Projeto de uma Arquitetura de Software Reflexiva para a Linguagem Xchart

Renato O. Stehling Hans K. E. Liesenberg

IC-UNICAMP
Caixa Postal 6176 - 13083-970
Campinas, SP
{renato.stehling, hans}@dcc.unicamp.br

Resumo

Xchart é uma linguagem visual, orientada a eventos e projetada para especificar controles de diálogo que não podem ser adequadamente descritos com as atuais propostas baseadas em DTEs (Diagramas de Transição de Estados). A arquitetura reflexiva proposta é uma extensão que não visa adaptar Xchart a nenhum novo contexto. O objetivo é controlar a complexidade do código descrito nessa linguagem, promovendo a separação de domínios e permitindo a intervenção na especificação de forma transparente. Nossa proposta procurou preservar ao máximo as características da linguagem Xchart. As novas estruturas são tratadas exclusivamente em tempo de compilação. O sistema de execução de Xchart (SE) não precisou ser alterado porque o código executável é o mesmo da abordagem convencional. Em tempo de execução, não há overhead relacionado ao mecanismo de reflexão adotado.

1. Introdução

O termo Xchart designa simultaneamente uma linguagem e um ambiente de desenvolvimento. A linguagem tem por objetivo descrever o controle de diálogo da interface de um sistema interativo. O controle de diálogo é o componente da interface responsável pela sintaxe da interação com o usuário, pelo *layout* dos objetos visuais e pela comunicação entre apresentação e aplicação. Xchart é uma linguagem visual. A maioria de seus recursos é representada por elementos gráficos, grande parte herdados da linguagem Statechart [1]. Especificações nas linguagens Xchart e Statechart são léxica e sintaticamente semelhantes. Da mesma forma que Statechart estende os DTEs (Diagramas de Transição de Estados), Xchart altera a semântica de alguns recursos herdados de Statechart, além de acrescentar novos recursos. Xchart possui recursos capazes de descrever interfaces que não são adequadamente contempladas ou não podem ser descritas com as propostas atuais (baseadas em DTEs). O ambiente Xchart é composto por ferramentas que auxiliam os usuários da linguagem a realizar tarefas típicas, por exemplo, edição de especificações, geração de código e simulação.

Embora a arquitetura proposta estenda os recursos de Xchart, as extensões propostas não visam adaptar Xchart a nenhum novo contexto. O objetivo é oferecer recursos que permitam controlar a complexidade das especificações. Assim, a especificação pode ser compreendida mais facilmente, reduzindo os custos de desenvolvimento e manutenção. Para atingir o objetivo de reduzir a complexidade, foram adicionados recursos que permitem que a especificação seja organizada de acordo com os domínios que a compõem. Cada domínio é

descrito de forma modular, de forma que possa ser analisado, compreendido e mantido independentemente. Essa organização facilita a intervenção de especialistas que, na maior parte dos casos, não precisam conhecer todos os domínios que compõem um sistema. Também foram criados mecanismos que permitem a intervenção no código descrito em Xchart de forma transparente, preservando a robustez da especificação quando elementos são adicionados/removidos em caráter temporário. Essa situação é comum, por exemplo, durante as etapas de depuração, testes e otimização da especificação. Utilizando os novos mecanismos, é possível preservar o código original e as adaptações realizadas. Assim, as adaptações podem ser habilitadas/desabilitadas tantas vezes quantas forem necessárias, sem nenhum tipo de prejuízo.

Para atingir os objetivos de separação de domínios e de intervenção na especificação de forma transparente, foram adicionados recursos à linguagem Xchart que adaptam sua arquitetura ao modelo reflexivo. Esse modelo é descrito na seção 2. Os recursos da linguagem e o ambiente Xchart são descritos na seção 3. Essa seção também mostra um exemplo de um gerenciador de diálogo modelado em Xchart. A seção 4 discute em detalhes a arquitetura reflexiva proposta. Ao longo de toda a seção são fornecidos exemplos de operações que podem ser realizadas utilizando o nível reflexivo. Nossas conclusões são discutidas na seção 5. Alguns trabalhos futuros são identificados na seção 6.

2. Reflexão computacional

O grande desafio para a maioria das tecnologias é o tratamento da complexidade. Nos sistemas de *software*, o conceito de modularidade tem sido amplamente utilizado com esse objetivo. Quando um sistema pode ser dividido em módulos com baixo acoplamento e alto grau de coesão conceitual, sua complexidade pode ser controlada mais facilmente. Tradicionalmente, um módulo pode ser visto como uma caixa-preta, que expõe sua funcionalidade através de uma interface bem-definida e esconde detalhes de sua implementação (Figura 1).

Um módulo deve ser projetado para que possa ser reutilizado em diferentes sistemas. Dessa forma, os custos de desenvolvimento e manutenção podem ser reduzidos. Para que um módulo possa ser reutilizado extensivamente, sua interface deve ser genérica o suficiente para que possa oferecer as funcionalidades requeridas por diferentes clientes. Adicionalmente, a estratégia escolhida para a implementação do módulo deve ser eficiente. A dificuldade de desenvolver módulos reutilizáveis é que os dois requisitos acima são contraditórios: apesar de eficazes, estratégias de implementação genéricas não são necessariamente eficientes em casos específicos. Por outro lado, estratégias de implementação eficientes não são necessariamente genéricas. Ao escolher a estratégia para a implementação de um módulo, o desenvolvedor se compromete com um subconjunto de perfis de clientes **{C'}**, para os quais a estratégia escolhida é otimizada. O conjunto **{C'}** é um subconjunto do conjunto **{C}**, formado por todos os possíveis perfis de clientes. O conjunto **{C}** é potencialmente infinito. Algumas observações importantes podem ser feitas em relação aos clientes com perfil não contemplado (**{C} - {C'}**), ou seja, para os quais a estratégia escolhida não é otimizada:

- Apesar do código escrito por esses clientes expressar de forma simples e elegante o comportamento desejado, a performance obtida é insatisfatória.
- Por não poder reutilizar o módulo de forma apropriada, esses clientes são forçados a codificar sua própria versão do módulo com a estratégia mais adequada a seu perfil. Assim, o código do cliente se torna maior, mais

complexo e menos robusto, aumentando os custos de desenvolvimento e manutenção do sistema como um todo.

Muitas vezes, para oferecer o máximo de eficiência a um subconjunto de clientes, o projetista otimiza não apenas a implementação do módulo, mas também sua interface. Nesses casos, são ocultadas funcionalidades e informações de estado que não são relevantes para o subconjunto de clientes contemplado. De fato, quanto mais funcionalidades e informações acerca do estado interno de um módulo estiverem disponíveis através de sua interface, mais complexa e ineficiente se torna sua implementação. Essas novas otimizações diminuem ainda mais a reusabilidade dos módulos já que, além da performance insatisfatória, clientes com perfil não contemplado também não terão acesso a eventuais funcionalidades e informações de estado de que possam necessitar.

