Programação Centrada em Instâncias em SMALLTALK

Paulo H. C. Lisboa phcl@di.ufpe.br
José Fernando Tepedino jftm@di.ufpe.br
Silvio Lemos Meira srlm@di.ufpe.br

Object Reflection Ltda.
fax: (081) 525-0225
Av. Belmino Correia 5039
Camaragibe PE 54771-000 Brasil

Universidade Federal de Pernambuco
Departamento de Informática
PO Box 7851
Recife PE 50732-970 Brasil

Abstract

This paper discusses the concept of instance-centered programming, describing a prototype that introduces many extensions to the SMALLTALK/V environment in order to make it more propitious to this programming style. This paper also presents some techniques that allow the prototype to incorporate instance-based systems features.

Resumo

Este artigo discute o conceito de programação centrada em instâncias, mostrando um protótipo com várias extensões introduzidas ao ambiente SMALLTALK/V para torná-lo mais propício a este estilo de programação. O artigo apresenta também algumas técnicas que permitem a incorporação de características dos sistemas baseados em instâncias no protótipo.

Motivação

Linguagens orientadas a objetos (LOOs) como SMALLTALK [7], EIFFEL [15] e C++ [20] tendem a supervalorizar a disponibilidade de recursos para a manipulação de classes, como por exemplo as ferramentas de acesso e definição dos métodos, variáveis e hierarquia, entre outros elementos que compõem a especificação de uma classe, relegando para o programador a tarefa de criar, guardar, localizar, mover, em suma, gerenciar as instâncias usadas durante o desenvolvimento de um sistema.

O ambiente PORTIA [6] apresenta o conceito de ambiente centrado em instâncias como algo que reforça as facilidades existentes para o tratamento dos objetos utilizados durante o desenvolvimento de sistemas. A apresentação de PORTIA deixa entretanto algumas dúvidas sobre a funcionalidade do ambiente, sua implementação e possíveis extensões para um sistema baseado em instâncias.

O nosso trabalho objetiva esclarecer as questões acima citadas, apresentando o conhecimento obtido na construção de um protótipo que adiciona e estende as ferramentas do ambiente SMALLTALK/V[1] para permitir uma programação centrada em instâncias [13].

1. Introdução

Um ambiente centrado em instâncias, segundo [6], é aquele que proporciona uma interação explícita com as instâncias, provendo facilidades para trabalhar diretamente com os objetos usados na criação, entendimento e depuração de aplicações. Os recursos providos por esse tipo de ambiente, entretanto, modificam tão significantemente o desenvolvimento de um sistema que caracterizam um novo estilo de programação, chamado programação centrada em instâncias. Nele, ao editar um método, pode-se, por exemplo, acessar e analisar instâncias-exemplo que representariam aqueles valores que o receptor, as variáveis e os parâmetros do método normalmente assumiriam em tempo de execução, permitindo que o programador execute, teste e entenda muito mais facilmente o método em questão, por trechos ou como um todo.

A programação centrada em instâncias tem um forte controle sobre as instâncias do sistema, fazendo com que a observação da reação das instâncias em determinadas condições seja um dos principais fatores na definição e modificação das classes e métodos do sistema. Este trabalho aborda uma série de questões relativas à programação centrada em instâncias e se encontra dividido da seguinte forma: a seção 2 esclarece a diferença entre os sistemas baseados em classes e os baseados em instâncias, para mostrar a ligação entre eles e a programação centrada em instâncias; a seção 3 apresenta alguns requisitos básicos para uma programação centrada em instâncias; a seção 4 descreve um protótipo com uma série de extensões introduzidas no ambiente SMALLTALK/V para auxiliar no entendimento deste trabalho; a seção 5 esclarece questões importantes na implementação do protótipo; a seção 6 mostra determinadas técnicas que podem ser usadas na continuação e ampliação deste trabalho, fazendo com que o ambiente adquira a flexibilidade existente nos sistemas baseados em instâncias e a seção 7 conclui, fazendo algumas considerações sobre a importância do trabalho.

2. Sistema Baseado em Instâncias *versus* Sistema Baseado em Classes

Os sistemas de programação orientados a objetos podem ser classificados, segundo [9; 17], em sistemas baseados em instâncias e baseados em classes.

