições iniciais e finais que são excludentes. Ao criar uma instância de **X**, o estado **X** permanece básico. Um estímulo externo que provoca a ocorrência dos tipos **e** e **f** habilita as duas transições iniciais dessa instância. O estado **X** é exclusivo, no máximo um dos seus subestados pode ser ativado. O processo de ativação de um estado ainda envolve a identificação de apenas uma transição inicial a ser executada. Ou seja, uma delas é selecionada, ou o estado **A** ou o estado **B** será ativado exclusivamente. Transições finais também podem ser excludentes, pois implicam na desativação do estado raiz por mais de uma transição, o que não é possível. Ou seja, se a atividade **a** é iniciada, então o estado **X** foi desativado pela transição rotulada com o tipo de evento primitivo **g**. Não seria possível, em um mesmo passo, desativar novamente o estado **X** para que a atividade **b** venha a ser executada.

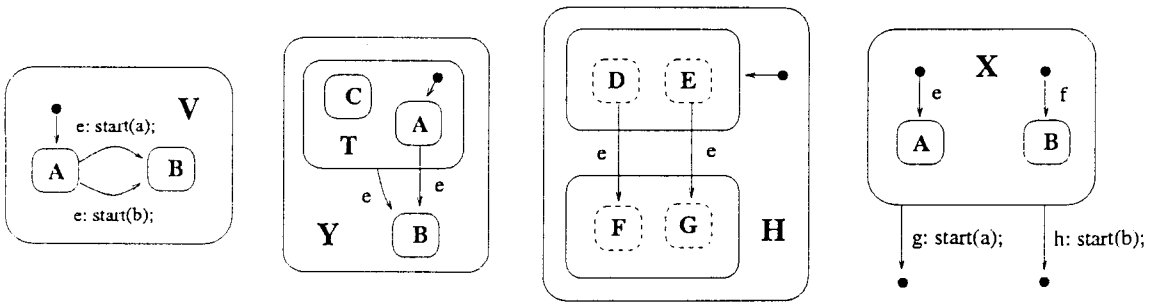


Figura 3-43: Casos de transições excludentes

Um vez identificados os casos de exclusão mútua entre transições, a seção seguinte apresenta o processo de seleção de uma dentre as transições excludentes.

Resolução de Exclusão Mútua entre Transições

A exclusão mútua entre transições é eliminada através da seleção de uma dessas transições. As demais não serão executadas no passo em questão. A exclusão mútua existente em um conjunto de transições é eliminada através da remoção de elementos desse conjunto. Pega-se duas transições quaisquer desse conjunto e se aplica as diretrizes abaixo. A transição selecionada permanece no conjunto, a outra é removida. Esse processo é repetido até que o conjunto se torne unitário. O elemento que restar é aquele selecionado para o conjunto fornecido.

Dadas duas transições excludentes **t1** e **t2**, as seguintes diretrizes são obedecidas, na ordem em que aparecem, para selecionar uma delas:

I. *A de maior prioridade é selecionada.*

Se o nível de prioridade de **t1** é superior ao de **t2**, então **t1** é selecionada — **t1** tem precedência sobre **t2**. Os casos de exclusão mútua entre transições nos Xcharts na Figura 3-43 podem ser resolvidos deterministicamente através da atribuição de níveis de prioridades apropriados às transições. As transições do Xchart **X** na Figura 3-43 possuem o mesmo nível de prioridade cujo valor é zero. Esse Xchart é refinado naquele da Figura 3-44, onde foram estabelecidos níveis de prioridades para as transições. O estado raiz permanece temporariamente básico quando uma instância do novo Xchart **X** é criada. Se um estímulo externo provoca a ocorrência dos tipos de evento **e**, **f**, **g** e **h**, então todas as transições desse Xchart formam um conjunto de transições excludentes. Se elas possuem níveis de prioridade distintos, então aquela de maior nível é selecionada, conforme processo comentado acima. Nesse exemplo, a transição inicial que ativa o estado **B** será a única executada.

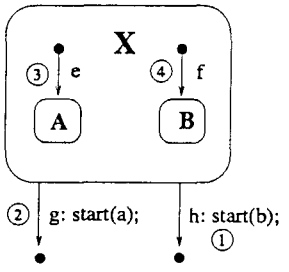


Figura 3-44: Prioridade entre transições

II. *A mais interna tem precedência sobre a mais externa.*

Se os níveis de prioridade de **t1** e **t2** são iguais, então a mais interna é selecionada. Diz-se que uma transição é mais *interna* que outra se o seu ACMP (Seção 3.3.14) é descendente do ACMP da outra. O Xchart **W** é exibido na Figura 3-45. Se o estado **A** de uma instância desse Xchart está ativo e um estímulo externo provoca a ocorrência do tipo de evento **e**, então essas transições se tornam excludentes: uma cujo destino é o estado **B** e outra cujo destino é **C**. Ambas possuem o mesmo nível de prioridade (zero), ou seja, a norma anterior não permite selecionar uma delas. Contudo, o ACMP da transição de **A** para **B** é o estado **AB**, que é descendente do estado **W**. O estado **W** é o ACMP da transição de **AB** para **C**. Em conseqüência, a transição de **A** para **B** é selecionada por ser a mais interna. No Xchart **U** tem-se uma situação análoga. A transição de **A** para **B** é mais interna que aquela de **C** para **D** (ambas possuem o mesmo nível de prioridade). Em um contexto onde essas transições se tornam excludentes, aquela de **A** para **B** é a selecionada.

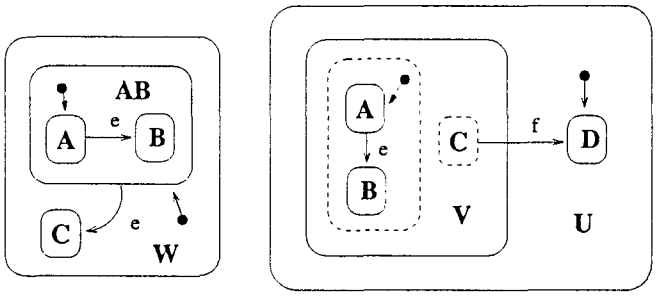


Figura 3-45: Transições excludentes

III. *Uma delas é arbitrariamente selecionada.*

Se nenhuma das normas I e II identifica qual a transição a ser selecionada entre as transições excludentes **t1** e **t2**, então uma delas é arbitrariamente selecionada, ou **t1** ou **t2**. Por exemplo, as transições dos Xcharts **V**, **Y** e **H** (Figura 3-43) são excludentes em instâncias desses Xcharts em contextos simples de serem elaborados. Nesses casos, as transições possuem o mesmo nível de prioridade, ou seja, a norma I não é suficiente para resolver a exclusão mútua assim como a norma II. Em conseqüência, a resolução da exclusão mútua é feita arbitrariamente, qualquer transição desses Xcharts é uma representante legítima da seleção.

A Figura 3-46 exibe o Xchart **X** contendo várias transições sem nenhum nível de prioridade estabelecido para elas. Se o estado **A** de uma instância desse Xchart está ativo e um estímulo externo provoca a ocorrência do tipo **e**, então três transições se tornam excludentes. A norma I não seleciona nenhuma delas. Contudo, a aplicação da norma II seleciona a mais interna, ou seja, aquela que conduz ao estado **B**. Em outro cenário, se o

estado **B** está ativo e o mesmo estímulo externo é sinalizado, então tem-se duas transições excludentes. Aquela para o estado **C** é selecionada, novamente pela aplicação da norma II. Se o estado **C** está ativo, a ocorrência de do tipo **e** habilita apenas a transição final, não há exclusão mútua, essa transição é executada. Por outro lado, se o estado **A** ou **B** está ativo e o estímulo externo a ser tratado provoca a ocorrência do tipo **f**, então nem a norma I e a norma II se aplicam para selecionar qual das transições que irá ativar o estado **D** ou **E**. A norma III arbitrariamente seleciona uma delas.

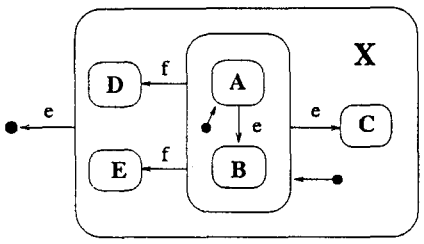


Figura 3-46: Mais exemplos de transições excludentes

3.3.18 História

Um processo de ativação é iniciado quando um estado **s** é ativado (Seção 3.3.16). Esse processo prossegue ativando os subestados de **s**, via transições iniciais, até que estados básicos finalizem o processo. *História* é um mecanismo alternativo para ativação de subestados onde as transições iniciais de **s** não são utilizadas. Por exemplo, na Figura 3-47, se a transição do estado **K** para o **X** é executada em uma instância do Xchart **SimpleHistory**, então a ativação do estado **X** sempre provocará a ativação do subestado **A**, conforme a transição inicial de **X**. Em contrapartida, se a transição de **K** para o símbolo de história **H** é executada, então o subestado de **X** a ser ativado dependerá do passado recente de ativação do estado **X**. As transições iniciais não são consultadas, nesse caso. Ou seja, tanto **A** quanto **B** pode ser o subestado de **X** a ser ativado.

A ativação de um estado por um símbolo de história pode, em circunstâncias especiais, fazer uso das transições iniciais. Por exemplo, se é a primeira vez que o estado **X** é visitado, então a transição inicial é executada, o que provoca a ativação do estado **A**. A transição inicial também é utilizada na primeira vez que o estado **X** é ativado após a execução da ação **clear(H)**.

Há dois tipos de história: *simples* e *estrela*. Se a ativação de um estado **s** se der por um símbolo de história simples, então o subestado de **s** a ser ativado será aquele que esteve ativo quando o estado **s** foi desativado pela última vez. Uma vez identificado esse subestado, o processo de ativação prossegue normalmente consultando as transições iniciais do subestado identificado e assim por diante. Se a ativação de um estado **s** se

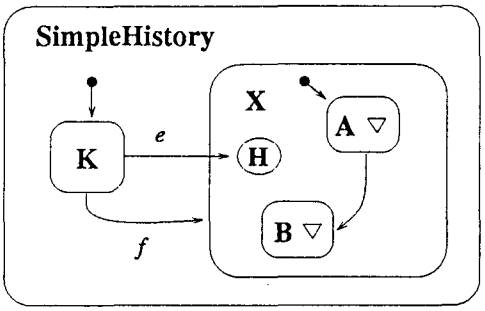


Figura 3-47: História simples **H**

der por um símbolo de história estrela, então o subestado imediato de **s** a ser ativado é identificado e ativado de forma similar ao que ocorre com um símbolo de história simples. Contudo, nesse caso, a ativação do subestado de **s** segue o mesmo processo anterior, ou seja, identifica-se o que esteve mais recentemente ativo e o ativa. Esse processo é recursivamente aplicado aos subestados mais recentemente ativos até que seja finalizado. As transições iniciais não são, salvo exceção, empregadas para ativar nenhum subestado de um estado que é ativado por um símbolo de história simples ou estrela.

Quando o estado **X** do Xchart **SimpleHistory** (Figura 3-47) é ativado pelo símbolo de história simples **H**, por exemplo, ou o estado **A** ou o **B** será ativado. A posterior ativação de um desses subestados prossegue conforme as suas transições iniciais. Em contrapartida, se o estado **X** fosse ativado por um símbolo de história estrela, então o subestado de **A** ou de **B** a ser ativado não seria identificado por uma transição inicial, mas por aquele que esteve mais recentemente ativo. Essa forma de ativação é recursivamente aplicada até que um estado básico finalize o processo.

A Figura 3-48 ilustra as diferenças entre história simples e estrela. A ativação do estado **X**, pela história simples **H**, provoca a ativação do estado **E** ou **F**, conforme o mais recentemente visitado subestado direto de **X**. Em conseqüência, ou o estado básico **B** ou o **D** será finalmente ativado. Se a ativação do estado **X** se der pelo símbolo de história estrela **H***, então a ativação de **X** irá ativar todos os seus subestados mais recentemente ativados. Ou seja, a lista de estados básicos candidatos à ativação inclui **A**, **B**, **C** e **D**, ao contrário do caso anterior, onde a lista está necessariamente restrita aos estados **B** e **D**. A ativação pelo símbolo de história simples decide entre os estados **E** e **F**. A ativação de **E** ou **F** se dá por transição inicial, o que limita as opções possíveis a **B** ou **D**. Se o estado **X** é ativado pela primeira vez ou pela primeira vez após a execução da ação **clear** para estes símbolos de história, então os subestados de **X** são ativados conforme as transições iniciais dos estados a serem ativados.

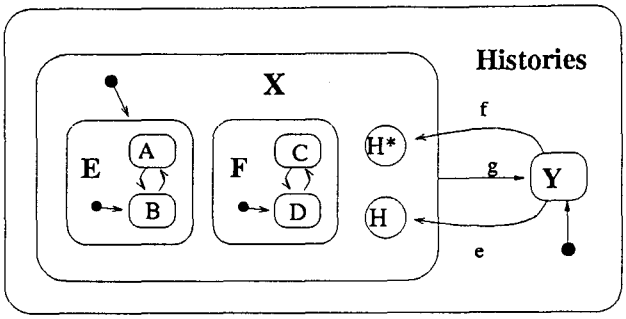


Figura 3-48: História estrela **H*** e simples **H**

3.3.19 Porta

Porta é um repositório onde o controle produzido por uma instância é depositado. A reação de uma instância a um estímulo externo torna-se visível a outras instâncias (subsistema reativo) quando o passo termina. Clientes, contudo, podem consultar parte das reações produzidas, em particular, aquelas que são depositadas em portas. Nesse sentido, uma porta é um meio através do qual a reação produzida por uma instância torna-se disponível para clientes.

Através de uma porta uma instância requer o início de uma atividade **a**, por exemplo, via execução da ação **start(a)**. Uma ação **sync(a)**, por exemplo, ainda interrompe a execução da instância até que o cliente pertinente recupere essa requisição, execute a atividade **a** e indique o fim de execução dessa atividade. Só após essa seqüência de ações a reação da instância prossegue.

Portas permitem classificar as reações produzidas por instâncias. Sem esse conceito todo o resultado produzido pela execução de um especificação em Xchart seria enviado para um único e grande receptor que é o ambiente externo. Portas são intermediários nesse percurso. A reação pode ser dividida entre portas e os clientes podem consultar apenas as portas em que têm interesse. Se a classificação da saída não é de interesse, então apenas uma única porta pode ser o destino de todas as reações produzidas por uma especificação. A identificação da porta onde determinada reação deve ser colocada é feita através de algumas ações, descritas na Seção 3.3.10.

A Figura 3-49 apresenta relacionamentos válidos entre instâncias, portas e clientes. A figura contém **N** instâncias, **K** portas e **J** clientes. Necessariamente **K** é maior ou igual a **N**, pois toda instância possui a sua própria porta padrão. Esta é a única restrição entre os valores de **K**, **N** e **J**. Isso não significa que uma instância obrigatoriamente deposita suas reações na sua porta padrão. Uma instância pode depositar suas reações em várias portas, quaisquer que sejam elas. Em conseqüência, uma porta pode ser o repositório de reações de várias instâncias. Um cliente pode consultar o conteúdo de várias portas ou

de uma única porta específica.

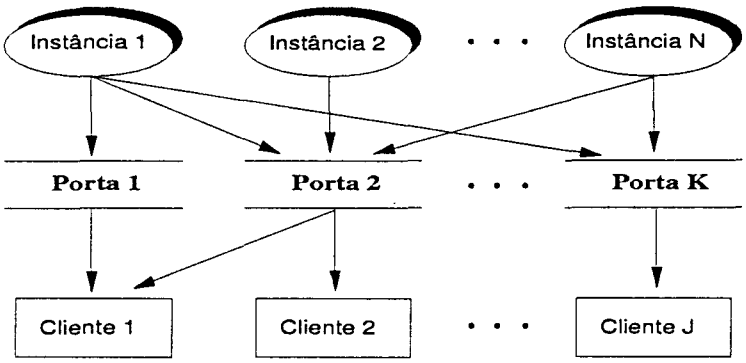


Figura 3-49: Fluxo de dados via portas entre instâncias e clientes.

Clientes distintos podem ser responsáveis pela execução de tarefas distintas. Nesse caso, cada cliente consulta apenas as portas onde são depositadas as atividades pertinentes as suas tarefas. Se nenhuma reação é depositada nas portas de interesse de um cliente, então a consulta do cliente a qualquer uma dessas portas informará que ela está vazia. Caso contrário, a primeira disponível é fornecida ao cliente requisitante.

Quando uma instância é criada a sua porta padrão é automaticamente criada. Quando uma instância é destruída, a sua porta padrão persiste até que não contenha nenhuma entrada. As demais portas (nomeadas) persistem durante toda a execução de uma especificação na linguagem Xchart e só podem ser "criadas" quando um diagrama é editado. Por exemplo, a presença das ações **start(e, p1)** e **start(e, p2)**, em um Xchart, significa que as portas **p1** e **p2** estarão disponíveis durante a execução da especificação pertinente.

3.4 Concorrência

Concorrência é geralmente definida como o compartilhamento de recursos em um intervalo de tempo. Embora esta definição seja compatível com uma implementação de estados concorrentes utilizando um único processador, é conveniente abordar concorrência de forma abstrata, independente da sua implementação. Esta seção fornece a interpretação de concorrência na linguagem Xchart.

A Figura 3-50 exibe o Xchart **E** e um diagrama de transição de estados (DTE). O estado **E** é concorrente, ou seja, os subestados **I** e **H** estarão ativos sempre que o estado **E** estiver ativo. Tanto o estado **I** quanto o estado **H** pode interferir no comportamento do outro, por exemplo, gerando um evento primitivo que irá repercutir no próximo micro-passo. Eles também podem capturar comportamentos que evoluem independentemente um do outro ou a concorrência existente entre atividades. Qualquer que seja o motivo pelo

qual concorrência é empregada ou os benefícios obtidos com o seu emprego, a semântica desse recurso deve ser claramente definida.

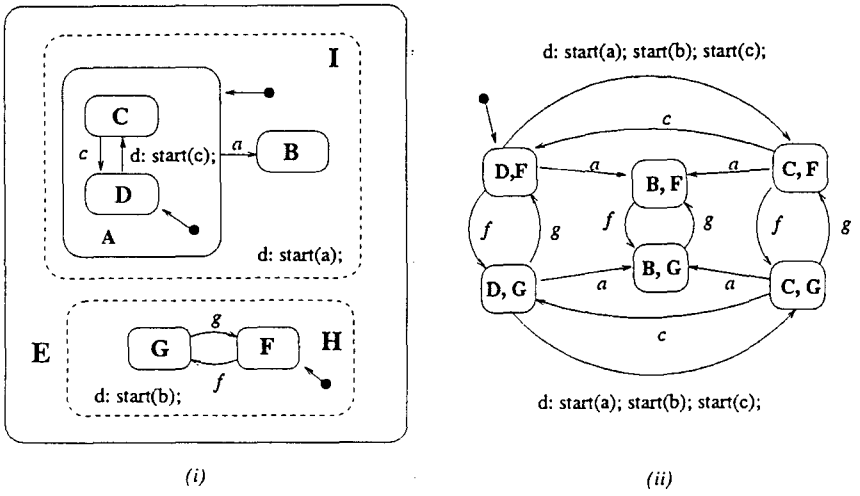


Figura 3-50: Um Xchart e um diagrama de transição de estados

Estados concorrentes tornam possível a execução concorrente de regras e transições. Por exemplo, duas regras pertencentes a subestados distintos de um estado concorrente podem ser executadas concorrentemente. Em particular, todas as transições de um micro-passo podem ser executadas concorrentemente. A execução concorrente de regras de um micro-passo (conjunto finito) é realizada através da execução concorrente de suas ações. Cada uma dessas ações pode ser constituída por uma seqüência finita de elementos de ação, que são atômicos (indivisíveis). Em conseqüência, um número finito de permutações dos elementos de ação dessas ações estabelecem todos os possíveis resultados obtidos pela execução de regras concorrentes. Cada uma dessas permutações pode ser capturada por um DTE. Por exemplo, se os estados **D** e **G** estão ativos em uma instância do Xchart **E**, na Figura 3-50, então a ocorrência do tipo **d** provoca a execução de três regras. Duas associadas a estados e a outra rotulando a transição do estado **D** para o estado **C**. As seqüências possíveis da execução de suas ações são

start(a); start(b); start(c);
start(b); start(a); start(c);
start(a); start(c); start(b);

O DTE captura apenas uma dessas seqüências:

start(a); start(b); start(c)

que rotula a transição de $(\mathbf{D}, \mathbf{G})$ para $(\mathbf{C}, \mathbf{G})$. Novamente, o conjunto de seqüências estabelece todos os possíveis resultados que podem ser obtidos no contexto comentado acima. Nenhum outro resultado pode ser obtido, independentemente da implementação da linguagem Xchart. Nas seqüências, o elemento de ação **start(c)** sempre é precedido pelo elemento **start(a)**, pois as regras, em um conjunto não concorrente de regras e/ou transições, são executadas antes das transições. A instância de **E** é responsável por produzir uma dessas seqüências. A execução de atividades, contudo, não é realizada pelo subsistema reativo. É possível, por exemplo, que a atividade **a** ainda esteja em execução quando a atividade **c** inicia sua execução. Entretanto, a atividade **a** foi iniciada antes da **b**, ou vice-versa, que foram iniciadas antes da **c**. O restante dessa seção comenta dois outros exemplos de concorrência.

A Figura 3-51 exibe o Xchart **Command**. A criação de uma instância do Xchart **Command** leva à ativação dos estados **GetArg** e **GetCmd** e, por último, à ativação dos subestados **WaitA** e **WaitC**. Nesta configuração, a ocorrência do tipo **cmd** provoca a transição do estado **WaitC** para **Cmd**. Analogamente, o tipo **arg** torna ativo o estado **Arg**. Após a ativação de **Arg**, por exemplo, uma nova ocorrência do tipo **arg** faz com que o estado **Arg** seja desativado e imediatamente reativado de forma similar ao que ocorreria com o estado **Cmd** quanto ao tipo **cmd**. Assim que ambos os estados **Arg** e **Cmd** tornarem-se ativos, a ocorrência do tipo de evento **enter** provoca a execução da ação **start(command)**, que não é executada de forma síncrona com a instância. Ou seja, várias atividades **command** podem estar em execução concorrentemente. A execução concorrente de atividades não é da responsabilidade de Xchart, mas o comportamento de **GetArg** e **GetCmd** (estados ortogonais) são regulados pela semântica da Xchart.

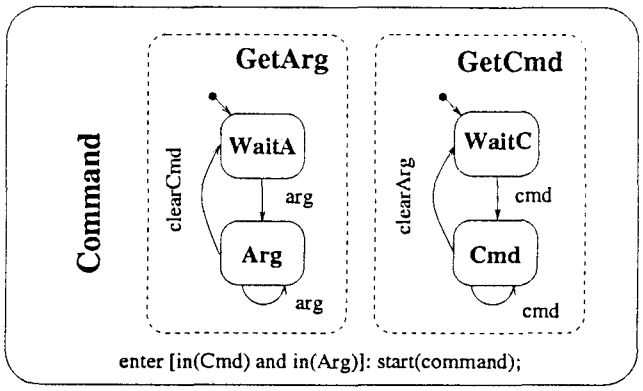


Figura 3-51: Concorrência e comunicação

A Figura 3-52 exibe o Xchart **Concurrency**. A criação de uma instância desse Xchart torna os estados **A** e **C** ativos. A instância criada permanece nessa configuração esperando pela ocorrência de estímulos externos. Se a ocorrência do tipo **e** é provocada por

um estímulo, então as transições de **A** para **B** e de **C** para **D** serão executadas concorrentemente. As possíveis seqüências de ações compatíveis com essa execução concorrente são

sync(a1); **sync(a2)**; ou
sync(a2); **sync(a1)**;

Nesse exemplo particular, ao contrário do Xchart **E** (Figura 3-50), não permite uma execução concorrente das atividades **a1** e **a2**. A execução da primeira seqüência acima, sem perda de generalidade, envolve a execução do elemento de ação **sync(a1)** seguido do elemento **sync(a2)**. O elemento **sync(a1)** só é executado após o término da execução da atividade **a1**. Só após essa condição o elemento **sync(a2)** é considerado para execução, provocando a ativação da atividade **a2**. Em conseqüência, as atividades **a1** e **a2** são executadas uma após a outra, embora suas execuções sejam disparadas por transições executadas concorrentemente. Se a concorrência entre essas atividades é desejável, então o elemento de ação **sync** deve ser substituído por **start**. Elementos de ação são comentados na Seção 3.3.10. Em particular, **atomic** permite tratar como atômica a execução de uma seqüência de elementos de ação.

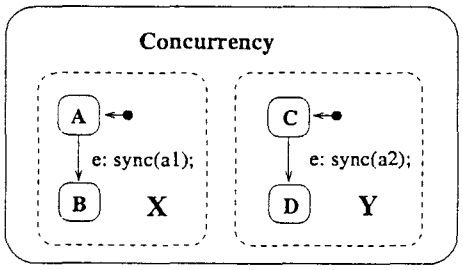


Figura 3-52: Concorrência do ponto de vista de Xchart

3.5 Comunicação e Sincronização

Estados concorrentes e instâncias podem trocar informações entre si e sincronizar suas operações. No exemplo da Figura 3-51, as condições **in(Arg)** e **in(Cmd)** estabelecem uma forma de comunicação e sincronização implícitas entre aqueles estados concorrentes. Comunicação também pode-se dar através de:

- *Variáveis de controle locais.* Na Figura 3-53, após a criação da instância do Xchart **XYZW**, um estímulo externo qualquer irá provocar a execução das transições de **X** para **Y** e de **Z** para **W**. A primeira é conseqüência da execução de **a := 1** quando

o estado **XYZW** é ativado. A segunda é provocada por $\mathbf{b := b + 1}$, cuja execução torna igual os valores de **a** e **b**. Essa igualdade é suficiente para a execução da transição de **Z** para **W** no micro-passo seguinte.

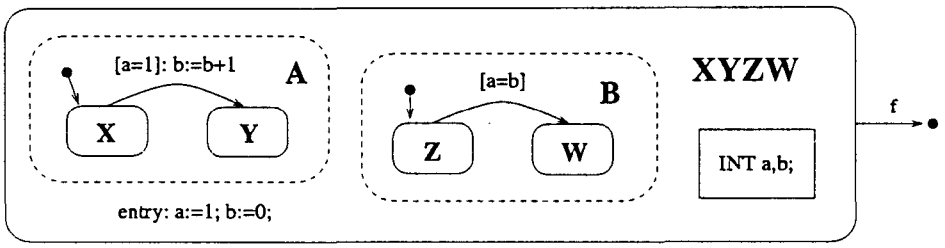


Figura 3-53: Mais um exemplo de concorrência e comunicação

- *Geração de eventos.* A Figura 3-54 ilustra a geração de um evento entre estados e instâncias. Supondo que os estados **C** e **E** estejam ativos, em uma instância de **G** e outra de **H**, a ocorrência do tipo **e** faz com que a regra $\mathbf{e : raise(n, all)}$ seja executada. A execução da ação dessa regra gera uma instância do tipo de evento primitivo **n** para todas as instâncias do subsistema reativo. Esse evento ocasiona, no micro-passo seguinte da instância do Xchart **G**, a transição do estado **C** para o **D**. Ao fim do passo, um estímulo externo é composto e enviado para todas as demais instâncias em execução. A instância de **H** também receberá esse estímulo, que será depositado na fila de estímulos externos da instância. Quando os estímulos externos precedentes forem tratados, se for o caso, o estímulo contendo o evento do tipo **n** será tratado por essa instância. Nessa ocasião, se o estado **E** estiver ativo, então a transição para o estado **F** é executada. Nesse exemplo ainda se observa que um evento do tipo **end** é suficiente para finalizar as instâncias de **G** e **H**.

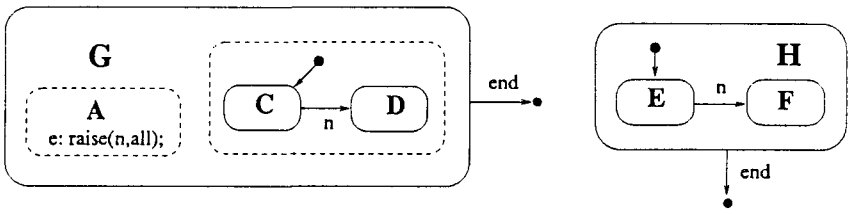


Figura 3-54: Repercussão de um evento entre estados e instâncias

- *Consultando o status de uma instância.* Na Figura 3-51 é apresentado um gatilho dependente da condição **in**, que identifica se um estado está ativo ou não. Uma variante dessa condição permite fazer referência a um estado particular de determinada instância (Seção 3.3.8).

- *Variáveis de controle globais.* A Figura 3-55 mostra um caso similar àquele da Figura 3-53. O Xchart **XYZW** foi transformado nos Xcharts **XY** e **ZW**. Quando uma instância do Xchart **ZW** é criada e, posteriormente, uma do Xchart **XY**, o comportamento obtido é parecido com o daquele comentado anteriormente. Quando um estímulo externo qualquer for sinalizado para a instância de **XY**, a transição de **X** para **Y** é executada e, em conseqüência, a transição de **Z** para **W** na instância de **ZW**. Essa segunda transição consulta o valor compartilhado das variáveis **a** e **b**, que se tornam iguais com a execução da primeira transição. Essa igualdade é condição necessária e suficiente para a execução dessa transição quando o estado **Z** está ativo.

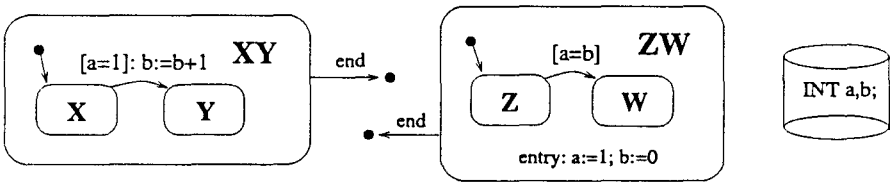


Figura 3-55: Compartilhando variáveis de controle entre instâncias

3.6 Ciclo de Vida de uma Instância

A animação de uma instância pode ser temporariamente "congelada" até que a execução de uma atividade chegue ao fim, independentemente do que ocorre no ambiente externo, que não pode ser controlado. Uma instância ainda pode reagir normalmente aos estímulos externos à medida que são recebidos. Para cada instante da execução de uma instância tem-se um status que caracteriza o comportamento da instância na circunstância em questão. Os status são: *ativo*, *inativo* e *inoperante*.

Quando *ativa*, uma instância de um Xchart é sensível a estímulos externos, que são acumulados e tratados à medida que ocorrem. Quando *inoperante*, uma instância não percebe alterações no ambiente externo, não executa ações (não produz controle). Todos os estímulos externos sinalizados para uma instância inoperante podem ser descartados ou não, conforme implementação de Xchart. Aquele que contém um evento esperado pela instância, entretanto, não é descartado. O estímulo externo que contém o evento esperado "fura" a fila de estímulos externos da instância. O estímulo que contém o evento não é considerado integralmente: apenas o evento esperado é utilizado, os demais elementos do estímulo são descartados. Os estímulos já presentes na fila de uma instância ativa, que se torna inoperante, são mantidos. Quando *inativa* uma instância não mais produzirá saída de qualquer espécie — isto ocorre quando uma instância é finalizada.

Uma instância torna-se inoperante em decorrência da execução dos elementos de ação **call**, **sync** ou **wait** (Seção 3.3.10). Se a ação executada é **wait**, então a instância torna-se ativa novamente apenas quando o evento esperado ocorrer ou quando o intervalo de tempo, eventualmente fornecido, expirar. Se a ação **sync** é executada, então a instância permanece inoperante durante a execução da atividade em questão. A instância que executa a ação **call** torna-se ativa novamente apenas quando a instância criada por esta ação for finalizada.

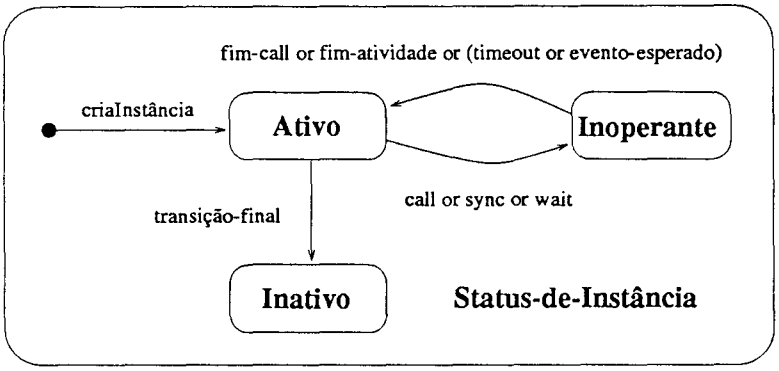


Figura 3-56: Ciclo de vida de uma instância de um Xchart

A Figura 3-56 mostra um Xchart que descreve o ciclo de vida de uma instância em Xchart. Nessa figura, o estado **Ativo** é atingido após a criação da instância, o que habilita a transição inicial. Uma vez ativa, uma transição final pode conduzir a instância para o estado **Inativo**, do qual nenhum outro estado pode ser atingido. Antes que a transição final da instância seja executada, a instância pode executar várias ações. Entre elas podem estar **sync**, **call** e **wait**, que conduzem a instância para o estado **Inoperante**, que é desativado quando a condição de saída desse estado, associada à ação concernente, for satisfeita.

3.7 Interferência de Atividades no Fluxo de Controle

Controle é produzido por uma instância à medida que estímulos externos são consumidos de sua fila, que é alimentada pelo ambiente externo. Atividades também podem gerar estímulos. Um estímulo gerado por uma atividade pode conter os seguintes elementos:

- *Mudança do valor de variáveis globais.*
  A alteração do valor de uma variável global **g** pode, por exemplo, provocar a execução de regras cujo gatilho é **ch(g)**, **tr(g < 0)** e outras.

- *Instância de tipo de evento primitivo.*
  A criação de um evento primitivo é o principal mecanismo utilizado para sinalizar mudanças no ambiente externo para uma instância.

Para que uma atividade sinalize a ocorrência de um estímulo externo é necessário o emprego de um protocolo de comunicação entre uma atividade e o sistema de execução da linguagem Xchart. Esse protocolo (IPX) diz respeito à implementação de Xchart que é comentada no Capítulo 5.

3.8 Organização de Especificações em Xchart

Os recursos que podem ser utilizados na composição de um diagrama descrito na linguagem Xchart foram definidos em seções anteriores. Um diagrama desta natureza é um dos componentes de uma especificação baseada na linguagem Xchart. Esta seção define os componentes de uma especificação em Xchart.

Um sistema especificado em Xchart compreende:

- definição de variáveis globais (Seção 3.3.4).
- definição de eventos (Seção 3.3.7).
- descrição de um ou mais diagramas em Xchart. Cada Xchart possui os seguintes elementos:
  - variáveis locais (Seção 3.3.4).
  - hierarquia de estados (Seção 3.3.2).
  - história (Seção 3.3.18).
  - transições (Seção 3.3.14).
  - regras (Seção 3.3.13).
  - níveis de prioridade (Seção 3.3.17).

A Figura 3-57 descreve uma especificação em Xchart. Essa especificação é formada pelos Xcharts **XchartA**, **XchartB** e **XchartC**, uma hierarquia de eventos e uma variável de controle global **g**. O Xchart **XchartA** define a variável local **l** assim como o Xchart **XchartC** define a variável local **x**.

A semântica formal de Xchart permite que uma especificação como aquela da Figura 3-57 possa ser automaticamente executada. Ou seja, a semântica descrita informalmente nas seções anteriores não precisa ser manualmente implementada. Ferramentas podem

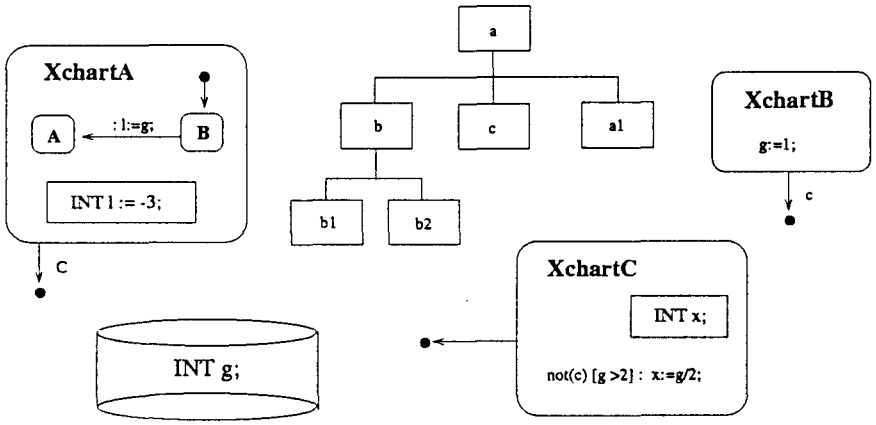


Figura 3-57: Uma especificação em Xchart

automatizar esse processo e prover todos os recursos necessários para dar suporte a tal execução.

A execução de uma especificação em Xchart é iniciada quando uma instância de qualquer um de seus Xcharts é criada. As condições em que tal instância é criada depende, entre outras, do sistema cujo controle foi modelado. Não existe um "Xchart inicial" ou uma instância para cada Xchart de uma especificação quando ela entra em execução. Não existem regras nesse sentido. A execução de uma especificação é finalizada quando há apenas uma instância em execução e ela é finalizada. Em outras palavras, a execução de uma especificação persiste desde o instante da criação da primeira instância até o instante em que não mais existem instâncias da especificação em execução. Por exemplo, se há uma única instância dos Xcharts da especificação da Figura 3-57 e a transição final dessa instância é executada, então a especificação também é finalizada. Se uma nova instância de algum diagrama dessa especificação é criada, então os valores de variáveis globais, por exemplo, serão novamente inicializados, conforme a especificação.

As operações para iniciar e finalizar a execução de uma especificação também são dependentes da implementação da linguagem Xchart. Uma linguagem de configuração pode ser utilizada, por exemplo, para estabelecer quais os nós de uma rede executam instâncias de quais Xcharts, ou ainda quantas instâncias são inicialmente criadas de um dado Xchart quando a especificação pertinente é iniciada. Contudo, os recursos de Xchart devem possuir a semântica apresentada independentemente de uma implementação particular.

3.9 Modelo de Execução

Em seções anteriores foram definidos termos e apresentados, isoladamente, os recursos da linguagem Xchart. A Seção 3.6 descreve o ciclo de vida de uma instância. Esta seção

descreve o modelo de execução de uma instância. São apresentados os mecanismos que regulam a reação de uma instância na presença de um estímulo externo.

Uma instância é criada em um instante **t0** e atinge sua configuração inicial em algum instante posterior **ti**. Desde o instante **ti** até o instante **tf**, no qual uma transição final é executada, uma instância continuamente trata estímulos externos recebidos ou aguarda pela ocorrência deles. Estímulos enviados a uma instância que ainda não foi criada ou após a sua finalização não produzem nenhum efeito. A Figura 3-58 ilustra esses e outros instantes de tempo comuns ao longo da execução de uma instância.