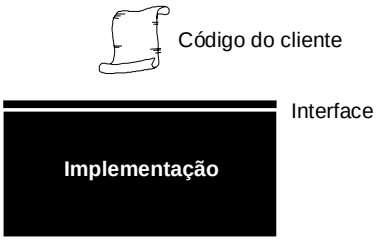

Figura 1 - O modelo de caixa preta

2.1 O modelo de implementação aberta

A prática tem mostrado que um módulo se torna mais útil e reutilizável (extensivamente) se, ao invés de esconder a sua implementação conforme o modelo de caixa-preta, ele a expõe de forma controlada, permitindo a seus clientes ajustá-la às suas necessidades específicas. Esse novo modelo de desenvolvimento é conhecido como **implementação aberta** [6]. O princípio básico do modelo de implementação aberta é que um módulo de *software* pode ser melhor reutilizado se for desenvolvido de forma a acomodar uma ampla faixa de estratégias para a sua implementação. A estratégia mais adequada é selecionada de acordo com o perfil de cada cliente. Esse princípio pressupõe que, ou uma única estratégia é adequada para todos os perfis de clientes, ou o perfil do cliente pode ser determinado automaticamente. Algumas questões relativas a esse modelo são: (1) o que fazer quando o perfil do cliente não puder ser determinado automaticamente? (2) O que fazer quando não existe uma estratégia apropriada para o perfil do cliente?

Com o objetivo de solucionar as questões identificadas acima, foi incorporado ao modelo de implementação aberta a noção de **metainterface**. Uma metainterface é uma segunda interface adicionada ao modelo de caixa-preta, independente da interface original e com objetivos distintos (Figura 2). A interface original expõe as funcionalidades do módulo, determina como os clientes irão utilizar essas funcionalidades e esconde detalhes da implementação. A metainterface, ao contrário, expõe a implementação do módulo de forma controlada, permitindo aos clientes ter acesso e alterar algumas de suas características. A metainterface não oferece acesso irrestrito à implementação, pelo contrário, a maior parte da

implementação permanece oculta, assim como no modelo de caixa preta. Somente detalhes considerados relevantes podem ser examinados e alterados.

Os clientes para os quais existem estratégias de implementação adequadas utilizam apenas a interface do módulo, conforme o modelo de caixa-preta. Os clientes para os quais não existe estratégia adequada, além de utilizarem a interface do módulo, também utilizam a metainterface para otimizar a implementação ao seu perfil. Esses clientes acabam escrevendo dois tipos de código: aquele que utiliza da funcionalidade do módulo e que é relativo ao domínio do sistema que está sendo implementado e um novo tipo de código, ao qual denominaremos **código implementacional**, que otimiza a implementação do módulo ao seu perfil.

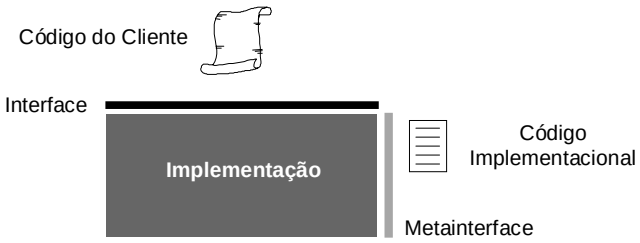

Figura 2 - Metainterface

O projeto de metainterfaces deve atingir dois objetivos muitas vezes conflitantes. Por um lado, as facilidades associadas à metainterface não podem prejudicar a implementação efetiva e eficiente da interface do módulo. Por outro lado, a metainterface deve oferecer aos clientes acesso a aspectos da implementação que possam ser utilizados para criar variações semânticas ou otimizações relevantes.

2.2 O modelo reflexivo

Um módulo/sistema com implementação aberta permite aos seus clientes utilizá-lo conforme o modelo da caixa-preta quando sua implementação for adequada ou, então, alterar sua implementação quando necessário (nesse caso, o modelo de caixa-preta é enfraquecido). Essa capacidade adicional dos clientes de intervir na implementação do módulo/sistema é conhecida como **reflexão implementacional** (RI). RI é uma visão alternativa do conceito de reflexão computacional proposto por Maes [7]. Segundo Maes, **reflexão computacional** (RC) é a atividade realizada por sistemas reflexivos quando esses agem (computam) sobre si mesmos, alterando seu próprio comportamento, estrutura ou status. Sistemas reflexivos são sistemas computacionais que incorporam dados (denominados **auto-representação)** que representam aspectos estruturais e/ou comportamentais do próprio sistema. Diversas atividades computacionais são inerentemente reflexivas, por exemplo, qualquer tipo de estatística sobre a execução de um sistema ou facilidades de monitoração e depuração de programas oferecidas pelos atuais ambientes de desenvolvimento.

O modelo reflexivo [8] sugere um novo paradigma para o desenvolvimento de sistemas. Nesse novo paradigma, um sistema é decomposto em pelo menos dois níveis: o nível base, que agrupa a implementação do sistema, e o nível reflexivo, que agrupa o código implementacional ou reflexivo. Utilizando esse modelo, os sistemas tendem a se tornar mais

estruturados, diminuindo sua complexidade, facilitando sua compreensão e sua manutenção. Algoritmos de políticas, bem como quaisquer outros algoritmos que não estejam diretamente relacionados ao domínio do sistema podem ser implementados no nível reflexivo sem a necessidade de alterações no código do nível base. Os sistemas cujas arquiteturas reflexivas trabalham em tempo de execução introduzem *overhead* adicional ao sistema porque, para executar uma operação no nível base, deve ser executado o código que a implementa no nível reflexivo. As arquiteturas que trabalham em tempo de compilação não introduzem esse tipo de *overhead*. Nessas arquiteturas, o código reflexivo modifica o compilador para que o código compilado se comporte de acordo com as intenções do usuário.

3. Xchart

Em Xchart [5], o controle de diálogo de uma interface é descrito por um conjunto de diagramas denominados **diagramas Xchart**, que servem para a criação de instâncias. Um diagrama Xchart é a menor unidade utilizada para representar controle. A partir de cada diagrama, pode ser criado um número arbitrário de instâncias. As instâncias dos diagramas Xchart interagem entre si (subsistema reativo) e com os demais subsistemas. Os componentes que interagem diretamente com as instâncias dos diagramas Xchart são denominados **clientes**. Os clientes sinalizam estímulos externos para as instâncias que, por sua vez, depositam suas reações em portas. A partir desse instante, qualquer cliente pode consultar a reação produzida na porta pertinente. A comunicação entre os clientes e as instâncias de Xchart é realizada através da **IPX** (Interface de Programação de Xchart) [2]. A IPX é uma interface orientada a objetos, utilizada para ter acesso aos recursos oferecidos pelo Sistema de Execução de Xchart (SE).