Sistemas baseados em instâncias [1; 11], como SELF [21], não possuem construtores sintáticos para a definição de classes, sendo toda sua programação realizada através de instâncias (chamadas protótipos), que podem ter sua estrutura e comportamento compartilhados por outras instâncias [3], onde um mecanismo similar à herança de classes permite redefinir simplesmente a diferença local [19]. O desenvolvimento baseado em instâncias, nesses sistemas, é muito defendido para as fases iniciais (prototipação) de um projeto, quando não existe uma idéia bem esclarecida sobre as classes envolvidas, mas sim dos objetos necessários à elaboração dos primeiros sistemas experimentais.

Linguagens ou sistemas orientados a objetos, como C++ e SMALLTALK, são baseados em classes e muitas vezes sugerem um desenvolvimento mais robusto e confiável do que aquele baseado em instâncias. Isto ocorre porque classes são entendidas como agrupadores de idéias e conceitos normalmente mais maduros, sendo elas que especificam, nos sistemas baseados em classes, o algoritmo de criação, a estrutura e comportamento das instâncias.

---

[1]Este trabalho foi desenvolvido no ambiente do SMALLTALK/V for Windows da DIGITALK.

Sistemas baseados em classes são projetados para situações onde existem muitos objetos com o mesmo comportamento, sendo um tanto impróprios na criação de classes que seguramente terão uma única instância. Objetos (ou instâncias) protótipos, por outro lado, são exemplos concretos de entidades do mundo real, ao invés de descrições de formato e inicialização como nas classes. Exemplos são mais fáceis de serem concebidos, entendidos e reutilizados pois não exigem um conhecimento inicial profundo das suas características conceituais em relação aos outros indivíduos do sistema.

Como um ambiente centrado em instâncias pode ser usado por um sistema baseado em classes ou instâncias, o estilo de programação do ambiente é independente da base de desenvolvimento da linguagem. Apesar das facilidades disponíveis nos ambientes centrados em instâncias complementarem perfeitamente bem as deficiências dos ambientes tradicionais para desenvolvimento baseado em classes —os quais freqüentemente enfatizam o uso de ferramentas para o gerenciamento das classes, esquecendo-se das instâncias— estas facilidades também são úteis nos sistemas baseados em instâncias que, por serem mais recentes, não são apoiados por sofisticados ambientes de desenvolvimento.

A elaboração de um protótipo deste trabalho em SMALLTALK despertou tal atenção para a importância do uso simplificado das instâncias no desenvolvimento de um sistema, que já estudamos (seção 6) a possibilidade de expandir este protótipo para a incorporação de características existentes no desenvolvimento baseado em instâncias, como a criação de instâncias sem classe. Esperamos assim a obtenção de um ambiente centrado em instâncias bastante versátil que aproveite as virtudes dos sistemas baseados em classes e dos sistemas baseados em instâncias [10].

3. Requisitos para uma Programação Centrada em Instâncias

Esta seção descreve os principais pontos levados em consideração na caracterização de um ambiente centrado em instâncias. O desenvolvimento de um protótipo foi a principal fonte de idéias para o entendimento dos itens abaixo discutidos, apesar de PORTIA, já mencionado, também ter servido de base para algumas dessas questões.

3.1. O poder de capturar os objetos visualizados na interface de programação

Os ambientes de interface gráfica ganharam destaque ao mesmo tempo em que a programação orientada a objetos se difundia. Existem aqueles que acreditam que a programação orientada a objetos criou a demanda por ferramentas de recursos visuais que auxiliem na compreensão do dinamismo com que os objetos interagem nesse tipo de desenvolvimento. O fato é que tipos abstratos de dados são excelentes para encapsular a estrutura e o comportamento de um objeto gráfico —qualquer interface gráfica é composta de unidades básicas de objetos gráficos que ajudam na composição de outros mais complexos. Sendo assim, um ambiente para programação centrada em instâncias deve ser capaz de distinguir os diferentes objetos gráficos da sua interface e prover um protocolo uniforme de acesso às entidades por eles representadas. Estas entidades no caso são aqueles objetos (ou instâncias) usados no desenvolvimento de uma aplicação. Por exemplo, um objeto gráfico com uma lista (*ListPane*) de nomes de classes deveria ter comandos para capturar a classe selecionada no momento, da mesma forma que um *ListPane* com nomes de métodos deveria ter um protocolo semelhante para captura do método selecionado. Uma janela aberta no ambiente também é um objeto gráfico que poderia ser capturado para facilitar o acesso a qualquer objeto por ela referenciado.