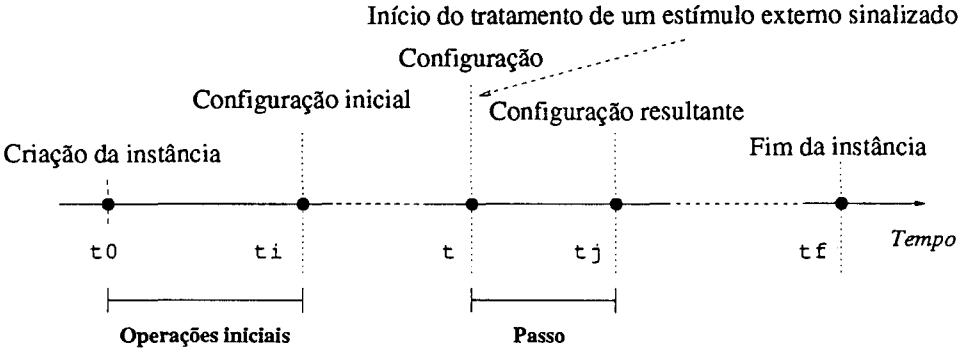


Figura 3-58: Instantes de tempo típicos da execução de uma instância

No instante **t0** a instância é criada. A criação de uma instância compreende a ativação do estado mais externo do Xchart subjacente. A ativação de um estado pode acarretar a execução de ações, conforme a Seção 3.3.16. As conseqüências de ativação do estado mais externo são executadas até o instante **ti**, no qual é obtida a configuração inicial da instância. A partir desse instante, a animação da instância criada é comandada por estímulos externos, que produzirão outras configurações, cada qual dependente do estímulo tratado e da configuração anterior. Por exemplo, no instante **t** é iniciado o tratamento de um estímulo externo sinalizado para a instância em questão. O estímulo é tratado conforme a configuração anteriormente atingida pela instância. Se nenhum estímulo externo ocorre entre **ti** e **t**, então a configuração no instante **t** é a mesma daquela no instante **ti**. O tratamento do estímulo termina no instante **tj**, quando uma nova configuração é obtida. Essa configuração persiste até que outro estímulo externo provoque alguma reação da instância, produzindo uma outra configuração e assim por diante. Um estímulo externo também pode, por exemplo, provocar a execução de uma transição final, que finaliza a execução da instância no instante **tf**.

Uma instância trata um único estímulo externo por vez. Durante a execução do passo entre os instantes **t** e **tj**, por exemplo, a ocorrência de um estímulo externo não interrompe aquele que iniciou o passo. Em outras palavras, após o instante **t** e antes do instante **tj**, todo e qualquer estímulo recebido pela instância é acrescido a sua fila

de estímulos e posteriormente será tratado, conforme a ordem de chegada. Entre estes instantes, contudo, uma instância pode alternar entre os status ativo e inoperante. Em particular, no instante **ti** a instância está ativa e, a partir do instante **tf**, inativa.

Um passo envolve a execução de elementos de ação de efeito local (por exemplo, atribuição de um valor a uma variável local) ou que têm implicações externas (por exemplo, alterar o valor de uma variável global ou gerar um evento para outra instância). Para o passo executado entre os instantes **t** e **tj**, as repercussões locais do estímulo cessam no instante **tj**. Os elementos que repercutem externamente têm efeito apenas a partir do instante **tj**. Para a instância que trata um estímulo externo, todas as suas conseqüências terminam com o fim do passo provocado pelo estímulo. Para o ambiente externo (pág. 43), as conseqüências de um passo são visíveis apenas após o seu término.

A Figura 3-59 divide o passo executado no intervalo **[t, tj]** em vários micro-passos. Nos instantes **t1**, **t2**, **t3**, ..., **tn** são obtidas configurações intermediárias (ou micro-configurações, conforme Capítulo 4). A primeira configuração intermediária de um passo é obtida como conseqüência da execução do primeiro micro-passo do passo. As demais configurações intermediárias são obtidas pela execução dos micro-passos seguintes. Em particular, o último micro-passo identifica a configuração resultante do passo, ou seja, a configuração obtida como reação ao estímulo externo tratado. Essa última configuração é visível externamente. Nessa figura, o termo estímulo externo acumulado refere-se à união do estímulo externo que provocou a execução do passo acrescentado de estímulos internos, isto é, estímulos gerados pela execução de micro-passos anteriores. Estímulos internos são compostos pelos mesmos elementos que compõem estímulos externos.

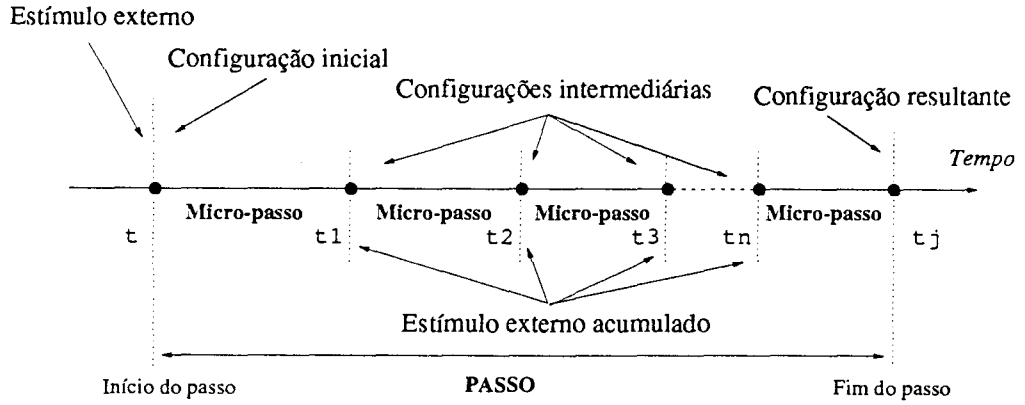


Figura 3-59: Instantes de tempo típicos da execução de um passo

Embora as configurações intermediárias contenham todo o status de uma instância, elas não são visíveis externamente. Elas são visíveis apenas no contexto da instância que executa o passo. Por exemplo, seja **x** uma variável global inteira cujo valor era **0** no início do passo. O início do passo ocorre no instante **t**. A primeira configuração intermediária,

obtida no instante **t1**, pode indicar que o valor de **x** foi alterado para **1** após a execução do primeiro micro-passo. O segundo micro-passo pode consultar o valor de **x** disponível na configuração intermediária, ou seja, o valor **1**. Embora **x** seja uma variável global, o valor **1** não pode ser observado pelo ambiente externo. Apenas o valor final, contido na configuração resultante, pode ser recuperado pelo ambiente externo.

Um micro-passo é parte da execução de um passo e sua realização está dividida em duas fases: identificação e execução, conforme a Figura 3-60. A fase de identificação é responsável por estabelecer o conjunto de regras e/ou transições a serem executadas na fase seguinte. Se uma transição é identificada para execução, então também são identificadas as ações decorrentes de desativações e ativações de estados provocadas pela transição. Sem perda de generalidade, o micro-passo detalhado na Figura 3-60 é aquele entre os instantes **t1** e **t2** na Figura 3-59. No instante **t1** (Figura 3-59), o estímulo externo que provoca a execução do passo pertinente é acumulado aos eventos gerados pelo primeiro micro-passo e, o conjunto resultante, forma o primeiro estímulo externo acumulado do passo e, após o primeiro micro-passo, é obtida a primeira configuração intermediária. Esses elementos são representados na Figura 3-60, respectivamente, pelo estímulo externo acumulado e configuração intermediária inicial. A Figura 3-60 ilustra que durante a fase de identificação de um micro-passo tanto o estímulo externo acumulado quanto a configuração intermediária não são alterados. Esse processo de identificação é finalizado no instante **tu**. Se nenhuma regra e/ou transição pode ser identificada para execução, então **tu** é igual a **t2**, assim como a configuração intermediária inicial torna-se a configuração resultante do micro-passo e do passo. Caso contrário, a partir do instante **tu** tem-se o início da execução do conjunto identificado de regras e/ou transições. O instante **t2** caracteriza o fim da execução dos elementos desse conjunto.

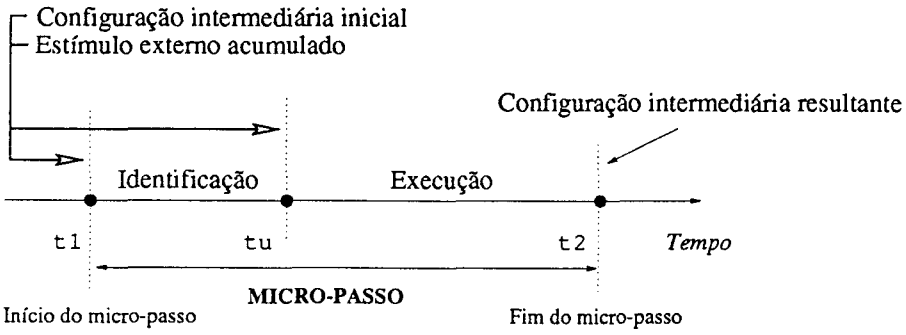


Figura 3-60: Detalhes de um micro-passo

A fase de execução de um micro-passo, ao contrário da fase de identificação, produz alterações na configuração e no estímulo externo acumulado de uma instância. A configuração e o estímulo resultantes, contudo, não são obtidos diretamente após uma execução indivisível das regras e/ou transições identificadas na fase anterior. Os elementos de ação

de uma dada regra, por exemplo, são executados seqüencialmente e a execução de um pode interferir no resultado da execução do seguinte. Por exemplo, para a execução da ação

$$\mathbf{x := 1;\ x := 2;\ x := x + 1;}$$

o valor final de **x** é 3. Após a execução do primeiro elemento o valor de **x** é 1; após o segundo é 2 e o terceiro acrescenta 1 ao valor 2. Em conseqüência, são necessárias as *configurações transientes*, que capturam o status de uma instância entre a execução de elementos de ação. Para a ação exemplificada, após a execução de **x := 1**, o valor da variável **x** não é aquele disponível no início do micro-passo conforme a configuração intermediária inicial, mas o valor **1**, conforme a configuração transiente inicial. Analogamente, após a execução do segundo elemento dessa ação, a segunda configuração transiente informa que o valor de **x** é 2.

As configurações transientes e estímulos externos transientes decorrentes da fase de execução do micro-passo da Figura 3-60 são ilustrados na Figura 3-61. A execução de uma regra significa a execução de sua ação. A execução de uma transição significa a execução das ações decorrentes das desativações de estados, a execução da ação da regra que rotula a transição e a execução das ações provocadas por ativações de estados. Todas essas ações são identificada na fase de identificação. As ações decorrentes de transições distintas são executadas concorrentemente.

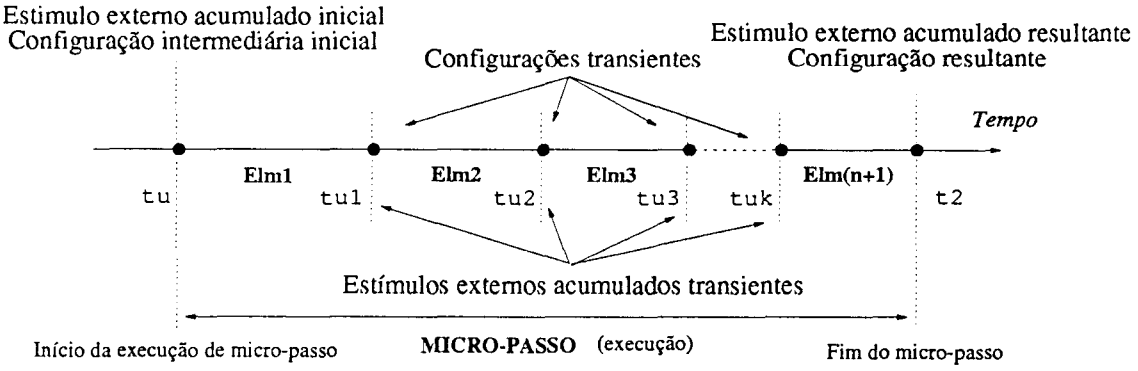


Figura 3-61: Detalhes da etapa de execução de um micro-passo

Nas Figuras 3-59 e 3-60, a execução do passo e dos micro-passos foi considerada finita. As conseqüências de um estímulo externo são finitas no sentido em que o conjunto de regras e/ou transições a serem executadas em sua decorrência é finito. Contudo, a execução de um micro-passo pode durar indefinidamente. Nessa fase alguns elementos de ação podem tornar a instância inoperante. A instância permanece nessa situação até que se torne ativa novamente, quando a execução é retomada. Ainda podem existir construções como

while (1 = 1) { a1; ...; an }

cujas execuções não possuem fim. Detalhes das operações realizadas em um micro-passo e em todo o processo de execução de um passo são fornecidos na seção seguinte.

3.10 Processo de Reação de uma Instância

A reação de uma instância à ocorrência de um estímulo externo é dividida em etapas, comentadas sucintamente na seção anterior. Nesta seção são descritas cada uma dessas etapas e o processo que envolve todas elas. Embora essa seção não contemple casos excepcionais, ela fornece uma visão geral mais detalhada do processo de reação de uma instância.

Se um estímulo externo, composto por um ou mais eventos, é sinalizado para uma instância, então transições e/ou regras podem ser executadas como conseqüência. Se uma transição é executada, possivelmente ocorrem alterações na configuração de estados. Abaixo seguem as operações realizadas por uma instância durante o processo de tratamento de um estímulo externo:

1. Início do passo.
2. Início da etapa de identificação de um micro-passo.
   (a) Identificar regras e/ou transições cujos gatilhos são habilitados com o estímulo externo que está sendo tratado.
   (b) Selecionar, dentre as regras e/ou transições identificadas acima, aquelas que são relevantes.
   (c) Eliminar todas as regras e transições que já foram executadas ou aquelas pertinentes a estados que foram ativados durante o passo em andamento. Isso garante que um passo seja finito.
   (d) Eliminar eventuais casos de exclusão mútua entre transições (Seção 3.3.17).
   (e) As transições restantes podem provocar a desativação de estados que contém regras que também fazem parte do conjunto obtido até o momento. Nesse caso, elimine tais regras.
   (f) Os elementos do conjunto de execução induzem uma seqüência de ações. As regras impõem a ordem definida na pág. 77. As transições prescrevem a ordem apresentada na Seção 3.3.14. Uma ação pertence a esta seqüência se e somente se ela é uma ação de uma regra ou transição que pertence ao conjunto de execução.

(g) Se a seqüência de ações obtida no item anterior é vazia, então a etapa de execução do micro-passo não produz nenhum efeito e o passo corrente é finalizado. Nesse caso, a próxima etapa é o item IV. Caso contrário, inicia-se a etapa de execução de um micro-passo.

3. Início da etapa de execução de um micro-passo.
Cada ação da seqüência identificada acima é executada conforme a ordem imposta pela seqüência. Algumas possuem uma relação de precedência entre elas que é satisfeita pela seqüência. As demais são arbitrariamente posicionadas nessa seqüência. A execução dessa seqüência satisfaz as seguintes observações:

(a) Alguns elementos de ação geram eventos primitivos. Eventos primitivos gerados em um micro-passo são acumulados àqueles já gerados desde o início do passo. O conjunto resultante é utilizado na etapa de identificação do próximo micro-passo.

(b) Se o estado **s** é ativado, então os eventos **en(s)** e **en(s, i)**, onde **i** é a instância pertinente, são gerados. Estes eventos também são acumulados ao longo da execução do micro-passo. Alternativamente, se **s** é desativado, os eventos **ex(s)** e **ex(s, i)** são gerados e acumulados.

(c) Atribuições podem alterar o valor de variáveis. Nesse caso, eventos do tipo **ch(exp)**, **tr(exp)** e outros podem ser gerados. Estes eventos também são acumulados.

(d) A manipulação de atividades também pode gerar eventos. Por exemplo, uma ação **start(a)** provoca a ocorrência do evento **started(a)**.

(e) Os eventos acumulados pela execução das ações são adicionados àqueles disponíveis no início do micro-passo. O conjunto resultante será a entrada do próximo micro-passo. Nesse ponto, repete-se o processo desde o item II.

4. A etapa de identificação do último micro-passo não identifica nenhuma regra ou transição a ser executada, em conseqüência, o passo é finalizado. Nesse ponto, todos os eventos gerados até o presente micro-passo serão sinalizados através de estímulos externos para as instâncias concernentes. Os estímulos externos serão acumulados nas filas de estímulos externos dessas instâncias. Essa operação pode ocorrer em instantes distintos para instâncias distintas. Não há relacionamento síncrono entre instâncias. Após estas operações, o passo é dito finalizado e a instância trata o próximo estímulo externo disponível ou aguarda a ocorrência de um.

3.11 Diferenças entre Xchart e Statechart

A existência de um grande número de variantes de Statechart torna necessário a identificação de características essenciais dessa linguagem para permitir uma comparação entre as variantes. Beeck [7] propõe uma lista de 26 características, que são utilizadas em recente reapresentação da semântica de Statechart [44]. Essa seção também faz uso de tais características que são comentadas abaixo (algumas foram fundidas) com ênfase nas linguagens Xchart e Statechart. A definição delas é minuciosamente coberta em [7]. Tal lista não é suficiente para ressaltar as diferenças entre Xchart e Statechart e, em conseqüência, várias outras características são acrescentadas. Isso facilita o posicionamento de Xchart no espaço de linguagens derivadas de Statechart. As observações, contudo, enfatizam recursos de Xchart e Statechart, sem considerar a aplicação delas nos domínios pertinentes.

A definição informal de Statechart [40] não estabelece claramente a semântica de algumas de suas construções. A Seção 7 de [40] apresenta algumas dessas construções para as quais o próprio texto sugere que a semântica formal deve resolver. A semântica formal de Statechart foi inicialmente apresentada em [45]. Essa semântica esclarece alguns itens e define outros não considerados no texto informal, embora não contemple todos os recursos (Capítulo 4). Uma análise cuidadosa dessa semântica revela diferenças com aquela apresentada em [44]. Por exemplo, os efeitos de um estímulo externo são finitos na primeira, enquanto a última semântica permite reação infinita. Todos os três textos, embora produzidos por Harel e outros, apresentam visões díspares associadas à Statechart. Em particular, [44] foi produzido com a pretensão de eliminar confusões acerca da semântica de Statechart. Mais recentemente, Harel e outros apresentam Statechart (orientada a objetos) [42] com diferenças semânticas que não se encontram em nenhum dos três textos anteriores. Para efeito da comparação abaixo, Statechart é definida conforme [44], exceto quando dito o contrário.

CONSTRUÇÕES GRÁFICAS OU EM FORMA DE TEXTO

Xchart é uma linguagem que faz uso significativo de construções gráficas. Alguns recursos, contudo, são exclusivamente usufruídos através de descrições em forma de texto, por exemplo, regras (Seção 3.3.13). Especificações em Xchart podem ser completamente descritas em forma de texto através da linguagem TeXchart (pág. 173). Em [44], não é feita nenhuma referência a uma versão em forma de texto de Statechart.

DISJUNÇÃO, CONJUNÇÃO E NEGAÇÃO DE EVENTOS

Eventos podem ser compostos com os operadores **or**, **and** e **not** (pág. 61). Statechart apresentada em [42], contudo, não permite tais construções.

EVENTOS E TRANSIÇÕES ASSOCIADOS A TEMPO

Eventos temporais (pág. 60) permitem programar um relógio para gerar eventos em determinados instantes ou após transcorridos intervalos preestabelecidos de tempo. Eventos gerados por relógios formam estímulos externos que são sinalizados para as instâncias pertinentes. Ou seja, podem perdurar em uma fila antes de serem tratados. Statechart, ao contrário, adota a hipótese síncrona (comentada abaixo), que permite que eventos temporais sejam tratados no instante em que ocorrem. Não é claro, contudo, a semântica para um evento temporal gerado durante uma reação infinita.

CONDIÇÕES

Xchart permite um rico conjunto de condições (Seção 3.3.8). Em particular, referências a estados podem ser realizadas além de estados concorrentes. Por exemplo, uma condição pode fazer referência a um estado de um outro Xchart subjacente a uma particular instância. Statechart não é clara a respeito de tais condições, embora permita "múltiplos diagramas".

ATRIBUIÇÃO

Xchart permite o uso de variáveis e atribuições a elas assim como Statechart.

TRANSIÇÕES ENTRE NÍVEIS DISTINTOS

Xchart permite transições que "atravessam as fronteiras" de vários estados, assim como Statechart.

HISTÓRIA

Xchart contempla o emprego de símbolos de história. Em Xchart esse conceito ainda é estendido: vários símbolos de história podem estar associados a um único estado, ao contrário de Statechart. Isso permite que símbolos distintos possam provocar comportamentos distintos quando utilizados. Por exemplo, se os símbolos de história **h1** e **h2** estão associados a um dado estado, então um deles pode estar "limpo" (efeito da execução da ação **clear**) enquanto o outro não. Ou seja, a ativação do estado pertinente por **h1** pode ser diferente da ativação por **h2**, mesmo sendo símbolos de história do mesmo tipo (Seção 3.3.18).

SEMÂNTICA OPERACIONAL, DENOTACIONAL, COMPOSICIONAL

Embora o emprego de Xchart como um modelo possa ser útil o suficiente, uma ferramenta que automaticamente executa descrições em Xchart pode ser ainda mais benéfica. O desenvolvimento de um ambiente de apoio ao emprego de Xchart (Capítulo 5), principalmente o sistema de execução de Xchart, beneficia-se da descrição da semântica operacional dessa linguagem (Capítulo 4). Essa semântica não é composicional e ainda não há uma semântica denotacional para Xchart.

HIPÓTESE SÍNCRONA

A hipótese síncrona prescreve que a reação a um estímulo externo seja produzida em zero unidades de tempo. Ou seja, a reação é instantânea e ocorre no mesmo instante em que o estímulo é recebido. Essa hipótese não é factível e pode conduzir a modelos não intuitivos [69]. Xchart não emprega essa noção. De fato, a duração não nula de um passo é um conceito explicitamente estabelecido em Xchart (Seção 3.9). A não adoção dessa hipótese permite a existência de expressões como aquela na pág. 74, onde é possível verificar, em um dado passo, se um estado foi desativado e posteriormente ativado. Em Statechart, tal expressão não é possível. A não adoção da hipótese síncrona ainda permite um ambiente de projeto e implementação mais realístico, conforme sugerido em [42, pág. 39].

DETERMINISMO

Se duas ou mais transições partindo de um único estado estão habilitadas, tem-se um caso de exclusão mútua entre elas. Essa exclusão mútua pode ser resolvida selecionando-se arbitrariamente uma delas (Seção 3.3.17). Alguns casos de não-determinismo presentes em Statechart e em algumas de suas variantes não ocorrem em Xchart. Por exemplo, [7] cita um exemplo onde uma transição habilitada parte de determinado estado e outra, também habilitada, parte de um ancestral desse estado. Em Xchart, esse cenário não caracteriza não-determinismo: a primeira transição tem precedência sobre a outra e sempre seria executada, exceto se as prioridades dessas transições disserem o contrário.

O não-determinismo pode estar presente em situações mais sutis. Por exemplo, não é prescrita uma ordem para a execução das ações decorrentes da execução de uma transição em Statechart. A hipótese síncrona cria uma concorrência na execução dessas ações. Se uma transição é executada, então as ações induzidas por essa execução são executadas concorrentemente em Statechart. Em Xchart, contudo, se a transição não envolver estados concorrentes, então a execução dessas ações pode ser preestabelecida com precisão (Seção 3.3.15).

INTERLEAVING E CONCORRÊNCIA REAL

Xchart captura concorrência real. As transições de um micro-passo são executadas paralelamente. Essa execução envolve a execução de ações. Algumas ações ou seqüências delas podem ser atômicas (Seção 3.3.10).

TEMPO DISCRETO/CONTÍNUO

A execução de uma instância de um Xchart é marcada pela execução de passos. Entre passos, a execução de uma instância não é visível ao subsistema reativo, embora tal execução ocorra durante uma ou mais unidades de tempo. Estímulos externos ocorridos

no intervalo de execução de um passo são acumulados em uma fila e tratados conforme a ordem de chegada. Eventos temporais são gerados por relógios e encapsulados em estímulos externos.

CONSISTÊNCIA GLOBAL

Xchart emprega consistência local assim como Statechart. Conforme [7], consistência global permite a construção de sentenças menos intuitivas do que aquelas em uma variante que emprega consistência local. A consistência local está intrinsecamente associada a causalidade.

CAUSALIDADE

A semântica é causal se um estímulo externo ocorre, direta ou indiretamente, para cada ação executada. Sem causalidade uma semântica intuitiva não é possível [7]. Xchart e Statechart são causais.

ATIVAÇÃO INSTANTÂNEA DE ESTADO

Em Xchart, se um estado é ativado em decorrência de determinado estímulo externo, então só poderá ser desativado em decorrência de um outro estímulo externo. Ao contrário, Statechart definida em [44] permite que um único estímulo externo provoque a ativação e posterior desativação de um estado. Statechart, conforme definida em [45], apresenta a mesma semântica de Xchart. Um estado, contudo, pode ser desativado e posteriormente ativado. Essa situação é típica de transições cuja origem e destino coincidem ou ainda, em Xchart, quando há uma transição de um dado estado para um de seus ancestrais.

EXECUÇÃO DE UM NÚMERO FINITO DE TRANSIÇÕES EM UM DADO INSTANTE

Embora uma reação não seja instantânea em Xchart, um estímulo externo provoca uma reação finita. Finita no sentido em que o número de regras/transições a serem executados é finito. Micro-passos são finitos e não há como eles se estimularem mutuamente causando uma execução infinita de um passo. Em particular, toda regra e transição é executada, no máximo, uma vez por passo.

Em Statechart [44] pode-se ter uma reação infinita, que seria provocada por um número infinito de passos desencadeados por um dado estímulo externo, ao contrário da definição em [45]. Um passo (elemento de super-passo) em Statechart, contudo, é finito.

PRIORIDADES

Prioridade é um atributo de uma transição. Em casos de não-determinismo, em tempo de execução, esse atributo pode ser utilizado para eliminar o não-determinismo (Seção 3.3.17). Beeck [7] cita trabalhos onde o conjunto de estados de origem de uma transição pode resolver o não-determinismo entre transições de mesma prioridade: quanto mais

"alto" o estado de origem, maior é a prioridade. Essa abordagem, contudo, é imprecisa. Em Xchart, por exemplo, a origem de uma transição pode conter estados em níveis hierárquicos distintos e, nesse caso, não é claro como aquela abordagem pode ser útil. Xchart possui proposta semelhante, mas baseada no conceito de ACMP (Seção 3.3.17).

Statechart também adota prioridade entre instâncias, mas apenas como um segundo recurso, ao contrário de Xchart. O primeiro elemento de resolução de transições excludentes em Statechart é a prioridade estrutural, conceito similar ao de ACMP, mas cujos resultados obtidos são inversos. Statechart ainda não esclarece como é resolvido o caso de exclusão mútua entre transições que empatam através do emprego desses dois recursos.

INTERRUPÇÃO DE MAIOR PRIORIDADE

Esse tipo de interrupção ocorre, conforme definição em [7], entre transições excludentes em Xchart. Dado um conjunto de transições excludentes, apenas uma delas pode ser executada por passo e, em conseqüência, Xchart dá suporte a interrupção *preemptive*. Statechart comporta-se de forma similar.

DISTINÇÃO ENTRE EVENTOS EXTERNOS E INTERNOS

Eventos externos referem-se a estímulos externos. Em Xchart eles são depositados em filas. Durante o tratamento de um estímulo externo, a fila de estímulos de uma instância pode conter outros estímulos que aguardam para serem tratados. Isso protege a execução de um passo sem que estímulos gerados durante um passo sejam perdidos. Eventos internos, em contrapartida, são visíveis no interior de um passo tanto em Xchart quanto em Statechart. Nessa última, contudo, apenas o passo seguinte aquele em que foi gerado percebe os eventos internos gerados no anterior.

EVENTOS LOCAIS/GLOBAIS

Eventos são, em Statechart e na grande maioria das variantes, sentidos em todo o diagrama. Em Xchart, eventos podem ser sinalizados apenas para um estado e seus descendentes, seja em todas as instâncias em execução ou apenas para um subconjunto delas. As instâncias ainda podem ser selecionadas pelo diagrama subjacente. Por exemplo, pode-se gerar um evento apenas para as instâncias de um dado diagrama. A visibilidade de um evento é comentada em detalhes na pág. 66.

EVENTOS DISCRETOS/CONTÍNUOS

Eventos em Statechart são discretos, ou seja, ocorrem em um determinado instante de tempo, antes e após o qual a ocorrência do evento pertinente não se verifica. A hipótese síncrona garante que ele será tratado no instante em que é gerado. Um evento interno, contudo, persiste apenas durante o passo seguinte àquele em que é gerado.

Em Xchart um evento conceitualmente é um acontecimento instantâneo. Contudo, eventos não são sinalizados para instâncias diretamente e no mesmo instante em que são gerados. Eles são encapsulados em estímulos externos, que podem perdurar na fila de estímulos de uma instância antes de serem tratados. O tratamento de um estímulo externo termina apenas no fim da reação em cadeia por ele provocada. Uma reação em cadeia pode se estender por vários micro-passos em Xchart e, necessariamente, dura uma ou mais unidades de tempo. Os eventos que fazem parte de um estímulo são acumulados, ao longo do passo, àqueles gerados nos micro-passos até o fim da reação. O efeito dos eventos gerados em um micro-passo, contudo, são visíveis apenas a partir do micro-passo seguinte, mas até o fim do passo. A Seção 3.9 fornece detalhes.

As observações acima podem ser utilizadas para comparar Xchart com outras variantes de Statecharts, conforme o trabalho realizado por Beeck [7]. Os itens seguintes, entretanto, não foram comentados por Beeck. Eles são necessários para fornecer uma visão mais abrangente das diferenças entre Xchart e Statecharts.

HIERARQUIA DE TIPOS DE EVENTOS

A organização de tipos de eventos primitivos em uma hierarquia (Seção 3.3.5) foi inspirada na variante de Statechart apresentada em [114]. Não são feitas referências a este recurso em [40, 42, 44, 45].

VISIBILIDADE DA REAÇÃO

Em Xchart, uma reação em cadeia iniciada por um estímulo externo provoca alterações que se tornam visíveis ao subsistema reativo apenas ao fim dessa reação. Clientes, contudo, podem recuperar, concorrentemente com a execução de uma instância, o conteúdo de portas (Seção 3.3.19). O texto [44] não é claro acerca do que ocorre entre múltiplos diagramas de Statechart.

CONFIGURAÇÃO CONSULTADA

Durante a execução de um micro-passo, os valores consultados, salvo explicitamente o caso do emprego de **step**, a configuração consultada é aquela do início do micro-passo, possivelmente alterada por ações já executadas. Ou seja, a cada execução de uma ação, uma nova configuração torna-se disponível e passa a ser consultada. Essa nova configuração é denominada de configuração transiente. Detalhes pertinentes são fornecidos na Seção 3.9. Tal abordagem reforça a natureza causal de Xchart. Ao contrário, Statechart limita-se a consultar a configuração disponível no início de um passo (ou micro-passo em Xchart), mesmo que ações anteriores tenham alterado o valor de uma variável, por exemplo. Nesse caso, enquanto a execução de

$$\mathbf{x} := 0;\ \mathbf{y} := \mathbf{x} + 1;$$

deposita o valor **1** na variável **y** em Xchart, a execução delas em Statechart depende do valor inicial de **x** no início do passo. Um efeito similar em Xchart é obtido através de

$$\mathbf{x := 0;\ y := step(x) + 1;}$$

O operador **step** é utilizado para consultar valores disponíveis no início do passo (pág. 74).

MÚLTIPLAS ATRIBUIÇÕES A UMA MESMA VARIÁVEL

Em Xchart, uma ação de uma regra ou transição é uma seqüência a ser executada na ordem fornecida. Em conseqüência, se apenas uma regra ou transição faz referência a uma variável **x** qualquer, então pode-se deterministicamente identificar o valor final dessa variável após a execução da ação correspondente. A ação

$$\mathbf{x := 0;\ x := x + 1;\ x := 2 * x;}$$

por exemplo, após executada, deposita o valor **2** na variável **x**. Em Statechart, contudo, a execução de tal ação tem resultado imprevisível [44, pág. 301]. Em particular, a semântica de Statechart definida em [45] não permite esse tipo de ação. Statechart permite múltiplas atribuições, com resultados previsíveis, desde que pertençam a passos distintos de um superpasso. Não há uma razão clara para não utilizar essa mesma abordagem quando da execução de uma ação composta como ilustrada acima. Xchart, nesse sentido, facilita uma conexão com linguagens de programação atualmente empregadas, onde a sentença acima seria interpretada de forma convencional.

ESTADO TEMPORARIAMENTE BÁSICO

Em Xchart, um estado do tipo exclusivo pode alternar entre básico e exclusivo ao longo da execução da instância pertinente (Seção 3.3.3). Essa abordagem generaliza o emprego de rótulos em transições. Dessa forma, se o gatilho de uma transição inicial não está habilitado, então o destino dessa transição não será ativado no passo em questão. Em Statechart, transições iniciais necessariamente possuem gatilhos nulos. Se uma transição inicial faz parte de uma transição composta que não pode ser completamente executada, então uma situação excepcional ocorre. Conforme [44], mais pesquisas são necessárias para lidar com essa situação em Statechart.

MÚLTIPLOS DIAGRAMAS

Uma especificação em Xchart é formada por um conjunto de diagramas. Em tempo de execução eles servem de molde para a criação de instâncias. Instâncias interagem entre si através de um modelo cuidadosamente elaborado (Seção 3.1). Uma abordagem

simplificada seria tratar diagramas como subestados descendentes de um estado fictício concorrente, conforme é sugerido em [44].

Um modelo que adequadamente contempla múltiplas instâncias envolve a criação de outros elementos necessários ao suporte desse modelo, como ocorre com Xchart. Por exemplo, em Xchart pode-se criar, através de uma ação, uma instância de um Xchart ou mesmo verificar se determinado estado de uma dada instância está ativo ou não. Um evento ainda pode ser sinalizado para um subconjunto das instâncias ativas. A existência de instâncias também influi na semântica de variáveis locais (não compartilhadas) e globais (compartilhadas entre instâncias). Conforme apresentado em [44], múltiplos Statechart não possuem tais recursos assim como carecem de um modelo coeso, o que dá origem a uma semântica **ad hoc**, por exemplo, para definir o comportamento do fim da execução de um diagrama (que na abordagem sugerida é uma espécie de subestado concorrente). Em Statechart não existe o conceito de instância. Dessa forma, se o controle de um artefato de software pode possuir, em tempo de execução, múltiplas instâncias que cooperam entre si, então o modelo oferecido por Xchart é superior.

Em Xchart, ainda é possível execuções "recursivas" de instâncias através da ação **call**. Essa ação só termina com o término (fim) da execução da instância criada por ela (pág. 69).

ORDEM DE EXECUÇÃO DE AÇÕES

A execução de uma transição induz a execução de uma seqüência de ações em Xchart. Se uma transição de **A** para **B** é executada, então as ações de saída de **A**, a ação que rotula a transição e posteriormente a ação de entrada em **B** são executadas nessa ordem. Statechart, em contrapartida, não estabelece nenhuma ordem. Uma seqüência que primeiro trata as ações de ativação de **B**, depois a ação que rotula a transição e posteriormente aquelas de **A**, ou ainda uma execução concorrente dessas ações não ocorre em Xchart.

EXECUÇÃO ATÔMICA DE AÇÕES

A modificador **atomic** permite que um subconjunto de ações de uma transição, por exemplo, seja executado de forma indivisível. Isso permite estabelecer o valor final da execução dessas ações mesmo na presença de concorrência (pág. 71). Em Statechart todas as ações são executadas em zero unidades de tempo e, dessa forma, não existe a noção de seqüência entre elas.

REGRAS

Xchart acrescenta o conceito de regras (Seção 3.3.13). Em [44] é introduzido o conceito de reações estáticas, que se assemelham sintaticamente às regras de Xchart. As

reações estáticas, contudo, não podem ser especificadas para um estado concorrente, ao contrário das regras em Xchart. A regra (Xchart) ou reação estática (Statechart)

$$: x := x + 1;$$

é executada uma única vez por passo (Xchart). O algoritmo na pág. 317 de [44] sugere que essa reação estática seja executada um número infinito de vezes, segundo Statechart.

REPRESENTAÇÃO DE ESTADOS CONCORRENTES

Xchart, ao contrário de Statechart, usa estados com contornos tracejados para designar subestados de um estado concorrente (Seção 3.3.3). Isso permite a descrição de regras no interior de estados concorrentes. Em Statechart, não é afirmado como reações estáticas são associadas a estados concorrentes.

VARIÁVEIS GLOBAIS E LOCAIS

Statechart e algumas de suas variantes fazem uso exclusivamente de variáveis globais, ou seja, podem ser utilizadas por todo o diagrama [7]. Em Xchart, contudo, variáveis podem ser locais e globais. Variáveis locais em Xchart tem efeito apenas nas instâncias geradas a partir do diagrama em questão. Variáveis globais em Xchart são aquelas compartilhadas entre todos os diagramas e, em conseqüência, por todas as instâncias desses diagramas. Há uma cópia única para essas variáveis compartilhada entre as instâncias. A Seção 3.3.4 fornece detalhes sobre variáveis.

TRANSIÇÕES PSEUDO-FINAIS

Além dos tipos de transições existentes em Statechart e comuns em algumas de suas variantes, Xchart acrescenta transições pseudo-finais e transições entre um estado e seu ancestral/descendente. Nos dois últimos casos as transições são ditas externas e internas, respectivamente (Seção 3.3.14). Nem Statechart e suas variantes conhecidas apresentam transições pseudo-finais. O texto [40] sugere a existência de transições internas, conforme exemplos apresentados. Em [45], transições internas e externas não são comentadas. Uma observação cuidadosa revela, contudo, que a definição formal de ACMP, naquele texto, não contempla tais transições.

MODELO DE IMPLEMENTAÇÃO

O modelo de implementação define como descrições em determinada linguagem, por exemplo, Xchart, transformam-se em código. O modelo de implementação é o meio através do qual descrições em Xchart ou Statechart tornam-se úteis durante a fase de implementação do controle capturado por tais linguagens. O modelo de implementação adotado por BETTERSTATE [2] é comentado em detalhes na Seção 5.4.

Os mecanismos utilizados para integração entre o "código" produzido via Statechart e os demais componentes de um sistema é pouco explorado em [40, 44]. Não é claro como código responsável por outras funções de um sistema interagem com o código automaticamente gerado a partir de descrições em Statechart. Xchart é uma proposta para o desenvolvimento de gerenciadores de diálogo: descrição e implementação. Os mecanismos empregados na descrição foram descritos nesse capítulo. Os capítulos seguintes, exceto o último, estão relacionados à implementação de Xchart, desde o apoio fornecido por ferramentas à conexão com o restante de um sistema (clientes) e exemplos.

Em [42], contudo, são sugeridos elementos pertinentes ao modelo de implementação da variante de Statechart comentada naquele trabalho. Tais elementos são comentados abaixo:

Condições e ações podem ser escritas diretamente na linguagem de implementação a ser utilizada, além das condições e ações oferecidas por Statechart (por exemplo, a condição **in(s)**). Essa abordagem não é recente. É a mesma empregada por BETTERSTATE [29], STATEMASTER [138], ESTEREL [9] e experimentada em [81]. A proposta atualmente empregada por Xchart evoluiu daquela apresentada em [81]. Subsistema reativo, clientes e portas são conceitos que existem no modelo de implementação de Xchart (Seção 3.1). Esse subsistema não é executado no mesmo espaço de endereçamento de clientes. Uma condição ou ação em Xchart empregam apenas os recursos fornecidos por Xchart, sem contar com o apoio de sentenças em linguagens como C ou C++. Dessa forma, recursos fornecidos por Xchart e linguagens de implementação não se confundem nem um exerce influência sobre o outro. Por exemplo, uma função em C executada em uma ação, pode não retornar, o que comprometeria o progresso da reação da instância.