Os diagramas Xchart são codificados utilizando-se o Editor de Xchart. O formato visual da descrição na linguagem Xchart é convertido pelo editor, de forma transparente e automática, em um formato texto conhecido como TeXchart [3]. Esse formato é utilizado para armazenar a especificação, para comunicação entre algumas ferramentas do ambiente e também como entrada para o compilador TeXchart. A conversão Xchart/TeXchart é possível nos dois sentidos, sem perda de informação. O compilador TeXchart gera estruturas de dados que serão utilizadas pelo Sistema de Execução de Xchart (SE). Esses dados são armazenados no formato FSDX (Formato de Sistema Descrito em Xchart) [4]. Os digramas Xchart são compostos, tipicamente, por estados, transições e regras. Os estados podem ser refinados em subestados, formando uma hierarquia de inclusão. Graficamente, são representados por um retângulo com cantos arredondados. As transições estabelecem relações entre estados. Toda transição possui, pelo menos, um estados origem, uma regra que a rotula e um estado destino. Ao ser executada, o estado origem é desativado, ativando o estado destino. Graficamente, um transição é representada por uma seta. As regras são utilizadas para estabelecer condições na qual um evento é interceptado e executar uma ação em resposta a esse evento.

3.1 Exemplo

A seguir, será apresentado um exemplo [5] de uma especificação em Xchart onde poderão ser identificados os principais recursos dessa linguagem. Será descrito o controle de diálogo da interface de um sistema que implementa o tradicional “jogo da velha”. Nesse jogo, o início de uma partida requer a existência de dois jogadores. Um dos jogadores inicia a partida escolhendo, com o *mouse*, uma posição no tabuleiro. Quando o *mouse* passa sobre uma

posição desocupada, uma marca cinza preenche a posição. Caso o botão do *mouse* seja pressionado sobre uma posição inválida, uma campainha soa indicando uma exceção. Quando o botão é pressionado sobre uma posição vazia, o símbolo associado ao jogador (**X** ou **O**) preenche definitivamente a posição e o controle é transferido para o jogador adversário. O controle é alternado entre os jogadores enquanto o jogo não termina. Quando o jogo termina, uma pequena animação é executada: uma para o vencedor, parabenizando-o, e outra estimulando o jogador derrotado. Se a partida termina empatada, então uma mesma animação é exibida para ambos. A finalização de uma partida pode ocorrer em qualquer momento, por iniciativa de qualquer jogador. O reinício exige um consenso entre ambos.

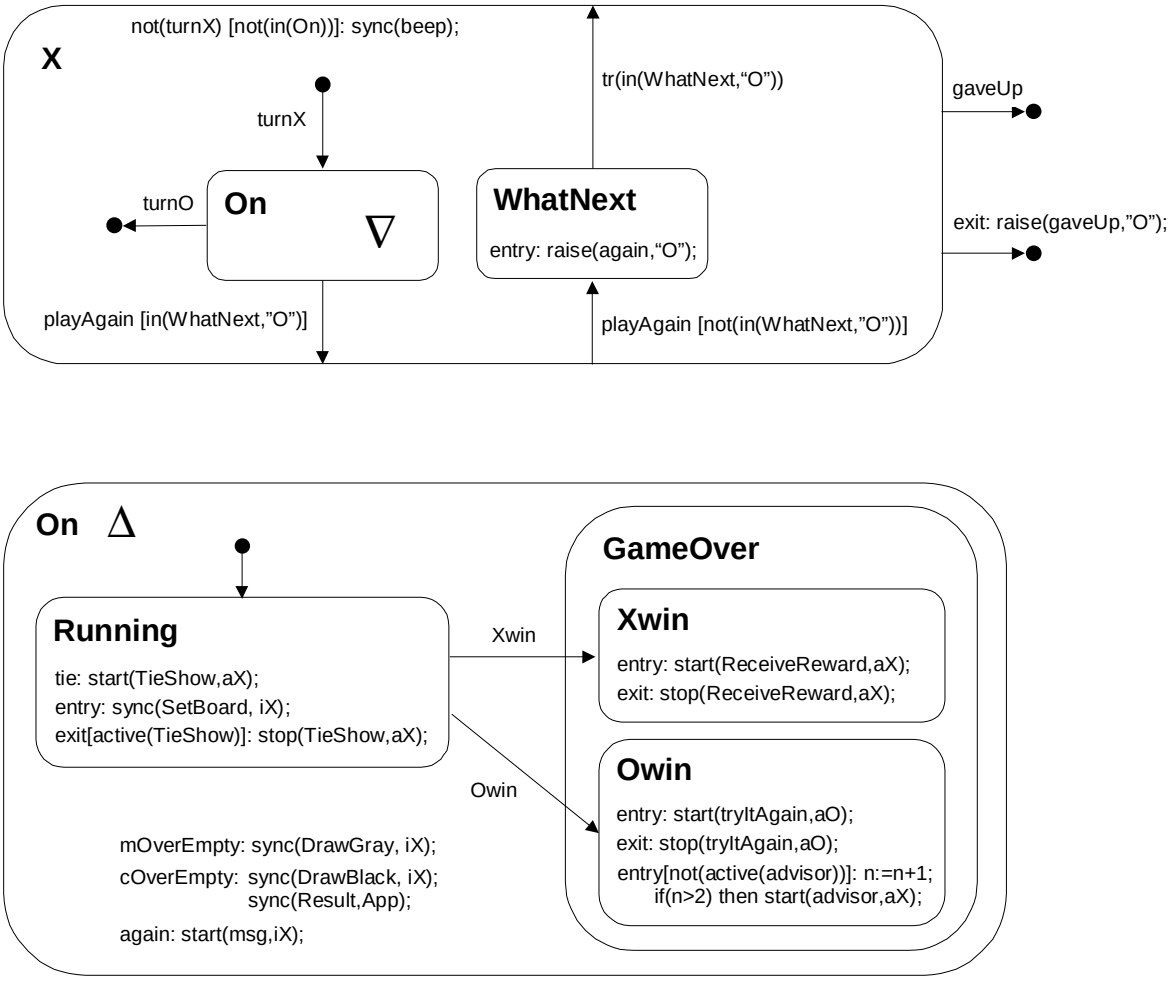

Figura 3 - Gerenciador de diálogo da interface do jogador X

O sistema que implementa o jogo é composto pelo **gerenciador de diálogo** (descrito em Xchart), por um cliente denominado ***Application*** (compartilhado entre os jogadores) e um par de clientes ***Animation*** e ***Interaction*** para cada jogador. Uma descrição mais detalhada desses clientes pode ser encontrada em [5]. A especificação do gerenciador de diálogo referente à interface do jogador **X** é mostrada na Figura 3. Quando uma instância do Xchart **X** é criada e o evento ***turnX*** é sinalizado, o subestado ***On*** é ativado. Esse subestado é desativado pelo evento

turnO e permanece desativado enquanto a jogada corrente é a do adversário. Os detalhes do estado ***On*** são exibidos separadamente.