3.1.1. O significado dinâmico dos nomes nas áreas de texto

As áreas de texto, principalmente o *TextPane*, são objetos gráficos que merecem destaque pela grande flexibilidade que adquirem dentro de um ambiente centrado em instâncias. Quase todas as

ferramentas de um ambiente de programação possuem áreas de texto onde palavras (*strings*) podem ser lidas ou escritas. Estas áreas de texto podem ser usadas como repositórios de instâncias que estão associadas a nomes locais. Mais de um nome local pode estar associado a uma mesma instância. Ao se inserir uma instância dentro de uma área de texto, um nome local é requerido para representar a nova instância. A captura de uma instância numa área de texto deve ocorrer pela seleção, dentro do texto, de uma palavra igual ao nome de uma instância previamente inserida nesta área de texto ou de uma expressão, seguido pelo protocolo padrão de captura. Dependendo da implementação, uma área de texto pode conter diversos contextos. Cada contexto significa um mapeamento independente de nomes a instâncias, ou seja, a seleção para outro contexto não alteraria o texto apresentado mas modificaria a relação entre os nomes e as instâncias associadas numa área de texto. A captura ou inserção de instâncias levaria sempre em consideração o contexto selecionado naquele momento para a área de texto em questão.

3.2. Possibilidade de mover os objetos capturados

A movimentação dos objetos pelas ferramentas da interface está intimamente ligada à captura dos mesmos. Um objeto normalmente é capturado a partir de uma ferramenta, para ser inserido e usado em outra, ou seja, ele é movido de uma ferramenta para outra na qual o programador precisa deste objeto. Além da definição do protocolo de inserção de objetos, que deve estar bem ajustado ao protocolo de captura, faz-se necessária a existência de um gerenciador dos objetos em movimento, que trate de questões como: Podemos movimentar vários objetos ao mesmo tempo? O primeiro a ser inserido é o último capturado? Podemos visualizar os objetos capturados? Eles têm algum nome que os diferencie? Este nome é local ou global?

A movimentação dos objetos pela interface pode ser implementada de diferentes formas. A nossa experiência mostrou que isto deve ser realizado de maneira bastante semelhante àquela encontrada nos modernos editores de texto (como *MS-Windows*, por exemplo). Da mesma forma que o *ClipBoard* no *Windows* é usado para armazenar textos, entre outras estruturas copiadas de diferentes aplicativos, o ambiente de programação pode conter um *ClipBoard* particular para as instâncias capturadas e usadas durante a programação. A próxima seção apresenta melhor a implementação de um *ClipBoard* de instâncias.

3.2.1. Identificação dos objetos

Dada a versatilidade para movimentar objetos, uma questão logo vem à tona: Como impedir a dificuldade para identificar uma mesma instância que pode estar sendo usada (com nomes distintos) por diferentes ferramentas? Não existe uma única solução para esta questão; a mais simples parece ser a adoção de identificadores únicos e globais para as instâncias movimentadas entre as ferramentas, identificação esta independente do nome local da instância.

3.3. Fácil localização das instâncias

A facilidade de se ter um número considerável de instâncias nas várias ferramentas do ambiente gera a dificuldade de localização destas instâncias pelo programador. A existência de recursos que simplifiquem esta localização é recomendável.

3.4. Facilidades para inspeção da estrutura e interface de um objeto

O estudo da estrutura e interface de um objeto resume o entendimento do seu funcionamento. As LOOs em geral oferecem ferramentas para este tipo de inspeção, que por vezes não são customizadas o suficiente para facilitar o seu uso. A estrutura de um objeto deve preferencialmente ser apresentada como

um grafo que se expande à medida que o usuário deseja um maior detalhamento. A apresentação dos métodos que compõem o comportamento de um objeto é equivalente à apresentação dos métodos na classe deste objeto, todavia é importante que não precisemos saber a classe do objeto para inspecionar tais métodos. A simplicidade com que as ferramentas de inspeção são ativadas a partir de um objeto do ambiente fazem relevante diferença no uso sistemático das mesmas.

3.5. Programação interativa auxiliada por instâncias-exemplo

Os recursos disponíveis para a programação centrada em instâncias possibilitam o uso de instâncias-exemplo durante o desenvolvimento. A flexibilidade conseguida nas áreas de texto é a principal responsável por tal proeza. Instâncias-exemplo podem ser inseridas, por exemplo, num contexto associado ao texto de um método para definirem valores aos diversos elementos (variáveis) do escopo de avaliação desse método, sendo associadas aos nomes dos parâmetros, variáveis locais, receptor, e assim por diante, para que o texto do método possa ser dinamicamente avaliado sem que todo o sistema esteja pronto ou em execução no momento. Esta propriedade chega a permitir que um simples trecho de um método seja avaliado sem alterações no seu código, independentemente do restante do método. Qualquer sistema pode ser muito mais agilmente testado ou compreendido se o programador puder observar o comportamento dos métodos ou trechos de código empregando certos contextos-exemplo.