Instâncias de diagramas são conceitos desconhecidos em [40, 44]. No contexto de desenvolvimento de interfaces, [85] já havia sugerido o emprego de um modelo onde unidades distribuídas (Xcharts) cooperariam na realização de suas funções. Em Xchart, instâncias não estão associadas a elementos de linguagens de programação como classes ou objetos em C++. Uma instância em Xchart é um diagrama em execução cuja saída é depositada em portas. Um objeto em C++ pode fazer referências exclusivas a uma determinada porta, assim como ela pode ser compartilhada entre vários módulos de um sistema desenvolvido em C. Em [42], diagramas em Statechart são associados a classes. Em tempo de execução, *objetos* desses diagramas trocam eventos ou invocam operações (chamada de métodos). Por exemplo, a sentença

```
<server> -> gen( <eventname> ( <parameters> ))
```

em um objeto Statechart é equivalente a uma ação **raise** (pág. 65). Parâmetros po-

dem ser fornecidos através de dados acoplados a eventos primitivos (Seção 3.3.6). A invocação de operações pode ser vista como uma atividade sinalizada para uma determinada porta. Em Xchart, contudo, pode-se ter chamadas assíncronas ou síncronas, enquanto em [42] tais operações são executadas de forma síncrona.

Transições são executadas "atomicamente". Statechart [42] emprega um modelo de execução de transições similar ao utilizado em [85]. Uma transição envolve a execução de várias ações. Enquanto a última dessas ações não for executada, nada mais poderá ser executado, pois a instância é "congelada" nesse intervalo. A versão adotada em Xchart é menos restritiva. A execução de uma ação **sync** permite que uma atividade seja executada, da mesma forma que o código fornecido em C++ junto com outras ações em Statechart. Se uma execução assíncrona é desejada, contudo, basta substituir a ação **sync** pela ação **start** (pág. 68). Nesse último caso, instâncias em Xchart são executadas concorrentemente com os demais componentes do sistema em questão.

Capítulo 4

Semântica Formal de Xchart

Many methodologies fail to rigorously define the semantics of the languages. Without a rigorous semantic definition, precise model behavior over time is not well defined and full executability and automatic code synthesis is impossible.
Harel e Gery [42]

It seems clear that formal methods are necessary for transforming software engineering into a discipline that is as well understood and well organized as other engineering disciplines, which rely on sound and well-tested mathematical models.
Luqi e Goguen [86, pág. 84]

A semântica operacional de uma linguagem é imprescindível para a execução de especificações nessa linguagem, principalmente para linguagens cujas sentenças podem ser combinadas em descrições sutis. Xchart foi informalmente definida no capítulo anterior onde, por clareza, casos excepcionais e outros não-triviais não foram tratados. Exemplos simples e específicos foram utilizados para ilustrar o comportamento de uma instância. Esse capítulo apresenta formalmente a semântica operacional de Xchart para o caso geral, que inclui questões ainda em aberto sobre a muito estudada semântica de Statechart. O sistema de execução de Xchart, no capítulo seguinte, descreve uma implementação da semântica aqui fornecida.

Há um grande interesse por questões semânticas de Statechart (Seção 4.1). Propostas alternativas para algumas delas são oferecidas no presente capítulo. Em particular, causalidade durante a execução de uma seqüência de ações resulta em uma semântica ainda mais desafiadora do que aquelas propostas para Statechart. Os recursos acrescentados e as diferenças entre Xchart e Statechart não são reapresentados aqui, eles foram abordados no capítulo anterior (Seção 3.11).

4.1 Considerações Iniciais

Os engenhosos mecanismos de Statechart tornam a definição semântica dessa linguagem uma tarefa não-trivial e suscetível a interpretações errôneas. O trabalho em [44], por exemplo, é parcialmente motivado por confusões acerca da semântica de Statechart. Glinz [33], por exemplo, atribui um comportamento a um exemplo (Statechart D, Figura 1b) de Statechart que é incompatível com as definições em [45] e mais recentemente [44]. Uma das primeiras propostas para a semântica formal de *Statechart* foi apresentada em [40]. Tal proposta não inclui definições de mecanismos importantes de Statechart, por exemplo, *history-star* e função de configuração. Algumas foram incluídas na versão mais recente [44]. 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 [7, 24, 58, 57, 69, 64, 108] 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 de controle de diálogo, onde especificações são manualmente convertidas em código em etapas posteriores de desenvolvimento. A proposta aqui apresentada inclui a especificação e a implementação de gerenciadores de diálogo a partir de descrições em Xchart. O emprego da semântica operacional é explicada pelo interesse na implementação de Xchart, que de fato ocorre, conforme o capítulo seguinte.

Entre as dificuldades, por exemplo, o emprego de consistência local torna a semântica mais difícil de ser definida [7]. Xchart vai além de consistência local: dado um conjunto de ações a serem executadas, um subconjunto de tal conjunto 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 esse grau de causalidade. Em outras palavras, 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 externo, micro-configurações que são resultantes da execução de micro-passos e micro-configurações transientes, obtidas entre a execução de ações.

A semântica de uma linguagem descreve o significado de uma especificação escrita nessa linguagem. Diz-se que uma semântica é operacional quando esta define *como* uma especificação atinge seus objetivos. Em geral, a semântica de uma linguagem é apresentada informalmente em linguagem natural. Por exemplo, pode-se interpretar a expressão a := b, comum em uma linguagem de programação convencional, como a substituição

do valor da variável **a** pelo valor da variável **b**. Essa definição é suficiente em alguns casos, no entanto, possui inconvenientes bem conhecidos que podem ser evitados com uma definição formal. A interpretação semântica de uma linguagem formal é especificada através da definição de mapeamentos entre as construções sintáticas da linguagem e os seus "significados". Por exemplo, os numerais cardinais são geralmente interpretados pelo mapeamento de cada numeral na quantidade por ele denotada, assim o numeral **10** (dez) é interpretado como o inteiro 10, que denota uma dezena.

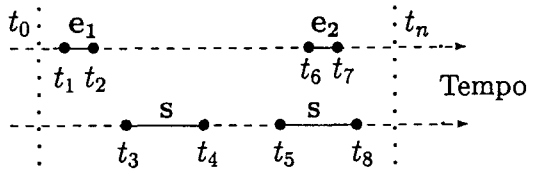
Uma visão geral de alguns conceitos da semântica de Xchart são fornecidos na seção seguinte. A sintaxe abstrata de Xchart é apresentada na Seção 4.3. Funções úteis à definição da semântica são definidas na Seção 4.4. A semântica operacional é apresentada na Seção 4.5. Para aqueles não familiarizados com o formalismo empregado, sugere-se [141] para detalhes. No fim do capítulo, pág. 165, são fornecidas tabelas contendo uma descrição sucinta dos símbolos e notação empregados além de facilitar a localização da definição de cada um desses elementos no presente capítulo.

4.2 Visão Geral

A descrição informal de Xchart (capítulo anterior) elucida comportamentos típicos de instâncias de Xcharts e serve de introdução ao conteúdo do capítulo corrente. A Seção 3.9, por exemplo, fornece conceitos importantes que são formalmente reapresentados aqui. Alguns deles são abordados abaixo.

Em Xchart um estado permanece ativo durante um intervalo de tempo, tem duração, persiste ativo ao longo do tempo por um período não nulo, por menor que seja. Convencionalmente, um evento ocorre em um instante particular do tempo, antes e após o qual esta ocorrência não se verifica. Contudo, eventos são capturados em Xcharts por estímulos externos, cujas existências consomem uma ou mais unidades de tempo. O tratamento de um estímulo externo também não é instantâneo.

Na figura seguinte são apresentados dois eixos. O superior registra os intervalos nos quais dois estímulos externos, $\mathbf{e_1}$ e $\mathbf{e_2}$, são tratados. Ou seja, os intervalos correspondentes às reações em cadeia provocadas por eles. No intervalo $[t_1, t_2]$ o estímulo $\mathbf{e_1}$ é tratado, enquanto o estímulo $\mathbf{e_2}$ é tratado no intervalo $[t_6, t_7]$. O inferior identifica os intervalos de tempo nos quais o estado **s** permanece ativo. Os eixos estão delimitados pelos instantes t_0 e t_n, que correspondem, respectivamente, ao instante inicial e final da execução de uma instância $\mathcal{I}$ de um Xchart. Os instantes de tempo $t_0, t_1, \ldots, t_8, t_n$ são tais que t_i precede t_j para $i < j$ e $n > 8$.



Nos intervalos $[t_0, t_1)$, (t_2, t_6) e $(t_7, t_n]$ da execução de $\mathcal{I}$, nenhum estímulo externo é mostrado, embora outros estímulos sejam necessários para que o estado s seja ativado nos instantes t_3 e t_5 e desativado nos instantes t_4 e t_8. Para $\mathcal{I}$, necessariamente o ambiente externo sinalizou primeiro a ocorrência de $\mathbf{e_1}$ e, posteriormente, a ocorrência de $\mathbf{e_2}$. Conforme o eixo inferior, o estado s permanece ativo apenas nos intervalos $[t_3, t_4]$ e $[t_5, t_8]$. Ou seja, $\mathbf{e_1}$ não provocou a ativação do estado s assim como $\mathbf{e_2}$ não provocou a desativação e posterior ativação desse estado. Em Xchart, nenhuma reação é instantânea, ou seja, se um estado é ativado, então ele permanece ativo por um período não nulo de tempo. Alternativamente, se um estado é desativado, então ele assim permanece por um período não nulo. Em ambos os casos, tais intervalos adicionais apareceriam no diagrama acima.

Se uma seqüência de estímulos externos $ex_0, ex_1, \ldots, ex_n$ é sinalizada para $\mathcal{I}$ nos instantes $t_0, t_1, \ldots, t_n$, onde o estímulo ex_i é sinalizado no instante t_i para $0 \leqslant i \leqslant n$, então $\mathcal{I}$ irá tratar tais estímulos na ordem de sinalização. Ou seja, primeiro será tratado o estímulo ex_0, cujo tratamento é finalizado no instante t'_0, posteriormente as conseqüências do estímulo ex_1 são obtidas no instante t'_1 e assim sucessivamente. O estímulo ex_n será tratado no instante t'_n, que seguramente é posterior ao instante t_n. Em Xchart, t'_i é um instante posterior ao instante t_i, ou seja, $t'_i - t_i > 0$.

Essa diferença superior a zero significa que Xchart não adota a hipótese síncrona, segundo a qual *entradas e saídas são produzidas em um mesmo instante de tempo.* As ações **call**, **sync** e **wait**, por exemplo, "interrompem" a execução de uma instância por um período que independe da instância que executa tais ações. A execução de ações de iteração (**for**, **while** e **do**) pode durar indefinidamente. Essas ações tornam a hipótese síncrona incompatível com a semântica de Xchart.

Convém ressaltar que reação instantânea não é factível e, de fato, qualquer instrução executada por um computador consome um período não nulo de tempo, por mais simples que seja a instrução e por mais eficiente que seja o computador. Em [9] é citado o emprego desta abstração (reação instantânea) em ESTEREL, assim como Statechart [40] e na linguagem reativa síncrona RS [131]. Em [57] é mostrado como esta abstração pode ser interpretada durante a execução de linguagens que a empregam. Embora tal abstração possa ser mantida, ela não pode ser implementada na prática. Em Xchart tal abstração não é empregada e, em conseqüência, não apresenta incompatibilidade entre a definição da sua semântica e a implementação correspondente.

Na Figura 4-1 é exibido o Xchart **S**. Este Xchart está subdividido em 3 subestados. Sejam **S1**, **S2** e **S3** os estados ativos no instante de tempo em que um estímulo externo

contendo apenas uma instância do tipo de evento **a1** é tratado. A transição para o estado **T1** será disparada gerando o evento **a2**, que provoca a execução da transição do estado **S2** para o estado **T2** que, por sua vez, provoca a execução da transição que ativa o estado **T3**. Nesse processo, desativações e ativações de estados e a execução de ações não são interrompidas. A reação em cadeia é preservada, por mais longa que seja e perdure. Essa reação é denominada de passo, enquanto cada transição, nesse exemplo particular, é executada em um micro-passo distinto.

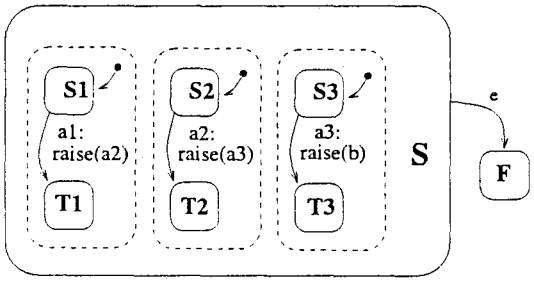


Figura 4-1: Reação em cadeia

A execução de uma instância de um Xchart é marcada por passos. Cada passo é dividido em uma seqüência de micro-passos. Nesta seqüência, cada micro-passo é conseqüência imediata da execução do micro-passo anterior. Tal causalidade é válida mesmo para a execução de uma única regra. Por exemplo, na Figura 4-2, se o estado **W** está ativo, então a transição de **W** para **Z** é executada na presença de qualquer estímulo externo que não contenha o evento **e** (essa condição é necessária e suficiente). A ação `raise(e)` é executada após a avaliação do gatilho e, portanto, não pode interferir ou anular o gatilho. Nesse caso particular, a execução da regra gera um evento cuja ocorrência é condição suficiente para não executar a regra.

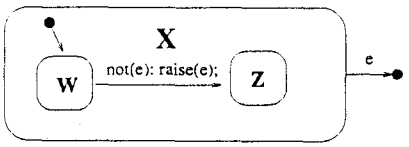


Figura 4-2: Causalidade

4.3 Sintaxe Abstrata

A sintaxe concreta está relacionada à gramática de uma linguagem, ou seja, às normas de obtenção de estruturas válidas nesta linguagem. Na especificação da sintaxe como

o domínio da interpretação semântica é conveniente evitar complicações semânticas irrelevantes como precedência de operadores. Em contrapartida, a sintaxe abstrata não é própria para a análise sintática. Contudo, é simples e suficiente para a definição da semântica.

A Seção 4.3.1 descreve os componentes de uma especificação na linguagem Xchart. Cada especificação pode compreender um ou mais diagramas Xcharts, que são definidos na Seção 4.3.2. Alguns dos conjuntos utilizados na definição da sintaxe abstrata de Xchart seguem abaixo:

- $\mathcal{N}$ de **numerais**. Representa o conjunto dos inteiros $\mathbb{Z}$. A correspondência entre os elementos do $\mathcal{N}$ e aqueles do $\mathbb{Z}$ é trivial. Por exemplo, $-\mathbf{2}$ é correspondente ao inteiro -2, $\mathbf{21}$ ao 21 e assim por diante.

$$\mathcal{N} := \{\ldots, -\mathbf{2}, -\mathbf{1}, \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \ldots\}$$

- OP_L de **operadores lógicos**. Representa as relações lógicas de igualdade, desigualdade, maior, menor, maior ou igual e menor ou igual:

$$OP_L := \{=, \neq, >, <, \geqslant, \leqslant\}$$

- OP_N de **operadores numéricos**. Representa as operações matématicas de soma, subtração, divisão e multiplicação:

$$OP_N := \{+, -, /, *\}$$

4.3.1 Sistema Descrito em Xchart

Um Sistema Descrito em Xchart ($\mathfrak{SDX}$) compreende uma ou mais descrições de Xcharts além de elementos compartilhados entre estas descrições:

$$\mathfrak{SDX} := \langle C_E, \varrho, A_T, P, V_L, V_N, D, I \rangle$$

onde

1. C_E de **classes de eventos**. Representa o conjunto de classes de eventos primitivos. Uma classe ou tipo de evento primitivo é um substrato a partir do qual instâncias de eventos primitivos são criadas.

2. **Função hierarquia** $\varrho : C_E \rightarrow 2^{C_E}$
   A função injetora[1] ϱ define para cada classe de eventos as suas classes derivadas. Seja $c_e \in C_E$ tal que

$$\varrho(c_e) = \{c_e^1, c_e^2, \ldots, c_e^n\},\ 1 \leq n \leq 2^{|C_E|}$$

[1]Uma função $f : A \rightarrow B$ é injetora se e somente se $f(a_1) \neq f(a_2)$ para quaisquer $a_1, a_2 \in A$ e $a_1 \neq a_2$.

Diz-se que c_e^i, $1 \leqslant i \leqslant n$, é uma classe derivada diretamente de c_e. Se $\varrho(c_e) = \phi$ então c_e não possui descendentes.

Fechos ϱ^+ e ϱ^*

$$\varrho^+(c) := \bigcup_{i>0} \varrho^i(c) \quad \text{e} \quad \varrho^*(c) := \bigcup_{i\geqslant 0} \varrho^i(c),$$

onde $\forall\, c \in C_E$, $\varrho(c) := \varrho^1(c)$, $\varrho^0(c) := \{c\}$,

$$\varrho^i(c) := \bigcup_{c' \in \varrho(c)} \varrho^{i-1}(c') \quad \text{para} \quad i > 1 \text{ e}$$

$$\forall\, j \geqslant k,\ \varrho^k(c) = \phi \Rightarrow \varrho^j(c) := \phi$$

3. A_T de **atividades**. Representa o conjunto de recursos funcionais do sistema cujo controle é modelado em Xchart.

4. P de **portas**. Representa o conjunto de destinos para as quais reações ou manipulações (início, fim, suspensão e outras) de atividades são sinalizadas.

5. V_L de **variáveis lógicas**. Cada elemento deste conjunto assume o valor lógico verdadeiro (**T**) ou falso (**F**). Esse conjunto inclui as variáveis globais, não associadas a um diagrama em particular. Variáveis globais são compartilhadas por todas as instâncias em execução.

6. V_N de **variáveis numéricas**. Cada elemento deste conjunto representa um repositório de um valor numérico, que é qualquer valor em $\mathbb{Z}$. (Naturalmente, uma implementação pode restringir esse domínio aos inteiros que podem ser representados com 16 bits, por exemplo. Esta e outras considerações pertinentes são evitadas neste texto.) Esse conjunto, à semelhança do anterior, inclui as variáveis globais numéricas.

7. D de **diagramas em Xcharts**. Cada elemento deste conjunto finito não vazio designa um diagrama em Xchart ou simplesmente Xchart.

$$D := \{d_1, d_2, \ldots, d_n\},\ n \geqslant 1$$

onde d_i é a descrição de um diagrama em Xchart para $1 \leqslant i \leqslant n$.

8. I de **identificadores de instâncias de Xcharts**. Cada elemento desse conjunto identifica uma única instância de um Xchart em D. Toda e qualquer instância pode ser identificada por um único elemento de I.

4.3.2 Diagrama em Xchart

Um $\mathfrak{SDX}$ é composto por um ou mais Xcharts que definem o conjunto D finito e não vazio. Um Xchart $d \in D$ é definido por vários conjuntos e funções:

$$d := \langle S, \rho, \psi_E, \Phi, \omega, H, \gamma, T, R, \Lambda, \mho \rangle$$

onde

1. S de **estados**. Conjunto finito não vazio que designa os estados convencionais, retângulos com cantos arredondados, que compõem um diagrama. Estados fictícios estão contidos no conjunto Φ (item 5).

2. **Função hierarquia** $\rho : S \to 2^S$
   Função injetora que estabelece os *subestados* de um estado. Seja $s \in S$ tal que

   $$\rho(s) = \{s_1, s_2, \ldots, s_n\},\ 1 \leq n \leq 2^{|S|}$$

   Diz-se que s_i, $1 \leqslant i \leqslant n$, é um subestado imediato de s. Se $\rho(s) = \phi$ então s não possui subestados.

3. **Fechos** ρ^+ e ρ^*

   $$\rho^+(s) := \bigcup_{i>0} \rho^i(s) \quad \text{e} \quad \rho^*(s) := \bigcup_{i \geqslant 0} \rho^i(s),$$

   onde $\forall\, s \in S,\ \rho(s) := \rho^1(s),\ \rho^0(s) := \{s\}$,

   $$\rho^i(s) := \bigcup_{s' \in \rho(s)} \rho^{i-1}(s') \quad \text{para} \quad i > 1 \text{ e}$$

   $$\forall\, j \geqslant k,\ \rho^k(s) = \phi \Rightarrow \rho^j(s) := \phi$$

4. **Função Tipo** $\psi_E : S \to \{\mathsf{AND}, \mathsf{OR}, \mathsf{BAS}\}$
   Designa o tipo estrutural (atributo imutável) de cada estado. Se $\psi_E(s) = \mathsf{BAS}$ então o estado s não possui subestados, $\psi_E(s) = \mathsf{BAS} \Leftrightarrow \rho(s) = \phi$. Se $\psi_E(s) = \mathsf{AND}$ então o estado s é dito *concorrente* e os estados $\rho(s)$ estão simultaneamente ativos quando s está ativo. Se $\psi_E(s) = \mathsf{OR}$, então o estado s é dito *exclusivo* e no máximo um dos estados em $\rho(s)$ está ativo quando s está ativo.

As definições abaixo são utilizadas na descrição dos demais elementos de um diagrama.

- **Ancestral comum mais próximo:** α
  Para um conjunto X não vazio, $X \in 2^S$ e $x \in S$, diz-se que x é o ancestral comum mais próximo de X, $\alpha(X) := x$, se e somente se as condições abaixo são satisfeitas:

  - $X \subseteq \rho^*(x)$
  - $\forall s \in S, X \subseteq \rho^*(s) \Rightarrow x \in \rho^*(s)$

- **Estados ortogonais:** $s_1 \perp s_2$
  Seja $s_1, s_2 \in S$. Diz-se que s_1 e s_2 são ortogonais, o que é denotado por $s_1 \perp s_2$, se e somente se:

  - $\psi_E(\alpha(\{s_1, s_2\})) = \mathsf{AND}$
  - $\rho^*(s_1) \cap \rho^*(s_2) = \phi$

- **Conjunto ortogonal**
  $X \in 2^{S \cup H}$ é ortogonal se e somente se X é unitário ou para quaisquer x_1 e x_2 distintos em X tem-se $x_1' \perp x_2'$ onde

$$x_i' := \begin{cases} x_i & \text{se } x_i \in S \\ \gamma(x_i) & \text{se } x_i \in H \end{cases}$$

  para $i = 1, 2$.

- **Conjunto ortogonal relativo a estado**
  X é ortogonal em relação a $s \in S$ se e somente se $X \subseteq \rho^*(s)$ e X é ortogonal.

- **Conjunto ortogonal maximal**
  X é um conjunto ortogonal maximal relativo a um estado s se X é ortogonal relativo a s e $\nexists y \in \rho^*(s)$ tal que $y \notin X$ e $X \cup \{y\}$ é ortogonal. Em particular, $\{s\}$ é ortogonal maximal relativo a s para todo $s \in S$.

- **Raiz:** $\mathcal{R}$
  A raiz $\mathcal{R}$ de um Xchart é definida como o estado mais externo deste Xchart. Dito de outra forma, $\mathcal{R} \in S \mid \forall s \in S, \mathcal{R} \notin \rho(s)$.

5. Φ de **estados fictícios**. Cada elemento desse conjunto representa um pequeno círculo, que é ou a origem de uma transição inicial, ou o destino de uma transição final ou ainda o destino de uma transição pseudo-final. Transições são definidas no item 12.

6. **Função ômega** $\omega : \Phi \rightarrow S$
   Um estado fictício $f \in \Phi$ pode estar associado a um estado convencional s e, nesse caso, $\omega(f) := s$. Se o estado fictício f é o destino de uma transição final, por exemplo, então não há associação entre f e um estado convencional. Nesse caso, a função ω não está definida para esse domínio particular, o que é denotado por $\omega(f) \equiv \lambda$.

7. H de **símbolos de história**. Um símbolo de história pode ser simples ou do tipo estrela e está associado a um estado que não é básico. H_s denota o conjunto dos símbolos simples e H_* do tipo estrela.

   - $H := H_s \cup H_*$
   - $H_s \cap H_* = \phi$

8. **Função gama** $\gamma : H \rightarrow S$
   Função que estabelece o estado s associado a cada símbolo de história.

9. V de **expressões numéricas**:

   - Se $k \in \mathcal{N}$ então $k \in V$
   - Se $v \in V_N$ então $v \in V$
   - Se $v \in V$ então $\mathbf{step}(v) \in V$
   - Se $v_1, v_2 \in V$ e $op_n \in OP_N$ então $v_1\, op_n\, v_2 \in V$

10. C de **expressões lógicas**:

    - $\kappa \in C$, κ é a expressão lógica nula.
    - $\{\mathbf{T}, \mathbf{F}\} \subseteq C$
    - Se $c \in V_L$ então $c \in C$
    - Se $s \in S$ e $i \in I$ então $\{\mathbf{in}(s), \mathbf{in}(s,i)\} \subseteq C$
    - Se $e \in E$ então $\mathbf{ny}(e) \in C$ (o conjunto E é definido abaixo)
    - Se $u, v \in V$ e $op_l \in OP_L$ então $u\, op_l\, v \in C$
    - Se $a \in A_T$ e $p \in P$ então $\{\mathbf{active}(a,p), \mathbf{hanging}(a,p)\} \subseteq C$
    - Se $c \in C$ então $\mathbf{step}(c) \in C$
    - Se $c, c_1, c_2 \in C$ então $\{c_1\ \mathbf{and}\ c_2, c_1\ \mathbf{or}\ c_2, \mathbf{not}(c)\} \subseteq C$

11. E de **eventos**:

- $\lambda \in E$, λ é o evento nulo.
- Se $c_e \in C_E$ então $c_e \in E$
- Se $c \in C$ então $\{\mathbf{tr}(c), \mathbf{fs}(c)\} \subseteq E$
- Se $v \in V \cup C$ então $\mathbf{ch}(v) \in E$
- Se $s \in S$ e $i \in I$ então $\{\mathbf{ex}(s), \mathbf{en}(s), \mathbf{ex}(s,i), \mathbf{en}(s,i)\} \subseteq E$
- Se $v \in V$ então $\{\mathbf{after}(v), \mathbf{every}(v)\} \subseteq E$
- Se $h, m, s \in \mathcal{N}$ então $\mathbf{at}(h : m : s) \in E$
- Se $a \in A_T$ e $p \in P$ então $\{\mathbf{started}(a,p), \mathbf{stopped}(a,p)\} \subseteq E$
- Se $e, e_1, e_2 \in E$ então $\{ e_1 \mathbf{\ and\ } e_2,\ e_1 \mathbf{\ or\ } e_2, \mathbf{not}(e)\} \subseteq E$

12. T de **transições**. $t := (X, Y)$ é uma transição do conjunto origem X para o conjunto destino Y. O conjunto T é formado pela união de transições de vários tipos:

$$T := T_c \cup T_f \cup T_i \cup T_{pf}$$

onde a transição t pertence ao conjunto de transições *convencionais* T_c se e somente se:

- $X \in 2^S$ e X é ortogonal.
- $Y \in 2^{S \cup H}$ e Y é ortogonal.
- $\{x, y\}$ não é ortogonal para qualquer que seja $x \in X$ e $y \in Y$.

As transições *finais* T_f são utilizadas para desativar instâncias de Xcharts. A transição t pertence a T_f se e somente se:

- $X \in 2^S$ e X é ortogonal.
- $Y = \{f\}$, $f \in \Phi$ e $\omega(f) \equiv \lambda$

As transições *iniciais* T_i são utilizadas durante o processo de ativação de um estado. A transição t pertence a T_i se e somente se:

- $X = \{f\}$, $f \in \Phi$
- $\omega(f) \in S$ e $\psi_E(\omega(f)) = \mathsf{OR}$
- $Y \subseteq 2^{\rho^+(\omega(f))}$ e Y é ortogonal.

As transições *pseudo-finais* T_{pf} provocam a ativação dos estados fictícios, que fazem parte do destino, mas não ativam os subestados desses estados fictícios. A transição t é um elemento de T_{pf} se e somente se:

- $Y = \{y_1, y_2, \ldots, y_l\}$, $1 \leq l$
- $\exists y \mid y \in Y$ e $y \in \Phi$
- Se $y \in Y$ e $y \in \Phi$ então $\omega(y) \in S$
- $(X, \{y'_1, y'_2, \ldots, y'_l\})$ é uma transição convencional onde

$$y'_i := \begin{cases} y_i & \text{se } y_i \in S \cup H \\ \omega(y_i) & \text{se } y_i \in \Phi \end{cases}$$

para $1 \leq i \leq l$.

13. A de **ações**. Representam as reações que podem ser produzidas por uma instância. Exceto a última ação, dita composta, as demais são executadas atomicamente, ou seja, suas execuções são indivisíveis. Uma ação é indivisível se ela não possui um estado intermediário entre o início e o fim da sua execução.

- $\epsilon \in A$, ϵ é a ação nula.
- Se $c \in V_L$ e $d \in C$ então $c := d \in A$
- Se $v \in V_N$ e $u \in V$ então $v := u \in A$
- Se $e \in C_E$ e $i \in I$ então $\{\mathbf{raise}(e), \mathbf{raise}(e, i)\} \subseteq A$
- Se $e \in C_E$ e $d \in D$ então $\mathbf{raise}(e, d) \in A$
- Se $e \in C_E$, $i \in I$ e $s \in S$ então $\mathbf{raise}(e, i, s) \in A$
- Se $e \in E$ e $v \in V$ então $\mathbf{wait}(e, v) \in A$
  $(\mathbf{wait}(e) \equiv \mathbf{wait}(e, \infty))$
- Se $a \in A_T$ e $p \in P$ então $\mathbf{sync}(a, p) \in A$
- Se $d \in D$ e $i \in I$ então $\{\mathbf{call}(d, i), \mathbf{run}(d, i)\} \subseteq A$
  (A implementação de **call**, por exemplo, pode evitar a necessidade de um identificador para a instância criada. Tais detalhes, contudo, são evitados no presente texto.)
- Se $a \in A_T$ e $p \in P$ então $\{\mathbf{start}(a, p), \mathbf{stop}(a, p)\} \subseteq A$
  (Naturalmente, $\mathbf{start}(e) \equiv \mathbf{start}(a, p)$ onde p é a porta padrão.)
- Se $a \in A_T$ e $p \in P$ então $\{\mathbf{suspend}(a, p), \mathbf{resume}(a, p)\} \subseteq A$
- Se $h \in H$ então $\mathbf{clear}(h) \in A$
- Se $a_i \in A$ para $0 \leqslant i \leqslant n$ e $0 \leq n$, então $a \in A$ onde $a := a_0 \,;\, a_1 \,;\, \cdots \,;\, a_n$. A ação a, formada pela seqüência de ações, é dita *composta*. Uma ação composta não é indivisível se $n > 0$.

- Se $a \in A$ é uma ação composta, então **atomic** { a } é uma ação composta indivisível.
- Se $a \in A$ é uma ação composta e $c \in C$, então **while** (c) $\{a\} \in A$
- Se $a \in A$ é uma ação composta e $c \in C$, então **do** $\{a\}$ **while** $(c) \in A$
- Se $x, y \in V_N$, $v_1, v_2 \in V$, $a \in A$ é uma ação composta e $c \in C$, então **for** ($x := v_1$; c; $y := v_2$) $\{a\} \in A$
- Se $a \in A$ é uma ação composta e $c \in C$, então **do** $\{a\}$ **while** $(c) \in A$
- Se $a_1, a_2 \in A$ são ações compostas e $c \in C$, então **if** (c) { a_1 } **else** { a_2 } $\in A$

14. G de **gatilhos**. Um gatilho é uma associação entre um evento e uma condição.

- $\eta \in G$, $\eta := \lambda[\kappa]$ é o gatilho nulo.
- Se $c \in C$ então $\{\textbf{entry}[c], \textbf{exit}[c]\} \subseteq G$
- Se $e \in E$ e $c \in C$ então $e[c] \in G$

15. R de **regras**. Elementos deste conjunto designam as regras que rotulam transições ou ocorrem no interior de estados:

- $\vartheta \in R$, $\vartheta := \eta : \epsilon$ é a regra nula.
- Se $g \in G$ e $a \in A$ então $g : a \in R$

Regras cujos gatilhos envolvem **entry** ou **exit** não podem rotular transições.

16. **Função associativa:** $\Lambda : R \rightarrow S \cup T$
A função associativa estabelece o estado ou a transição associada a uma determinada regra:

$$\Lambda(r) := \begin{cases} s & r \text{ está associada ao estado } s \\ t & r \text{ rotula a transição } t \end{cases}$$

17. **Função prioridade:** $\mho : T \rightarrow \mathbb{Z}$
Fornece o número de prioridade de uma transição. Dadas as transições $t_1, t_2 \in T$, diz-se que t_1 tem prioridade sobre a transição t_2 se e somente se $\mho(t1) > \mho(t2)$.

4.4 Definições

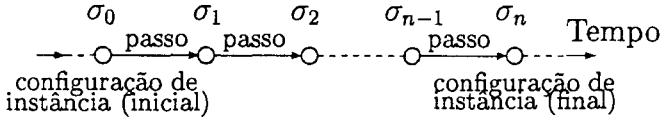
As definições seguintes são utilizadas na definição da semântica da linguagem Xchart. Algumas delas são pertinentes a um Xchart d, $d \in D$, ou a uma instância $\mathcal{I} \in I$ de um diagrama em D. Sem perda de generalidade, quando a definição diz respeito a uma instância $\mathcal{I}$, supõe-se que d é o diagrama em Xchart subjacente. A definição da semântica se baseia nas conseqüências de um estímulo externo tratado por uma instância $\mathcal{I}$ no intervalo $[\sigma_i, \sigma_{i+1})$. O estímulo é considerado pela instância em um instante σ nesse intervalo.

4.4.1 Instância de Xchart

Uma instância de um Xchart refere-se à animação desse Xchart. Para um dado Xchart podem existir zero ou mais instâncias em um dado instante. Para toda instância $\mathcal{I}$ de um Xchart $d \in D$ de um dado $\mathfrak{SDX}$ define-se uma seqüência de instantes de tempo significativos em relação à $\mathcal{I}$:

$$\Sigma_{\mathcal{I}} := \{\sigma_0, \ldots, \sigma_n\}, n \geqslant 0$$

Tais instantes são ilustrados na figura abaixo onde $\Sigma_{\mathcal{I}}$ é um subconjunto de todos os instantes de tempo Σ. Do ponto de vista de $\mathcal{I}$, o tempo real contínuo desde o instante em que é criada até aquele em que é destruída é dividido em intervalos definidos por tais instantes. Estes instantes estão relacionados às configurações de instância obtidas por $\mathcal{I}$ dados os estímulos externos tratados pela instância. Esses instantes não indicam os momentos em que tais estímulos são sinalizados, mas aqueles em que configurações de instância são obtidas. Uma configuração só pode ser obtida através da execução de um passo que, por sua vez, só ocorre na presença de um estímulo externo. O conjunto $\Sigma_{\mathcal{I}}$ define todos os instantes nos quais tais configurações de instâncias, para uma dada instância $\mathcal{I}$, são alteradas. Em particular, σ_0 é o instante de tempo inicial da criação de $\mathcal{I}$ e σ_n é o instante da destruição desta instância. Quando a instância em questão puder ser deduzida do contexto então será usado apenas Σ, por simplicidade.



Se nenhum estímulo externo é sinalizado para $\mathcal{I}$, após um instante σ_i, $0 \leq i < n$, então $\mathcal{I}$ permanece na configuração de instância obtida no instante σ_i. Para qualquer instante $\sigma < \sigma_0$ ou $\sigma \geqslant \sigma_n$ a instância $\mathcal{I}$ não existe, isto é, não provoca nenhuma reação nem recebe estímulos externos. Em contrapartida, no intervalo $[\sigma_0, \sigma_n)$ podem ser sinalizados

estímulos externos para cujas implicações deve haver uma semântica bem definida. Em particular, tal semântica deve estabelecer, para um estímulo tratado conforme a configuração no instante σ_i, qual a configuração resultante no instante σ_{i+1}. Essa semância inclui, basicamente, a definição das reações provocadas pelo estímulo, considerado em um instante σ no intervalo $[\sigma_i, \sigma_{i+1})$ e as conseqüências dessas reações, quando executadas. No intervalo $[\sigma_i, \sigma]$, a configuração da instância é visível externamente e coincide com aquela do instante σ_i. No intervalo (σ, σ_{i+1}) a configuração da instância não é visível externamente. Esse intervalo protege a reação em cadeia provocada pelo estímulo externo. Apenas no instante σ_{i+1} a nova configuração torna-se disponível e visível externamente.

4.4.2 Conjunto de Instâncias

Instâncias podem ser criadas e destruídas durante a execução de um $\mathfrak{SDX}$. A execução de um $\mathfrak{SDX}$ inicia-se quando a primeira instância de um diagrama do $\mathfrak{SDX}$ é criada e é finalizada quando a última instância em execução de um diagrama do $\mathfrak{SDX}$ é destruída e não mais existirem instâncias para serem criadas. Denote-se por $k_i \geqslant 0, 1 \leqslant i \leqslant n$ e $n := |D|$, a quantidade de instâncias, no instante σ, do Xchart $d_i \in D$ de um $\mathfrak{SDX}$ em execução. Seja ainda d_i^j a j-ésima instância de d_i. Logo, no instante σ, o conjunto de instâncias em execução de todo o $\mathfrak{SDX}$ é dado por

$$\mathfrak{I}_\sigma := \bigcup_{i=1}^{n} \{d_i^j \mid 1 \leqslant j \leqslant k_i\}$$

Não existe relação de precedência entre instâncias de $\mathfrak{I}_\sigma$. Xchart não prescreve nenhum mecanismo de justiça para garantir que estímulos externos sinalizados para uma dada instância sejam tratados em algum instante, em vez de perdurarem indefinidamente na fila da instância. A implementação de Xchart deve lidar com tais questões. Ainda convém ressaltar que passos (reações em cadeia) são atômicas. Em um dado instante σ, no máximo uma instância de $\mathfrak{I}_\sigma$ encontra-se reagindo à um estímulo externo. Dessa forma, a granulosidade mínima de concorrência entre instâncias é um passo. No máximo um único passo encontra-se em execução em um dado instante.

4.4.3 Ciclo de Vida de Instâncias

Conforme apresentado na Seção 3.6 (pág. 100), uma instância alterna, ao longo de sua existência, por vários estados. A semântica formal apresentada na Seção 4.5 descreve a reação de uma instância enquanto ativa. A reação de uma instância inativa é trivial, pois ela não produz nenhum tipo de controle ou saída. Analogamente, uma instância inoperante aguarda pela ocorrência de elemento externo à instância, não produzindo nenhum efeito enquanto tal ocorrência não se verifica.

As instâncias ativas concorrem para tratarem estímulos externos que possivelmente existam em suas filas. Se alguma instância encontra-se executando um passo, então as demais aguardam pelo término desse passo. Em contrapartida, só pode existir uma instância inoperante, pois ela estaria no interior de uma reação em cadeia (passo) e apenas uma instância pode estar executando um passo, por vez. Instâncias inativas podem existir em qualquer instante da execução de um sistema.

4.4.4 Função Tipo

Durante a execução de uma instância $\mathcal{I}$, a ativação de um estado exclusivo s dessa instância, $\psi_E(s) = \mathsf{OR}$, pode ou não conduzir à ativação de um de seus subestados imediatos, dependendo das regras que rotulam as transições iniciais de s. Isto torna possível o tipo de um estado exclusivo alternar para básico e, posteriormente, retornar ao tipo exclusivo. A função

$$\psi : S \rightarrow \{\mathsf{OR}, \mathsf{BAS}, \mathsf{AND}\}$$

designa o tipo de um estado da instância $\mathcal{I}$ no instante $\sigma_i \in \Sigma_{\mathcal{I}}$. Se $\psi_E(s) = \mathsf{BAS}$, então $\psi(s) = \mathsf{BAS}$ para todo $\sigma \in \Sigma_{\mathcal{I}}$. Analogamente, $\psi(s) = \mathsf{AND}$ se $\psi_E(s) = \mathsf{AND}$. Se $\psi_E(s) = \mathsf{OR}$, então $\psi(s) = \mathsf{OR}$ ou $\psi(s) = \mathsf{BAS}$, conforme a existência ou não, respectivamente, de algum subestado de s que esteja ativo na instância $\mathcal{I}$ no instante σ_i. Em contrapartida, o tipo estrutural de um estado ψ_E, qualquer que seja ele em qualquer instante da execução da instância, é sempre o mesmo.