O estado ***On*** contém algumas regras. Uma delas é executada na ocorrência do evento ***mOverEmpty***, sinalizado quando o indicador do *mouse* passa sobre uma posição não preenchida do tabuleiro. Nesse caso, a reação é a execução da atividade ***DrawGray***. Se o evento gerado é ***cOverEmpty***, então jogador pressionou o botão do *mouse* sobre uma posição vazia do tabuleiro. A reação é preencher a posição com um **X** e avaliar o resultado da jogada (a jogada pode resultar em empate ou em vitória de um dos jogadores). O estado ***On*** agrega os subestados ***Running*** e ***GameOver***. Quando o estado ***On*** é ativado, o subestado ***Running*** é ativado em seguida. O estado ***Running*** contém uma regra que trata os casos de empate (interceptando o evento ***tie*** e executando a atividade ***TieShow***). As transições rotuladas pelos eventos ***Xwin*** e ***Owin*** (partindo do estado ***Running***) capturam, respectivamente, a vitória do jogador **X** ou a vitória do jogador **O**. Essas transições 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.

Se o botão *Exit* é pressionado, o evento ***Exit*** é sinalizado e a transição final do Xchart **X** é executada. Ao tratar esse evento, a instância de **X** sinaliza para a instância de **O** o evento ***gaveUp***, que também provoca a finalização da instância do Xchart **O**. Se o botão *Restart* é pressionado, então o evento ***playAgain*** é sinalizado, provocando a ativação do estado ***WhatNext***. Esse estado só pode ser desativado se algum jogador deseja finalizar a aplicação ou quando o jogador adversário concorda em iniciar uma nova partida.

4. Arquitetura reflexiva para a linguagem Xchart

As vantagens associadas à reflexão computacional motivaram o projeto de uma arquitetura reflexiva para a linguagem Xchart. Outro fator decisivo foi que, embora existam arquiteturas reflexivas para linguagens de programação de diversos paradigmas, não temos conhecimento de nenhum trabalho similar em linguagens orientadas a eventos, como é o caso de Xchart. O paradigma de eventos favorece a criação de uma arquitetura reflexiva pois a transferência de controle entre os níveis base e reflexivo pode ser realizada de forma natural, através da interceptação de eventos. Nossa arquitetura visa atingir dois objetivos básicos: (1) promover a separação de domínios através da organização da especificação em múltiplos níveis e (2) permitir a adaptação do comportamento e da estrutura da especificação, de forma transparente, utilizando o nível reflexivo. Em ambos os casos, o código descrito no nível reflexivo poderia ser descrito diretamente no nível base, utilizando apenas os recursos disponíveis na linguagem Xchart. A vantagem em utilizar o nível reflexivo é conceitual e diz respeito à organização da especificação. A decomposição em níveis, de acordo com os múltiplos domínios envolvidos, é uma alternativa que diminui a complexidade da especificação, facilitando sua compreensão e sua manutenção e aumentando sua robustez. Nesse tipo de organização, cada domínio pode ser analisado, compreendido e mantido de forma mais modular, facilitando a intervenção de especialistas que, na maioria dos casos, não precisam conhecer todos os domínios que compõem um sistema.

A arquitetura proposta procurou preservar ao máximo as características da linguagem Xchart. Foram criadas apenas três novas estruturas: (1) diagrama reflexivo; (2) elemento de ação *reuse*; (3) elemento de ação *remove*. Conforme será mostrado, essas três estruturas são simples e podem ser tratadas exclusivamente em tempo de compilação. Assim, em relação ao ambiente Xchart, apenas o compilador TeXchart precisa ser adaptado. O sistema de execução

de Xchart permanece inalterado. De acordo com o modelo reflexivo, os elementos de ação *reuse* e *remove* não seriam necessários explicitamente. Seria função do Sistema de Execução de Xchart (SE) realizar essas operações de forma transparente (a operação a ser realizada seria uma das metainformações disponíveis quando um evento fosse interceptado). Resolvemos adicionar, a principio, os elementos de ação *reuse* e *remove* à linguagem (relaxando o modelo reflexivo) por dois motivos: (1) tornar mais claras as operações associadas a esses elementos e (2) facilitar uma implementação inicial da arquitetura (em tempo de compilação). A medida que nossa arquitetura for estendida e passar a ser tratada em tempo de execução (seção 5), inevitavelmente, esses elementos de ação deixarão de existir explicitamente na linguagem Xchart.

4.1 Diagrama reflexivo

Um diagrama reflexivo pode ser considerado uma especialização de um diagrama Xchart. A principal diferença entre esses dois tipos de diagramas é a restrição de que um diagrama reflexivo deve estar associado a pelo menos um diagrama Xchart ao qual estende, ou então, a outro diagrama reflexivo (formando uma torre reflexiva). Para diferenciar os dois tipos de diagramas, adotaremos a convenção de que, visualmente, um diagrama reflexivo apresenta um preenchimento cinza. A associação entre dois diagrama reflexivos ou entre um diagrama reflexivo e um diagrama Xchart é representada por uma linha unindo os diagramas.

A implementação do sistema (corresponde ao nível base) deve sempre ser modelada utilizando-se diagramas Xchart. Os demais domínios (ou extensões do nível base) devem ser descritos utilizando-se diagramas reflexivos. A arquitetura proposta é parcialmente reflexiva pois um diagrama Xchart pode ter ou não um diagrama reflexivo associado a si. Utilizando essa arquitetura, é possível adicionar, remover ou alterar regras, transições e estados em qualquer um dos níveis da hierarquia de estados de um diagrama Xchart. Para isso, o diagrama reflexivo deve repetir os estados do diagrama Xchart que serão estendidos. No caso de extensões a um subestado da hierarquia, o diagrama reflexivo deve conter o ramo da hierarquia ao qual o subestado pertence, até o nível desse subestado. Somente as regras e transições do diagrama Xchart que serão estendidas precisam ser repetidas no diagrama reflexivo. A separação de domínios em diferentes diagramas existe apenas no nível da especificação. Durante a compilação, o código dos diagramas reflexivos é mesclado ao código dos diagramas Xchart aos quais estão associados. A responsabilidade pelo código descrito no nível reflexivo, assim como acontece quando os domínios estão todos misturados nos diagramas Xchart (abordagem convencional), é do programador.

A seguir, serão apresentados alguns exemplos de diagramas reflexivos alterando diferentes características dos diagramas Xchart aos quais estão associados. Para facilitar a compreensão, em todos os exemplos, o diagrama Xchart é denominado **D**. O diagrama reflexivo associado ao diagrama Xchart é denominado **D-REFLEXIVO.** Com a intenção de esclarecer melhor como é feita a mesclagem desses diagramas, todo exemplo também irá conter um diagrama denominado **D-ESTENDIDO** (na prática, o diagrama obtido após a mesclagem possui o mesmo identificador do diagrama Xchart)**,** que é o diagrama conceitual obtido após a mesclagem do diagrama reflexivo e do diagrama Xchart. O código executado pelo SE é obtido da compilação de **D-ESTENDIDO**.

A Figura 4 mostra dois exemplos de como adicionar regras a um diagrama Xchart utilizando diagramas reflexivos. No exemplo (a), o diagrama Xchart **D** aparece vazio. O diagrama **D-REFLEXIVO** estende **D** adicionando a regra `E[x<0]: x:=x+1;`. O exemplo (b) mostra

a adição de uma regra no interior de um dos subestados do diagrama **D**. Observe que o subestado **B** e a transição de **A** para **B** não precisaram ser repetidos em **D-REFLEXIVO** porque esses elementos não sofreram qualquer tipo de alteração. O diagrama **D-REFLEXIVO** referenciou apenas o subestado **A**, adicionando a esse estado a regra E[x>1]: y:=TRUE;.