4. Um Protótipo em Desenvolvimento

Aqui descrevemos um protótipo que estende o ambiente SMALLTALK Windows incluindo todos os requisitos citados na seção anterior. Como a movimentação das instâncias representadas pelos objetos gráficos requer a programação de cada janela do ambiente para suportar esta extensão, resolvemos centralizar nossos esforços nas áreas de texto, em particular na classe *TextPane*, que é usada em várias janelas do ambiente convencional. Assumimos que um bom entendimento do protótipo descrito a seguir exige um conhecimento prévio do ambiente SMALLTALK. A figura 1 mostra a janela *WorkArea* resultante da substituição da classe *TextPane* por uma subclasse melhorada desta última na janela *Workspace* existente em SMALLTALK/V. A descrição de cada componente da janela é feita a seguir.

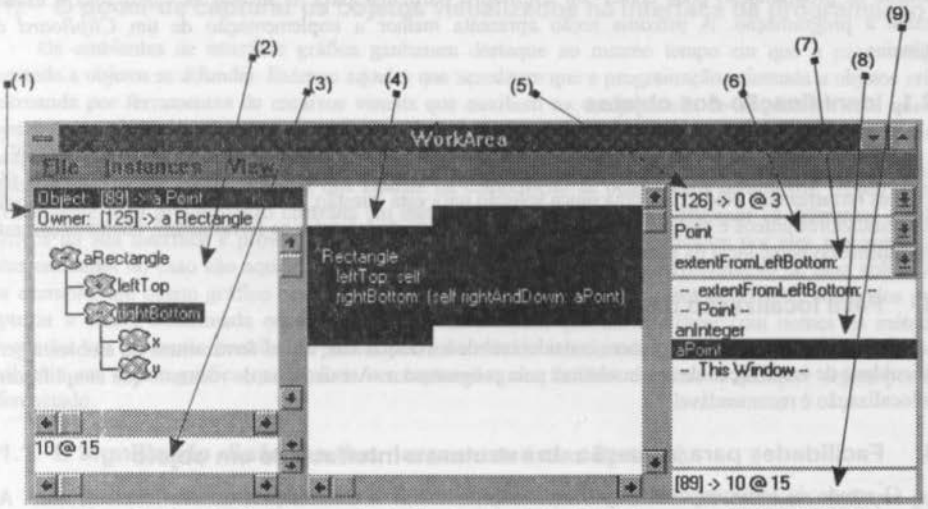

Figura 1. A janela *WorkArea* resultante de extensões da janela *Workspace* em SMALLTALK/V.

(1) Campos de identificação do objeto selecionado e seu proprietário segundo o *pane* gráfico de inspeção - O campo *Object:* identifica a instância selecionada no grafo, chamada *rightBottom* neste exemplo. O proprietário (*Owner*) é a instância *aRectangle* que contém *rightBottom* como uma das suas variáveis de instância (a outra se chama *leftTop*);

(2) *Pane* gráfico de inspeção da estrutura dos objetos - A apresentação visual da estrutura de um objeto simplifica bastante sua inspeção e entendimento. Os ítens 1, 2 e 3 apresentados nesta janela trabalham integrados para realizar, de forma extremamente simplificada, toda a funcionalidade conseguida com as janelas de inspeção do ambiente convencional de SMALLTALK/V. A estrutura de um objeto inspecionado pelo *pane* gráfico é expandida (ou aprofundada) por demanda, à medida que o usuário percebe que um nó da estrutura tem ramificações (o sinal '+' dentro de um nó indica que este objeto tem variáveis de instância que não estão sendo visualizadas). A opção *Drop as Root* no menu deste pane permite que um novo objeto, anteriormente capturado, seja analisado na posição raiz do grafo. Detalhes sobre a operacionalidade do *pane* gráfico e técnicas empregadas para indicar nós compartilhados fogem ao escopo deste trabalho e são tratados em [13];

(3) Área de texto simples com a descrição impressa (*printString*) do objeto selecionado no *pane* gráfico de inspeção - Além desta descrição impressa, essa área de texto permite que o programador avalie qualquer expressão em SMALLTALK, cujo resultado poderá ser atribuído à variável de instância selecionada, a qual é contida pelo objeto *Owner* mostrado no item 1;