4.4.5 Ancestral Estrito

Para um conjunto $X \in 2^S$ e $X \neq \phi$, diz-se $x \in S$ é o ancestral estrito de X, $\alpha^+(X) := x$, se e somente se as condições abaixo forem satisfeitas:

1. $\psi(x) = \mathsf{OR}$
2. $X \subseteq \rho^+(x)$
3. $\forall s \in S \mid \psi(s) = \mathsf{OR}$ então $X \subseteq \rho^*(s) \Rightarrow x \in \rho^*(s)$

4.4.6 Transição Interna/Externa

Toda transição possui um atributo, definido graficamente, que estabelece se ela é *interna* ou *externa*. A transição rotulada por **te**, Figura 4-3, é dita externa, enquanto **ti** é interna. Se uma transição (X, Y) é interna, então X é um conjunto unitário e a aresta que une X a Y não ultrapassa a fronteira do único estado em X. Se X não é unitário ou se tal aresta atravessa a fronteira do estado em X, então a transição é externa.

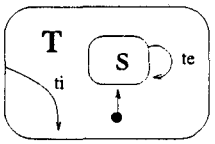


Figura 4-3: Transição externa e interna

4.4.7 Ancestral de Transição

O α de uma transição $t := (X, Y)$ convencional ou pseudo-final é definido abaixo.

$$\alpha(t) := \begin{cases} s' & \text{se } X = Y = \{s \in S\}, t \text{ é } \textit{externa} \text{ e } s \in \rho(s'). \\ s'' & \text{se } X = Y = \{s'' \in S\} \text{ e t é } \textit{interna}. \\ s & \text{se } \exists s \in U_t \mid U_t \subseteq \rho^*(s). \\ \alpha^+(U_t) & \text{nenhum dos casos anteriores.} \end{cases}$$

onde o conjunto U_t é definido como segue:

$$U_t := X \bigcup_{y \in Y} \left\{ \begin{array}{ll} \omega(y) & \text{se } y \in \Phi \text{ e } \omega(y) \in S \\ \gamma(y) & \text{se } y \in H \\ y & \text{se } y \in S \end{array} \right\}$$

4.4.8 Configuração de Estados

Uma configuração $X \subseteq 2^S$ de estados relativa ao estado s é um conjunto ortogonal relativo a s onde $x \in X \Rightarrow \psi(x) = \mathsf{BAS}$.

Configuração maximal

A execução de uma instância é marcada por instantes de tempo significativos (pág. 134). Para cada um desses instantes estão associadas várias informações acerca do estado da instância. Uma delas é a configuração maximal de estados relativa à raiz da instância. Uma configuração $X \subseteq 2^S$ maximal de estados relativa ao estado s é uma configuração de estados relativa ao estado s e um conjunto ortogonal maximal relativo a s.

Configurações maximais da raiz

$\mathcal{X}_0^n := (X_0, X_1, \ldots, X_n)$ é uma seqüência de configurações maximais de estados relativas à raiz do Xchart de $\mathcal{I}$ se:

1. X_i, $0 \leq i \leq n$, é a configuração maximal de estados relativa à raiz do Xchart subjacente a $\mathcal{I}$ no instante σ_i da execução dessa instância. (Instantes significativos de uma instância são definidos na pág. 134.)

2. X_0 é a configuração inicial de estados de $\mathcal{I}$ (pág. 148).

3. $n+1$ é o número total de configurações até um dado instante da execução de $\mathcal{I}$. Ou seja, no instante σ_i, a seqüência pertinente é dada por $\mathcal{X}_0^i$.

Note que não necessariamente $X_i \neq X_j$ para $i \neq j$. Em particular, quando um estímulo externo sinalizado no intervalo $[\sigma_i, \sigma_{i+1})$ provoca uma reação que não envolve a execução de transições, então necessariamente $X_i = X_{i+1}$.

4.4.9 Função π

Identifica a transição inicial de um dado estado s a ser executada em conseqüência da ativação de s. As transições iniciais relativas a um estado s são as transições $t := (f, Y)$ tais que $\omega(f) = s$. Em particular, se nenhuma transição inicial está habilitada, então nenhuma transição é selecionada e, nesse caso, $\pi(s) \equiv \lambda$. Caso contrário, $\pi(s) := t$ se e somente se:

1. A transição t é uma transição inicial relativa ao estado s e t está habilitada.

2. Se o conjunto $T' := \{t_1, t_2, \ldots, t_n\}$ com duas ou mais transições iniciais relativas a s e habilitadas é identificado, então para toda transição $t \in T'$, t não é selecionada se $\exists t' \in T'$ tal que $\mho(t') > \mho(t)$.

3. Se nenhuma transição satisfaz as condições acima, então, conforme mencionado anteriormente, $\pi(s) \equiv \lambda$. Se apenas uma transição t satisfaz estas condições, então $\pi(s) := t$. Se um conjunto de transições com pelo menos dois elementos satisfaz tais restrições, então uma delas é arbitrariamente selecionada.

4.4.10 Função Histórica

Dada a seqüência $\mathcal{X}_0^n$ e um estado s onde $\psi_E(s) = \mathsf{OR}$, a função histórica $\tau(s, \mathcal{X}_0^n)$ define o subestado de s que esteve ativo quando s foi ativado pela "última" vez, conforme $\mathcal{X}_0^n$. Em particular, diz-se que $\tau(s, \mathcal{X}_0^n) \equiv \lambda$ nos seguintes casos:

- Se s não foi ativado em nenhum momento da execução da instância.

- Se nenhum subestado de s foi ativado quando s tornou-se ativo pela última vez.

- Se a ação **clear(h)** foi executada em micro-passo anterior. Durante a execução de um passo pode-se "limpart" o símbolo de história antes que ele seja utilizado, em micro-passo posterior, na ativação do estado correspondente. Se houve uma ação **clear**, então necessariamente, para efeito do passo, o símbolo está limpo, pois um estado não pode ser ativado duas vezes em um mesmo passo.

Nos demais casos, o conjunto I_s é utilizado na definição de τ.

$$I_s := \{i \mid 0 \leqslant i \leqslant n \text{ e } \rho^*(s) \cap X_i \neq \phi\}$$

Se $I_s = \phi$, então $\tau(s, \mathcal{X}_0^n) \equiv \lambda$. Caso contrário, seja $j \in I_s$ tal que $j \geqslant i$ para todo $i \in I_s$. Se $\exists s' \in \rho(s)$ tal que $\rho^*(s') \cap X_j \neq \phi$ então $\tau(s, \mathcal{X}_0^n) := s'$. Se não existe tal s', então $\tau(s, \mathcal{X}_0^n) \equiv \lambda$.

4.4.11 Ação Ativação/Desativação

Dadas as regras r_a e r_d, $r_a :=$ **entry** : a_a e $r_d :=$ **exit** : a_d. A ação a_a é dita ação de ativação e a ação a_d é dita ação de desativação.

4.4.12 Função de Configuração

A reação de uma instância à ocorrência de um estímulo externo pode compreender a execução de um conjunto de regras e/ou transições. (A definição dos elementos que constituem esse conjunto é fornecida posteriormente.) Seja Z um subconjunto desses elementos formado apenas pelas transições. A execução dos elementos de Z estabelece uma nova configuração maximal de estados do estado raiz. A função $C(Z, \mathcal{X}_0^n)$ define a configuração maximal de estados obtida pela execução das transições do Z conforme a seqüência de configurações maximais de estados do estado raiz $\mathcal{X}_0^n = (X_0, \ldots, X_n)$.

A definição de C faz uso das definições de outras duas funções: $\mathcal{M}$ e $\mathcal{F}$. A função $\mathcal{M}$ identifica estados que *implicitamente* fazem parte do destino de uma transição. Por exemplo, a Figura 4-4 (pág. 142) exibe a hierarquia de estados de um Xchart e uma transição t, cujo destino é o símbolo de história **h** no interior do estado **B**. O destino ainda inclui implicitamente os estados **C**, **E** e **D**, que estarão ativos na próxima configuração juntamente com o estado **B**. Os estados **C**, **E** e **D** são subestados de estados do tipo **AND** que estarão ativos na próxima configuração.
O conjunto $\mathcal{M}(t)$ é composto, possivelmente, por símbolos de história e estados não básicos. Esses destinos são tratados de formas distintas e envolvem, possivelmente, um processo onde subestados desses elementos são recursivamente ativados até que estados básicos sejam ativados e interrompam o processo. A função inflar $\mathcal{F}(\mathcal{M}(t), \mathcal{X}_0^n)$ fornece uma configuração maximal de estados relativa ao estado $\alpha(t)$ para a transição t e a seqüência $\mathcal{X}_0^n$ de configurações maximais da raiz do Xchart pertinente. Esta configuração é obtida através da ativação dos subestados pertinentes aos elementos do conjunto $\mathcal{M}(t)$, conforme o processo de ativação de estados.

A função C deve identificar todas as configurações maximais de estados relativas aos ancestrais comuns mais próximos de todas as transições em Z, ou seja,

$$C(Z, \mathcal{X}_0^n) := \mathcal{F}\left(\bigcup_{t \in Z} \mathcal{M}(t), \mathcal{X}_0^n \right)$$

Função maximizar

A função maximizar $\mathcal{M} : T \rightarrow 2^{S \cup H}$ é tal que dada uma transição $t := (X, Y)$, a função $\mathcal{M}(t)$ identifica o conjunto ortogonal maximal relativo a $\alpha(t)$. Por conveniência é útil definir S_t como o conjunto formado por estados do tipo AND, que são descendentes de $\alpha(t)$ e ancestrais dos elementos em Y. Estes estados estarão ativos na próxima configuração. Um estado s pertence a S_t se e somente se existe y em Y tal que:

1. $\psi_E(s) = \mathsf{AND}$
2. $s \in \rho^+(\alpha(t))$
3. $\rho^+(s) \bigcap \left\{ \begin{array}{ll} y & \text{se } y \in S \\ \gamma(y) & \text{se } y \in H \end{array} \right\} \neq \phi$

Definido S_t, diz-se que um estado $y \in \mathcal{M}(t)$ se $y \in Y$ ou y é tal que $\exists s \in S_t$ que satisfaz as seguintes condições:

1. $y \in \rho(s)$
2. $\rho^*(y) \bigcap_{\forall y' \in Y} \left\{ \begin{array}{ll} y' & \text{se } y' \in S \\ \gamma(y') & \text{se } y' \in H \end{array} \right\} = \phi$

A condição 2 acrescenta estados que não estão explicitamente contidos no destino da transição t e evita o acréscimo de estados que estarão ativos mas não formam um conjunto ortogonal. Dessa forma, $\mathcal{M}(t)$ contém Y e estados que estarão ativos na próxima configuração e não são ancestrais nem descendentes de elementos em Y, se for o caso.

Função inflar

Dados um conjunto ortogonal maximal X composto por estados e símbolos de história relativo ao estado $\alpha(X)$ e a seqüência $\mathcal{X}_0^n$ de configurações maximais da raiz, a função $\mathcal{F}$ define uma configuração maximal de estados relativa a $\alpha(X)$. $\mathcal{F}$ é recursivamente definida de acordo com os elementos de X. $\forall x \in X$:

1. Se $x \in S$ então:

- Se $\psi_E(x) = \mathsf{BAS}$ então $x \in \mathcal{F}(X, \mathcal{X}_0^n)$
- Se $\psi_E(x) = \mathsf{OR}$ então
  - Se $\pi(x) \not\equiv \lambda$ então $\mathcal{F}(\mathcal{M}(\pi(x)), \mathcal{X}_0^n) \subseteq \mathcal{F}(X, \mathcal{X}_0^n)$
  - Se $\pi(x) \equiv \lambda$ então $x \in \mathcal{F}(X, \mathcal{X}_0^n)$
- Se $\psi_E(x) = \mathsf{AND}$ então $\mathcal{F}(\rho(x), \mathcal{X}_0^n) \subseteq \mathcal{F}(X, \mathcal{X}_0^n)$

2. Se $x \in H_s$ então
   - Se $\tau(\gamma(x), \mathcal{X}_0^n) \not\equiv \lambda$ então $\mathcal{F}(\{\tau(\gamma(x), \mathcal{X}_0^n)\}, \mathcal{X}_0^n) \subseteq \mathcal{F}(X, \mathcal{X}_0^n)$
   - Se $\tau(\gamma(x), \mathcal{X}_0^n) \equiv \lambda$ e $\pi(\gamma(x)) \neq \phi$ então $\mathcal{F}(\mathcal{M}(\pi(\gamma(x))), \mathcal{X}_0^n) \subseteq \mathcal{F}(X, \mathcal{X}_0^n)$
   - Se $\tau(\gamma(x), \mathcal{X}_0^n) \equiv \lambda$ e $\pi(\gamma(x)) \equiv \lambda$ então $\gamma(x) \in \mathcal{F}(X, \mathcal{X}_0^n)$
3. Se $x \in H_*$ então $\mathcal{F}_*(\gamma(x), \mathcal{X}_0^n) \subseteq \mathcal{F}(X, \mathcal{X}_0^n)$

Em particular, $\mathcal{F}(\phi, \mathcal{X}_0^n) := \phi$.

Função inflar estrela

Dado um estado s, associado a um símbolo de história $h \in H_*$, a função $\mathcal{F}_*$ fornece a configuração maximal de estados do estado s, conforme $\mathcal{X}_0^n$, caso sua ativação ocorra pelo h.

1. Se $\tau(s, \mathcal{X}_0^n) \equiv \lambda$ e $\pi(s) \equiv \lambda$ então $s \in \mathcal{F}_*(s, \mathcal{X}_0^n)$
2. Se $\tau(s, \mathcal{X}_0^n) \equiv \lambda$ e $\pi(s) \not\equiv \lambda$ então $\mathcal{F}(\mathcal{M}(\pi(s)), \mathcal{X}_0^n) \subseteq \mathcal{F}_*(s, \mathcal{X}_0^n)$
3. Se $\tau(s, \mathcal{X}_0^n) \not\equiv \lambda$ então $s' := \tau(s, \mathcal{X}_0^n)$
   - Se $\psi_E(s') = \mathsf{BAS}$ então $s' \in \mathcal{F}_*(s, \mathcal{X}_0^n)$
   - Se $\psi_E(s') = \mathsf{OR}$ então $\mathcal{F}_*(s', \mathcal{X}_0^n) \subseteq \mathcal{F}_*(s, \mathcal{X}_0^n)$
   - Se $\psi_E(s') = \mathsf{AND}$ então $\bigcup_{s'' \in \rho(s')} \mathcal{F}_*(s'', \mathcal{X}_0^n) \subseteq \mathcal{F}_*(s, \mathcal{X}_0^n)$

4.4.13 Conseqüências da Execução de Transição

Uma transição t provoca a desativação e ativação de estados. Todos os estados que serão ativados ou desativados são descendentes de $\alpha(t)$. Contudo, nem todos eles serão, necessariamente, desativados e/ou ativados. O conjunto S_d^t é composto por todos os estados que serão desativados em conseqüência da execução da transição t. Um estado s pertence a S_d^t se e somente se:

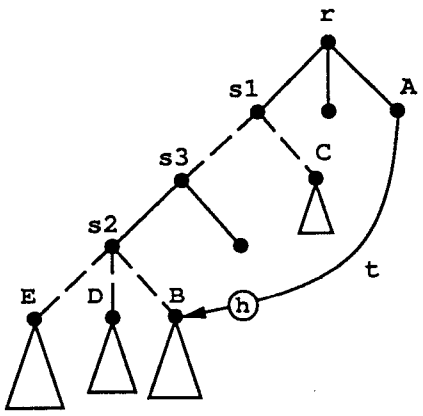


Figura 4-4: Hierarquia de estados de um Xchart

- $s \in \rho^{+}(\alpha(t))$
- $\rho^{*}(s) \cap X \neq \phi$, onde X é a configuração maximal de estados de $\mathcal{R}$.

O conjunto de estados S_a^t a ser ativado em conseqüência da execução de t é tal que $s \in S_a^t$ se e somente se:

- $s \in \rho^{+}(\alpha(t))$
- $\rho^{*}(s) \cap C(\{t\}, \mathcal{X}_0^n) \neq \phi$, onde $\mathcal{X}_0^n$ é a seqüência de configurações maximais de estados da raiz $\mathcal{R}$.

Ordem de desativação/ativação

O processo de ativação/desativação de estados, causado por uma transição t, estabelece uma seqüência $\vec{S_d^t}$ na qual os estados desativados S_d^t pela execução da transição t são desativados e outra, $\vec{S_a^t}$, na qual os estados S_a^t a serem ativados por t são ativados. Sejam s_1 e s_2 dois estados tais que $s_1, s_2 \in S_d^t \cup S_a^t$, $s_1 \neq s_2$ e X a configuração maximal de estados de $\mathcal{R}$. Conforme tais definições, as normas abaixo estabelecem a ordem de ativação/desativação de estados.

1. Se $s_1 \in \rho^{+}(s_2)$ e $s_1, s_2 \in S_d^t$ então s_1 é desativado antes da desativação de s_2.
2. Se $s_1 \in \rho^{+}(s_2)$ e $s_1, s_2 \in S_a^t$ então s_1 é ativado após a ativação de s_2.
3. Se $s_1 \in S_d^t$ e $s_2 \in S_a^t$ então s_1 é desativado antes da ativação de s_2.
4. Se $s_1, s_2 \in S_d^t$ e $s_1 \perp s_2$ então qualquer um deles pode ser arbitrariamente desativado antes do outro.

5. Se $s_1, s_2 \in S_a^t$ e $s_1 \perp s_2$ então qualquer um deles pode ser arbitrariamente ativado antes do outro.

Os itens 4 e 5 acima mostram que estados ortogonais permitem a existência de mais de uma seqüência correta $\vec{S_a^t}$ e $\vec{S_d^t}$ para os respectivos conjuntos S_a^t e S_d^t. Ao executar uma transição, uma dessas seqüências é arbitrariamente selecionada e executada para a desativação e ativação, se existir mais de uma.

Seqüência de execução de ações

Toda transição t rotulada por uma regra r induz uma seqüência $\Gamma(t)$ de ações de desativação e de ativação além da ação da regra que rotula a transição. Seja $\vec{S_d^t}$ a seqüência de estados a serem desativados, na ordem desta seqüência, em decorrência de t, e $\vec{S_a^t}$ a seqüência de estados a serem ativados.

$$\vec{S_d^t} := (s_d^1, s_d^2, \ldots, s_d^u) \text{ e } \vec{S_a^t} := (s_a^1, s_a^2, \ldots, s_a^v)$$

Tanto $\vec{S_d^t}$ quanto $\vec{S_a^t}$ induz uma seqüência de ações que devem ser executadas em decorrência da desativação/ativação de estados, provocadas por t. A seqüência de ações de desativação $(a_d^1, a_d^2, \ldots, a_d^k), 0 \leqslant k \leqslant u$, é obtida através da identificação de estados em $\vec{S_d^t}$ que contêm ações de desativação. Se a_d^m e a_d^n são elementos desta seqüência, $1 \leqslant m, n \leqslant k, m \neq n$, então existem s_d^f e s_d^g na seqüência $\vec{S_d^t}, 1 \leqslant f, g \leqslant u, f \neq g$, tais que a_d^m e a_d^n são causadas, respectivamente, pela desativação de s_d^f e s_d^g onde $f < g \Rightarrow m < n$. Analogamente é definida a seqüência de ações de ativação $(a_a^1, a_a^2, \ldots, a_a^l), 0 \leqslant l \leqslant v$. Dessa forma, dado que $r = g : a$ está associada à transição t, ou seja, $\Lambda(r) = t$, a seqüência de ações provocadas pela execução de t é definida como segue:

$$\Gamma(t) := (a_d^1; a_d^2; \cdots; a_d^k; a; a_a^1; a_a^2; \cdots; a_a^l)$$

4.4.14 Ação Parcial e Ação Total

Seja $L := \{r_0, \ldots, r_n\}$ um conjunto de regras onde $r_i := g_i : a_i$ para $0 \leqslant i \leqslant n$. Diz-se que a ação $a := a_0; \cdots; a_n$ é a *ação parcial* associada a L. A *ação total* é dada pela ação $a' := a'_0; a'_1; \cdots; a'_n$ onde a'_i é definida abaixo para $0 \leqslant i \leqslant n$.

$$a'_i := \begin{cases} \Gamma(r_i) & \text{se } \exists t \in T \mid \Lambda(t) = r_i \\ a_i & \text{caso contrário} \end{cases}$$

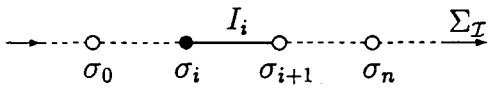
A próxima seção preocupa-se com a definição de um conjunto L de regras e transições a serem executadas e, posteriormente, como ele é executado e os efeitos produzidos por tal execução.

4.4.15 Eventos Primitivos

E_p denota o conjunto de eventos primitivos. Um evento primitivo é uma instância de uma classe (tipo) de evento primitivo. Um evento primitivo $ep \in E_p$ se e somente se $\exists c \in C_E$ tal que c é a classe de evento a partir da qual a instância ep foi criada.

4.5 Semântica Operacional

O eixo do tempo mostrado abaixo ilustra o intervalo de tempo I_i e os instantes de tempo típicos da execução de uma instância. Em particular, $I_i := [\sigma_i, \sigma_{i+1})$ onde o instante $\sigma \in I_i \Leftrightarrow \sigma_i \leqslant \sigma < \sigma_{i+1}$, $\sigma_i \geqslant \sigma_0$, $\sigma_{i+1} \leqslant \sigma_n$ e $0 \leqslant i < n$. No intervalo I_0 ocorre o primeiro passo e no intervalo I_{n-1} ocorre uma transição final (último passo), que finaliza a execução de $\mathcal{I}$. No instante σ_0 a instância está ativa e a partir do instante σ_n ela está inativa.



No instante σ_{i+1} é obtida a configuração de instância (definida adiante) provocada pelo estímulo externo tratado no intervalo I_i. A semântica da execução de uma instância é descrita por meio de definições formais das conseqüências desse estímulo externo e da configuração de $\mathcal{I}$ no instante σ_{i+1}. A configuração de instância define os estados ativos de uma instância, os valores de variáveis e outras informações pertinentes. Em outras palavras, a semântica de Xchart define precisamente tais configurações pelas quais uma instância passa, ao longo do tempo, ocasionadas pela execução de regras/transições provocadas por estímulos externos. Sem estímulos externos uma instância não produz nenhuma reação assim como sua configuração permanece inalterada.

Durante a reação provocada pelo estímulo externo, eventos podem ser gerados para outras instâncias. Por exemplo, a execução da ação **raise(e, i)** irá repercutir na instância i através de um estímulo externo contendo uma instância do tipo de evento e. O estímulo externo, contudo, não é sinalizado para i no instante em que a ação é executada. Apenas ao fim da reação (passo), no instante σ_{i+1}, tornam-se visíveis as reações da instância $\mathcal{I}$. Nesse instante, todos os estímulos gerados são sinalizados.

Ainda convém ressaltar que um estímulo externo, definido abaixo, não necessariamente é tratado no instante em que é sinalizado. Estímulos externos são enfileirados na ordem em que são sinalizados. O tratamento de cada um deles obedece esta ordem. Algumas das reações podem interromper a execução da instância por tempo indeterminado impedindo que um estímulo sinalizado para essa instância seja tratado assim que é sinalizado — uma reação em Xchart consome mais de zero unidades de tempo para ser executada.

No restante desse texto é apresentada a definição do que ocorre em um intervalo I_i da execução de uma instância $\mathcal{I}$. Os elementos doravante definidos referem-se a esse intervalo, cujas conseqüências tornam-se visíveis no instante σ_{i+1}.

4.5.1 Estímulo Externo

Π é um estímulo externo tratado no intervalo I_i cujas conseqüências tornam-se visíveis na configuração de instância obtida no instante σ_{i+1}. Π é um conjunto de eventos *simultâneos*. Se $e \in \Pi$, então e é um dos elementos abaixo:

- $e \in E_p$
- $\mathbf{ch}(v), v \in V_N \cup V_L$
- $\mathbf{en}(s,i), i \in \mathfrak{I}$ e s um estado da instância i.
- $\mathbf{ex}(s,i), i \in \mathfrak{I}$ e s um estado da instância i.
- $\mathbf{every}(k)$ para algum $k \in \mathcal{N}$
- $\mathbf{started}(a,p)$ para $a \in A_T$ e $p \in P$
- $\mathbf{stopped}(a,p)$ para $a \in A_T$ e $p \in P$
- $alarme(r,id)$ para $r \in R$ e id um identificador de **after**, **every**, **at** ou **wait** no interior da regra r.

Um estímulo externo pode ser gerado por clientes, por outras instâncias do subsistema reativo ou pelo relógio.

4.5.2 Relógio

A execução de um $\mathfrak{SDX}$ beneficia-se da existência, no ambiente onde é executado, de um relógio. Esse relógio é responsável por tarefas associadas a cronômetros: sinalizar o término de um período de tempo ou um instante específico. A programação desse relógio é realizada através de alarmes. O conjunto K representa todos os alarmes programados em um dado instante. A definição de K e o comportamento do relógio são definidos abaixo. Um alarme é representado por (i, r, id, k, t) onde:

- $i \in \mathfrak{I}$
- r é uma regra da instância i

- *id* identifica, dada a regra r, o evento **after**, **at**, **every** ou a ação **wait** correspondente ao alarme.
- k são as unidades de tempo que separam o alarme de ser sinalizado.
- t é o instante de tempo em que o alarme foi criado.

K é um conjunto de alarmes tais que dado $(i, r, id, k, t) \in K$, tem-se que:

1. Se $\Lambda(r) \in S$ então
   - o estado $\Lambda(r)$ permanece ativo desde o instante t ou, caso contrário,
   - o *id* refere-se a um **wait** ou **at**. A ação **wait** espera indeterminadamente por um evento. O evento **at** é gerado em um instante preestabelecido. Em ambos os casos, o instante t não é relevante. Se *id* não se refere a nenhum desses elementos, então refere-se ao evento **every** ou **after**. Nesses dois últimos casos, necessariamente o estado pertinente a $\Lambda(r)$ deve permanecer ativo desde o instante em que foi ativado e registrado por t. Isso ocorre porque tais eventos estão associados a um intervalo de permanência no estado pertinente.
2. Se $\Lambda(r) \in T$ e $\Lambda(r) = (X, Y)$ então
   - os estados em X permanecem ativos desde o instante t ou, caso contrário,
   - o *id* não se refere a **after** ou **every**. À semelhança do item anterior, tais eventos são gerados apenas se os estados que compõem a origem da transição $\Lambda(r)$ permanecem ativos desde o instante em que foram ativados até o intervalo preestabelecido pelo evento. Se essa condição não é satisfeita, então o evento pertinente não é gerado. Por exemplo, se uma transição possui dois estados como origem e um rótulo **after(10s)**, então tais estados devem permanecer ativos por 10 segundos para que esse evento seja gerado. Se um deles é desativado e posteriormente ativado, enquanto o outro permanece ativo, então a contagem de 10 segundos é reiniciada.
3. Se $k = 0$ então este elemento é removido do conjunto K e o estímulo externo contendo apenas $alarme(r, id)$ é sinalizado para a instância i.
4. Se $k > 0$ então após transcorrida uma unidade de tempo (r, id, k, t) é substituído por $(r, id, k - 1, t)$ em K.
5. Se a instância l é destruída, então todos os alarmes tais que $i = l$ são removidos do conjunto K.

6. Se um estado de uma instância i torna-se ativo em um instante t e há uma regra r associada a esse estado contendo **after, at, every** ou **wait**, que é identificado por id, então o alarme (i, r, id, k, t) é acrescentado ao conjunto K onde k é o intervalo ou instante de tempo associado a id.

4.5.3 Configuração de Instância

A configuração de $\mathcal{I}$ estabelece o *status* completo de $\mathcal{I}$ no instante σ_i. A instância $\mathcal{I}$ permanece com essa configuração até a ocorrência de um estímulo externo. A ocorrência de tal estímulo em um instante σ, $\sigma_i < \sigma$ induz uma "nova" configuração, que é fornecida pela configuração inicial de instância. A configuração inicial, definida abaixo, inclui o estímulo externo sinalizado. A configuração de instância é dada por $\Upsilon := (X, \Theta, \xi, \Omega, K, \mathcal{X}_0^{i-1})$ onde:

1. X é uma configuração maximal de estados do estado raiz da instância $\mathcal{I}$ no instante σ_i.

2. Θ é um conjunto de variáveis lógicas cujos valores são verdadeiros a partir de algum instante σ tal que $\forall c \in \Theta$ e $\Theta \subseteq V_L, c = \mathit{true}$ em $[\sigma, \sigma_{i+1})$ para $\sigma_i \leqslant \sigma$. Ou seja, todas as variáveis que ao terminarem o passo anterior continham o valor verdadeiro.

3. A função ξ estabelece o valor de uma variável ou constante numéricas

$$\xi : \{V_N \cup \mathcal{N}\} \rightarrow \mathbb{Z}$$

   $\xi(v) := x$ se o valor da variável v é x em $[\sigma, \sigma_{i+1})$ para $\sigma_i \leqslant \sigma$. Se $v \in \mathcal{N}$, então $\xi(v) := v'$ onde v' é o inteiro correspondente a v (por exemplo, $-\mathbf{2}$ corresponde ao inteiro -2).

4. Ω é o conjunto de símbolos de história para os quais não há informação de história disponível, $\Omega \subseteq H$.

5. K é o conjunto de alarmes.

6. $\mathcal{X}_0^{i-1} := (X_0, X_1, \ldots, X_{i-1})$ é a seqüência de configurações maximais da raiz onde X_j é a configuração maximal de estados da raiz da instância $\mathcal{I}$ no instante σ_j, $0 \leqslant j \leqslant i-1$.

A configuração $\Upsilon := (X, \Theta, \xi, \Omega, K, \mathcal{X}_0^{i-1})$ é utilizada na identificação da reação da instância à ocorrência do estímulo Π, que está associado ao instante σ_{i+1}. Ou seja, a reação da instância a esse estímulo irá repercutir produzindo uma nova configuração de instância

no instante σ_{i+1}. Em particular, uma instântica $\mathcal{I}$ permanece com a configuração obtida no instante σ_i até que um estímulo externo seja tratado. Sem a ocorrência de Π, a configuração do instante σ_i permanece inalterada. A definição de configuração inicial de instância (fornecida abaixo), ao contrário da definição de configuração de instância, inclui o estímulo externo a ser tratado pela instância. Observação: um estímulo externo também pode ser gerado pelo relógio conforme o conjunto K.

4.5.4 Configuração Inicial de Instância

$\Upsilon_0 := (X_0, \Theta, \xi, \Omega, K, \phi)$ define a configuração de uma instância imediatamente após a sua criação. De fato, a instância é criada em um instante anterior a σ_0. No instante σ_0 todas as ações de ativação provocadas pela ativação do estado raiz e seus subestados são finalizadas e a configuração inicial torna-se disponível. A instância criada permanece nesta configuração desde o instante σ_0 até a ocorrência do primeiro estímulo externo. Quando este estímulo é tratado, a nova configuração torna-se visível a partir do instante σ_1. A configuração Υ_0 é tal que:

- $X_0 := \mathcal{F}(\{\mathcal{R}\}, \phi)$ é a configuração inicial de estados da instância. Alternativamente, um conjunto ortogonal X' pode ser fornecido de forma que $X_0 := \mathcal{F}(\mathcal{M}((\mathcal{R}, X')), \phi)$. A configuração inicial de estados é uma configuração de estados maximal do estado raiz. No primeiro caso, ações de ativação deverão, eventualmente, ser executadas.

- Θ é tal que contém todas as variáveis cujo valor inicialmente fornecido é verdadeiro. Se $v \in V_L$ e $v \notin \Theta$, então o valor de v é falso. O valor inicial é fornecido juntamente com a definição da variável. A ativação da raiz do Xchart subjacente à instância pode, contudo, alterar esse conjunto. Ações de entrada em estado podem tornar variáveis inicialmente verdadeiras em falsas e vice-versa.

- $\xi(v) := 0$ se não foi fornecido nenhum valor inicial, no momento da criação da instância, para a variável v. Se a variável v possui um valor inicial k, então $\xi(v) := k$. A ativação da raiz do Xchart, conforme comentado acima, pode alterar o valor de variáveis.

- $\Omega := \phi$

- $K := \phi$

4.5.5 Reação de Instância

Dada uma configuração Υ de uma instância $\mathcal{I}$ e um estímulo externo sinalizado para esta instância, ela reage à ocorrência do estímulo através da execução de um passo Δ. O con-

junto $\{\Delta_1, \Delta_2, \ldots, \Delta_n\}$, $n \geqslant 1$, denota as reações (passos) possíveis dados a Υ e o estímulo externo sinalizado. Em particular, se $n > 1$ então a reação é dita não-determinística. Um dos principais objetivos da semância aqui descrita é a definição de um passo Δ_i para algum i tal que $1 \leqslant i \leqslant n$. A execução do passo Δ_i irá conduzir a instância a uma nova configuração.

4.5.6 Passo: Visão Geral

Um passo é um conjunto finito de regras e transições. Um passo também é utilizado para caracterizar a execução dos elementos desse conjunto. A definição da semântica apresentada neste texto define a configuração Υ' obtida por uma instância após tratar um estímulo externo sinalizado na configuração Υ dessa instância. A configuração Υ' é conseqüência da execução do passo Δ provocado pelo estímulo externo Π na configuração Υ, o que é denotado por

$$\langle \Pi, \Upsilon \rangle \stackrel{\Delta}{\rightarrow} \Upsilon'$$

Um passo é formado por uma seqüência finita e maximal de micro-passos.

4.5.7 Micro-passo: Visão Geral

Todas as regras e transições que compõem um passo Δ estão ordenadas em uma seqüência de micro-passos $\mu\Delta_1, \ldots, \mu\Delta_n$ que formam uma partição de Δ. Cada micro-passo é um conjunto de uma ou mais regras/transições. Micro-passo também denomina a execução dos elementos desse conjunto. O primeiro micro-passo é conseqüência imediata do estímulo externo em questão. A execução desse micro-passo induz uma micro-configuração de instância. Essa micro-configuração pode provocar a execução de um novo micro-passo, ocasionado pelo estímulo externo (sinalizado no início do passo) juntamente com os eventos gerados pelo primeiro micro-passo. A execução do segundo micro-passo induz uma nova micro-configuração e assim por diante.

Micro-passos capturam as conseqüências indiretas de um estímulo externo. O primeiro micro-passo é diretamente causado pelo estímulo externo. Os demais micro-passos de uma reação em cadeia, se houverem, são decorrências da execução dos micro-passos anteriores. Ou seja, o micro-passo $\mu\Delta_{i+1}$ é conseqüência da execução do micro-passo anterior $\mu\Delta_i$, que possivelmente gerou eventos, atualizou o valor de variáveis, expressões ou executou outra ação que permitiu a identificação de um novo conjunto de regras/transições a ser executado.

4.5.8 Micro-configuração

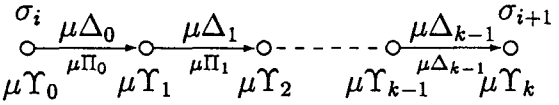
Uma micro-configuração de instância define a configuração de uma instância entre micro-passos. Um passo, composto por k micro-passos, pode ser escrito da seguinte forma:

$$\langle \mu\Pi_0, \mu\Upsilon_0 \rangle \overset{\mu\Delta_0}{\rightarrow} \langle \mu\Pi_1, \mu\Upsilon_1 \rangle \cdots \overset{\mu\Delta_{k-1}}{\rightarrow} \mu\Upsilon_k$$

onde

- $\mu\Delta_0, \mu\Delta_1, \ldots, \mu\Delta_{k-1}$ é a seqüência de micro-passos pertinentes ao passo Δ.
- $\mu\Upsilon_0$ e $\mu\Upsilon_k$ são, respectivamente, as micro-configurações inicial e final associadas à configuração Υ. Em particular, $\Upsilon = \Upsilon_0$ e $\Upsilon' = \Upsilon_k$.
- $\mu\Pi_i$ é a união do estímulo externo Π com os eventos gerados pela execução dos $i-1$ micro-passos anteriores onde $0 \leqslant i < k$ e $\mu\Pi_0 = \Pi$.

Um passo, executado no intervalo I_i, é abstratamente ilustrado abaixo.



Sendo que $\mu\Upsilon := (\mu X, \mu\Pi, \mu\Theta, \mu\xi, \mu\Omega, \mu Y, \mu Z)$ é uma micro-configuração de instância relativa a
$\Upsilon := (X, \Theta, \xi, \Omega, K, \mathcal{X}_0^{i-1})$ onde:

1. μX é uma configuração parcial de estados, $\mu X \subseteq X$. Um estado s pertence a μX se e somente se s permanece ativo desde o início do passo
2. $\mu\Pi$ acumula aos elementos do estímulo externo Π os eventos ocorridos ao longo do passo, $\Pi \subset \mu\Pi$. O conjunto resultante é formado por eventos tratados como *simultâneos*, independentemente da duração da execução de um passo.
3. $\mu\Theta$ contém todas as variáveis lógicas cujos valores são verdadeiros, ou seja, são verdadeiros desde o início do passo ou tornaram-se verdadeiras a partir de algum micro-passo.
4. $\mu\xi$ fornece o valor corrente de uma expressão numérica $\mu\xi(v)$ ou aquele disponível no início do passo $\mu\xi(\mathbf{step}(v))$. A definição de $\mu\xi$ estabelece como uma expressão numérica é avaliada em Xchart.

   $$\mu\xi : \{v, \mathbf{step}(v) \mid v \in V\} \rightarrow \mathbb{Z}$$

   onde

- $\mu\xi(\mathbf{step}(v)) := \xi(v)$
- $\mu\xi(\mathbf{step}(\mathbf{step}(v))) := \mu\xi(\mathbf{step}(v))$
- Se o valor corrente (no interior do passo onde é consultado) da variável v é k, então $\mu\xi(v) := k$
- Se $k \in \mathcal{N}$ então $\mu\xi(k) := \mu\xi(\mathbf{step}(k)) := \xi(k)$
- Se $x, y \in V$, $op_n \in OP_N$ então $\mu\xi(x\ op_n\ y) := \mu\xi(x)\ op_n\ \mu\xi(y)$
- Se $x, y \in V$ e $op_n \in OP_N$ então $\mu\xi(\mathbf{step}(x\ op_n\ y)) := \mu\xi(\mathbf{step}(x))\ op_n\ \mu\xi(\mathbf{step}(y))$

5. μY é uma configuração parcial de estados. Todo estado básico s ativado em micro-passos anteriores é elemento desse conjunto. Durante a execução de um passo tem-se a seguinte invariante: $X \cap \mu Y = \phi$.

6. μZ é o conjunto de regras já executadas em micro-passos anteriores do passo em questão, $\mu Z \subseteq R$.