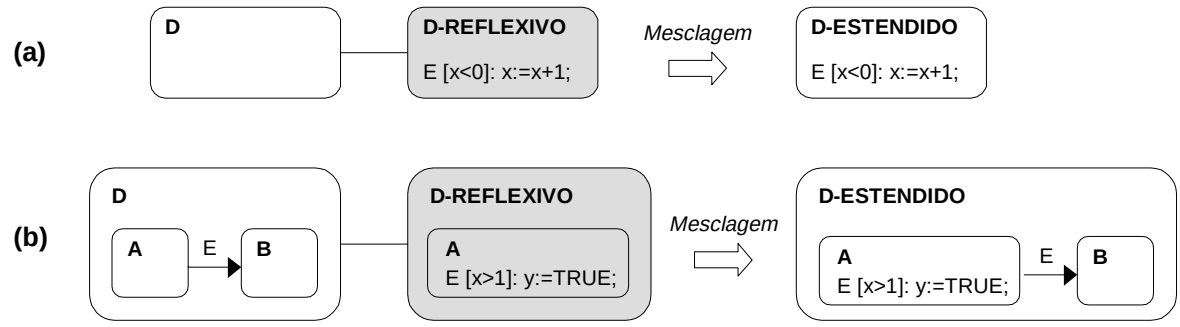

Figura 4 – Exemplos de adição de regras

A Figura 5 mostra como utilizar um diagrama reflexivo para adicionar transições e estados a um diagrama Xchart. O diagrama **D-REFLEXIVO** estende o diagrama **D** adicionando o estado **C** e a transição do estado **A** para o **C** (rotulada pelo evento E1). Essa transição é utilizada para ativar o estado **C**. Novamente, como não houve nenhum tipo de extensão ao estado **B** ou à transição de **A** para **B** (rotulada pelo evento E), esses elementos não precisaram ser referenciados em **D-REFLEXIVO**. Embora o estado **A** também não tenha sido estendido, o diagrama **D-REFLEXIVO** precisou referenciá-lo para que a transição de **A** para **C** pudesse ser completamente caracterizada no nível reflexivo.

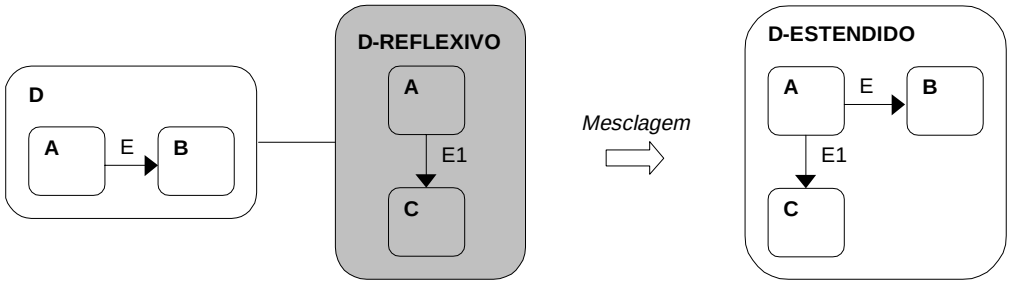

Figura 5 – Exemplo de adição de transição e estado

4.2 Elemento de ação *reuse*

Muitas vezes, o domínio descrito no nível reflexivo precisa adicionar alguns elementos de ação a uma regra já existente no nível base. Para simplificar esse tipo de operação, foi criado um novo elemento de ação denominado *reuse*. *Reuse* é utilizado para reutilizar a ação de uma

regra do nível base no nível reflexivo, evitando cópias redundantes dessa ação. Durante a mesclagem dos diagramas, *reuse* é substituído pela ação da regra à qual referencia. O procedimento para utilização de *reuse* é o seguinte: (1) repetir o gatilho da regra do diagrama Xchart que será estendida no diagrama reflexivo (dentro do estado correspondente); (2) referenciar a ação dessa regra utilizando *reuse*; (3) adicionar os novos elementos de ação antes e/ou depois de *reuse*. Caso o gatilho da regra seja repetido mas *reuse* não seja utilizado, a ação da regra do nível base (diagrama Xchart) será substituída pela ação descrita no nível reflexivo (diagrama reflexivo).

A Figura 6 mostra dois exemplos de utilização de *reuse* para estender a ação de uma regra. No exemplo (a), **D-REFLEXIVO** adiciona o elemento de ação y:=0; ao final da ação da regra descrita no diagrama **D**. A utilização de *reuse* antes de y:=0; (em **D-REFLEXIVO**) determina essa posição. No exemplo (b), a regra E[x<0]: x:=x+1;, descrita no diagrama **D**, foi referenciada em **D-REFLEXIVO** sem a utilização de *reuse* (apenas o gatilho E[x<0] foi repetido). Nessas condições, a ação descrita em **D** (x:=x+1;) é substituída pela ação descrita em **D-REFLEXIVO** (x:=0;).

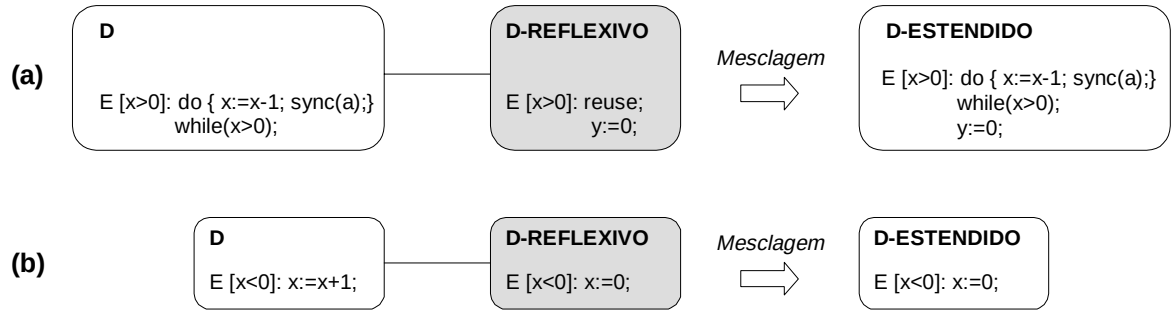

Figura 6 – Exemplos de alteração de regras

4.3 Elemento de ação *remove*

Para realizar adaptações em um diagrama Xchart, além da capacidade de estendê-lo, deve ser possível também remover elementos desse diagrama. Para realizar esse tipo de operação, foi criado o elemento de ação *remove*. Esse elemento é utilizado, basicamente, para remover uma regra descrita no nível base. Para remover uma regra, basta referenciá-la no nível reflexivo (repetindo seu gatilho) substituindo a ação original pelo elemento de ação *remove*. Durante a mesclagem dos diagramas, a presença de *remove* no nível reflexivo causa a eliminação da regra correspondente no nível base.

A remoção de regras pode ser utilizada para remover, indiretamente, transições e estados. Toda transição é rotulada por uma regra (mesmo que essa regra tenha evento, condição e ação nulos). Quando a regra que rotula uma transição é removida, o processo de mesclagem encarrega-se de eliminar também a transição rotulada por essa regra. A remoção (no nível reflexivo) de todas as transições que ativam um estado implica em que esse estado jamais será ativado. Nessa situação, o estado também é removido durante a mesclagem dos diagramas. A eliminação de um estado implica na eliminação de todos os seus subestados e também de todas as regras e transições descritas em seu interior. A remoção de uma transição ou de um

estado pode provocar, implicitamente, a remoção de outras transições e estados, gerando um efeito cascata.

A Figura 7 mostra dois exemplos da utilização de *remove*. No exemplo (a), a utilização de *remove* em **D-REFLEXIVO** elimina a regra descrita no diagrama **D**. No exemplo (b), a utilização de *remove* na regra que rotula a transição de **A** para **B** elimina essa transição. Os estados **A** e **B** foram repetidos em **D-REFLEXIVO** para que a transição pudesse ser completamente caracterizada. A remoção da transição implica também na remoção do estado **B**.

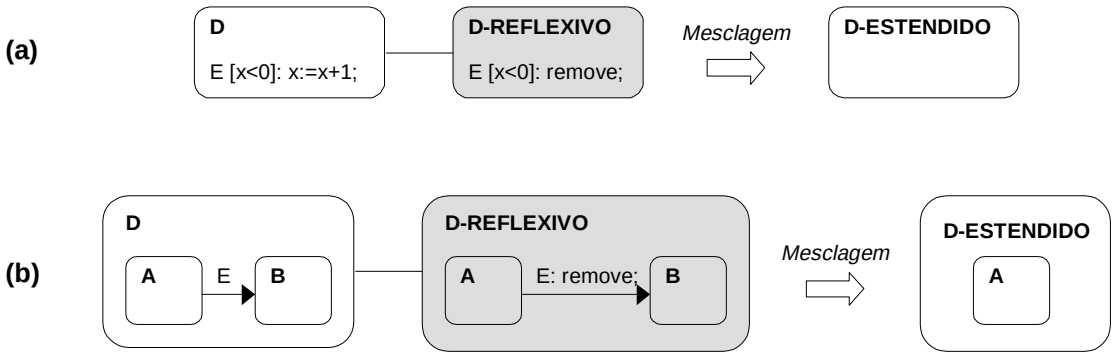

Figura 7 – Exemplos de remoção de regras, transições e estados

4.4 Exemplos

A seguir, serão descritos dois exemplos de utilização da arquitetura reflexiva proposta. Ambos os exemplos utilizam a especificação descrita na seção 3.1 como base para algumas extensões e adaptações utilizando o nível reflexivo. Nos dois exemplos, o diagrama **X** (mostrado em detalhes na Figura 3) será apenas referenciado. O controle de volume (∇) será utilizado para simbolizar que os detalhes desse diagrama estão descritos em outro local.

No exemplo da Figura 8, o diagrama reflexivo (**X-reflexivo**) é utilizado para acrescentar um novo domínio ao diagrama Xchart **X**. O novo domínio é formado por estatísticas sobre a execução das instâncias de **X.** O diagrama reflexivo adiciona regras para calcular o número de empates, vitórias e derrotas do jogador **X.** Estes valores são armazenados nas variáveis *vtie*, *vXwin* e *vOwin*. Essas variáveis são zeradas pela regra: entry: vtie:=0; vXwin:=0; vOwin:=0; sempre que o estado raiz é ativado. Embora essa regra esteja contida no estado **X-reflexivo** (estado raiz do diagrama reflexivo), ela refere-se ao estado raiz do diagrama Xchart. Essa é a única regra acrescentada pelo diagrama **X-reflexivo**. As demais regras já existem no diagrama **X** e são estendidas, através da adição de elementos de ação, no diagrama **X-reflexivo**.

As regras no interior dos estados ***Xwin*** e ***Owin*** são utilizadas para atualizar, respectivamente, as variáveis *vXwin* e *vOwin*. O primeiro elemento de ação dessas regras incrementa a respectiva variável. Como essas regras já existem no nível base, para que a ação original seja executada, essa ação é referenciada pelo elemento de ação *reuse* após o incremento da variável. Embora não tenha sido realizada nenhuma extensão nos estados ***GameOver*** e ***On***, esses estados foram repetidos no diagrama reflexivo porque são ancestrais dos estados que foram estendidos (***Xwin*** e ***Owin***). A regra no interior do estado ***Running***

incrementa a variável *vtie* quando acontece um empate. O estado ***WhatNext*** não precisou ser repetido no diagrama reflexivo porque não foram realizadas alterações nesse estado. Também não foi necessário repetir as regras e as transições contidas nos estados referenciados porque esses elementos não foram alterados no diagrama reflexivo.