Micro-configuração inicial

$\mu\Upsilon_0 := (X, \Pi, \Theta, \mu\xi_0, \Omega, \phi, \phi)$ é a micro-configuração inicial para um estímulo externo Π sinalizado para a instância $\mathcal{I}$ cuja configuração é $\Upsilon := (X, \Theta, \xi, \Omega, K, \mathcal{X}_0^{i-1})$.

Micro-configuração sucessora

Dado duas micro-configurações de sistema $\mu\Upsilon_1 = (\mu X_1, \mu\Pi_1, \mu\Theta_1, \mu\xi_1, \mu\Omega_1, \mu Y_1, \mu Z_1)$ e $\mu\Upsilon := (\mu X, \mu\Pi, \mu\Theta, \mu\xi, \mu\Omega, \mu Y, \mu Z)$, ambas relativas a $\Upsilon := (X, \Theta, \xi, \Omega, K, \mathcal{X}_0^{i-1})$ e ao estímulo externo Π, diz-se que $\mu\Upsilon$ é uma sucessora possível de $\mu\Upsilon_1$, $\mu\Upsilon_1 \preccurlyeq \mu\Upsilon$, se e somente se:

1. $\mu X \subseteq \mu X_1$
2. $\mu\Pi_1 \subseteq \mu\Pi$
3. $\mu Y_1 \subseteq \mu Y$
4. $\mu Z_1 \subseteq \mu Z$

4.5.9 Micro-configuração Transiente

A execução de um micro-passo envolve a execução da ação total pertinente a esse micro-passo e a desativação/ativação de estados entre outras operações. A execução de cada elemento da ação total pode interferir na execução dos elementos seguintes. Por exemplo,

se a atribuição **v := 1** é seguida da **u := v**, então o valor atribuído à variável **u** será 1, pois esse é o valor da variável **v** após a execução da primeira atribuição. Analogamente, a execução de **x := ny(e)** coloca o valor falso na variável **x** se alguma ação anterior a essa atribuição é **raise(e)** ou o evento **e** pertence ao estímulo que desencadeou o passo em questão.

O conceito de micro-configuração captura o estado de uma instância entre micro-passos, ou seja, no interior de um passo. Essa micro-configuração é utilizada durante a identificação de regras habilitadas e relevantes, que possivelmente irão compor um micro-passo. Após a identificação de um micro-passo, essa micro-configuração deve sofrer pequenas alterações para contemplar as mudanças provocadas pela execução do micro-passo identificado. Micro-configuração transiente é uma micro-configuração que captura o estado de uma instância durante a execução de um micro-passo, ou seja, entre a execução dos elementos (ações) que compõem a ação total provocada pelo micro-passo. Uma micro-configuração transiente é utilizada apenas durante o processo de execução de um micro-passo.

Conforme as definições abaixo

- $\mu\Upsilon := (\mu X, \mu\Pi, \mu\Theta, \mu\xi, \mu\Omega, \mu Y, \mu Z)$
- $\mu\Delta := \{r_1, r_2, \ldots, r_k\}$ é o micro-passo *em execução* na $\mu\Upsilon$.
- $a := a_1; a_2; \cdots; a_n$ é a ação total relativa a $\mu\Delta$
- r_l^i é o elemento de $\mu\Delta$ cuja ação total contém a ação a_i, onde $1 \leqslant i \leqslant n$ e $1 \leqslant l \leqslant k$.

a micro-configuração transiente inicial $\mu\Upsilon_0^T$ e a micro-configuração transiente $\mu\Upsilon_i^T$ após a execução da ação a_i, $1 \leqslant i \leqslant n$ são definidos a seguir.

Micro-configuração transiente inicial

A micro-configuração transiente inicial relativa a um micro-passo $\mu\Delta$ e à micro-configuração $\mu\Upsilon$ é dada por

$$\mu\Upsilon_0^T := (\mu X_0^T, \mu\Pi, \mu\Theta, \mu\xi, \mu\Omega_0^T, \mu Y_0^T, \mu Z_0^T)$$

onde, para todo $1 \leqslant l \leqslant k$:

- $\mu X_0^T := \mu X - \left(\mu X \bigcap_{\forall \Lambda(r_l) \in T} \rho^*(\alpha(\Lambda(r_l))) \right)$

- $\mu Y_0^T := \mu Y \bigcup_{\forall \Lambda(r_l) \in R} C(\{\Lambda(r_l)\}, \mathcal{X}_0^n)$
- $\mu\Omega_0^T := \mu\Omega - \left\{ h \mid h \in H \text{ e } \gamma(h) \in S_a^{\Lambda(r_l) \in T} \right\}$
- $\mu Z_0^T := \mu Z \cup \mu\Delta$

Micro-configuração transiente $\mu\Upsilon_i^T$

A micro-configuração transiente $\mu\Upsilon_i^T$, obtida imediatamente após a execução da ação a_i, $1 \leqslant i \leqslant n$, é dada por

$$\mu\Upsilon_i^T := (\mu X_0^T, \mu\Pi_i^T, \mu\Theta_i^T, \mu\xi_i^T, \mu\Omega_0^T, \mu Y_0^T, \mu Z_0^T)$$

onde

1. $\mu X_0^T, \mu\Omega_0^T, \mu Y_0^T$ e μZ_0^T são definidos pela micro-configuração transiente inicial $\mu\Upsilon_0^T$. Esses elementos não alteram o seu conteúdo durante a execução de ações. Os valores pertinentes são obtidos no início do micro-passo e permanecem até que um novo micro-passo os altere.
2. Se $a_i = \mathbf{raise(e)}$ então $\mu\Pi_i^T := \mu\Pi_{i-1}^T \cup \{e\}$, caso contrário, $\mu\Pi_i^T := \mu\Pi_{i-1}^T$
3. Se $a_i = \mathbf{wait(e, v)}$ então a instância torna-se inoperante até que o evento **e** ocorra ou o tempo fornecido em **v** expire.
4. Se $a_i = \mathbf{sync(a, p)}$ então a atividade **a** é sinalizada na porta **p** e, enquanto essa atividade não for finalizada a instância permanece inoperante.
5. Se $a_i = \mathbf{call(x, n)}$ então uma instância **n** do Xchart **x** é criada. A instância que executa esta ação permanece inoperante até que a instância **n** seja finalizada.
6. Se $a_i = \mathbf{start(a, p)}$ então a atividade **a** é iniciada na porta **p**.
7. Se $a_i = \mathbf{stop(a, p)}$ então a atividade **a** é finalizada na porta **p**.
8. Se $a_i = \mathbf{suspend(a, p)}$ então a atividade **a** torna-se suspensa na porta **p**.
9. Se $a_i = \mathbf{resume(a, p)}$ então a atividade **a** tem sua execução retomada na porta **p**.
10. Se $a_i = \mathbf{v := exp}$, $\mathbf{v} \in V_N$ e $\mathbf{exp} \in V$, então $\mu\xi_i^T(\mathbf{v}) := \mu\xi_{i-1}^T(\mathbf{exp})$
11. Se $a_i = \mathbf{c := exp}, \mathbf{c} \in V_L$ e $\mathbf{exp} \in C$, então
    - Se $\mu\Upsilon_{i-1}^T \rightarrow \mathbf{exp}$ então $\mu\Theta_i^T := \mu\Theta_{i-1}^T \cup \{\, \mathbf{c} \,\}$

- Se $not(\mu\Upsilon^T_{i-1} \rightarrow \mathbf{exp})$ então $\mu\Theta^T_i := \mu\Theta^T_{i-1} - \{\mathbf{c}\}$
  As definições de $\rightarrow$ e $\leftarrow$, fornecidas nas seções seguintes, aplicam-se à micro-configurações. Uma micro-configuração transiente, contudo, define uma micro-configuração de forma trivial (Seção 4.5.10).

12. Se a_i =**clear(h)** e $h \in H$, então o símbolo de história torna-se "limpo". Ou seja, a próxima ativação seguindo esse símbolo irá conduzir a execução de transições iniciais. Em micro-passo posterior, o símbolo de história pode tornar-se "sujo".

4.5.10 Micro-configuração obtida a partir de $\mu\Upsilon^T_i$

Dada uma ação total $a := a_1; a_2; \cdots; a_n$, cada uma das micro-configurações transientes $\mu\Upsilon^T_i$, $1 \leq i \leq n$, induz uma micro-configuração correspondente. Em particular, a execução do último elemento de uma ação total dá origem a uma nova micro-configuração. Ou seja, a execução da ação a_n dá origem a micro-configuração resultante, conforme definição de micro-configuração resultante (pág. 162).
Durante a execução de um passo transições podem ser executadas. Em conseqüência, estados tornam-se inativos, reduzindo o conjunto de estados que permanecem ativos, enquanto outros tornam-se ativos e são acumulados em μY (item 1 acima). Ações das regras do passo podem gerar eventos que são acumulados em $\mu\Pi_i$ (item 2). Se regras não são executadas e eventos não são gerados, então a igualdade presente nesses itens modelam tal situação.
Quando um estímulo externo é sinalizado ou ao fim da execução de um micro-passo, tem-se uma micro-configuração resultante que pode provocar a execução de regras. Se for o caso, elas formarão um novo micro-passo onde cada uma de suas regras estão, necessariamente, habilitadas conforme a micro-configuração. Abaixo segue a definição formal de gatilho habilitado, cuja definição é empregada na identificação de regras habilitadas.

4.5.11 Evento *ocorre* em Micro-configuração

Na identificação de regras habilitadas é necessário estabelecer se determinado evento *ocorre* conforme uma determinada micro-configuração. Os itens seguintes fornecem esta definição. Eles compartilham os elementos abaixo:

- $\mu\Upsilon := (\mu X, \mu\Pi, \mu\Theta, \mu\xi, \mu\Omega, \mu Y, \mu Z)$
- $\mu\Upsilon$ é relativa à $\Upsilon := (X, \Theta, \xi, \Omega, K, \mathcal{X}_0^{i-1})$.
- Π o estímulo externo tratado na Υ.
- $\mu\Upsilon_1$ é uma micro-configuração em decorrente de Π.

- $\langle \mu\Pi, \mu\Upsilon_1 \rangle \stackrel{\mu\Delta}{\rightarrow} \mu\Upsilon$, ou seja, $\mu\Upsilon_1 \preccurlyeq \mu\Upsilon$ onde $\mu\Delta$ é um micro-passo executado na $\mu\Upsilon_1$ e $\mu\Pi$ é o conjunto Π acumulado de eventos gerados por micro-passos que conduziram à configuração $\mu\Upsilon_1$.
- $\mu\Upsilon_0$ é a micro-configuração inicial relativa à Υ.

Seja e um evento de uma regra r. Diz-se que e ocorre em $\mu\Upsilon$, $\mu\Upsilon \leftarrow e$, conforme segue:

1. $\mu\Upsilon \leftarrow \lambda$
2. Se $e \in C_E$ então $\mu\Upsilon \leftarrow e$ se e somente se $\exists e_p \in \mu\Pi$ tal que e_p é instância da classe c tal que $c \in \varrho^*(e)$
3. Se $c \in C$ então $\mu\Upsilon \leftarrow \mathbf{fs}(c)$ se uma das condições abaixo for satisfeita:
   - $\mu\Upsilon_1 \leftarrow \mathbf{fs}(c)$ ou
   - $\mu\Upsilon_1 \rightarrow c$ e $c \stackrel{\mu\Delta}{\leftarrow} \mathbf{F}$. Ou seja, c era verdadeira na micro-configuração anterior (veja definição de $\rightarrow$ a seguir) e o $\mu\Upsilon$ atribuiu o valor falso a variável c.
4. Se $c \in C$ então $\mu\Upsilon \leftarrow \mathbf{tr}(c)$ se uma das condições abaixo for satisfeita:
   - $\mu\Upsilon_1 \leftarrow \mathbf{tr}(c)$ ou
   - $\mu\Upsilon_1 \rightarrow \mathbf{not}(c)$ e $c \stackrel{\mu\Delta}{\leftarrow} \mathbf{T}$
5. Se $v \in V_L$ ou $v \in V_N$ então $\mu\Upsilon \leftarrow \mathbf{ch}(v)$ se
   - $\mu\Upsilon_1 \leftarrow \mathbf{ch}(v)$ ou
   - $\mu\Delta \models \mathbf{ch}(v)$
6. Se $s \in S$ então $\mu\Upsilon \leftarrow \mathbf{ex}(s)$ se
   - $\mu\Upsilon_1 \leftarrow \mathbf{ex}(s)$ ou
   - $\mu\Delta \models \mathbf{ex}(s)$
7. Se $s \in S$ então $\mu\Upsilon \leftarrow \mathbf{en}(s)$ se
   - $\mu\Upsilon_1 \leftarrow \mathbf{en}(s)$ ou
   - $\mu\Delta \models \mathbf{en}(s)$
8. Se s é um estado de um Xchart e i uma instância desse Xchart, $i \neq \mathcal{I}$, então:
   - $\mu\Upsilon \leftarrow \mathbf{ex}(s,i) \Leftrightarrow \mathbf{ex}(s,i) \in \Pi$

- $\mu\Upsilon \leftarrow \mathbf{en}(s,i) \Leftrightarrow \mathbf{en}(s,i) \in \Pi$

9. Se $v \in V$ então $\mu\Upsilon \leftarrow \mathbf{after}(v)$ se e somente se $alarme(r,id) \in \Pi$ e id identifica o evento $\mathbf{after}(v)$ da regra r.

10. Se $v \in V$ então $\mu\Upsilon \leftarrow \mathbf{every}(v)$ se e somente se $alarme(r,id) \in \Pi$ e id identifica o evento $\mathbf{every}(v)$ da regra r.

11. Se $h,m,s \in \mathcal{N}$ então $\mu\Upsilon \leftarrow \mathbf{at}(h:m:s)$ se e somente se $alarme(r,id) \in \Pi$ e id identifica o evento $\mathbf{at}(h:m:s)$ da regra r.

12. Se $a \in A_T$ então

    - $\mu\Upsilon \leftarrow \mathbf{started}(a) \Leftrightarrow \mathbf{started}(a) \in \Pi$
    - $\mu\Upsilon \leftarrow \mathbf{stopped}(a) \Leftrightarrow \mathbf{stopped}(a) \in \Pi$

13. Se $e_1, e_2 \in E$ então:

    - $\mu\Upsilon \leftarrow e_1$ **and** e_2 se e somente se $\mu\Upsilon \leftarrow e_1$ e $\mu\Upsilon \leftarrow e_2$
    - $\mu\Upsilon \leftarrow e_1$ **or** e_2 se e somente se $\mu\Upsilon \leftarrow e_1$ ou $\mu\Upsilon \leftarrow e_2$

14. Se $e \in E$ então $\mu\Upsilon \leftarrow \mathbf{not}(e)$ se e somente se e não ocorre, isto é, $not(\mu\Upsilon \leftarrow e)$.

4.5.12 Micro-configuração *satisfaz* Condição

Seja c uma expressão lógica, $c \in C$. Diz-se que $\mu\Upsilon$ *satisfaz* c, $\mu\Upsilon \rightarrow c$, conforme segue:

1. $\mu\Upsilon \rightarrow \kappa$
2. $\mu\Upsilon \rightarrow \mathbf{T}$
3. $\mu\Upsilon$ não satisfaz $\mathbf{F}$, i.e., $not(\mu\Upsilon \rightarrow \mathbf{F})$
4. Se $c \in V_L$ então:
    - $\mu\Upsilon \rightarrow c \Leftrightarrow c \in \mu\Theta$
    - $\mu\Upsilon \rightarrow \mathbf{step}(c) \Leftrightarrow c \in \Theta$
5. Se $s \in S$ então:
    - $\mu\Upsilon \rightarrow \mathbf{step}(\mathbf{in}(s)) \Leftrightarrow X \cap \rho^*(s) \neq \phi$
    - $\mu\Upsilon \rightarrow \mathbf{in}(s)$ se e somente se $\mu X \cap \rho^*(s) \neq \phi$ ou $\mu Y \cap \rho^*(s) \neq \phi$

6. Se s é um estado de um Xchart e i uma instância desse Xchart, $i \neq \mathcal{I}$, então:

   - $\mu\Upsilon \rightarrow \mathbf{in}(s,i) \Leftrightarrow \mathbf{in}(s,i) \in \Theta$
   - $\mu\Upsilon \rightarrow \mathbf{step}(\mathbf{in}(s,i)) \Leftrightarrow \mathbf{in}(s,i) \in \Theta$
   - $\mu\Upsilon \rightarrow \mathbf{ex}(s,i) \Leftrightarrow \mathbf{ex}(s,i) \in \Theta$
   - $\mu\Upsilon \rightarrow \mathbf{step}(\mathbf{ex}(s,i)) \Leftrightarrow \mathbf{ex}(s,i) \in \Theta$

7. Se $e \in E$ então:

   - $\mu\Upsilon \rightarrow \mathbf{ny}(e) \Leftrightarrow not(\mu\Upsilon \leftarrow e)$
   - $\mu\Upsilon \rightarrow \mathbf{step}(\mathbf{ny}(e)) \Leftrightarrow \mu\Upsilon_0 \rightarrow \mathbf{ny}(e)$

8. Se $u, v \in V$, $R \in OP_L$ então:

   - $\mu\Upsilon \rightarrow uRv \Leftrightarrow \mu\xi(u)\ R\ \mu\xi(v)$
   - $\mu\Upsilon \rightarrow \mathbf{step}(uRv)$ se e somente se $\mu\Upsilon_0 \rightarrow \mathbf{step}(uRv)$

9. Se $a \in A_T$ então

   - $\mu\Upsilon \rightarrow \mathbf{active}(a) \Leftrightarrow \mathbf{active}(a) \in \mu\Theta$
   - $\mu\Upsilon \rightarrow \mathbf{step}(\mathbf{active}(a))$ se e somente se $\mu\Upsilon_0 \rightarrow \mathbf{active}(a)$
   - $\mu\Upsilon \rightarrow \mathbf{hanging}(a)$ se e somente se $\mathbf{hanging}(a) \in \mu\Theta$
   - $\mu\Upsilon \rightarrow \mathbf{step}(\mathbf{hanging}(a))$ se e somente se $\mu\Upsilon_0 \rightarrow \mathbf{hanging}(a)$

10. Se $c_1, c_2 \in C$:

    - $\mu\Upsilon \rightarrow c_1 \mathbf{\ and\ } c_2 \Leftrightarrow \mu\Upsilon \rightarrow c_1$ e $\mu\Upsilon \rightarrow c_2$
    - $\mu\Upsilon \rightarrow \mathbf{step}(c_1 \mathbf{\ and\ } c_2)$ se e somente se $\mu\Upsilon \rightarrow \mathbf{step}(c_1)$ e $\mu\Upsilon \rightarrow \mathbf{step}(c_2)$
    - $\mu\Upsilon \rightarrow c_1 \mathbf{\ or\ } c_2 \Leftrightarrow \mu\Upsilon \rightarrow c_1$ ou $\mu\Upsilon \rightarrow c_2$
    - $\mu\Upsilon \rightarrow \mathbf{step}(c_1 \mathbf{\ or\ } c_2)$ se e somente se $\mu\Upsilon \rightarrow \mathbf{step}(c_1)$ ou $\mu\Upsilon \rightarrow \mathbf{step}(c_2)$
    - $\mu\Upsilon \rightarrow \mathbf{not}(c_1) \Leftrightarrow not(\mu\Upsilon \rightarrow c_1)$
    - $\mu\Upsilon \rightarrow \mathbf{step}(\mathbf{not}(c_1))$ se e somente se $not(\mu\Upsilon \rightarrow \mathbf{step}(c_1))$

11. Se $c \in C$ então $\mu\Upsilon \rightarrow \mathbf{step}(\mathbf{step}(c)) \Leftrightarrow \mu\Upsilon \rightarrow \mathbf{step}(c)$

4.5.13 Gatilho Habilitado

Dado uma micro-configuração de sistema $\mu\Upsilon$, diz-se que $\mu\Upsilon$ *habilita* o gatilho $g \in G$, $\mu\Upsilon \rightsquigarrow g$, $g := e[c]$, $e \in E$ e $c \in C$, conforme segue:

$$\mu\Upsilon \rightsquigarrow g \Leftrightarrow \mu\Upsilon \leftarrow e \text{ e } \mu\Upsilon \rightarrow c$$

4.5.14 Regra Habilitada, Relevante e Regras Consistentes

Um micro-passo é necessariamente composto por regras e transições habilitadas, relevantes e consistentes. Estes três atributos são definidos nessa seção. Toda transição está associada a uma regra. A definição abaixo leva em consideração regras associadas a estados e aquelas associadas a transição. Quando uma regra estiver associada a transição, então a transição possui o mesmo atributo que a regra pertinente. Ou seja, se a regra r rotulando uma transição está habilitada, então a transição pertinente também está habilitada. Essa seção também fornece os critérios para identificar um subconjunto de regras consistentes de um conjunto de regras habilitadas, relevantes e não-consistentes.

Regra habilitada

Dada a regra $r = g : a$ e a micro-configuração $\mu\Upsilon$, r está *habilitada* em $\mu\Upsilon$ se e somente se $\mu\Upsilon \rightsquigarrow g$.

Regra relevante

Uma regra r é *relevante* na $\mu\Upsilon$ se e somente se as condições abaixo são satisfeitas:

1. Se $\Lambda(r) \in S$ então $\rho^*(\Lambda(r)) \cap \mu X \neq \phi$
2. Se $\Lambda(r) \in T$ e $\Lambda(r) = (B, Y)$ então $\exists x \in \mu X$ tal que $b \in B \Rightarrow x \in \rho^*(b)$
3. $r \notin \mu Z$

Regras consistentes

Sejam L_T e L_S subconjuntos disjuntos[2] de um conjunto de regras L onde

$$L_T := \{t \mid \exists r \in L, \Lambda(r) \in T \text{ e } \Lambda(r) = t\}$$

$$L_S := \{s \mid \exists r \in L, \Lambda(r) \in S \text{ e } \Lambda(r) = s\}$$

L é dito *consistente* se e somente se

[2] A e B são ditos conjuntos disjuntos se $A \cap B = \phi$.

1. $\forall t_1, t_2 \in L_T \Rightarrow \alpha(t_1) \perp \alpha(t_2)$
2. $\nexists s \in L_S \mid s \in \bigcup_{t \in L_T} S_d^t$

Dessa definição uma transição final é inconsistente com qualquer outra regra/transição. A seção seguinte estabelece um processo de identificação de um subconjunto consistente de um conjunto inconsistente de regras.

Seleção de regras consistentes

Dado um conjunto $L = \{r_1, r_2, \ldots, r_n\}$ de regras habilitadas e relevantes, apenas um subconjunto maximal de L, que também é consistente, pode formar um micro-passo. Este subconjunto é identificado obedecendo-se, na ordem em que aparecem, as diretrizes abaixo até que a inconsistência entre duas regras seja resolvida pela identificação de uma delas a ser eliminada do conjunto L. Ou seja, se a diretriz 1 não eliminar a inconsistência, então passa-se a aplicar a diretriz 2 e assim por diante, até que a diretriz 4, eventualmente, selecione de forma arbitrária uma a ser eliminada. Seja $r_i, r_j \in L$ não-consistentes para algum $1 \leqslant i, j \leqslant n$ e $i \neq j$. Estas diretrizes eliminam elementos de L para torná-lo consistente.

1. Se $\Lambda(r_i) \in T$ e $\Lambda(r_j) \in R$, então a regra r_j é eliminada do conjunto L. Nesse caso, o estado que contém a regra r_j será desativado pela transição r_i. Toda e qualquer transição tem prioridade sobre toda e qualquer regra associada a um estado.
2. Se $\mho(r_i) > \mho(r_j)$, sem perda de generalidade, então r_j é eliminada do conjunto L.
3. Se $\rho^*(\alpha(\Lambda(r_i))) \cap \rho^*(\alpha(\Lambda(r_j))) \neq \phi$ e
   $\alpha(\Lambda(r_i)) \in \rho^+(\Lambda(r_j))$, então r_j é eliminada do L.
4. Se as condições anteriores não elimina a inconsistência entre as regras r_i e r_j, então arbitrariamente uma delas é selecionada e removida do conjunto L.

4.5.15 Micro-passo

Micro-passos capturam a ordem em que os subconjuntos que formam uma partição de um passo são executados. Um micro-passo é um conjunto de regras habilitadas, relevantes e consistentes identificadas em uma dada micro-configuração de uma instância. As transições de um micro-passo são executadas simultaneamente, enquanto as regras associadas a estados são executadas em uma seqüência bem definida. A execução de um micro-passo modifica o valor de variáveis, o conjunto de estados ativos, gera eventos e outras, o que define uma nova micro-configuração. Este processo continua até que não seja mais

possível identificar um conjunto de regras habilitadas, relevantes e consistentes. Ao fim deste processo, esta seqüência torna-se um passo observável externamente.

Durante a execução de um micro-passo, **step**(c) e **step**(v) fornecem os valores de expressões lógicas e numéricas disponíveis no início do passo. Se o valor corrente dessas expressões são desejados, então deve-se fazer referência a c e v. Estes valores possivelmente são diferentes daqueles disponíveis no início do passo, pois podem ter sido alterados pela execução de micro-passos anteriores. **ny**(e) também é sensível aos micro-passos. Esta construção assume valor verdadeiro se e não ocorreu no presente passo, i.e., desde o primeiro micro-passo na seqüência. Por outro lado, eventos, condições e expressões que não envolvem **step** e **ny** não são alterados nos micro-passos de um passo.

Restrições

Seja $\mu\Upsilon$ uma micro-configuração de uma instância. $\mu\Delta$ é um micro-passo de $\mu\Upsilon$ se e somente se:

1. $\mu\Delta$ é um conjunto não vazio de regras.
2. Se $r \in \mu\Delta$ então r está habilitada na $\mu\Upsilon$.
3. $\mu\Delta$ é relevante.
4. $\mu\Delta$ é um conjunto de regras consistentes.
5. $r \in \mu\Delta \Rightarrow r \notin \mu Z$.

Concorrência em Micro-passo

As ações decorrentes das transições de um micro-passo são executadas concorrentemente. Ou seja, se $t_1, t_2 \in T$ e $t_1, t_2 \in \mu\Delta$, então as ações $\Gamma(t_1)$ são executadas concorrentemente com as ações $\Gamma(t_2)$. As ações causadas por cada uma dessas transições, contudo, são executadas conforme a seqüência de ações pertinentes, ou seja, não há concorrência entre as ações de $\Gamma(t_1)$ ou de $\Gamma(t_2)$.
As ações de um micro-passo decorrentes de regras associadas a estados são executadas conforme a hierarquia e os tipos de estados envolvidos. As regras associadas a um estado s são executadas concorrentemente assim como as regras associadas a estados s_1 e s_2 tais que $s_1 \perp s_2$. Se s_1 é ancestral de s_2, sem perda de generalidade, então as regras de s_1 são executadas antes das regras de s_2. Ou seja, entre estados que mantém uma relação ancestral/descendente, as regras associadas ao estado ancestral têm precedência sobre aquelas associadas ao estado descendente. A primeira regra associada a um estado

descendente é executada somente após o fim da execução de todas as regras associadas ao estado ancestral.

4.5.16 Efeitos de um Micro-passo

Um micro-passo pode causar mudanças na configuração de uma instância ou de outras instâncias do $\mathfrak{SDX}$ em execução. Abaixo são identificadas (definições 8, 9, 10, 11 e 12) as mudanças ocorridas em decorrência da execução de um micro-passo. Para esta finalidade são empregadas as seguintes definições:

- $\mu\Upsilon := (\mu X, \mu\Pi, \mu\Theta, \mu\xi, \mu\Omega, \mu Y, \mu Z)$
- $\mu\Upsilon_1 := (\mu X_1, \mu\Pi_1, \mu\Theta_1, \mu\xi_1, \mu\Omega_1, \mu Y_1, \mu Z_1)$
- $\Upsilon := (X, \Theta, \xi, \Omega, K, \mathcal{X}_0^{i-1})$
- $\mu\Upsilon_1 \preccurlyeq \mu\Upsilon$ (ambas relativas a Υ)
- $\mu\Delta := \{r_1, r_2, \ldots, r_k\}$ é um micro-passo executado na $\mu\Upsilon_1$
- $\mathcal{X}_0^n := (X_0, X_1, \ldots, X_n)$
- $a := a_0; a_1; \cdots; a_n$ (ação total causada pelo $\mu\Delta$)

Ordem de Execução de Ações

Dada a ação total a, $1 \leqslant i, j \leqslant n$, $i \neq j$, se a ação a_i é executada antes de a_j, então $i < j$.

Alteração do Valor de Variáveis

Diz-se que $\mu\Delta$ *atribui* o valor k, lógico ou numérico, à variável v do tipo correspondente, o que é denotado por $v \overset{\mu\Delta}{\leftarrow} k$, conforme segue:

1. Se $v \in V_L$ e $c \in C$ então $v \overset{\mu\Delta}{\leftarrow} \mathbf{T} \Leftrightarrow \exists i \mid a_i = v := c$ e $\mu\Upsilon \rightarrow c$
2. Se $v \in V_L$ e $c \in C$ então $v \overset{\mu\Delta}{\leftarrow} \mathbf{F} \Leftrightarrow \exists i \mid a_i = v := c$ e $\mu\Upsilon \rightarrow \mathbf{not}(c)$
3. Se $v \in V_N$, $x \in \mathbb{Z}$ e $u \in V$ então $v \overset{\mu\Delta}{\leftarrow} x \Leftrightarrow \exists i \mid a_i = v := u$ e $\mu\xi(u) = x$

Alteração do Valor de Expressões

Diz-se que $\mu\Delta$ *atribui* o valor k, lógico ou numérico, à expressão *exp* do tipo correspondente, o que é denotado por $exp \overset{\mu\Delta}{\Leftarrow} k$ se uma das condições abaixo é satisfeita:

1. $\mu\Delta$ não altera o valor de *exp*, ou seja, o valor de *exp* permanece k ou
2. $\mu\Delta$ altera o valor de *exp* tal que o novo valor de *exp* torna-se k

Símbolo de História — Esquecendo o Passado

Diz-se que $\mu\Delta$ limpa o símbolo de história h se $\exists i \mid a_i = \mathbf{clear}(h)$ e $\nexists j, r, s \mid j > i, s = \gamma(h), r \in \mu\Delta, \Lambda(r) \in T$ e $s \in S_a^{\Lambda(r)}$. Por outro lado, se $\exists j, r, s \mid j > i, s = \gamma(h), r \in \mu\Delta, \Lambda(r) \in T$ e $s \in S_a^{\Lambda(r)}$ então diz-se que $\mu\Delta$ visita o estado s.

Gerando Eventos

Diz-se que $\mu\Delta$ *gera* e, o que é denotado por $\mu\Delta \models e$, $e \in E$, conforme segue:

1. Se $e \in E_p$ e $c \in C_E$ então $\mu\Delta \models e$ se e somente se $\exists\, i \mid a_i = \mathbf{raise}(c)$ e e é o evento primitivo gerado pela execução de a_i.
2. Se $a \in A_T$ e $p \in P$ então $\mu\Delta \models \mathbf{started}(a, p) \Leftrightarrow \exists\, i \mid a_i = \mathbf{start}(a, p)$
3. Se $a \in A_T$ e $p \in P$ então $\mu\Delta \models \mathbf{stopped}(a, p) \Leftrightarrow \exists\, i \mid a_i = \mathbf{stop}(a, p)$
4. Se $s \in S$, então
   - $\mu\Delta \models \mathbf{ex}(s) \Leftrightarrow \exists r \in \mu\Delta \mid s \in S_d^{\Lambda(r)}$
   - $\mu\Delta \models \mathbf{en}(s) \Leftrightarrow \exists r \in \mu\Delta \mid s \in S_a^{\Lambda(r)}$
5. Se $s \in \overline{D_S}$ e $i \in \mathfrak{I}$, então
   - $\mu\Delta \models \mathbf{ex}(s, i) \Leftrightarrow \exists r \in \mu\Delta \mid s \in S_d^{\Lambda(r)}$
   - $\mu\Delta \models \mathbf{en}(s, i) \Leftrightarrow \exists r \in \mu\Delta \mid s \in S_a^{\Lambda(r)}$
6. Se $v \in V_L \cup V_N$ e k é um valor correspondente, então $\mu\Delta \models \mathbf{ch}(v) \Leftrightarrow v \overset{\mu\Delta}{\Leftarrow} k$

Micro-configuração Resultante

Diz-se que $\mu\Upsilon$ é obtida pela execução da ação total $a := a_1; a_2; \cdots; a_n$, relativa ao micro-passo $\mu\Delta$, que é obtido de uma micro-configuração $\mu\Upsilon_1$, conforme segue:

$$\mu\Upsilon := \mu\Upsilon_n^T$$

A Seção 4.5.10 apresenta detalhadamente o significado dessa definição.

4.5.17 Passo

Um passo Δ é uma seqüência maximal de micro-passos $\mu\Delta_0, \ldots, \mu\Delta_m$. Seja Υ uma configuração de instância. Δ é um passo de Υ onde:

1. $\mu\Delta_i$ é um micro-passo executado na micro-configuração $\mu\Upsilon_i$, $0 \leqslant i \leqslant m$.
2. $\mu\Upsilon_0$ é a micro-configuração inicial de Υ.
3. $\mu\Upsilon_{i+1}$ é a micro-configuração de sistema resultante da execução de $\mu\Delta_i$ para $0 \leqslant i \leqslant m$. Ou seja,

$$\langle \mu\Delta_i, \mu\Upsilon_i \rangle \rightarrow \mu\Upsilon_{i+1}$$

   e, naturalmente, $\mu\Upsilon_i \preccurlyeq \mu\Upsilon_{i+1}$.
4. O conjunto $\mu\Delta_0 \cup \mu\Delta_1 \cup \cdots \cup \mu\Delta_m$ é um conjunto consistente de regras.
5. Se a regra r está habilitada em $\mu\Upsilon_{m+1}$ então $\{r\} \cup \mu\Delta_0 \cup \cdots \cup \mu\Delta_m$ não é consistente, i.e., Δ é maximal. Na Figura 4-5, se os estados **A** e **B** estão ativos e ocorre o evento **e**, a transição para o estado **C** não irá ocorrer em uma suposta reação em cadeia de tamanho três, pois ela não forma, juntamente com as anteriores, um conjunto consistente de regras.

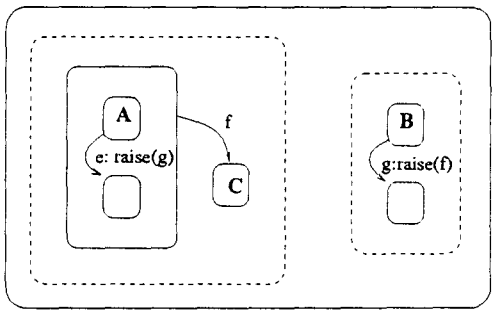


Figura 4-5: Cenário possível de regras não consistentes

6. Um passo Δ faz com que o sistema atinja uma nova configuração de sistema Υ', ou seja, $\langle \Delta, \Upsilon \rangle \rightarrow \Upsilon'$. A configuração obtida é aquela da última micro-configuração atingida pela seqüência de micro-passos que compõem um passo, $\Upsilon' = \mu\Upsilon_{m+1}$. Uma vez que não há transições habilitadas, relevantes e consistentes dessa última micro-configuração de sistema a instância $\mathcal{I}$ permanece nesta configuração à espera de um novo estímulo externo.

4.5.18 Configuração Resultante

$\Upsilon' := (X, \Pi, \Theta, \xi, \Omega, K, \mathcal{X}_0^i)$ é a configuração de sistema obtida da configuração Υ após a execução de $\Delta := (\mu\Delta_0, \ldots, \mu\Delta_m)$, ou seja, $\langle \Delta, \Upsilon \rangle \rightarrow \Upsilon'$ conforme definido abaixo:

1. $\mu\Upsilon_{m+1}$ é a micro-configuração resultante da execução de $\mu\Delta_m$ e na qual não há regras consistentes que podem ser executadas.

2. $X := \mu X_{m+1} \cup \mu Y_{m+1}$

3. Π é o conjunto de eventos primitivos ocorridos no intervalo I_i

4. Para $c \in V_L$, c é verdadeiro em $\sigma_i \Leftrightarrow \mu\Upsilon_{m+1} \rightarrow \mathbf{step}(c)$. Θ é o conjunto de condições primitivas verdadeiras no intervalo $[\sigma, \sigma_{i+1})$ para algum $\sigma \geqslant \sigma_i$.

5. Para $v \in V_N$, o valor de v no instante σ_{i+1} é x se e somente se $\mu\xi_{m+1}(\mathbf{step}(v)) = x$. $\xi(v) := x$ se o valor de v em $[\sigma, \sigma_{i+1})$ é x para algum $\sigma \geqslant \sigma_i$.

4.5.19 Conseqüências de um Passo

Seja Υ a configuração da instância $\mathcal{I}$ e o passo $\Delta := (\mu\Delta_0, \ldots, \mu\Delta_m)$ executado nessa configuração. A execução de Δ provoca, possivelmente, mudança de valores de variáveis e outras. Algumas delas, que são de interesse de outras instâncias, tornam-se visíveis apenas ao fim do passo. As mudanças de interesse externo são acumuladas no conjunto Π^g onde $r \in \Pi^g$ se e somente se:

- r refere-se à mudança do valor de uma variável global. Tal mudança pode ocasionar a ocorrência de eventos **ch**, **tr** e **fs**.

- r é um evento gerado pelo passo. Formalmente, $r \in \Pi^g$ se e somente se $\exists i \mid \mu\Delta_i \models r$, $0 \leq i \leq m$. Os eventos **tr** e **fs** podem ser gerados indiretamente através da ocorrência de **ch**.

- r é uma das ações de controle de atividades, ou seja, **sync**, **start**, **stop**, **resume** ou **suspend**. Nesses casos, tais elementos são depositados em portas à medida em que são gerados. Ou seja, o fim do passo não determina o instante em que tais reações tornam-se visíveis. A ação **sync**, por exemplo, exige o término da atividade pertinente (comunicação com o exterior) para que a reação em cadeia da instância possa prosseguir.

4.5.20 Execução de uma Instância

A seqüência $\{(\Upsilon_0, \Delta_0, \Pi_0^g), \ldots, (\Upsilon_i, \Delta_i, \Pi_i^g)\}, i \geqslant 0$, refere-se à execução de uma instância $\mathcal{I}$ onde:

1. $\Upsilon_0 := (X_0, \Pi_0, \Theta_0, \xi_0, \tau_0)$, X_0 é a configuração inicial e (Π_0, Θ_0, ξ_0) são estímulos externos ocorridos no primeiro intervalo de tempo I_0
2. Δ_j é um passo de Υ_j executado no instante de tempo σ_{j+1} para $1 \leqslant j \leqslant i$.
3. Υ_{j+1} é a configuração obtida de Υ_j após a execução do passo $\Delta_j, 1 \leqslant j \leqslant i$.
4. Π_j^g é o conjunto de reações produzidas pelo passo Δ_j dada a configuração Υ_j e $1 \leqslant j \leqslant i$.
5. Se $\exists j \mid i \leq j$, então Υ_j é a configuração final da instância, $\Delta_j := \phi$ e $\Pi_j^g := \phi$.