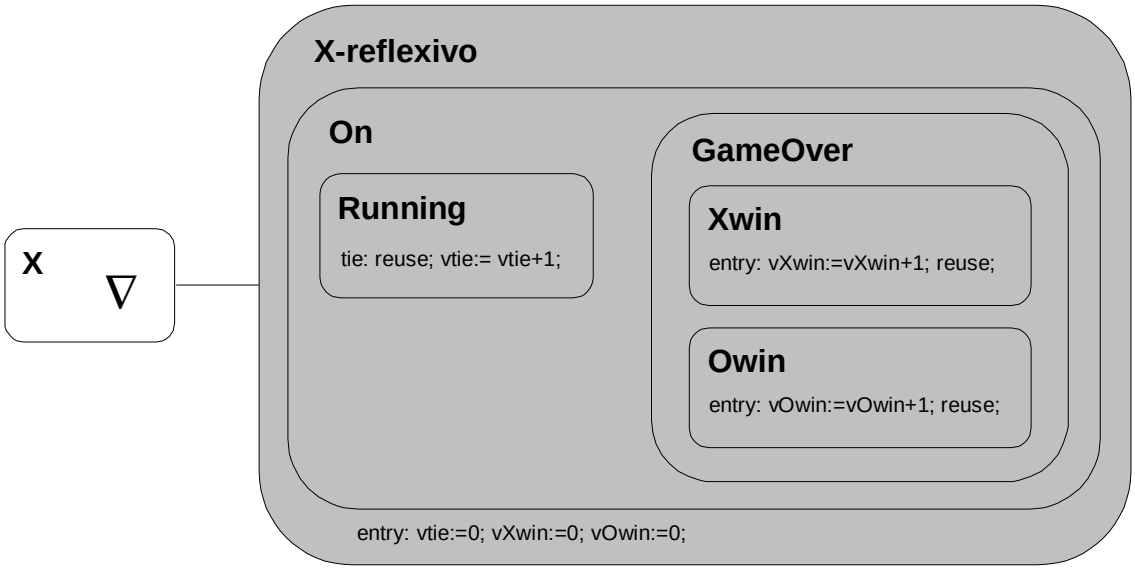

Figura 8 – Exemplo da separação de domínios utilizando o nível reflexivo

No exemplo da Figura 9, o diagrama reflexivo (**X-reflexivo**) é utilizado para alterar, temporariamente, a especificação descrita no diagrama Xchart **X**. A alteração consiste em substituir o estado ***GameOver*** por regras no interior do estado ***Running***. O objetivo é verificar, em tempo de execução, o impacto dessas alterações em relação à performance. Para remover o estado ***GameOver***, as transições do estado ***On*** para os subestados ***Xwin*** e ***Owin*** (que ativam o estado ***GameOver***) foram removidas. Para remover as transições, foi necessário remover as regras que as rotulam utilizando o elemento de ação *remove*. Ao remover o estado ***GameOver***, os estados ***Xwin*** e ***Owin*** (e as regras em seu interior) são automaticamente removidos. As novas regras para iniciar/finalizar as animações exibidas aos jogadores foram adicionadas no interior do estado ***Running***. No diagrama reflexivo, o evento ***Xwin*** passa a ser interceptado por uma regra no interior do estado ***Running***. A ação dessa regra dispara a animação ***ReceiveReward***. A animação é finalizada quando o estado ***Running*** é desativado. O tratamento da vitória do jogador O é equivalente.

4.5 Torre reflexiva

Os exemplos descritos nas seções anteriores associaram apenas um diagrama reflexivo a cada diagrama Xchart, embora essa não seja uma restrição. A relação entre diagramas Xchart e diagramas reflexivos é do tipo *m:n*. Um diagrama reflexivo pode estar associado a múltiplos diagramas Xchart, assim como um diagrama Xchart pode ter múltiplos diagramas reflexivos associados a si. A associação de um diagrama reflexivo a múltiplos diagramas Xchart (*m:1*) permite que o nível reflexivo seja utilizado não apenas para separar domínios, mas também

para agrupar elementos comuns a diversos diagramas Xchart. No caso de existirem múltiplos diagramas reflexivos associados ao mesmo diagrama Xchart (1:*n*), passa a existir uma torre reflexiva. Nessa torre, o primeiro diagrama reflexivo (r_1) é associado diretamente ao diagrama Xchart (*x*). Nesse caso, *x* é o nível base em relação a r_1. O segundo diagrama reflexivo (r_2) é associado ao diagrama reflexivo r_1, e assim sucessivamente. O *n*-ésimo diagrama reflexivo é associado ao diagrama r_{n-1}. Para o diagrama reflexivo r_n, o nível base corresponde à mesclagem do diagrama Xchart *x* com os *n*-1 diagramas reflexivos anteriores a r_n na torre reflexiva. A mesclagem dos diagramas é realizada de forma *bottom-up*. Inicialmente, o diagrama Xchart *x* é mesclado ao diagrama reflexivo r_1. O diagrama intermediário obtido dessa mesclagem (m_1) é então mesclado ao diagrama reflexivo r_2. Esse processo continua até que o diagrama intermediário m_{n-1} seja mesclado ao diagrama reflexivo r_n. Diagramas reflexivos cujos elementos são referenciados por outro diagrama reflexivo devem precedê-lo na torre reflexiva.

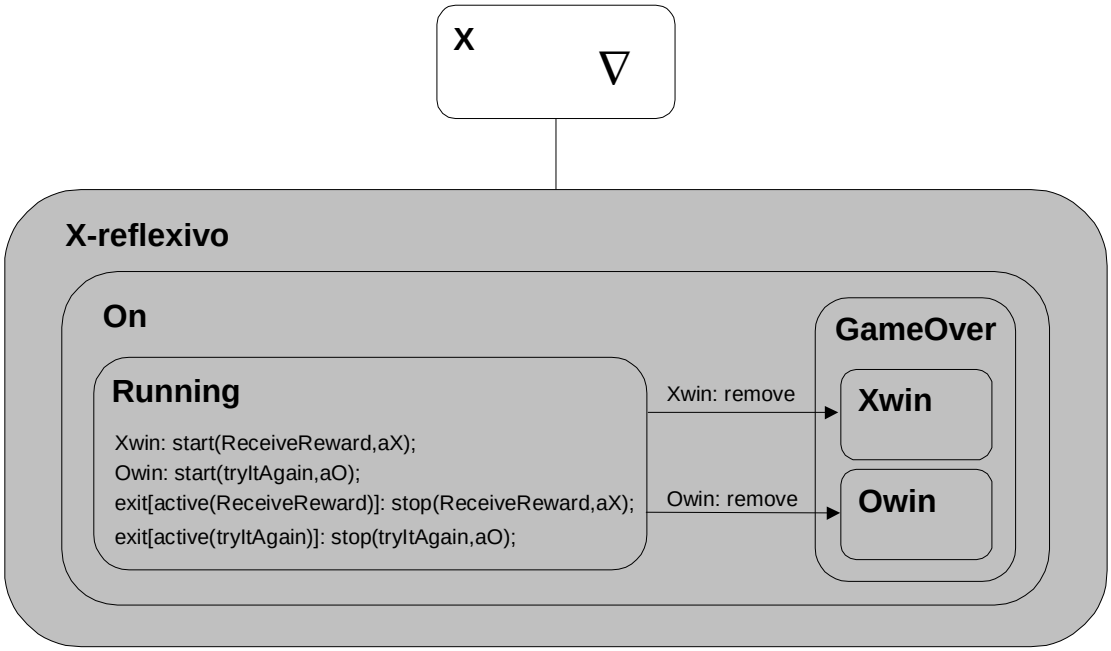

Figura 9 – Exemplo de intervenção no nível base utilizando o nível reflexivo

4.6 Implementação

A implementação da arquitetura proposta implica em alterações em dois componentes do ambiente Xchart: o editor de Xchart e o compilador TeXchart (para realizar a mesclagem dos diagramas). Conforme será discutido, algumas extensões imediatas de nossa arquitetura também irão requerer alterações no sistema de execução (SE) e na IPX. No caso dessas extensões, os processos de mesclagem e compilação dos diagramas (tal como será discutido a seguir) deverão ser adaptados.

A mesclagem dos diagramas Xchart e reflexivos pode ser realizada antes ou durante a compilação da especificação. A mesclagem da descrição textual dos diagramas (em TeXchart)

antes da compilação é a abordagem mais simples. Nesse caso, nem mesmo o compilador TeXchart precisaria ser alterado. No entanto, a abordagem ideal é realizar a mesclagem a medida que os diagramas são compilados. Algumas extensões imediatas que irão implicar em grandes mudanças na implementação da arquitetura, assim como a impossibilidade de realizar testes que comprovem a eficácia dessa arquitetura sem utilizar um editor de Xchart, nos levaram a escolher, a princípio, a opção mais simples para a implementação. A mesclagem das torres reflexivas acontece antes da compilação da especificação, utilizando a descrição textual dos diagramas (em TeXchart). Essa implementação é temporária e tem por objetivo, simplesmente, viabilizar a utilização de nossa arquitetura. O processo de mesclagem detecta apenas erros relativos à referência em diagramas reflexivos de elementos inexistentes nos respectivos diagramas Xchart (basicamente, regras que utilizam *reuse* ou *remove*). Qualquer elemento descrito do nível reflexivo e que não faça parte do nível base é considerado uma extensão ao nível base e não gera erro durante a mesclagem.