4.6 Tabelas de Símbolos Utilizados

Símbolo	Função	Pág.
$\models$	gera	162
$\perp$	ortogonal	129
$\leftarrow$	ocorre	154
$\rightarrow$	satisfaz	156
$v \overset{\Delta}{\leftarrow} k$	Δ atribui k a v	161
$\preccurlyeq$	sucede	151
$\rightsquigarrow$	habilita	158
$\equiv$	domínio não definido	130

Tabela 4.1: Notação empregada

Nome	Semântica
ch(n)	Retorna **T** se o valor de n foi alterado durante o passo em andamento.
step(x)	Obtém o valor de x no interior do micro-passo em andamento
en(s)	Análoga à semântica abaixo
ex(s)	Retorna **T** se em algum momento do passo atual o sistema passa de uma configuração que continha subestados de s para uma que não continha; mesmo que tenha voltado a conter.
fs(x)	entre estados exclusivos
in(s)	T se o sistema estava em s no início do passo em andamento.
ny(e)	Retorna T se o evento e não ocorreu no intervalo entre o início do passo em andamento e o início do micro-passo em andamento.
tr(x)	Retorna T (**T**) se x mudou do valor F (**F**) para T durante o passo em andamento. Se x é sempre T durante o passo $tr(x) = F$. Se x inicia-se com T e torna-se F e depois volta a ser T, tem-se $tr(c) = T$

Tabela 4.2: Funções "visíveis" externamente

Nome	Semântica
$\xi(x)$	Retorna o valor de x
$\mu\xi(x)$	Equivale a $\xi(x)$ ou $\xi(step(x))$ conforme a solução do conflito
$\alpha(X)$	Retorna o ancestral comum mais próximo do conjunto de estados X.

Tabela 4.3: Funções internas

Nome			Finalidade	Pág.
Alpha	α		identifica ancestral comum mais próximo	128
	α		ancestral de transição	137
	α^+		ancestral estrito	136
Beta	β		porta padrão	126
Gama	γ		associa símbolo de história a um estado	130
		Γ	seqüência de ações provocadas por regra	143
Delta	δ		hierarquia de ativação de estados por default	126
Delta		Υ	configuração de sistema	126
Epsilon	ϵ		ação nula	126
Eta	η		gatilho nulo	126
Teta		Θ	conjunto de condições primitivas externas	126
Vartheta	ϑ		regra nula	126
Kappa	κ		condição nula	126
Lambda	λ		evento nulo e elemento inexistente de vários conjuntos	126
Lambda		Λ	identifica estado ou transição associada a regra	133
Mu	μ		prefixo indicador de micro-passo e outros	126
Ksi	ξ		estabelece o valor de variável ou expressão numéricos no interior de um passo	126
Pi	π		identifica transição inicial a ser executada	138
Pi		Π	conjunto de eventos primitivos externos ocorridos no instante i	144
Ro	ρ		estabelece os descendentes de um estado	128
Ro	ρ^+	ρ^*	fechos de ρ	128
Varrho	ϱ		estabelece descendentes de classe de eventos	126
Varrho	ϱ^+	ϱ^*	fechos de ϱ	127
Sigma	Σ		conjunto de instantes de tempo típicos da execução de uma instância	134
Tau	τ		retém história recente de uma configuração	126
Úpsilon		Δ	conjunto de regras denominado de passo	126
Fi	ϕ		conjunto vazio	126
Fi		Φ	conjunto de estados fictícios	129
Psi	ψ		identifica tipo corrente de um estado	136
Psi	ψ_E		identifica tipo estrutural de um estado	128
Ômega	ω		associa estado fictício a estado	130
		Ω	símbolos de história sem informação disponível	147
Mho	$\mho$		estabelece prioridade de uma regra	133

Tabela 4.4: Letras gragas utilizadas

Nome	Descrição	Pág.
A	ações	132
A_T	atividades	127
C	expressões lógicas	126
C	função de configuração	139
C_E	classes de eventos	126
D	descrições de Xcharts	127
E	eventos	126
E_p	eventos primitivos	144
$\mathcal{F}$	função inflar	140
$\mathcal{F}_*$	função inflar estrela	141
G	gatilhos	133
H	conjunto de símbolos de história	130
H	conjunto de símbolos de história simples	130
H_*	conjunto de símbolos de história estrela	130
I	conjunto de identificadores de instâncias	127
$\mathcal{I}$	uma instância de um Xchart	134
$\mathfrak{I}$	conjunto de instâncias	135
K	conjunto de alarmes	145
$\mathcal{M}$	função maximizar	140
$\mathcal{N}$	conjunto de valores numéricos	126
OP_L	operadores lógicos	126
OP_N	operadores numéricos	126
P	portas	127
R	regras	133
$\mathcal{R}$	raiz	129
S	estados de um diagrama em D	128
S_a^t	estados ativados em conseqüência da transição t	142
S_d^t	estados desativados em conseqüência da transição t	141
$\vec{S_a^t}$	seqüência obtida de S_a^t	142
$\vec{S_d^t}$	seqüência obtida de S_d^t	142
$\mathfrak{SDX}$	Sistema Descrito em Xchart	126
T	transições	131
T_c	transições convencionais	131
T_i	transições iniciais	131
T_f	transições finais	131
T_{pf}	transições pseudo-finais	131
V	expressões numéricas	130
V_L	variáveis lógicas	127
V_N	variáveis numéricas	127
X	configuração de estados	137
$\mathbb{Z}$	conjunto dos inteiros	126

Tabela 4.5: Símbolos utilizados

Capítulo 5

Ambiente Xchart

To be useful a specification must be both precise and complete. Such a specification is, inherently, going to be lengthy and difficult either to analyze or to understand. Generally, specialized training is necessary to generate such a specification. These barriers are the motivation for the use of tools in the formal methods arena.
Working Group on Formal Methods in HCI and Software Engineering
Len Bass [128, pág. 15]

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. A primeira seção desse capítulo comenta as ferramentas disponíveis no ambiente e utilizadas durante a etapa de desenvolvimento (projeto e implementação) de um sistema interativo.

A Seção 5.2 apresenta o sistema de execução de Xchart (SE). Esse software 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 apresentada no capítulo anterior. As propostas existentes para softwares similares são pouco conhecidas. Algumas são comerciais, enquanto outras contemplam diagramas com poucos recursos. Entre elas é desconhecida, contudo, uma que ofereça execução distribuída de instâncias ou as noções citadas acima.

Projetistas não necessariamente precisam conhecer o complexo software que executa Xcharts. Programadores, contudo, deverão conectar o restante do sistema ao gerenciador de diálogo, fornecido pelo SE conforme diagramas em Xchart. Essa conexão é realizada via o protocolo IPX (Seção 5.3). O capítulo seguinte apresenta, além de exemplos do emprego de Xchart, como o ambiente pode ser útil à essa tarefa.

5.1 Desenvolvimento usando Xchart

O processo de desenvolvimento de uma interface envolvendo o emprego de Xchart é exibido na Figura 5-1. Esse processo assemelha-se ao de desenvolvimentos baseados em modelos conforme o processo descrito em [125]. Essa semelhança é natural, pois Xchart é uma proposta de extensão de modelos de diálogos empregados em abordagens baseadas em modelos.

Uma especificação em Xchart de um gerenciador de diálogo é produzida após a definição do projeto da interação da interface correspondente. Tal especificação é produzida pelo projetista do software da interface (Figura 5-1). O projetista utiliza o projeto da interação de uma interface para produzir o gerenciador de diálogo em Xchart usando o Editor de Xchart. Especificações em Xchart são armazenadas em meio estável através da linguagem TeXchart. O projetista não precisa conhecer a linguagem TeXchart. O editor transparente e automaticamente converte Xcharts (diagramas) em sentenças na forma de texto de TeXchart e vice-versa. Especificações em TeXchart, contudo, não são apropriadas para serem interpretadas em tempo de execução. O Sistema de Execução de Xchart (SE) utiliza um outro formato que é gerado pelo compilador TeXchart a partir de especificações em TeXchart. Em tempo de execução, o SE lê o conteúdo de um arquivo gerado pelo compilador TeXchart para executar instâncias de Xcharts da especificação.

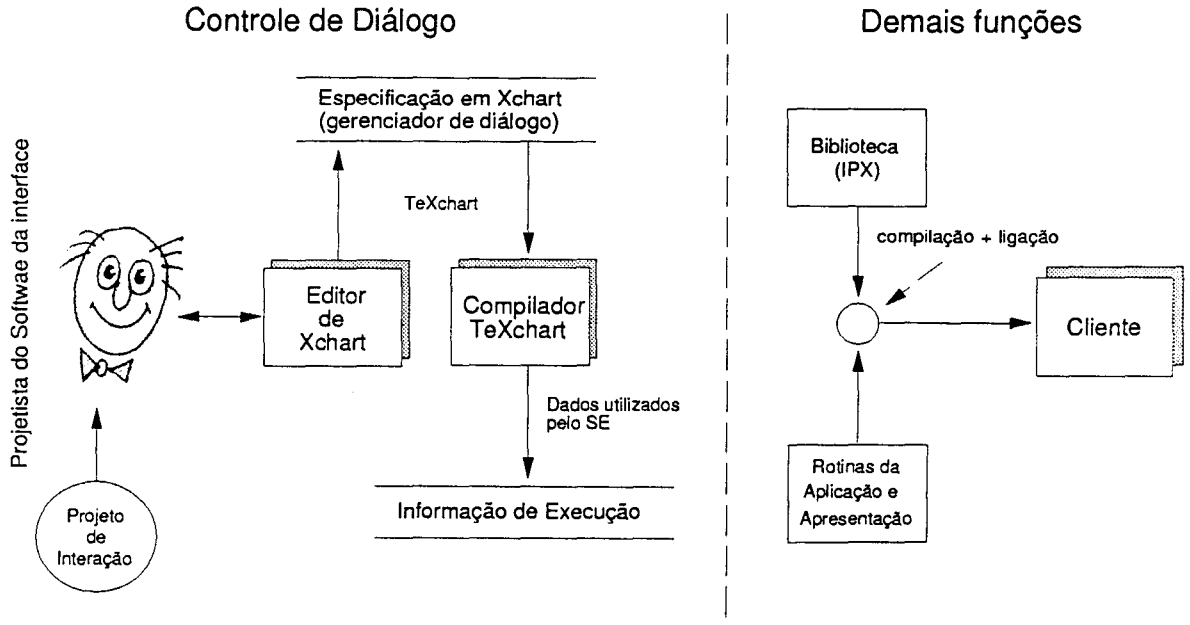


Figura 5-1: Processo de desenvolvimento usando Xchart

Um sistema interativo não compreende apenas as funções que podem ser capturadas

por Xchart ou realizadas pelo gerenciador de diálogo. As demais funções são realizadas por outros componentes. Aqueles componentes que interagem com instâncias de Xcharts (sinalizam estímulos externos e/ou recuperam reações provocadas por instâncias) são denominados de **clientes**.

Um cliente que realiza tarefas de apresentação recebe, em geral, eventos de baixo nível correspondentes às ações dos usuários e os converte em eventos de mais alto nível. Os eventos de baixo nível são fornecidos por toolkits. Os eventos de alto nível são, possivelmente, sinalizados para instâncias de Xcharts via a IPX. A Figura 5-2 ilustra essa organização. Um cliente de aplicação possui uma organização análoga. Possivelmente tal cliente faz uso de um banco de dados, onde os dados da aplicação são armazenados. Tal cliente processa tais dados e, quando conveniente, sinaliza a ocorrência de eventos que interessam às instâncias do gerenciador de diálogo. Clientes também são responsáveis pela execução de atividades. Em ambos os tipos de cliente, a comunicação com o gerenciador de diálogo é realizada basicamente através de eventos sinalizados para instâncias e controle de atividades recuperadas de portas.

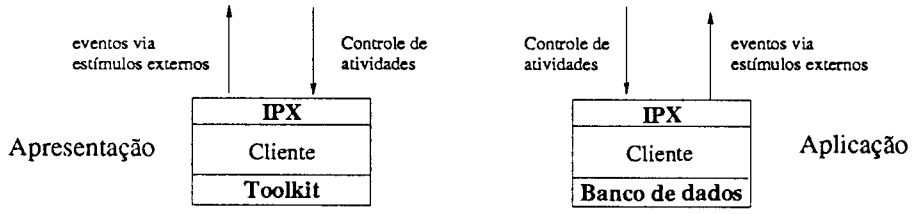


Figura 5-2: Organização típica de clientes de apresentação e aplicação

Um sistema interativo é geralmente composto por uma coleção de clientes, que podem ser executados concorrentemente, possivelmente distribuídos sobre nós de uma rede. A comunicação entre clientes não é de responsabilidade de Xchart, mas a comunicação entre clientes e instâncias de Xcharts é provida pelo sistema de execução de Xchart através da IPX. O conceito de cliente não é uma restrição à organização do software de um sistema interativo. Xchart impõe um isolamento do gerenciador de diálogo. Entretanto, os demais componentes são organizados conforme as necessidades do sistema em desenvolvimento. As Figuras 3-1 (pág. 43) e 3-3 (pág. 44) ilustram essa separação e sugerem que as funções não pertinentes ao gerenciador de diálogo sejão realizadas por clientes cujas organizações internas não são prescritas.

O desenvolvimento de um cliente (lado direito da Figura 5-1) envolve a compilação do código pertinente (apresentação ou aplicação) e ligação com a biblioteca IPX. Um cliente pode compreender todo um sistema interativo, exceto o gerenciador de diálogo, que é fornecido através de diagramas em Xcharts animados pelo SE. Alternativamente, o cliente mostrado na Figura 5-1 pode corresponder a apenas um thread de um conjunto

deles que, em tempo de execução, compreendem o sistema interativo.

Independentemente da arquitetura do sistema interativo em desenvolvimento, clientes podem conter código tanto de apresentação quanto da aplicação. Independência de diálogo, por exemplo, pode não existir em um cliente. Por outro lado, clientes distintos podem cuidar isoladamente das funções de apresentação e aplicação de um sistema interativo. O uso de Xchart não é suficiente para assegurar a independência de dálogo do sistema resultante. Um cliente pode, por exemplo, possuir uma rotina que realiza divisões entre inteiros e é responsável por emitir aviso quando o denominador é zero.

Em Xchart, o controle de diálogo pode ser desenvolvido isoladamente das demais funções de um sistema interativo. Antes que esses desenvolvimentos possam progredir isolados, contudo, deve haver um consenso acerca da comunicação entre o gerenciador de diálogo e os demais componentes. Esse consenso deve estabelecer portas e instâncias que farão uso dessas portas além de clientes que recuperarão reações de algumas dessas portas. Clientes ainda deverão estar associados às instâncias para as quais eles sinalizarão a ocorrência de estímulos. Após a definição dessa comunicação, diagramas em Xcharts e clientes podem ser desenvolvidos separadamente. Em tempo de execução, instâncias, clientes e a comunicação entre eles executarão as funções do sistema interativo em questão. Compromissos similares entre desenvolvimentos isolados de componentes de um sistema interativo são empregados e sugeridos em outras abordagens [28, 121].

Conforme a Figura 5-1, o processo de desenvolvimento empregando Xchart é relativamente simples. O SE definido na Seção 5.2, contudo, revela a necessidade de mecanismos sofisticados para dar vida a um diagrama em Xchart. Uma versão sucinta da execução típica de sistemas que empregam Xchart é apresentada abaixo. Posteriormente seguem breves descrições sobre o editor de Xchart, a linguagem, o compilador e a biblioteca TeXchart, que são as ferramentas que dão apoio ao emprego de Xchart em tempo de desenvolvimento. A IPX é apresentada na Seção 5.3 e propostas correlatas na Seção 5.4.

5.1.1 Modelo de Execução usando Xchart

Segundo a abordagem Xchart, um sistema interativo é organizado em uma coleção de clientes e um conjunto de diagramas em Xchart. Após o início da execução desse sistema, clientes são criados (processos leves distintos, por exemplo) e estes, fazendo uso da IPX, requisitam a manipulação de instâncias de Xcharts (criação, sinalização de eventos, obtenção de resultados e outros). A interação entre clientes e SE encerra-se com o fim das instâncias (execução de transições finais) ou quando os clientes são finalizados.

O SE é uma máquina que recebe estímulos externos e os trata conforme a configuração das instâncias para os quais foram sinalizados. A reação é depositada em portas e recuperada por clientes, que utilizam a IPX para tal.

5.1.2 Editor Xchart

Um editor para Xchart envolve requisitos desde a qualidade da interação oferecida por esse software (usabilidade) até algoritmos sofisticados para o leiaute automático desse tipo de diagrama. Em [62] é apresentado um editor para Xchart cujo principal objetivo foi implementar um algoritmo para leiaute automático de Xcharts. Não houve muita preocupação acerca da usabilidade do editor propriamente dito. Essa questão está sendo investigada em outro trabalho. O projeto de um editor para Xchart é apresentado em [73].

O editor ainda é responsável por converter diagramas em descrições TeXchart e vice-versa além de interagir com o compilador TeXchart.

5.1.3 Linguagem TeXchart

TeXchart (*Textual Xchart*) permite representar, em forma de texto, as construções da linguagem visual Xchart. TeXchart é especialmente útil quando não se encontra disponível um dispositivo gráfico capaz de manipular descrições gráficas em Xchart além de permitir o armazenamento de Xcharts em um formato legível e passível de ser automaticamente processado. TeXchart está para Xchart assim como Statext [64] está para Statecharts.

A conversão TeXchart/Xchart é possível nos dois sentidos sem perda de informação. Um projetista pode especificar, usando um editor gráfico, diagramas em Xchart e manusear, quando desejar ou for conveniente, especificações equivalentes em forma de texto. As construções de TeXchart são simples e, embora de fácil interpretação humana, sua estrutura e elementos básicos foram projetados visando, principalmente, o desempenho de um compilador.

Não se espera que desenvolvedores tenham contato direto com TeXchart. O emprego similar de uma linguagem em forma de texto também se verifica em outras abordagens, por exemplo, MASTERMIND [127]. A descrição da linguagem TeXchart e exemplos de Xcharts e suas versões TeXchart correspondentes podem ser obtidos em [79].

5.1.4 Compilador TeXchart

O compilador TeXchart converte especificações na linguagem TeXchart em estruturas de dados utilizadas pelo Sistema de Execução de Xchart (SE). O modelo desses dados e os artifícios para tornar eficiente a interpretação de um estímulo externo por uma instância são relativamente complexos. Um extenso documento descreve detalhadamente os dados produzidos pelo compilador TeXchart [80]. Esses dados são armazenados em um formato denominado FSDX (Formato de Sistema Descrito em Xchart). A descrição do formato FSDX é apresentado em [77].

A versão atual do compilador TeXchart compreende mais de 12000 linhas de código em C. O projeto do software da versão atual do compilador visou minimizar e isolar questões dependentes do sistema operacional e linguagens empregados na implementação desse projeto. Em conseqüência, a atual versão é executada em ambiente Unix e Win32 sem diferenças no código para estas duas plataformas. Uma versão na linguagem Java também foi considerada nesse projeto.

5.1.5 Biblioteca TeXchart

A biblioteca TeXchart é um modelo orientado a objetos que fornece suporte à implementação de um compilador e de um editor de TeXchart. Versões dessas ferramentas contêm código redundante que dificulta a manutenção de ambos. Por exemplo, ao interagir com um editor de Xchart o projetista pode fornecer uma expressão numérica qualquer cuja análise léxica, sintática e semântica são fornecidas pela biblioteca TeXchart. Dessa forma, essas e outras funções que também são necessárias em um compilador podem ser compartilhadas entre essas ferramentas.

O projeto da biblioteca TeXchart envolveu um cuidadoso estudo das necessidades típicas de um editor de Xchart e uma coordenação com as funções necessárias em um compilador. Embora as funções sejam compartilhadas, os resultados e as suas execuções são interpretados de forma distinta. Por exemplo, enquanto um estado exclusivo sem transição inicial é tratada como um erro pelo compilador, o editor deve permitir tal especificação temporariamente "inválida" até que o projetista complete a especificação com uma transição inicial. Em conseqüência, as classes dessa biblioteca podem armazenar temporariamente sentenças "inválidas" (requisito do editor) ou impedir essa flexibilidade quando o compilador emprega tais classes. A descrição dessa biblioteca pode ser obtida em [77].

5.2 Sistema de Execução de Xchart (SE)

O Sistema de Execução de uma linguagem é o software responsável, em tempo de execução, pela execução de especificações nessa linguagem. O Sistema de Execução de Xchart ou apenas SE, implementa a semântica formal apresentada no capítulo anterior. Esta seção descreve a arquitetura de software do SE.

O fluxo de controle e de informações entre componentes de uma interface, em tempo de execução, é uma área relativamente inexplorada onde não há consenso [51]. Xchart é proposta para capturar um componente de uma interface e, em conseqüência, o SE necessariamente lida com tal área pouco explorada. A versão do SE aqui apresentada é fruto de experiências com versões precursoras. Em [81] é apresentada uma delas.

A Figura 5-3 ilustra um sistema interativo composto de m clientes em execução sobre n processadores $P_1, P_2, \ldots, P_n$ $(m \geq n)$. Se $m = n = 1$, então um único programa em execução em um dado processador faz uso de Xchart através de um único cliente.

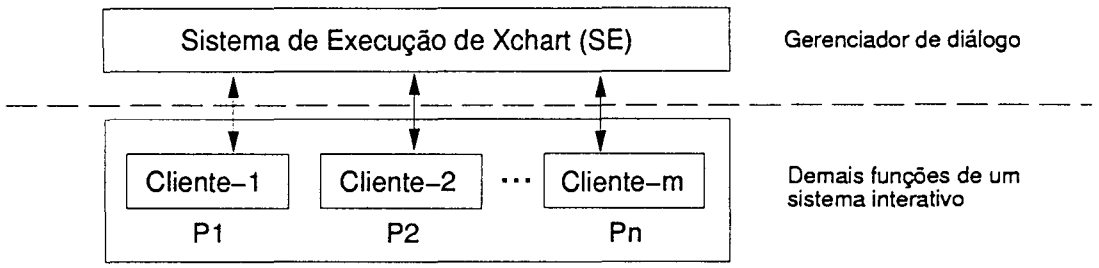


Figura 5-3: Relacionamento entre clientes e SE. Modelo em camadas do SE

Cada cliente esta associado a um único processo (programa em execução). Cada processo, contudo, pode conter um ou mais clientes. Se um processo contém mais de um cliente, então a execução deles pode ser seqüencialmente coordenada pelo programa pertinente ou eles podem residir em *threads* (processos leves) distintos, que são executados concorrentemente. Um cliente interage com pelo menos uma instância de um Xchart ou recupera o controle depositado em uma ou mais portas.

Quando uma instância é criada, o SE é responsável por alocar e iniciar as informações sobre essa instância. Quando tal instância é finalizada, o SE deve remover a memória alocada para a instância. Quando um cliente sinaliza a ocorrência de um estímulo externo, o SE é responsável por depositar tal estímulo na fila de estímulos correta e, quando for o momento, identificar a reação a tal estímulo, possivelmente depositando o controle de atividades em portas e os eventos gerados pela reação nas filas de outras instâncias. O SE é responsável por todas as tarefas necessárias a animação de uma instância: desde a atribuição de um valor a uma variável até a sinalização de um estímulo para um subconjunto das instâncias em execução, possivelmente em processadores distintos conectados por uma rede.

O SE e clientes são executados concorrentemente, possivelmente distribuídos. Esse modelo tem semelhanças com o descrito em [133], proposto para simplificar e dividir em módulos aplicações interativas cujas interfaces exigem o emprego de múltiplos threads de controle, simultâneos, independentes e distribuídos.

5.2.1 Arquitetura do SE

O SE divide-se em dois grandes componentes: o Gerente de Distribuição (GD) e o Servidor Xchart (SX). Dessa forma, a Figura 5-3 pode ser refinada na arquitetura mostrada na Figura 5-4. Em um cenário típico, um cliente gera um estímulo externo para uma dada

instância de um Xchart, o SX recebe o estímulo e, caso necessário, aciona o GD com o intuito de obter/fornecer informações necessárias à interpretação do estímulo sinalizado. A interpretação é depositada em portas. A reação de uma instância é distribuída em uma ou mais portas e deve ser recuperada pelos clientes interessados no conteúdo dessas portas.

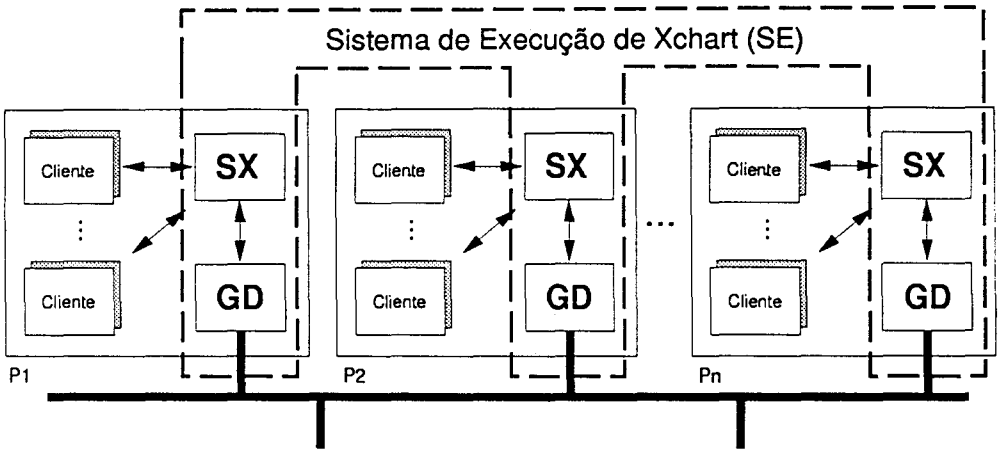


Figura 5-4: Organização de sistemas interativos em tempo de execução (Xchart)

A Figura 5-4 ainda mostra que clientes em execução em um dado processador comunicam-se com um único Servidor Xchart (SX) em execução nesse processador.

5.2.2 Gerente de Distribuição (GD)

O Gerente de Distribuição ou GD é responsável pelo suporte à distribuição: controle de concorrência, localização de recursos, atomicidade e outros. Os serviços oferecidos pelo GD são exclusivamente utilizados por um SX. Em particular, se clientes não forem executados em processos distintos, então os recursos oferecidos pelo GD não serão requisitados por um SX. A implementação do GD fez uso de um modelo em camadas. Uma camada de mais baixo nível oferece suporte à comunicação em grupo [3], que é utilizada para fornecer recursos de mais alto nível conforme descrito em [25].

5.2.3 Servidor Xchart (SX)

O SX é responsável pelas tarefas do SE que são executadas em um dado processador. Cada SX manipula um conjunto de instâncias. Se os clientes de um sistema interativo são executados em um único processador, por exemplo, então as tarefas do SE resumem-se àquelas realizadas pelo SX.

O SX é subdividido nos componentes ilustrados na Figura 5-5. Esses componentes são executados em um único processo por SX. Clientes são executadas em processos distintos. Essa execução em processos distintos permite que o SX continue em execução, assim como outros clientes, independentemente do comportamento de um outro cliente.

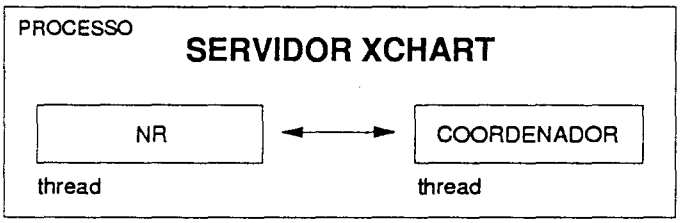


Figura 5-5: Arquitetura do Servidor Xchart (SX)

As funções do SX são realizadas por dois componentes: o Núcleo Reativo (NR) e o Coordenador. Cada um deles é executado concorrentemente em seu próprio *thread.*

5.2.4 Coordenador

O Coordenador mantém dados pertinentes a instâncias em execução e faz uso do relógio do sistema além de outras tarefas dependentes do ambiente operacional. O Coordenador eventualmente faz uso dos serviços providos pelo GD (por exemplo, requisita *broadcast* de um evento).

5.2.5 Núcleo Reativo (NR)

O Núcleo Reativo ou NR é o componente que interpreta a ocorrência de um estímulo externo sinalizado para uma dada instância. Grande parte da semântica de Xchart é implementada por esse componente. Se o NR é alimentado com um estímulo externo e a configuração de uma instância, então uma reação é identificada e depositada em portas. Essa reação é estabelecida pelo NR com o eventual apoio dos demais componentes do SE, se for necessário. O NR é o único componente do SE que pode alterar a configuração de uma instância.

A versão disponível do NR contém cerca de 4000 linhas de código em ANSI C, que manipulam a estrutura de dados gerada pelo compilador TeXchart. A implementação do NR foi projetada com o propósito de tratar rapidamente um estímulo externo. Ou seja, reduzir o tempo decorrido entre o instante que um estímulo externo é sinalizado para uma dada instância e o tempo em que a reação é produzida pelo NR. Esse intervalo de tempo depende de vários fatores. A complexidade do diagrama subjacente, o status corrente da instância e o uso de variáveis compartilhadas são alguns deles.

Embora a grande variedade de fatores torne difícil uma avaliação do desempenho do NR, os exemplos até o momento utilizados mostram que os resultados são satisfatórios no domínio de gerenciadores de diálogo. Alguns mecanismos foram especificamente definidos com o intuito de melhorar o desempenho. Dependências [80], por exemplo, permitem que apenas um subconjunto das regras/transições relevantes e que podem ser habilitadas por um dado estímulo externo sejam consideradas. Uma versão simplificada do algoritmo empregado pelo NR é descrita em [81].

5.2.6 Fluxo de Dados entre Componentes

O fluxo de dados entre alguns dos componentes do SE de Xchart é sucintamente exibido na Figura 5-6. O NR interpreta uma instância (Objetold) de um diagrama (Xchart) conforme o estímulo sinalizado (Estímulo) e a configuração da instância pertinente (Status). O NR pode necessitar de informações peculiares a instâncias em execução em outros processadores e, nesse caso, uma requisição (Reação interna) é enviada para o SX apropriado por intermédio do GD. Durante a execução de uma especificação é comum a alteração de valores de variáveis e da configuração de estados, alterando o antigo Status e produzindo um outro (Novo Status). Variáveis globais (Valores) são gerenciadas pelo GD. Eventos gerados em uma ação para outras instâncias são emitidas para o GD (Reação Interna), que se encarrega de entregá-las ao SX pertinente.

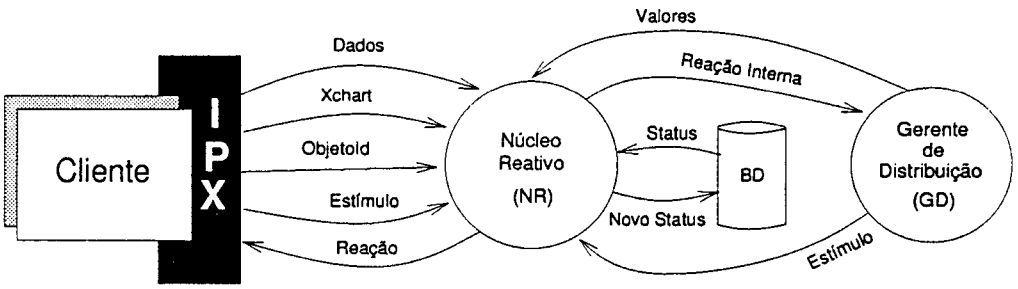


Figura 5-6: Fluxo de dados entre alguns componentes do SE

5.3 Interface de Programação de Xchart (IPX)

Clientes sinalizam estímulos externos para instâncias de Xcharts e recuperam a reação produzida por instâncias e depositada em portas através da Interface de Programação de Xchart ou IPX. A IPX é o único meio de comunicação entre clientes e instâncias. Instâncias são mantidas pelo SE e o SX é o componente responsável por diretamente comunicar-se com clientes via IPX. A Figura 5-7 ilustra essa comunicação.

A IPX é uma ferramenta para ser utilizada por um programador. Um cliente pode sinalizar a ocorrência de um estímulo para uma instância independentemente do local, onde o cliente e a instância em questão são executados. O mesmo também é válido para portas. Reações em portas podem ser recuperadas independentemente da localização de um cliente. A IPX realiza essa função transparentemente.

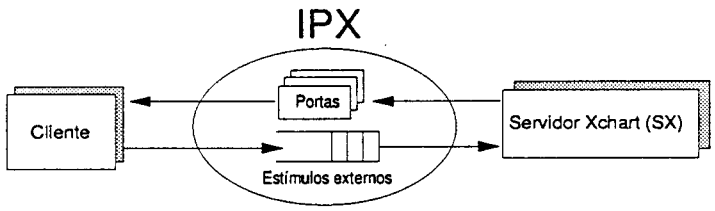


Figura 5-7: Comunicação entre SX e Cliente

A IPX permite que clientes possam concorrentemente sinalizar a ocorrência de estímulos externos assim como recuperar informações depositadas em portas. Uma descrição exaustiva da IPX e vários exemplos do seu emprego podem ser obtidos em [83].

5.4 Propostas Correlatas

Uma única ferramenta que gera código executável a partir de um Statechart foi consultada durante a execução do trabalho corrente. Isso, contudo, não é um indicador ruim, pois apenas uma outra ferramenta conhecida (STATEMATE [43]) realiza tal tarefa. A proposta apresentada em [42] considera algumas questões típicas de modelos de implementação. Tais questões são discutidas na pág. 118.

BETTERSTATE [2] é um produto comercial que implementa uma variante de Statecharts. BETTERSTATE automaticamente gera código pertinente aos diagramas fornecidos interativamente através de um editor.

Especificações geradas pelo BETTERSTATE não são tratadas de forma homogênea entre as várias linguagens para as quais código pode ser gerado. Por exemplo, o rótulo de uma transição contém um evento apenas quando o código gerado é VISUAL BASIC, DELPHI ou faz uso de MFC. Em outras palavras, se código em C é gerado a partir de um diagrama, então os rótulos de transição não possuem eventos, apenas condição e ação. Nesse caso, eventos e estímulos externos são conceitos inexistentes. Eles são manualmente simulados através de valores de variáveis compartilhadas entre o código gerado por BETTERSTATE e o código que faz uso da saída produzida por essa ferramenta. Essa restrição não existe em Xchart assim como o responsável por uma descrição em Xchart não necessariamente precisa compreender detalhes da implementação de diagramas. Essa última proposta não fere requisito de teste apresentado em [144].

A inexistência de uma versão oficial de Statecharts [44] ainda faz da linguagem apoiada por BETTERSTATE uma outra variante de Statecharts. Uma transição do estado **A** para o estado **B** provoca a execução de várias ações: ação que rotula a transição, ação de entrada no estado **B** e ação de saída do estado **A**. Em Xchart, uma seqüência diferente dessa é empregada. Primeiro a ação de saída do estado **A** é executada, seguida da ação da transição e posteriormente da ação de entrada no estado **B**. Em Xchart, o estado de origem de uma transição é desativado antes da ativação do estado de destino da transição. Essa convenção parece mais intuitiva do que aquela empregada por BETTERSTATE. Diferenças entre Xchart e a versão "oficial" de Statecharts são comentadas na Seção 3.11 (pág. 110).

Em BETTERSTATE, o responsável pela especificação em Statecharts precisa conhecer mecanismos empregados pelo código gerado por essa ferramenta. Construções sofisticadas que podem aparecer em eventos, condições e ações em Xchart, por exemplo, devem ser implementadas manualmente quando BETTERSTATE é empregado. BETTERSTATE fornece os mecanismos básicos para transição entre estados. A avaliação de eventos, condições e ações é realizada pelo sistema de execução da linguagem hospedeira empregada, por exemplo, C. Em conseqüência, é comum, por exemplo, produzir uma especificação cujo código gerado por BETTERSTATE possui erros sintáticos. O usuário deve, nesses casos, identificar manualmente qual a sentença em Statecharts que contém o erro sintático, corrigi-lo através do editor e reutilizar o gerador de código. Em Xchart tais inconvenientes não ocorrem. Uma especificação em Xchart é validada e posteriormente interpretada em tempo de execução através de um protocolo (IPX) que fornece alto nível de abstração.

Em Xchart, um número arbitrário de instâncias de um mesmo diagrama pode ser criado e manipulado pelo SE. Em BETTERSTATE, uma única "instância" de cada diagrama pode ser executada. Se duas "instâncias" pertinentes aos diagramas `chart1` e `chart2`, por exemplo, devem ser escalonadas para execução, então o código

```
while (1) {
  CHRT_chart1(0);
  CHRT_chart2(0);
}
```

ou similar, deve ser manualmente produzido.

Embora essa proposta possa ser suficiente para muitos casos, Xchart apresenta um nível de abstração mais elevado. Instâncias são escalonadas para execução pelo SE, elas podem trocar eventos e compartilhar variáveis. Sistemas que fazem uso dos recursos de Xchart não exigem que os seus programadores conheçam os mecanismos de execução de Xcharts ou utilizem recursos do sistema operacional para usufruir de concorrência. Com o intuito de enfatizar essa diferença, se o sistema de execução de BETTERSTATE é chamado de threads concorrentes, então o código que realiza tais chamadas deve manualmente

tratar problemas decorrentes de concorrência. Em Xchart, esse tratamento é realizado pelo SE.

Em contrapartida, os benefícios de Xchart apresentam um custo: desempenho. Alguns exemplos foram implementados em ambos os ambientes com o intuito de obter indicadores de desempenho. Observou-se que, na média, o código gerado pelo BETTERSTATE trata o triplo de estímulos externos que Xchart trata em um mesmo intervalo de tempo. Tal investigação fez uso de diagramas simplificados incluindo apenas os recursos presentes na variante de Statechart adotada pelo BETTERSTATE (transição). Em conseqüência, eventos, condições e ações de Xchart não foram utilizados, pois em BETTERSTATE, tais elementos são implementados na linguagem hospedeira (C foi utilizada). Por exemplo, o evento $\mathbf{tr(x > 2)}$ não é possível de ser gerado pelo sistema de execução de BETTERSTATE. O tamanho dos programas obtidos da versão do BETTERSTATE, contudo, apresentavam quase o dobro do tamanho daqueles gerados pela abordagem Xchart. Se estados concorrentes são utilizados, então as diferenças de desempenho são menores. BETTERSTATE usa o sistema operacional para tratar estados concorrentes. O SE, ao contrário, implementa o seu próprio escalonador.

Capítulo 6

Desenvolvendo Interfaces em Xchart

Like physics, medicine, manufacturing, and many other disciplines, ... We must experiment with techniques to see how and when they really work, to understand their limits, and to understand how to improve them. We must learn from application and improve our understanding.
Victor R. Basili, Proceedings of ICSE'96, page 443

A qualidade de uma ferramenta só pode ser determinada através da observação dos resultados da sua utilização em casos reais e representativos. Nos capítulos anteriores a abordagem Xchart e seus fundamentos foram apresentados. O capítulo precedente, em particular, abordou uma proposta de software que ampara o emprego de Xchart. No presente capítulo são apresentados exemplos do uso de Xchart no desenvolvimento de gerenciadores de diálogo.

O capítulo mostra como modelos de gerenciadores de diálogos descritos em Xchart podem ser implementados em arquiteturas de softwares de sistemas interativos baseadas em agentes (Seção 6.7). Grande parte das propostas similares não se preocupam com a implementação dos modelos que oferecem ou com a conexão do software gerado com arquiteturas de software. Muitas propostas baseadas em linguagens semelhantes desconsideram questões importantes para a comunidade de interfaces, como independência de diálogo (Seção 6.6). Conforme ressaltado no Capítulo 2, poucas propostas contemplam o desenvolvimento de interfaces concorrentes como aquela exemplificada na Seção 6.5. As demais seções exemplificam como recursos alguns dos recursos de Xchart não disponíveis em Statecharts podem ser efetivamente empregadas nesse domínio.

Os exemplos fornecem indícios propícios à Xchart em relação a propostas similares assim como também sugerem a necessidade de esforços futuros (Seção 6.8).