Os objetivos de nossa arquitetura são relativos à organização e à adaptação do gerenciador de diálogo descrito em Xchart. A eficácia da arquitetura deve ser testada pelos usuários de Xchart durante a codificação e/ou manutenção de uma especificação. Nesse aspecto, o editor de Xchart é indispensável. Na realidade, o editor de Xchart funciona como um complemento à nossa arquitetura. A organização da especificação em níveis possibilita oferecer ao usuário diferentes “visões” da especificação e avançados recursos de navegação. É possível oferecer ao usuário, por exemplo, a capacidade de visualizar cada domínio da especificação individualmente, habilitar/desabilitar domínios através de comandos simples, exibir/ocultar diagramas, domínios e torres reflexivas. Somente com a existência de um editor que faça uso efetivo das possibilidades oferecidas pela arquitetura reflexiva proposta é possível realizar testes que comprovem sua eficácia.

5. Conclusões

Nossa motivação para o projeto de uma arquitetura reflexiva para a linguagem Xchart se baseou em dois fatores: as vantagens associadas ao modelo reflexivo e a inexistência de arquiteturas reflexivas em linguagens do paradigma de eventos. A arquitetura proposta visa atingir dois objetivos básicos: (1) promover a separação de domínios através da organização da especificação em múltiplos níveis e (2) permitir a adaptação do comportamento e da estrutura da especificação, de forma transparente, utilizando o nível-reflexivo. A vantagem em utilizar o nível reflexivo é conceitual e diz respeito à organização da especificação. A separação de domínios em diferentes diagramas existe apenas no nível da especificação (durante o processo de compilação, o código dos diagramas reflexivos é mesclado ao código dos diagramas Xchart aos quais estão associados).

Um cuidado especial durante o projeto da arquitetura foi preservar o máximo possível as características da linguagem Xchart. Foram criadas apenas de três novas estruturas para essa linguagem: (1) diagrama reflexivo; (2) elemento de ação *reuse*; (3) elemento de ação *remove*. Essas três estruturas são simples e podem ser tratadas exclusivamente em tempo de compilação. O sistema de execução não precisa ser alterado porque o código executável é o mesmo da abordagem convencional (sem arquitetura reflexiva). Em tempo de execução, não há *overhead* associado ao mecanismo de reflexão adotado. Apesar de contrariar o modelo reflexivo, os elementos de ação *reuse* e *remove* foram adicionados explicitamente à linguagem Xchart para (1) facilitar a compreensão das operações associadas a esses elementos e (2) para facilitar uma implementação inicial (em tempo de execução). A medida que nossa arquitetura

for estendida e passar a ser tratada em tempo de execução, esses elementos de ação deixarão de existir explicitamente na linguagem e as operação associadas passarão a ser realizadas de forma transparente pelo sistema de execução (SE).

O impacto na implementação das extensões que serão discutidas na próxima seção, assim como a inexistência, atualmente, de um editor de Xchart que permita realizar testes que comprovem a eficácia de nossa arquitetura, nos levaram a escolher, a princípio, uma simplificada para a implementação da arquitetura proposta. Nessa abordagem, a mesclagem das torres reflexivas acontece antes da etapa de compilação, utilizando a descrição textual dos diagramas (em TeXchart). Essa implementação é temporária e tem por objetivo, simplesmente, viabilizar a utilização de nossa arquitetura. A mesclagem de uma torre reflexiva acontece iterativamente, de forma *bottom-up*.

6. Trabalhos futuros

Para testar a eficácia de nossa arquitetura é indispensável a utilização de um editor de Xchart que explore todo o seu potencial. Esse editor funciona como um complemento à nossa arquitetura. Também é indispensável a criação de uma metodologia para auxiliar o usuário na utilização da arquitetura reflexiva proposta. Essa metodologia deve guiar o usuário na identificação dos diversos domínios que compõem o gerenciador de diálogo a ser implementado e escolher a melhor organização para a torre reflexiva, determinando-se quais os domínio serão descritos em níveis distintos, qual a precedência entre esses níveis, etc.

Da forma como foi proposta, nossa arquitetura prevê a habilitação/desabilitação de níveis reflexivos (domínios) em tempo de compilação. Caso o usuário deseje habilitar/desabilitar algum domínio, toda a especificação precisa ser recompilada. Uma extensão imediata de nossa arquitetura é permitir a habilitação/desabilitação de níveis reflexivos em tempo de execução. A partir do instante que os diagramas reflexivos passam a ser conhecidos em tempo de execução, passa a existir também a possibilidade de sinalizar eventos diretamente para esses diagramas, aumentando o potencial de nossa arquitetura. Para oferecer suporte a essas operações, o sistema de execução de Xchart precisa ser adaptado.

Embora especificações em Xchart tenham a capacidade de invocar atividades, essas atividades interagem com o sistema de execução de Xchart de forma bastante restrita (através da IPX), limitando o seu potencial reflexivo. Caso as atividades pudessem ter acesso (pelo menos para consulta) a configuração de uma instância, elas poderiam ser utilizadas para fins reflexivos em relação ao gerenciador de diálogo, ou seja, poderiam existir atividades que pertencessem ao próprio subsistema reativo e cuja finalidade seria participar na execução desse subsistema. Esse tipo de acesso é possível através de uma metainterface (ortogonal à IPX).

7. Referências bibliográficas

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

[2] F.N. Lucena and H.K.E. Liesenberg. *Interface de Programação de Xchart*. Em preparação, http://www.dcc.unicamp.br/proj-xchart/texchart, 1998.

[3] F.N. Lucena, C.N. Júnior, T.H. Yunes, H.K.E. Liesenberg and L.E. Buzato. *Especificação da Linguagem TeXchart*. Em preparação, http://www.dcc.unicamp.br/projects/Xchart/texchart, 1998.

[4] F.N. Lucena,, C.N. Júnior and H.K.E. Liesenberg. *Biblioteca TeXchart*. Em preparação, http://www.dcc.unicamp.br/proj-xchart/texchart, 1998.
[5] F.N. Lucena. *Xchart: Um Modelo de Especificação e Implementação de Gerenciadores de Diálogo.* Tese de Doutorado, DCC/IMECC/UNICAMP, Campinas/SP. Dez 1997.
[6] G. Kiczales. *Beyond the Black Box: Open Implementation*. IEEE Software, 13(1): 8,10-11. 1996.
[7] P. Maes. *Concepts and Experiments in Computational Reflection*. OOPSLA 87, ACM Sigplan Notices, 22(12):147-155. Dec 1987.
[8] R.O. Stehling. *Projeto e Implementação de uma Arquitetura Reflexiva para a Linguagem Xchart*. Tese de Mestrado, IC-UNICAMP. Mar. 1999.