6.1 Introdução

Exemplos de Xchart podem ser obtidos em [84, 85]. Um emprego complexo de Xchart é descrito em [78], onde o controle de diálogo do sistema de geoprocessamento SPRING, desenvolvido pelo INPE, foi parcialmente capturado em Xchart. As principais interações foram modeladas através de dezenas de diagramas em Xchart. Outros exemplos do emprego de Xchart podem ser encontrados em [76], onde os recursos de Xchart são cobertos em exemplos elaborados com o intuito de elucidar sutilezas semânticas de Xchart. Os exemplos aqui comentados não incluem aqueles citados acima.

Os exemplos fornecidos não cobrem todos os recursos de Xchart e não são apresentados como versões "ótimas", em qualquer que seja o sentido, de diagramas em Xchart. Ou seja, diagramas equivalentes com um menor número de estados ou variáveis, por exemplo, podem ser desenvolvidos. A qualidade das interfaces correspondentes também pode ser melhorada. Em alguns exemplos apenas um esboço de tela (função da apresentação de uma interface) é fornecido. A facilidade de uso e aprendizado das interfaces exibidas não é o foco das atenções, mas o controle subjacente a cada uma delas.

A implementação de gerenciadores de diálogo de interfaces também faz parte da proposta do ambiente Xchart. A implementação desses componentes de interfaces exige uma proposta para a organização do software de interfaces. Atualmente não existem "regras de ouro" para organizar o software de uma interface. Em conseqüência, divisões funcionais distintas daquelas dos exemplos, fornecidos no presente capítulo, podem ser estabelecidas. Esse capítulo abstrai de tais questões. O principal objetivo é identificar o controle de "interfaces" e mostrar como Xchart comporta-se na descrição e implementação desse controle.

6.2 Transição Inicial e Pseudo-final

Transições iniciais podem não estar habilitadas quando o estado que as contém for ativado. Em conseqüência, o estado em questão permanece básico. Uma transição pseudo-final torna o estado destino temporariamente básico pelo menos até o término do passo no qual é executada. Tais comportamentos podem ser positivamente empregados para modelar situações típicas de interfaces, conforme ilustra o exemplo abaixo.

Em uma aplicação típica de uma vídeo locadora, algumas informações sobre vídeos podem ser exibidas a um usuário enquanto outras são fornecidas apenas quando requisitadas. Por exemplo, no esboço de tela cujo título é MAIN WINDOW, Figura 6-1, são apresentados o nome e o código de uma fita de vídeo além de um botão (**More Info**).

A partir dessa tela, quando o usuário deseja obter mais informações sobre a fita de vídeo cujo nome e código são exibidos, ele faz uso do botão **More Info**. Quando esse

botão é pressionado, a tela MAIN WINDOW é substituída por aquela de título DETAILED WINDOW, onde mais detalhes sobre a fita de vídeo são exibidos. Nessa nova tela, o botão utilizado é **Less Info**. Quando pressionado, esse botão substitui a DETAILED WINDOW pela MAIN WINDOW.

O controle comentado acima pode ser capturado pelo Xchart apresentado no lado direito da Figura 6-1. Quando uma instância desse Xchart é criada, o estado mais externo, **Main**, permanece temporariamente básico até que a única transição inicial esteja habilitada. Ao ativar o estado **Main**, a atividade **ShowMain** é executada (regra especial **entry**). Essa atividade é responsável por apresentar a janela MAIN WINDOW. A ocorrência do evento **evML** provoca a execução da única transição inicial, que ativa o estado **Detailed**. Esse evento ocorre quando o usuário pressiona o botão **More Info**. A ativação do estado **Detailed** provoca a execução da atividade **ShowDetails**, que acrescenta à MAIN WINDOW, informações sobre a fita de vídeo em questão.

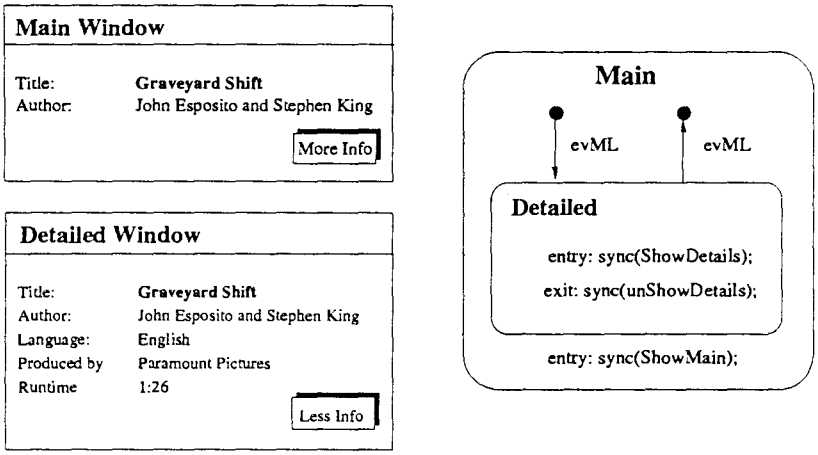


Figura 6-1: Capturando o controle de interfaces em Xchart

Se o estado **Detailed** estiver ativo e o evento **evML** é gerado, então a transição pseudo-final exibida é executada. Tal evento é provocado pela ação do usuário ao pressionar o botão **Less Info**. Nesse caso, o estado **Detailed** é desativado e o estado **Main** permanece temporariamente básico. Quando o estado **Detailed** é desativado, a atividade **unShowDetails** é executada. Essa atividade remove algumas das informações da janela DETAILED WINDOW, retornando ao conteúdo da MAIN WINDOW. Essa atividade ainda altera o rótulo do botão **Less Info** para **More Info**. A atividade **ShowDetails** realiza a troca de rótulos inversa.

6.3 Regras

Statecharts [40, 45] não possui o conceito de regra. São comuns situações onde uma ação deve ser executada sem que seja natural a execução de uma transição. A inexistência desse recurso é suprida pela criação de estados artificiais para simular estas situações. Por exemplo, na Figura 6-2, o estado **NODE** existe com o único propósito de modelar o conceito de regra em Xchart. O Statechart **ROOT** foi extraído de [24, pág. 69]. Essa figura ainda apresenta o Xchart equivalente, sem a necessidade de um estado "artificial".

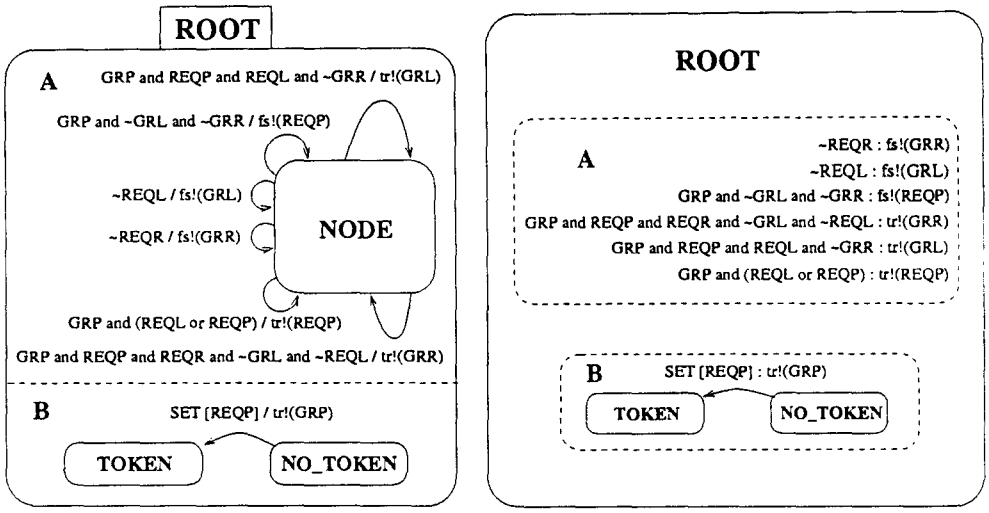


Figura 6-2: Um Statechart (obtido de [24, pág. 69]) e um Xchart equivalente

A versão de Statecharts apresentada em [44] contempla um recurso similar a regras, denominado de reação estática. Comentários sobre tal reação são feitos na página 117.

6.4 Hierarquia de Tipos de Eventos

O Xchart exibido na Figura 6-3 ilustra o emprego de hierarquia de tipos de eventos primitivos. O Xchart verifica, conforme a entrada fornecida pelo usuário, se se trata de uma especificação válida de uma placa de automóvel, ou seja, três letras seguidas por quatro dígitos.

Uma entrada válida é identificada se o estado **Valid** está ativo. Caso contrário, o estado **NotValid** está ativo e a entrada fornecida até o momento em questão não é válida. O usuário pode fazer uso do teclado para fornecer dígitos, letras ou comandos (apagar último caractere). Cada uma dessas ações é capturada de forma abstrata por um tipo de evento primitivo. Na hierarquia apresentada (Figura 6-3, lado direito), por exemplo, o tipo **letter** representa toda e qualquer letra (maiúscula ou não). O tipo **digit**

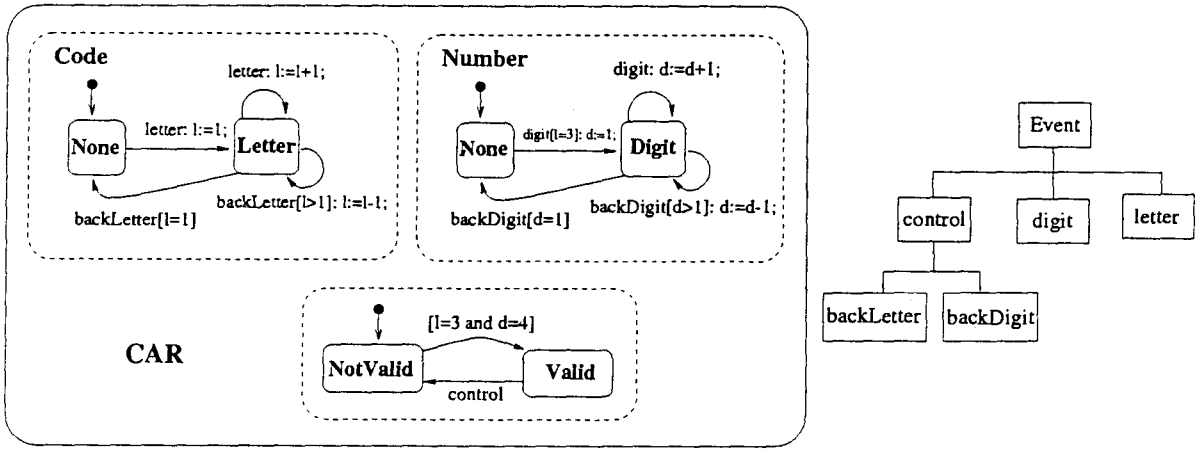


Figura 6-3: Gerenciador de edição de placas de automóveis

representa todo e qualquer numeral arábico (0 a 9) e o tipo **control** é refinado nos tipos **backLetter** e **backDigit**. Os dois últimos referem-se à remoção do último caractere fornecido, letra ou dígito, respectivamente.

O tipo **control** é utilizado na transição do estado **Valid** para o estado **NotValid**. Inicialmente o estado **NotValid** está ativo. Se o usuário remove uma letra ou dígito de uma entrada válida (estado **Valid** ativo), então a entrada torna-se inválida. Essa ação do usuário é capturada pelo tipo **control**. Esse tipo pode ser provocado tanto pelo evento **backLetter** quanto pelo evento **backDigit**. A hierarquia de tipos permite substituir o evento **backLetter or backDigit** por simplesmente **control**, sem prejuízo para a semântica desejada. Nesse caso, o cliente em questão não irá sinalizar a ocorrência do tipo **control**, mas a ocorrência de um dos seus tipos descendentes.

O mapeamento entre o significado abstrato de um tipo de evento primitivo e as ações dos usuários correspondentes é realizado pelo componente de apresentação ou clientes em Xchart. Ou seja, esse componente, em tempo de execução, é responsável por detectar as ações dos usuários correspondentes a cada um dos tipos e, quando ocorrerem, sinalizar o estímulo externo pertinente para a instância apropriada. A Figura 5-2, pág. 171, ilustra esse fato.

6.5 Sistemas Interativos Multi-Usuário

O jogo (*Tic-Tac-Toe*) descrito abaixo é realizado entre dois jogadores (X e O). O tabuleiro exibido na Figura 6-4 mostra o jogo (sistema) do ponto de vista de um dos jogadores: o jogador X. O jogador O vê um tabuleiro análogo. Os jogadores X e O alternam suas jogadas ao selecionar uma posição do tabuleiro para depositar as suas respectivas marcas.

Quando o mouse passa sobre uma posição desocupada, uma marca cinza, pertinente ao jogador que realiza o movimento, preenche a vaga. Um jogador não percebe tais movimentos do seu adversário. Se é a vez de um dado jogador e ele pressiona algum botão do mouse, então uma marca preta preenche definitivamente a posição. Se não é a vez do jogador, então nenhum feedback é oferecido quando o jogador movimenta o mouse. Ao pressionar um botão do mouse sobre uma posição inválida, uma campainha soa indicando uma exceção.

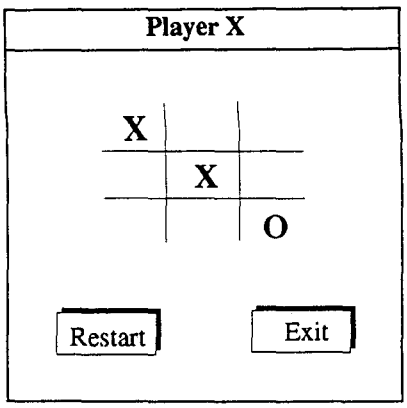


Figura 6-4: Visão do jogador X em um instante de uma partida

Quando o jogo termina, uma pequena animação é executada: uma para o jogador vencedor, parabenizando-o e outra estimulando o jogador derrotado. Se os jogadores empatam, então uma mesma animação é exibida para ambos. Essas animações persistem até que eles recomecem uma nova partida ou finalizem a execução do jogo. A finalização pode ocorrer em qualquer momento de uma partida por iniciativa de qualquer jogador. O reinício exige um consenso entre eles. Esse exemplo é utilizado pela simplicidade e pelos importantes recursos de interfaces multi-usuário [53]:

- Controle alternado entre jogadores.
- Cada jogador pode configurar a sua própria interface.
- Visão replicada: o tabuleiro.
- Interação independente: o jogador pode realizar, a qualquer momento, as operações que diretamente não alteram o conteúdo do tabuleiro.
- Operações simultâneas: cada jogador pode finalizar a partida a qualquer momento.
- Manipulação direta: mover o mouse sobre o tabuleiro provoca o emprego de uma marca cinza nas posições onde é possível realizar uma jogada.

- Consenso entre os jogadores. Os jogadores iniciam uma nova partida como fruto do consenso entre eles. Esse recurso é introduzido pelo presente trabalho e não faz parte da aplicação apresentada em [53].

Antes de comentar a descrição do gerenciador de diálogo em Xchart, as funções desse e dos demais componentes do sistema devem ser estabelecidas. Para esse exemplo são identificados os clientes ilustrados na Figura 6-5. Há um grupo de clientes (**Animation** e **Interaction**) para cada jogador e um cliente **Application** compartilhado entre eles. O cliente **Animation** é responsável pelas animações produzidas pelo sistema. O cliente **Interaction** recebe as entradas do jogador e as mapeia para eventos em Xchart. Esse cliente ainda fornece feedback imediato para algumas ações de um jogador além de executar atividades como **beep** (soa a campainha).

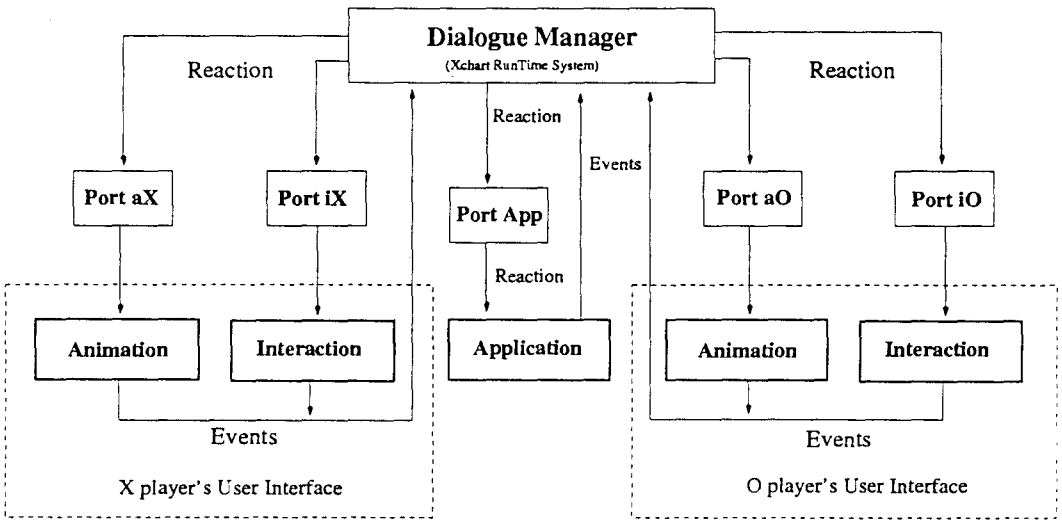


Figura 6-5: Organização do software do jogo Tic-Tac-Toe

Para cada jogador os seus clientes **Animation** e **Interaction** estão em execução no processador pertinente. Assim, conforme a Figura 6-5, cada jogador "interage" com os seus próprios clientes. Detalhes da organização interna de cada cliente não são apresentados. A sinalização de eventos (estímulos externos) e a recuperação de reações de portas é realizada via IPX. Nesse exemplo, jogadores executam seus lances de computadores distintos e, transparentemente, estímulos são trocados entre instâncias e clientes além da troca apenas entre instâncias. Por simplicidade, essa figura exibe apenas o relacionamento entre portas, clientes e o gerenciador de diálogo do sistema descrito acima. Pelo mesmo motivo, algumas das funções de clientes não serão comentadas: iniciar o sistema, manipulação de estruturas de dados e outras.

O cliente **Application** é responsável pela semântica do jogo: (i) armazenar o status de cada uma das posições do tabuleiro; (ii) indicação do próximo jogador a realizar a sua

jogada e (iii) uma indicação de qual jogador realizou o primeiro movimento na partida em questão (o jogador que inicia a partida alterna entre execuções consecutivas). As funções do gerenciador de diálogo bem como a comunicação desse componente com os clientes (portas e eventos) são estabelecidos durante a explanação do Xchart **X** (Figura 6-6). Em especial, a tarefa de identificar o consenso entre os jogadores é realizada pelo gerenciador de diálogo.

Os clientes identificados ressaltam a concorrência existente nesse sistema. Se eles forem implementados como processos distintos ou *threads*, distribuídos ou não sobre nós de uma rede, por exemplo, não importa para Xchart. A organização do código de clientes está além dos interesses e restrições de Xchart. O restante da seção comenta o Xchart **X** (Figura 6-6).

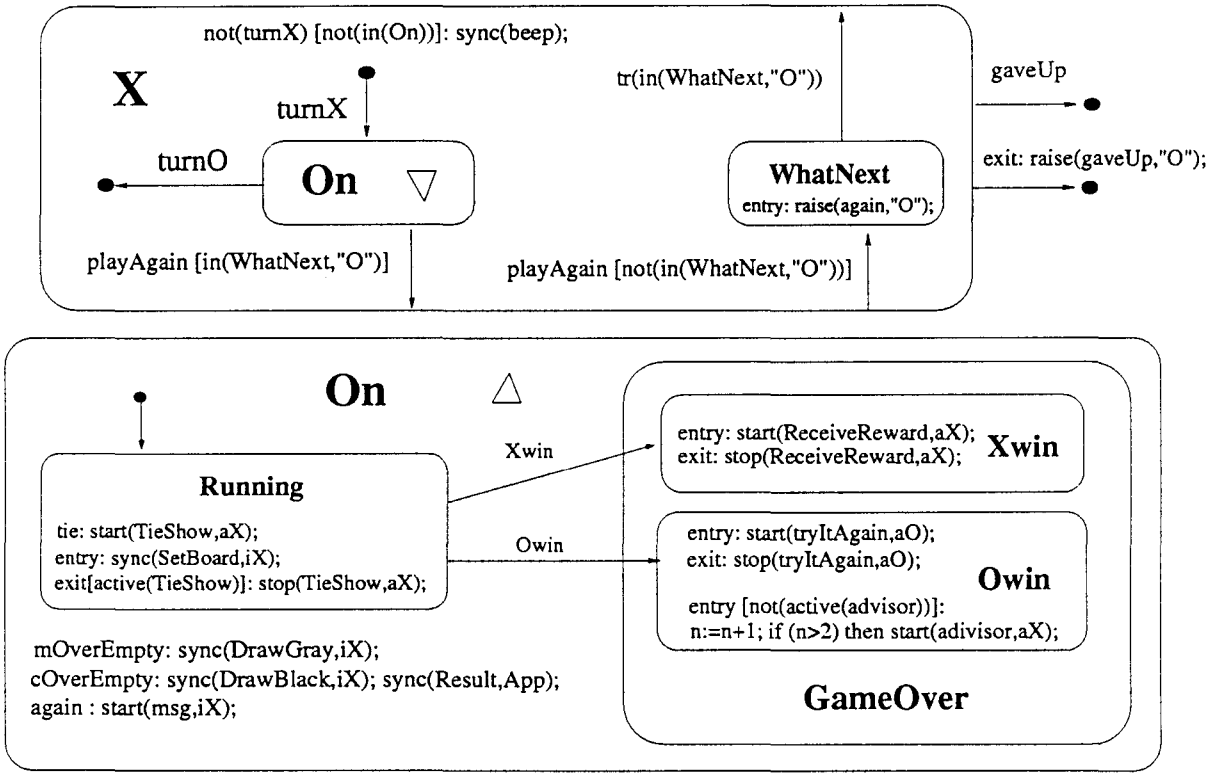


Figura 6-6: Gerenciador de diálogo da interface para o jogador X

Na Figura 6-6 vemos a descrição do gerenciador de diálogo para o jogador X. A descrição equivalente para o jogador O é análoga. Ela pode ser facilmente obtida através de pequenas modificações. Em tempo de execução, uma instância desse Xchart fornecerá as funções desse gerenciador para os clientes do sistema que interagem com o jogador X. Analogamente, uma instância do Xchart O executará as mesmas funções para o jogador O. Cada uma dessas instâncias ainda irá interagir com a outra através de eventos, por

exemplo, na identificação de consenso para a realização de um novo jogo entre os jogadores. O restante dessa seção descreve, do ponto de vista do jogador X, o comportamento do Xchart **X** (Figura 6-6). Os recursos ∇ e Δ (pág. 53) são utilizados na partição da hierarquia de **X**.

Quando uma instância do Xchart **X** é criada, o subestado **On** não é ativado. O estado **On** é ativado se o evento **turnX** ocorrer após a criação da instância. Esse evento é sinalizado por um cliente apenas se há outro jogador interessado em jogar. Quando o estado **On** é ativado pela primeira vez, o evento **turnX** ainda habilita o jogador X a realizar a sua primeira jogada. Esse e outros eventos gerados por clientes são sinalizados através da IPX. O SE encarrega-se de depositá-lo na fila de estímulos da instância pertinente que o recupera e, ao tratá-lo, ativa o seu estado **On**.

O estado **On** permanece desativado quando a jogada corrente é do adversário e, nesse caso, a ocorrência de qualquer evento que não seja o evento **turnX** soa uma campainha, conforme regra no interior de **X**, informando que a ação do jogador não é permitida no momento. O jogador poderia, por exemplo, estar tentando ocupar uma posição vazia durante a jogada do adversário. O estado **On** é desativado pelo evento **turnO**, gerado pelo cliente **Application**, que é responsável por estabelecer a ordem de jogadas entre os jogadores. Quando o jogador O realiza o seu movimento, o estado **On** da instância de **X** está desativado e, após a jogada, o cliente **Application** sinaliza a ocorrência do evento **turnX** para a instância de **X**.

O estado **On** contém algumas regras. Uma delas é executada na ocorrência do evento **mOverEmpty**, que é gerado quando o indicador do mouse passa sobre uma posição não preenchida no tabuleiro. É função do cliente **Interaction** detectar essa situação e sinalizar tal ocorrência para a instância pertinente. Nesse caso, a reação é a execução síncrona da atividade **DrawGray**. Essa reação é depositada na porta **iX** (Figura 6-5). O cliente **Interaction** associado ao jogador X é responsável por tratar as reações produzidas nessa porta. Tal cliente irá recuperar essa reação e executá-la. Ao fim da execução dessa atividade, a instância de X pode prosseguir. Se o evento gerado é **cOverEmpty**, então o jogador pressionou um botão do mouse sobre uma posição vaga do tabuleiro. A reação é preencher a posição com um X em preto e requisitar ao cliente **Application** o resultado de tal jogada através da porta **App**. A jogada pode resultar em empate ou vitória de um dos jogadores, o que é capturado pelo evento **Xwin** ou **Owin**. Esses eventos são gerados pelo cliente **Application**. A outra regra do estado **On** é comentada posteriormente. O estado **On** é refinado nos estados **Running** e **GameOver**.

Quando o estado **On** é ativado o subestado **Running** é ativado em seguida, provocando a execução da atividade **SetBoard**, que exibe o tabuleiro em seu estado inicial, ou seja, sem nenhuma marca. Essa atividade é de responsabilidade do cliente **Interaction** e, em conseqüência, é depositada na porta **iX**. Esse estado ainda contém regra que trata

casos de empate. Se não há vencedor, então o evento **tie** é sinalizado para ambas as instâncias pelo cliente **Application**. Ao receber o estímulo externo correspondente, a instância de **X** deposita a execução assíncrona da atividade **TieShow** na porta **aX**. De forma análoga, a instância de **O** deposita o mesmo controle na porta **aO**. Ao desativar o estado **Running**, a regra **exit** desse estado verifica se a atividade **TieShow** está em execução e, se for o caso, requisita a interrupção dessa atividade. Por simplicidade, esse estado é apresentado apenas parcialmente.

As transição rotuladas com **Xwin** e **Owin** capturam, respectivamente, a vitória do jogador X ou a vitória do jogador O. Esses eventos são gerados pelo cliente **Application**, se for o caso, durante a execução da atividade **Result**, executada após a jogada de cada jogador. As transições que partem de **Running** têm destinos explícitos no interior do estado **GameOver**.

A ativação de um dos subestados de **GameOver** estabelece o feedback a ser apresentado a cada jogador em caso de vitória de um deles. Se o vencedor é o jogador X, então a animação é disparada pela execução de **start(ReceiveReward, aX)**. A porta **aX** enfileira as reação que controlam atividades cujas execuções são de responsabilidade do cliente **Animation** associado ao jogador X. A atividade é executada de forma assíncrona, ou seja, a instância pode continuar reagindo a estímulos à medida que o cliente **Animation** executa tal atividade. Em particular, se o usuário desejar interromper a animação, ele pode, por exemplo, finalizar a aplicação ou pedir para que o jogo reinicie. Se o botão **Exit** é pressionado, então a transição final do Xchart **X** é executada em decorrência do evento **Exit**, sinalizado pelo cliente **Interaction**. Ao tratar esse evento, qualquer uma das instâncias sinaliza para a outra o evento **gaveUp**, que também provoca a finalização da instância. Se o botão **Restart** é pressionado, então o evento **playAgain** é gerado, causando a desativação do estado **GameOver** e, em conseqüência, interrompendo a animação através de **stop(ReceiveReward, aX)**.

O evento **playAgain** pode ser gerado por qualquer um dos clientes **Interaction** para a sua respectiva instância. Ele pode provocar a ativação do estado **WhatNext**, que só pode ser desativado se algum jogador deseja finalizar a aplicação ou quando o outro jogador também deseja iniciar uma nova partida. Se o jogador X deseja iniciar uma nova partida e o estado **WhatNext** da instância pertinente é ativado, então o evento **again** é sinalizado para a instância do Xchart **O** através da ação **raise(again,"O")**. Nesse caso, o jogador X tomou a iniciativa de iniciar uma nova partida e o seu estado **WhatNext** será ativado, pois o estado **WhatNext** da instância do Xchart **O** não está ativo. Se o jogador **O** toma a iniciativa, então o estado **X** permanecerá temporariamente básico, conforme a transição interna do estado **On** para **X**. Se o jogador X toma a iniciativa, então o estado **WhatNext** permanecerá ativo até que o estado **WhatNext** na instância de **O** torne-se ativo, caracterizando o consenso capturado pelo evento **tr**. Em ambos os

casos, se os jogadores desejam iniciar uma nova partida, então o estado **X** permanecerá temporariamente básico até que o cliente **Application**, através dos eventos **turnX** ou **turnO**, decida quem irá iniciar a partida.

Qualquer que seja a instância que receba o evento **again** irá provocar a execução da atividade **msg**, que exibe uma mensagem informado que o adversário deseja jogar novamente. A regra no interior do estado **On** captura esse comportamento. No interior do estado **Owin** ainda é apresentada uma regra de entrada que conta quantas partidas foram perdidas pelo jogador X. A terceira derrota provoca a execução assíncrona da atividade **advisor**. Essa atividade é executada pelo cliente **Animation** e pode corresponder a um agente que irá sugerir os próximos lances para o jogador X. A condição dessa regra impede que o referido "auxiliar" seja executado caso a atividade pertinente esteja em execução.

6.6 Independência de Diálogo em Xchart

Essa seção ilustra um caso onde não há independência de diálogo e outro onde tal independência é obtida através de uma reorganização do software de um sistema interativo com o apoio de Xchart.

O trecho de código abaixo ilustra um tipo indesejável de acoplamento comum entre interface e aplicação. Sem perda de generalidade, pode-se assumir que tal código pertence à aplicação (cliente de aplicação). Ele executa operações pertinentes a um domínio (função de aplicação) e, em seguida, habilita opção SAVE de menu (função de interface). Essa opção de menu deve permanecer desabilitada quando os dados da aplicação não foram alterados e habilitada em caso contrário.

```
retType app_function (partype a) {
   // Codigo da aplicacao (realiza calculos, atualiza valores ...)
   // vem aqui.
                            /* Feedback                        */
   menuItemSave->Enable(1); /* Codigo de responsabilidade */
}                           /* da interface                    */
```

Nesse exemplo, o projetista de software transferiu uma responsabilidade de função da interface para a aplicação, pois o código responsável pelas funções da aplicação estabelece uma reação da interface. Se o projeto da interface for refeito e uma barra de ferramentas for acrescentada com o botão **Save**, por exemplo, então o código abaixo terá que ser alterado para contemplar o botão acrescentado. Nesse caso, o cliente de aplicação não se apresenta independente da forma de interação do usuário com a interface. Conforme o código, a aplicação ainda é responsável pela manutenção da consistência entre o que é apresentado pela interface e os seus próprios dados.

A organização das funções desse código de acordo com a arquitetura PAC (arquiteturas foram discutidas no Capítulo 2), por exemplo, produziria um objeto presentation que cuidaria da apresentação da opção de menu e um objeto control, que seria responsável por manter a consistência entre o objeto presentation e o objeto abstraction, que realizaria os cálculos pertinentes. O código acima pode ser ilustrado através da Figura 6-7, lado esquerdo, que é uma outra versão para a mesma arquitetura apresentada na Figura 2-2, pág. 15. Nesta figura, o código de aplicação e apresentação fundem-se em um mesmo componente que diretamente requisitas serviços do toolkit subjacente para prover função de apresentação. Essa organização pode ser substituída por aquela da Figura 6-7 (lado direito), conforme comentado abaixo.

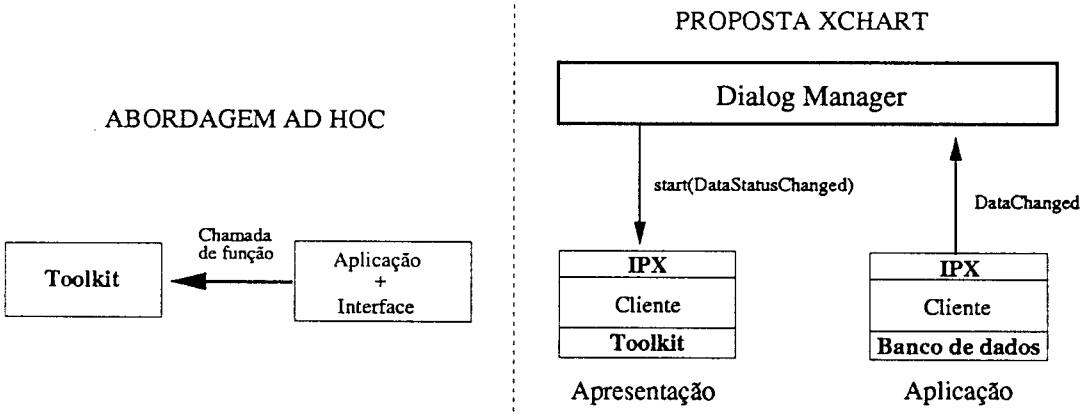


Figura 6-7: Organização **ad hoc**. Independência de diálogo via Xchart

Esse acoplamento indesejável pode, alternativamente, ser eliminado. Em uma nova versão, a detecção de que dados foram alterados não significa que a opção de menu será habilitada. Tal decisão é deixada para o legítimo responsável: a interface (cliente de apresentação). A aplicação não estabelece o feedback a ser apresentado ao usuário. O cliente apropriado sinaliza, através do evento **DataChanged** a mudança de valores de dados. A interpretação desse evento fica a cargo de diagramas em Xchart. A reação produzida seria executada por um cliente de apresentação.

```
retType new_app_function (partype a) {
   // Codigo da aplicacao (realiza calculos, atualiza valores ...)
   // vem aqui.

   XC_Send(DataChanged); /* IPX (emprego simplificado)    */
                         /* Imune a mudancas na interface */
}
```

Ao contrário do caso anterior, o novo código não tem conhecimento acerca da reação da interface. É possível alterar a reação como, por exemplo, soar uma campainha ou ainda substituir o toolkit utilizado por um outro sem efeitos colaterais para o novo código. As alterações necessárias seriam efetuadas, nesses casos, exclusivamente no cliente de apresentação (interface).

O exemplo ilustra como o acoplamento entre interface e aplicação pode ser reduzido com o apoio de Xchart. A interação entre interface e aplicação é, contudo, uma questão bem mais complexa do que o exemplo acima pode sugerir (Seção 2.3.5).

6.7 Xchart e Arquiteturas de Software

Nessa seção é apresentado o emprego de Xchart na implementação das arquiteturas PAC e MVC. Na Figura 6-8 é exibida uma simples caixa de diálogo que é utilizada como exemplo. Ela contém 13 objetos de interação típicos. Em geral eles correspondem a 13 instâncias de widgets e, em conseqüência, 13 callbacks são necessárias. Essa caixa de diálogo é empregada para a seleção de uma cor no padrão RGB.

O usuário pode selecionar uma das seis cores predefinidas no interior do grupo **Colors** ou fornecer uma cor através do seu valor RGB via as barras de rolagem, por exemplo. A cor corrente preenche um retângulo no canto superior direito da caixa de diálogo. Se o usuário arrasta a barra de rolagem, por exemplo, então o valor numérico para a cor pertinente é devidamente atualizado bem como a cor do retângulo. Ao atuar sobre as barras de rolagem uma das cores preestabelecidas pode ser obtida e, nesse caso, automaticamente o item pertinente torna-se selecionado no grupo **Colors**. O mesmo ocorre em qualquer uma das três opções de entrada. Se o usuário digita o valor 255 para vermelho, verde e azul, por exemplo, então as barras de rolagem são devidamente atualizadas, o item correspondente a branco no grupo **Colors** torna-se selecionado e a cor branca é utilizada para preencher o retângulo que apresenta a cor corrente.

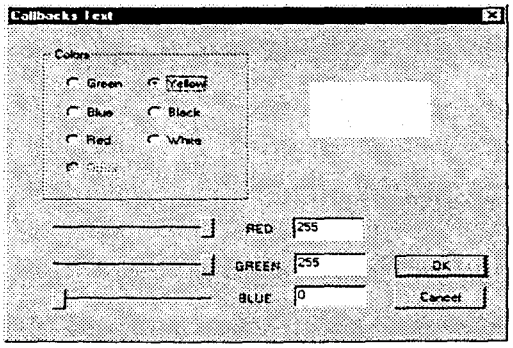


Figura 6-8: Caixa de diálogo para seleção de cor

As duas seções seguintes mostram como o código dessa caixa de diálogo pode ser organizado segundo as arquiteturas PAC e MVC, respectivamente. A identificação de agentes em arquiteturas baseadas em múltiplos agentes pode ser realizada com o apoio de diretrizes como aquelas apresentadas em [5, págs. 175-184].

6.7.1 Arquitetura PAC

A hierarquia PAC, mostrada na Figura 6-9, exibe uma possível organização do software para o exemplo da caixa de diálogo. Por simplicidade, apenas três dos sete agentes descendentes de **Colors** são mostrados nessa figura. Os agentes não apresentados são pertinentes aos itens de cores **White**, **Green**, **Blue** e **Yellow**.

Na raiz dessa hierarquia o agente PAC **Raiz** mantém a consistência entre os itens de **Colors**, as barras de rolagem e os valores fornecidos para cada cor. Ou seja, é de responsabilidade do objeto control da raiz dessa hierarquia substituir o item selecionado **Black** por **Red** quando o usuário rolar a barra de rolagem apropriada da extremidade esquerda para a direita. O agente **Barras de rolagem** detecta a nova cor "pura" que, através do objeto control transfere tal informação para o objeto control da raiz e assim por diante. O objeto presentation do agente da raiz é responsável por preencher o retângulo com a cor corrente. O objeto abstraction representa a cor corrente em formato reconhecido pela aplicação.

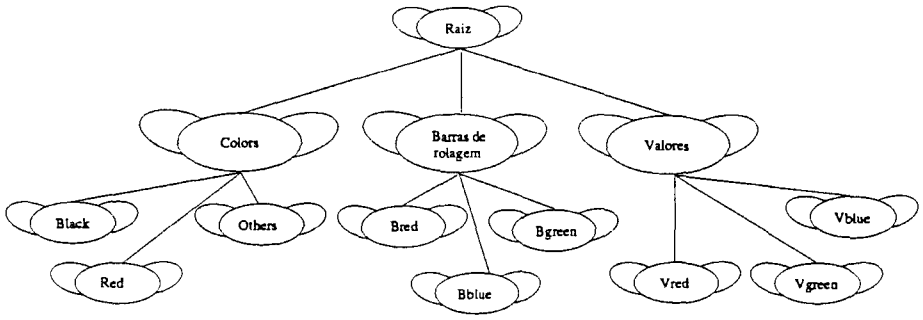


Figura 6-9: Arquitetura PAC para o exemplo

6.7.2 Arquitetura MVC

Na arquitetura MVC são identificados vários objetos: um objeto model que representa a cor corrente (cor selecionada pelo usuário); um objeto view que preenche o retângulo de cor corrente; dois objetos controller (um para cada um dos botões) e treze objetos view-controller: um para o item **Other** do grupo **Colors**, seis representando as cores

preestabelecidas, três para cada uma das barras de rolagem e os outros três para cada um dos campos de edição.

Os objetos view-controller desempenham funções de objetos view e objetos controller. Cada um desses objetos relacionados aos elementos do grupo **Colors**, por exemplo, recebe entrada do usuário (controller), quando o mesmo seleciona um deles ou é automaticamente selecionado quando uma das cores predefinidas é obtida através de manipulação das barras de rolagem (ou ainda fornecendo os valores através dos campos de edição). Analogamente, cada barra de rolagem também representa um objeto controller, quando o usuário arrasta a barra, e um objeto view que reflete, conforme a posição da barra, o valor da cor correspondente.

Ao se abrir a caixa de diálogo, pode ser fornecido um valor RGB para a cor inicial. Nesse caso, o único objeto model sinaliza a necessidade dos seus objetos view associados serem atualizados. O protocolo dessa comunicação é fornecido em detalhes na página 19. De forma análoga, quando um dos objetos view-controller recebe alguma entrada do usuário, então os demais objetos view-controller são devidamente avisados, se for o caso, assim como o objeto model. Ao arrastar uma das barras de rolagem, por exemplo, o objeto view-controller pertinente avisa os demais objetos view-controller e o objeto model para que eles possam se atualizar.

6.7.3 Capturando o C de PAC e o C de MVC em Xchart

Xchart permite capturar funções da faceta control de um agente PAC ou de um objeto controller de um agente MVC. Embora as funções dessas elementos apresentem muitas semelhanças, existem diferenças (Seção 2.3.3). A descrição em Xchart comentada abaixo é mais próxima da faceta de controle de um agente PAC, pois esta é responsável pela comunicação entre agentes. Uma das características de instâncias de diagramas Xcharts é a possibilidade de troca de estímulos entre elas. Essa tarefa não é explicitamente atribuída a nenhum dos objetos MVC. Nesse exemplo, contudo, um objeto controller passa a assumir a responsabilidade pela troca de informações entre agentes na arquitetura MVC.

A Figura 6-10 exibe três Xcharts e uma hierarquia de tipos de eventos. O conjunto especifica o controle da caixa de diálogo. Uma instância do Xchart **Color** reflete os três estados em que cada item do grupo **Colors** (caixa de diálogo) pode se encontrar. Por exemplo, se o estado **Selected** da instância identificada por "Black" está ativo, então o item **Black** encontra-se marcado na caixa de diálogo. Para cada item do grupo **Colors** há uma instância desse Xchart. Apenas uma instância desse Xchart pode estar com o estado **Selected** ativo. Esse Xchart, dessa forma, captura a faceta controle de um agente PAC ou ainda o comportamento de um objeto view-controller de MVC conforme a organização do software realizada na presente seção.

Uma das formas de ativar o estado **Selected** é pela execução da transição que parte do estado **Unselected**. Quando **Selected** é ativado, o evento **Unselect** é enviado para todas as instâncias de **Color** através da regra **entry**. Qualquer outra instância cujo estado **Selected** esteja ativo terá este estado desativado pela transição que conduz ao estado **Unselected**. Isso significa que a seleção dos itens na caixa de diálogo é exclusiva.

A regra **entry** do Xchart **Controller**, executada quando uma instância desse Xchart é criada, envia o evento **black** para a instância "Black". Esse evento é descendente de **Cor** (hierarquia de tipos de eventos) e, desse modo, a sua ocorrência também significa a ocorrência de **Cor**. Por simplicidade, apenas as regras relativas às cores vermelho, verde e azul são apresentadas. No interior de **Controller**, a regra

in(Max,"R") and in(Min,"G") and in(Min,"B") : raise(select,"yellow");

estabelece um relacionamento entre os elementos de interação da caixa de diálogo: quando a instância "R" de **RGB** estiver com o estado **Max** ativo e as instâncias "G" e "B" com os respectivos estados **Min** ativos, então a cor amarela, item exclusivo do grupo **Colors**, deve ser selecionada. Nesse caso, uma regra em Xchart foi utilizada para manter esse relacionamento entre os elementos dessa interface. No sentido inverso, quando o evento **Mselect** é gerado (ação do usuário sobre o mouse em um dos itens de **Colors**), as barras horizontais de rolagem e os campos de valores são devidamente atualizados através de eventos específicos gerados para as instâncias pertinentes. Por exemplo,

red : raise(max,"R"); raise(min,"G"); raise(min,"B");

atualiza as três instâncias que representam as barras de rolagem.

Dado o comportamento descrito acima, as arquiteturas PAC e MVC poderiam ser implementadas através dos recursos do ambiente Xchart. Um cliente de apresentação pode gerenciar as facetas de apresentação da hierarquia de agentes PAC (Figura 6-9) ou ainda os objetos view-controller de MVC (Seção 6.7.2). Um cliente de aplicação pode fornecer as facetas de abstração (Figura 6-9) ou o único objeto model de MVC.

Uma outra organização poderia identificar um cliente de apresentação para cada faceta da hierarquia de agentes PAC (Figura 6-9). Tal organização permitiria a execução concorrente de cada um dos elementos da interface. Contudo, esse comportamento não é o esperado para a interface e não foi utilizado. Em contrapartida, as facetas de controle são retratadas com mais fidelidade. Há um Xchart **Color**, por exemplo, cujas instâncias fornecerão o controle de cada um dos subagentes do agente **Colors**.

As comunicações entre os elementos definidos pelas arquiteturas seriam realizadas através de eventos trocados entre clientes/instâncias e entre instâncias. A comunicação entre um objeto view-controller e o objeto model, por exemplo, seria realizada entre o cliente de apresentação e o cliente de aplicação, possivelmente envolvendo eventos e/ou a recuperação de controle depositado em portas.

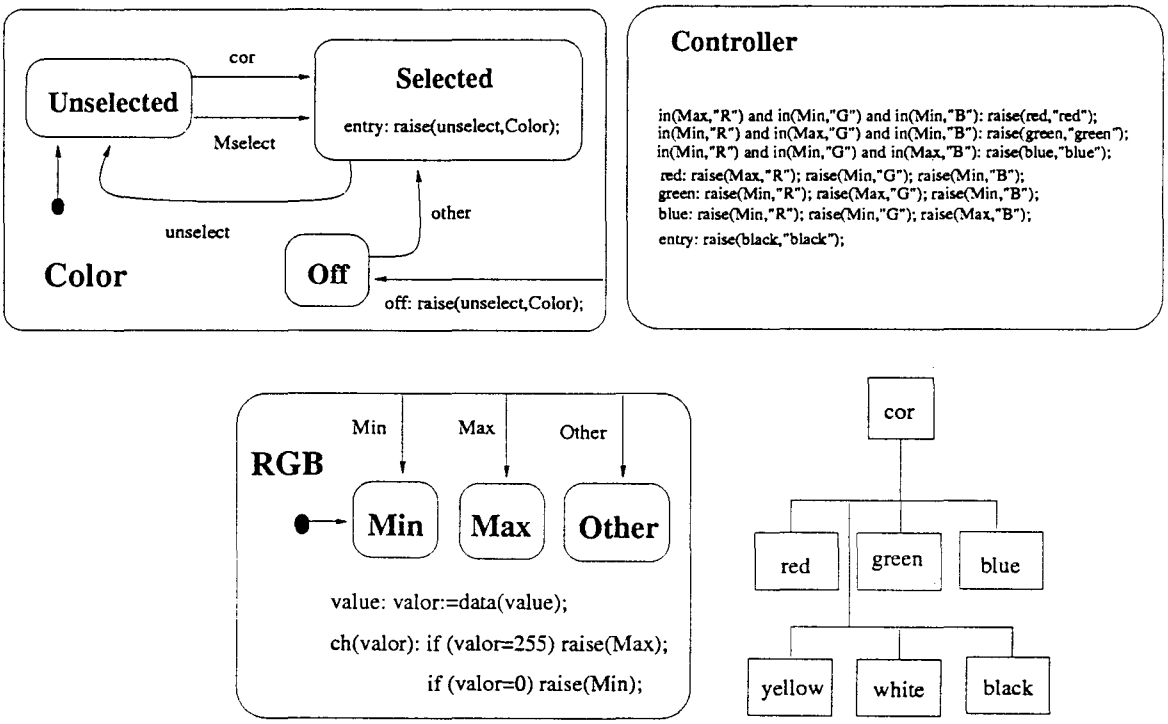


Figura 6-10: Comportamentos e interações entre elementos de interface

6.8 Avaliação do Uso de Xchart

Evitou-se, no presente capítulo, fornecer detalhes de implementação dos exemplos envolvendo o protocolo IPX. Exemplos pertinentes podem ser obtidos em [83]. Os exemplos exploraram uma combinação de esboços de tela (apresentação), linguagem natural e descrições em Xchart para elucidar as funções de controle das interfaces apresentadas. Esse trio reforça a união benéfica entre linguagens informais e formais, conforme sustentado em [92].

A organização do software de um sistema interativo através de clientes e portas não exige o emprego de callbacks, cujo emprego possui inconvenientes [96]. Um cliente não necessita registrar um recurso oferecido a outro módulo com tal módulo antecipadamente, como acontece com o uso de callbacks. Se uma callback é substituída por outra, por exemplo, então necessariamente código deve ser modificado e novamente compilado. Em Xchart, o relacionamento entre clientes é intermediado por instâncias de Xcharts. Um cliente não precisa nem mesmo ter conhecimento da existência de um outro. Naturalmente, se toolkits convencionais são utilizados, então é inevitável o emprego de callbacks. Contudo, elas perdem o papel de destaque e são utilizadas exclusivamente no interior de clientes. Em alguns casos, o número de callbacks pode ser reduzido com o apoio de ferra-

mentas especializadas. Por exemplo, ao selecionar um dos itens do grupo **Colors**, o evento correspondente ao item selecionado pode ser automaticamente gerado pela substituição da callback associada ao widget por acesso ao protocolo IPX.

Clientes e portas ainda podem ser efetivamente utilizados para eliminar o acoplamento entre interface e aplicação, conforme ilustrado na Seção 6.6. Embora não seja o propósito de Xchart, a linguagem e o ambiente pertinentes mostraram-se suscetíveis de serem empregados em sistemas organizados segundo as arquiteturas PAC e MVC (Seção 6.7).

Os exemplos apresentaram alguns dos recursos de Xchart que não existem em Statechart (Seções 6.2, 6.3 e 6.4). Em todos eles, a noção de instância favorece a descrição do controle de elementos cujas implementações, em geral, empregam o paradigma de objetos. O controle de cada um dos itens do grupo **Colors**, Seção 6.7, ilustra um emprego positivo da noção de instâncias e o relacionamento entre elas.

Outros recursos, contudo, podem ser incorporados para facilitar algumas tarefas. Alguns exemplos exibidos no presente capítulo, por exemplo, sugerem que parametrização pode ser efetivamente empregada em Xchart (trabalhos futuros, contudo, deverão ser realizados). As variáveis de Xchart não residem no mesmo espaço de endereçamento de clientes, o que dificulta consultas dos clientes aos valores de variáveis de instâncias via IPX e impede, no sentido inverso, que instâncias consultem valores de memória gerenciada por clientes. Na versão corrente da linguagem Xchart, a regra

e[f(x)] : y− > left = (char∗)0;

não é permitida. Tal comportamento teria que ser substituído, por exemplo, por

e and f : sync(atv);

onde o tipo de evento "artificial" **f** teria que ser criado para modelar a situação na qual a função **f(x)** (responsabilidade de um cliente) retorna um valor verdadeiro. A atividade **atv** seria executada de forma síncrona. Em C essa atividade poderia ser implementada da seguinte forma:

```
void atv (void) {
  y->left = (char*)0;
}
```

As ferramentas [2] e [42] permitem o compartilhamento de espaços de endereçamentos proibido em Xchart. Essa proibição veio de experiências com o sistema de execução proposto em [81]. A simulação também é dificultada com esse recurso [144]. Embora útil em muitos casos, esse recurso permite que clientes interfiram diretamente no comportamento de uma instância de forma imprópria, o que se desejava evitar. Trabalhos futuros podem revelar a necessidade de "relaxar" essa restrição.

Capítulo 7

Conclusões

Acerca da descoberta do enfraquecimento da camada de ozônio, Carl Sagan disse:
Um resultado importante para todos os habitantes da terra proveio do que poderia parecer a pesquisa menos realista, mais abstrata e menos prática, compreender a química de elementos secundários na atmosfera superior de um outro mundo [*Vênus*].
Carl Sagan [pág. 271]
Pálido Ponto Azul
Uma Visão do Futuro da Humanidade no Espaço

7.1 Posfácio

Os altos custos do desenvolvimento do software de interfaces permanecem, mesmo com os progressos e ganhos significativos proporcionados por algumas abordagens. Muitas delas alongam ao máximo uma tecnologia baseada, principalmente, nos atuais ambientes de programação. Por exemplo, frameworks como MFC e vários toolkits fazem uso de C++, assim como o recente AWT faz uso de Java. VISUAL BASIC é outra abordagem que tem mostrado bons resultados no meio industrial. Os resultados positivos, contudo, estão aquém de satisfatórios.

Essa tese apresentou uma abordagem que não tenta estender o sucesso parcial das ferramentas que dominam o mercado ou os ambientes de produção. Embora tenha havido uma preocupação em tornar a proposta apresentada compatível com a tecnologia existente e empregada, Xchart não significa um pouco mais de "açúcar". Xchart acredita no futuro de abordagens baseadas em modelos (Seção 2.4.1).

Não há como antecipar a tecnologia que irá se sobressair e, ao mesmo tempo, os benefícios das abordagens atualmente utilizadas parecem distanciar-se cada vez mais das crescentes necessidades. Um conjunto de modelos de alto nível pode ser a alternativa para o código de baixo nível atualmente produzido pela dupla programador e ambiente

de programação. Tal conjunto de modelos ainda pode produzir melhores resultados onde nenhum modelo ou abordagens incipientes são utilizados para o projeto do software de interfaces. Por exemplo, trabalhos recentes fazem uso de pseudocódigo para capturar as funções de uma interface [67]. Embora pseudocódigo possa ser bem escrito, também não acreditamos nessa abordagem. Xchart é uma proposta promissora. Trabalhos futuros ainda terão que ser realizados para tornar realidade o desenvolvimento de interfaces através do emprego de tais modelos e, em particular, Xchart.

7.2 Trabalhos Futuros

We also learned that formal specification and design are effective under some but not necessarily all circumstances. Their effectiveness may be improved by supplementing them with other approaches, so that in concert they address most of the likely problems of sofware development.
Pfleeger e Hatton [67]

RECURSOS GRÁFICOS

Os recursos gráficos de Xchart são, na maioria, herdados de Statecharts. Trabalhos com aspectos visuais de Xchart, por exemplo, emprego de cores, podem eventualmente melhorar a legibilidade dessa linguagem. A aplicação do conhecimento acerca de percepção visual pode, por exemplo, identificar construções mais legíveis do que as atualmente empregadas para representar os mesmos conceitos.

FUNCIONALIDADE

Xchart foi projetada com o principal objetivo de facilitar o desenvolvimento de gerenciadores de diálogo. Esse, contudo, não é o único componente de uma interface. A comunicação dele com os demais componentes de um sistema interativo precisa ser realizada. A inserção de Xchart em uma abordagem baseada em modelos também exigirá alguma forma de conexão com outros modelos. Alternativamente, Xchart pode ser "integrada" a uma linguagem orientada a objetos como sugerido para Statechart em [42]. Ou ainda, Xchart poderia contemplar mais recursos para facilitar essa integração. Contudo, em ambos os casos, perde-se nitidez quanto ao emprego de Xchart. A distinção entre linguagem de especificação e implementação torna-se tênue. Uma conexão "implícita" poderia, por exemplo, evitar os problemas que as propostas de linguagens de programação de sistemas interativos visam solucionar [97, págs. 115-117]. Xchart é apresentada, no corrente trabalho, como uma linguagem de especificação executável. Em conseqüência, são evitadas as definições de recursos, por exemplo, do ponto de vista de desempenho, que envolvem considerações de implementação típicas de linguagens de baixo nível. Statechart, ao contrário, define o "escopo" de uma transição, por exemplo,

baseando-se no desempenho do código gerado como o fator "mais importante" nessa decisão [44, pág. 329]. Xchart adota a abordagem supostamente de baixo desempenho, mas que consideramos ser mais intuitiva.

EFICIÊNCIA E GERAÇÃO DE CÓDIGO

A versão corrente do ambiente Xchart interpreta especificações em Xchart. Enquanto essa abordagem fomenta o processo iterativo do projeto de interfaces, o desempenho é prejudicado. Experiências iniciais (Capítulo 5) sugerem que o desempenho pode ser melhorado, principalmente se o modelo de implementação for alterado. Por exemplo, condições e ações em Xchart não são convertidas em código C, que se beneficia dos inúmeros esforços para otimização desse código. Essa última opção, contudo, dificulta a simulação de descrições onde diagramas podem compartilhar variáveis com clientes, assim como rotinas e outras. Esse inconveniente é discutido em [144].

EXPERIMENTAÇÃO

O emprego de Xchart em casos reais e práticos deve ser estimulado. Só a partir dessas experimentações o aperfeiçoamento dessa linguagem será obtido assim como ocorre com Statechart. Por exemplo, em [40] e [45], transições excludentes não é uma questão claramente abordada. O problema é identificado mas nenhuma proposta é apresentada como em [44] (nove anos mais tarde), onde a transição associada a estado de mais alto nível recebe maior prioridade. Em [42] (um ano após a última proposta), contudo, há uma inversão: aquela associada a estado de mais baixo nível recebe mais prioridade. Por último, a proposta apresentada em [44] é acompanhada de uma outra, que será adotada em futuras versões do STATEMATE. Além dos recursos da linguagem, que podem sofrer ajustes ao longo do tempo, Xchart propõe mecanismos para a integração entre diagramas que descrevem controle e a arquitetura de software do sistema pertinente. Trabalhos semelhantes praticamente inexistem [123]. Em conseqüência, evoluções como aquelas citadas acima para Statechart serão fomentadas pelo emprego de Xchart e pelas observações dos usuários da última.

OUTROS DOMÍNIOS

Statechart é aplicada em uma variedade de domínios [7]. A definição de Xchart, em oposição, focaliza um domínio específico (pág. 37). Outros domínios, contudo, podem ser favorecidos com os recursos de Xchart [71]. Experiências do emprego de Xchart no desenvolvimento de sistemas reativos em geral, por exemplo, permitiria uma comparação mais apropriada entre Xchart e outras linguagens utilizadas nesse contexto, inclusive Statechart e suas variantes.

ESFORÇOS DE IMPLEMENTAÇÃO

Embora trabalhos consideráveis já tenham sido realizados (Capítulo 5), uma versão em

Java e um novo projeto de algumas das ferramentas são tarefas previstas para serem realizadas. A versão em Java, por exemplo, sempre foi um objetivo perseguido. Sem o amparo de ferramentas úteis e de qualidade, a avaliação do impacto de Xchart fica prejudicado em relação aos elaborados projetos de outras abordagens. Um simulador, por exemplo, pode auxiliar o usuário de um Xchart assim como a ferramenta descrita em [15] traz benefícios aos usuários de Statechart.

7.3 Considerações Finais

Xchart é uma linguagem que ainda deve passar por um processo de evolução. Da mesma forma que C/C++, e mesmo JAVA (bem recente) têm passado por mudanças, é natural esperar alterações também em Xchart. Observações sobre Xcharts deverão ser coleciona-das e analisadas à medida que ela for empregada. O fruto desse processo e a execução dos trabalhos futuros propostos podem revelar a necessidade de adaptações.

Acreditamos que as contribuições (Seção 1.3, pág. 4) foram obtidas. As diferenças entre Xchart e Statechart (Seção 3.11) são frutos de uma tentativa para equipar melhor uma proposta para descrever gerenciadores de diálogo. O modelo de controle (instâncias de diagramas, clientes e portas) e o ambiente de apoio à Xchart adequam-se mais aos requisitos de gerenciadores de diálogo do que o emprego **ad hoc** de notações para a descrição de controle em geral. Muitas dessas últimas são empregadas nesse domínio sem a consideração de itens importantes como arquiteturas de software, por exemplo. Essa crença é fundamentada nas observações do emprego de Xchart (Seção 6.8) e nas dimensões do problema em questão (Capítulo 2). Felizmente, a semântica formal de Xchart também é fornecida. Essa semântica e a sua implementação fornecem um alicerce sólido onde experiências e os trabalhos futuros terão origem.

Entendemos que as contribuições significam um progresso do ponto de vista de abor-dagens baseadas em modelos para desenvolvimento de gerenciadores de diálogo. Em particular, aqueles modelos que fazem uso de máquinas de estados finitos. Os produtos da indústria, apesar das dificuldades, ainda continuarão com a preferência dos programa-dores. Linhas de pesquisas similares àquela adotada no corrente trabalho terão que atingir um grau de desenvolvimento e maturidade bem superiores antes de povoarem prateleiras.

Bibliografia

[1] Heather Alexander. Structuring Dialogues Using CSP. In Michael Harrison and Harold Thimbleby, editors, *Formal Methods in Human-Computer Interaction*, pages 273–295, 1990. Cambridge University Press.

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

[3] Edilmar L. Alves. *Port System:* Sistema de Comunicação em Grupo para o Ambiente Xchart. Master's thesis, DCC/IMECC/UNICAMP, Campinas/SP, Fevereiro 1996.

[4] Ronald M. Baecker and William A. S. Buxton, editors. *Readings in Human-Computer Interaction – A Muldisciplinary Approach.* Morgan Kaufmann, 1987.

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

[6] 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.

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

[8] Thomas Berlage. Using Taps to Separate the User Interface from the Application Code. *UIST*, pages 191–198, November 1992.

[9] Gérard Berry and Georges Gonthier. The ESTEREL Synchronous Programming Language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, November 1992.

[10] Michael A. Bertrand and Willian R. Welch. Programming Windows Using State Tables. *Supplement to Dr. Dobb's Journal*, December 1991.

[11] Barry W. Boehm. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, pages 61–72, 1988.

[12] Alan Borning and robert Duisberg. Constraint-Based Tools for Building User Interfaces. *ACM Transactions on Graphics*, 5(4):345–374, October 1986.

[13] Judy Brown. Exploring Human-Computer Interaction and Software Engineering Methodologies for the Creation of Interactive Software. *SIGCHI Bulletin*, 29(1):32–35, January 1997.

[14] Luiz Eduardo Buzato. *Management of Object-Oriented Action-Based Distributed Programs.* PhD thesis, University of Newcastle upon Type, October 1994.

[15] João W. L. Cangussu, Paulo C. Masiero, and José C. Maldonado. Execução Programada de Statecharts. *Revista Brasileira de Computação*, 7(2):3–14, Jun 1994.

[16] L.M.F. Carneiro, D.D. Cowan, C.J.P. Lucena, and D. Smith. An Experience Using JASMINUM - Formalization Assisting with the Design of User Interfaces. *ICSE'94 Workshop on Software Engineering and Human-Computer Interaction*, pages 141–158, May 1994.

[17] D. A. Carr. *A Compact Graphical Representation of User Interface Interaction Objects.* PhD thesis, University of Maryland, DCS, 1995.

[18] J. Carroll and D. Long. *Theory of Finite Automata.* Prentice Hall, Inc., 1989.

[19] Uli H. Chi. Formal Specification of User Interfaces: A Comparison and Evaluation of Four Axiomatic Approaches. *IEEE Software*, SE-11(8):671–685, August 1985.

[20] Ed. M. Clarke, J. M. Wing, and **et al.** Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.

[21] D. Coleman, F. Hayes, and S. Bear. Introducing ObjectCharts or How to Use Statecharts in Object-Oriented Design. *IEEE TSE*, 18(1):9–18, January 1992.

[22] Rational Software Corporation. Unified Modeling Language (UML) Summary, September 1997. `http://www.rational.com/uml/`.

[23] Alan M. Davis. A Comparison of Techiniques for the Specification of External System Behavior. *Communications of the ACM*, 31(9):1098–1115, September 1988.

[24] Nancy Day. A Model Checker for Statecharts (Linking CASE Tools with Formal Methods). Master's thesis, University of British Columbia, Vancouver, Canada, 1993. Department of Computer Science.

[25] Luciana de Paula Brito. Suporte de Transação Atômica para a Linguagem Xchart. Master's thesis, DCC/IMECC/UNICAMP, Campinas/SP, 1996.

[26] Ernst Denert. Specification and Design of Dialogue Systems with State Diagrams. *International Computing Symposium*, pages 417–423, 1977.

[27] J. F. DeSoi and W. M. Lively. Survey and Analysis of Nonprogramming Approaches to Design and Development of Graphical User Interfaces. *Information and Software Technology*, 33(6):413–424, July 1991.

[28] Stephen W. Draper and D. A. Norman. Software Engineering for User Interfaces. *IEEE Software*, SE-11(3):252–258, March 1985.

[29] Doron Drusinsky. How To Make Statecharts Work for You. *BetterState Tutorial*, 1997. `http://www.isi.com/Products/BetterState/`.

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

[31] T. Faison. Object-Oriented State Machine. *Software Development*, 1(3):37–50, 1993.

[32] Mark A. Flecchia and R. Daniel Bergeron. Specifying Complex Dialogs in ALGAE. In *Proceedings of the ACM CHI+GI'87*, pages 229–234, April 1987.

[33] Martin Glinz. An Integrated Formal Model of Scenarios Based on Statecharts. *LNCS*, 989:254–271, 1995.

[34] T. C. Nicholas Graham and Tores Urnes. Relational Views as a Model for Automatic Distributed Implementation of Multi-User Applications. *CSCW 92 Proceedings*, pages 59–66, November 1992.

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

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

[37] Mark Green and Robert Jacob. SIGGRAPH'90 Workshop Report: Software Architectures and Metaphors for Non-WIMP User Interfaces. *Computer Graphics*, 25(3):229–235, July 1991.

[38] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems.* Kluwer Academic Publishers, 1993. ISBN 0-7923-9311-2.

[39] Anthony Hall. Using Formal Methods to Develop an ATC Information System. *IEEE Software*, 13(2):66–76, March 1996.

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

[41] D. Harel. Biting the Silver Bullet. *IEEE Computer*, pages 8–20, January 1992.

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

[43] 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. Produto comercializado pela i-Logix, Inc., `http://www.ilogix.com/`.

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

[45] 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.

[46] Michael Harrison and Harold Thimbleby, editors. *Formal Methods in Human-Computer Interaction.* Cambridge University Press, 1990. ISBN 0-521-37202-X.

[47] H. Rex Hartson and Deborah Hix. Toward Empirically Derived Methodologies and Tools for Human-Computer Interface Development. *Int. J. Man-Machine Studies*, pages 477–494, 1989.

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

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

[50] Pascal Van Hentenryck and Vijay Saraswat **et al.** Strategic Directions in Constraint Programming. *ACM Computing Surveys*, pages 701–726, December 1996.

[51] Ralph D. Hill. Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction — The Sassafras UIMS. *ACM TOG*, 5(3):179–210, July 1986.

[52] Ralph D. Hill. Event-Response Systems — A Technique for Specifying Multi-Thread Dialogues. In *Proceedings of the ACM CHI+GI'87*, pages 241–248, April 1987.

[53] 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.

[54] Deborah Hix and H. Rex Hartson. *Developing User Interfaces: Ensuring Usability Through Product & Process*. John Wiles & Sons, Inc., 1993. ISBN 0471-57813-4.

[55] Hyoung Seok Hong, Jeong Hyun Kim, Sung Deok Cha, and Yong Rae Kwon. Static Semantics and Priority Schemes for Statecharts. *COMPSAC' 95*, pages 114–120, 1995. Dallas, Texas, USA.

[56] Scott E. Hudson and Ian Smith. Ultra-Lightweight Constraints. *User Interface Software Technology*, pages 147–155, November 1996.

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

[58] C. Huizing, R. Gerth, and W. P. de Roever. Modelling Statecharts Behaviour in a Fully Abstract Way. In *Lecture Notes in Computer Science*, pages 271–294. Springer-Verlag, 1988. Number 299.

[59] W. D. Hurley and J. L. Sibert. Modeling User Interface-Application Interactions. *IEEE Software*, pages 71–77, January 1989.

[60] 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.

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

[62] Osvaldo Severino Junior. Smart: Um Editor Gráfico para os Diagramas Xchart. Master's thesis, DCC/IMECC/UNICAMP, Campinas/SP, Fevereiro 1996.

[63] David J. Kasik, Michelle A. Lund, and Henry W. Ramsey. Reflections on Using a UIMS for Complex Applications. *IEEE Software*, pages 54–61, January 1989.

[64] Y. Kesten and A. Pnueli. Timed and Hybrid Statecharts and their Textual Representation. *Lecture Notes in Computer Science*, 571:591–620, 1992.

[65] Daniel Klein. Developing Applications with the Alpha UIMS. *Interactions*, 2(4):48–65, October 1995.

[66] 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.

[67] Shari Lawrence. Investigating the Influence of Formal Methods. *IEEE Computer*, 30(2):33–43, February 1997.

[68] Geoff Lee. *Object-Oriented GUI Application Development.* Prentice-Hall, 1993.

[69] K.R.P.H. Leung and D.K.C. Chan. Extending Statecharts with Duration. In *Proceedings of the 12th Annual International Computer Software and Application Conference*, Seoul, Korea, August 1996. IEEE Press.

[70] Claus Lewerentz and Thomas Lindner, editors. *Formal Development of Reactive Systems.* Springer-Verlag, 1995. LNCS 891.

[71] Hans Kurt Edmund Liesenberg, Fábio Nogueira de Lucena, and Luiz Eduardo Buzato. Toward Dinamically Contextualized Image Maps. *Em preparação*, 1997. URL: `http://www.dcc.unicamp.br/proj-xchart/preparacao/`.

[72] Fábio N. Lucena. Construção de Interfaces Homem-Computador: O Uso de Estadogramas na Especificação e Implementação de Interfaces. Master's thesis, DCC/IMECC/UNICAMP, Campinas/SP, 1993.

[73] Fábio N. Lucena, Claudine S. Badue, Kleber V. Cardoso, Ricardo Pereira da Silva, and Rogério de Paula Carvalho. Projeto de Interação de um Editor para Xchart. *Em preparação*, 1997. `http://www.dcc.unicamp.br/proj-xchart/editor/`.

[74] 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.

[75] 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.

[76] Fábio N. Lucena and Carlos N. Júnior. Exemplos Comentados de Xchart. *Em preparação*, 1997. `http://www.dcc.unicamp.br/proj-xchart/exemplos/`.

[77] Fábio N. Lucena, Carlos N. Júnior, and Hans K.E. Liesenberg. Biblioteca TeXchart. *Em preparação*, 1997. `http://www.dcc.unicamp.br/proj-xchart/texchart/`.

[78] Fábio N. Lucena, Carlos N. Júnior, and Tallys H. Yunes. Descrição da Interface Homem-Computador do SPRING usando Xchart. *Em preparação*, 1997. `http://www.dcc.unicamp.br/proj-xchart/spring/`.

[79] Fábio N. Lucena, Carlos Neves Júnior, Tallys Hoover Yunes, Hans K.E. Liesenberg, and Luiz Eduardo Buzato. Especificação da Linguagem TeXchart. *Em preparação*, 1997. `http://www.dcc.unicamp.br/projects/Xchart/texchart/`.

[80] Fábio N. Lucena, Carlos Neves Júnior, Tallys Hoover Yunes, Hans K.E. Liesenberg, and Luiz Eduardo Buzato. Desenvolvimento do Servidor Xchart. *Em preparação*, 1997. `http://www.dcc.unicamp.br/proj-xchart/tools/`.

[81] 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.

[82] Fábio N. Lucena and Hans K. E. Liesenberg. Fundamentos de Interfaces Homem-Computador. *Em preparação*, 1997. `http://www.dcc.unicamp.br/proj-xchart/hci/`.

[83] Fábio N. Lucena and Hans K. E. Liesenberg. Interface de Programação de Xchart. *Em preparação*, 1997. `http://www.dcc.unicamp.br/proj-xchart/ipx/`.

[84] 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.

[85] 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.

[86] Luqi and Joseph A. Goguen. Formal Methods: Promises and Problems. *IEEE Software*, 14(1):73–85, 1997.

[87] Jock Mackinlay. Automating the Design of Graphical Presentations of Relational Information. *ACM Transactions on Graphics*, 5(2):110–141, April 1986.

[88] Aeron Marcus and Andries van Dam. User-Interface Developments for the Nineties. *IEEE Computer*, pages 49–57, September 1991.

[89] Satoshi Matsuoka, Shin Takahashi, Tomihisa Kamada, and Akinori Yonezawa. A General Framework for Bidirectional Translation between Abstract and Pictorial Data. *ACM Transactions on Information Systems*, 10(4):408–437, October 1992.

[90] Susan E. McDaniel, Gary M. Olson, and Judith S. Olson. Methods in Search of Methodology – Combining HCI and Object Orientation. In *Human Factors in Computing Systems CHI'94 Conference Proceedings*, pages 145–151, 1994.

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

[92] Bertrand Meyer. On Formalism in Specifications. *IEEE Software*, pages 6–26, January 1985.

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

[94] Brad A. Myers. User Interface Software Tools. HCI Institute, Carnegie Mellon University. URL: `http://www.cs.cmu.edu/~bam/toolnames.html`.

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

[96] Brad A. Myers. Separating Application Code from Toolkits: Eliminating the Spagetti of Call-backs. *UIST*, pages 211–220, November 1991.

[97] Brad A. Myers, editor. *Languages for Developing User Interfaces.* J&B, 1992.

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

[99] Brad A. Myers. User Interface Software Tools. *Transactions on Computer-Human Interaction*, 2(1):64–103, March 1995.

[100] Brad A. Myers. User Interface Software Technology. *ACM Computing Surveys*, 28(1):189–191, 1996.

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

[102] Brad A. Myers, Rich McDaniel, Rob Miller, Alan Ferrency, Patrick Doane, Andrew Faulring, Ellen Borison, Andy Mickish, and Alex Klimovitski. The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transactions on Software Engineering*, 23(6):347–365, June 1997.

[103] Brad A. Myers and Mary Beth Rosson. Survey on User Interface Programming. In *CHI'92 Proceedings*, pages 195–202, Monterey, California, May 1992.

[104] Hisashi Nakatsuyama, Makoto Murata, and Koji Kusumoto. A New Framework for Separating User Interfaces from Application Programs. *SIGCHI Bulletin*, 23(1):88–91, January 1991.

[105] William M. Newman. A System for Interactive Graphical Programming. *Proceedings of the Spring Joint Computer Conference*, pages 47–54, 1968.

[106] J. Nielsen. Iterative User-Interface Design. *Computer*, 26(11):32–41, Nov 1993.

[107] D. R. Olsen. Presentational, Syntactic and Semantic Components of Interactive Dialogue Specifications. In Günther E. Pfaff, editor, *User Interface Management Systems*, pages 125–133. Springer-Verlag, 1985.

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

[109] Jeff Prosise. *Programming Windows 95 with MFC.* Microsoft Press, 1996.

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

[111] Dave Roberts, Dick Berry, Scott Isensee, and John Mullaly. Developing Software Using OVID. *IEEE Software*, 14(4):51–57, August 1997.

[112] B. Rodini. Crafting WinApps with STD. *Computer Language*, 7(3):45–50, 1990.

[113] Christopher Rouff. Formal Specification of User Interfaces. *SIGCHI Bulletin*, 28(3):27–33, July 1996.

[114] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, N.J., 1991.

[115] D. Salber, L. Nigay, and Joëlle Coutaz. Extending the Scope of PAC-Amodeus to Cooperative Systems. *Proceedings of CSCW*, pages 22–26, October 1994.

[116] José M. F. Scafi. GUIS: Gerência de Interface de Usuário baseada em Statecharts. Master's thesis, Universidade Federal de São Carlos, Depto. de Computação, 1992.

[117] B. Selic, G. Gullekson, and Paul T. Ward. *Real-Time Object-Oriented Modeling.* John Wiley & Sons, Inc., 1994.

[118] Yen-Ping Shan. Mode: A UIMS for Smalltalk. *ACM Sigplan Notices*, 25(10):258–268, October 1990. ECOOP/OOPSLA Proceedings.

[119] Rosemeire Shibuya, Rosângela D. Penteado, and Paulo César Masiero. Geração de Código a partir de Modelos Comportamentais Especificados por Statecharts. *VIII Simpósio Brasileiro de Engenharia de Software*, pages 253–267, October 1994.

[120] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, second edition, 1992.

[121] Gurminder Singh. VU: Visual User-Interface Design. In *The Visual Computer*, pages 230–241. Springer-Verlag, 1990.

[122] Gurminder Singh and Mark Green. Automating the Lexical and Syntatic Design of Graphical User Interfaces: The UofA UIMS. *ACM Transactions on Graphics*, 10(3):213–254, July 1991.

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

[124] Terence R. Smith. A Digital Library for Geographically Referenced Materials. *IEEE Computer*, 29(5):54–60, May 1996.

[125] P. Szekely. Retrospective and Challenges for Model-Based Interface Development. In *Proceedings of the 3rd Int. Eurographics Workshop on Design, Specification and Verification of Interactive Systems DSV-IS'96*, pages 1–27, June 1996.

[126] P. Szekely, P. Luo, and R. Neches. Beyond Interface Builders: Model-Based Interface Tools. *Human Factors in Computing Systems*, pages 383–390, April 1993.

[127] P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher. Declarative interface models for user interface construction tools: the Mastermind approach. *In Engineering for Humand-Computer Interaction, L. Bass and C. Unger Eds. Chapman & Hall*, 1996.

[128] Richard N. Taylor and Joëlle Coutaz, editors. *Software Engineering and Human-Computer Interaction*, volume 896 of *LNCS*. Springer-Verlag, 1995.

[129] Harold Thimbleby and Ian H. witten. User Modeling as Machine Identification: New Design Methods for HCI. In H. Rex Hartson and Deborah Hix, editors, *Advances in Human-Computer Interaction*, pages 58–86. Ablex Publishing, 1993.

[130] Roger Took. Surface Interaction: A Paradigm and Model for Separating Application and Interface. In *Human Factors in Computing Systems, Proceedings SIGCHI'90*, pages 35–42, Seattle, WA, April 1989.

[131] S.S. Toscani and L.F. Monteiro. Apresentação da Linguagem Reativa Síncrona RS. *VIII Simpósio Brasileiro de Engenharia de Software*, pages 63–77, October 1994.

[132] Laura A. Valaer and Robert G. Babb. Choosing a User Interface Development Tool. *IEEE Software*, 14(4):29–39, August 1997.

[133] A. van Dam, S. Abi-Ezzi, C. Bass, R. Carey, and M. Tarlton. Graphics Software Architecture for the Future. *Computer Graphics*, 26(2):389–390, July 1992.

[134] Lynette van Zijl and Deon Mitton. Using Statecharts to Design and Specify a Direct-Manipulation User Interface. *Southern African Computer Symposium*, pages 51–68, 1991.

[135] Anthony I. Wasserman. Developing Interactive Information Systems with the User Software Engineering Methodology. In B. Shackel, editor, *Human-Computer Interaction — Interact'84*, pages 611–617, 1984.

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

[137] Anthony I. Wasserman. Toward a Discipline of Software Engineering. *IEEE Software*, 13(6):23–31, November 1996.

[138] Pierre D. Wellner. Statemaster: A UIMS based on Statecharts for Prototyping and Target Implementation. In *Proceedings SIGCHI'89*, pages 177–182, April 1989.

[139] Alan Wexelblat. Explicitly Modal Interfaces for Business Professionals. *Interactions*, 1(4):58–66, October 1994.

[140] Alan Wexelblat. Letters & Updates. *Interactions*, 3(1):5–8, January 1996.

[141] G. Winskel. *The formal Semantics of Programming Languages.* MIT Press, 1993.

[142] Catherine A. Wood and Philip D. Gray. User Interface - Application Communication in the Chimera User Interface Management System. *Software – Practice and Experience*, 22(1):63–84, January 1992.

[143] Bradley T. Vander Zanden and Scott A. Venckus. An Empirical Study of Constraint Usage in Graphical Applications. *User Interface Software Technology*, pages 137–146, November 1996.

[144] L. Zucconi. Building Testable Software. *SIGSOFT*, 21(5):51–55, September 1996.

Índice