(4) Área de texto para rascunho e avaliação de expressões - Além das funções que já eram realizadas pela área de texto do *Workspace* convencional, este *pane* permite associar instâncias às palavras aqui escritas, de forma que nomes de variáveis e parâmetros possam adotar valores *default*. Isto possibilita a avaliação parcial de expressões, assumindo-se os valores *default* nas variáveis ou parâmetros não inicializados no trecho de código avaliado. A forma mais simples de associar um nome a uma instância é realizar uma operação *drop* (inserção no contexto selecionado) quando este nome se encontra hachurado no texto. Entre as opções de menu deste *pane*, existem comandos para capturar, inspecionar graficamente ou revisar os métodos do objeto retornado de uma expressão avaliada;

(5) Lista que seleciona o objeto *self* - A palavra *self* aparece constantemente no corpo dos métodos em SMALLTALK para designar a instância receptora da mensagem em processamento. Uma vez que a área de texto para rascunho (item 4) pode estar sendo usada para avaliar trechos de código provenientes de um método, qualquer instância pode precisar assumir o papel de objeto *self*. Várias instâncias candidatas a *self* podem ser coletadas nesta lista que, após a escolha, mostra apenas aquela selecionada;

(6) Lista que seleciona o contexto classe - O mapeamento corrente de nomes a instâncias utilizado para avaliação de expressões na área de rascunho (item 4) depende dos contextos selecionados. Cada contexto possui o seu mapeamento. Uma grande versatilidade na nossa implementação é que o mapeamento corrente é o resultado da sobreposição dos mapeamentos dos contextos *Window*, classe e método, onde a prioridade cresce nesta seqüência, proporcionando ao contexto método a maior precedência. O contexto *Window* é local e relativo à janela em questão, sendo sua existência condicionada à da janela. Este contexto é usado para simplificar alguns testes rápidos que normalmente são executados em janelas de curta existência. Um contexto classe, assim como um contexto método (item 7), é global e tem sua existência independente da janela que o utiliza, o que possibilita seu acesso a partir de qualquer outra ferramenta do ambiente que utilize contextos. Qualquer nome pode ser usado para identificar um contexto classe, mesmo que não exista uma classe com o nome escolhido. A denominação **contexto classe** resultou do fato desta lista normalmente possuir diversos contextos com nomes iguais aos de classes existentes no sistema; isto ocorre porque a

janela *Class Hierarchy Browser* estendida neste trabalho possibilita a associação de contextos às classes dos métodos analisados e testados com os recursos aqui descritos; veremos isto melhor mais adiante;

(7) Lista que seleciona o contexto método - Assim como um contexto classe (item 6), um contexto método é global, independente da existência da janela que o utiliza e pode assumir qualquer nome, apesar de freqüentemente ter nomes iguais aos nomes de métodos pelo fato destes também poderem estar associados a contextos. Assim como os métodos de uma classe, a lista de contextos método depende do contexto classe selecionado, isto é, ela muda quando um novo contexto classe é selecionado;

(8) Lista de nomes das instâncias inseridas nos contextos selecionados - Este pane apresenta a lista de nomes do mapeamento resultante da composição do contexto *Window*, contexto classe e contexto método selecionados. A listagem dos nomes é dividida segundo o contexto (*Window*, classe ou método) em que se encontram;

(9) Campo de identificação da instância selecionada na lista de nomes - Para facilitar a identificação das instâncias na lista de nomes (item 8), este campo é atualizado toda vez que um novo nome é selecionado, mostrando a identificação da instância a ele associada.

Para facilitar a diferenciação entre as diversas instâncias usadas com a ajuda de nossas extensões, adotamos a seguinte notação nos campos de identificação (itens 1 e 9):

[*Número-de-Identificação*] -> *Descrição-da-Instância*, onde

Número-de-Identificação é um inteiro positivo usado como identificador global, o qual é diferente para cada instância (identificada) do sistema. As instâncias identificadas do sistema são normalmente aquelas que já foram movimentadas pelas janelas do ambiente.

Descrição-da-Instância corresponde à descrição impressa (*printString*) da instância ou, dependendo da janela, a um *string* com a informação da classe dessa instância.

A visualização da janela *WorkArea* mostra que ela resulta da junção de uma área de rascunho convencional com um conjunto de *panes* necessários a uma inspeção gráfica, além dos já citados contextos. Isto decorreu da constatação, neste trabalho, de que as janelas de rascunho convencionais freqüentemente precisavam ativar janelas de inspeção sobre os objetos em uso e vice-versa. A utilização conjunta desses recursos, em substituição à classe *TextPane*, também foi feita em outras janelas convencionais de SMALLTALK/V, sendo a janela *Class Hierarchy Browser* (figura 2) um exemplo disso.

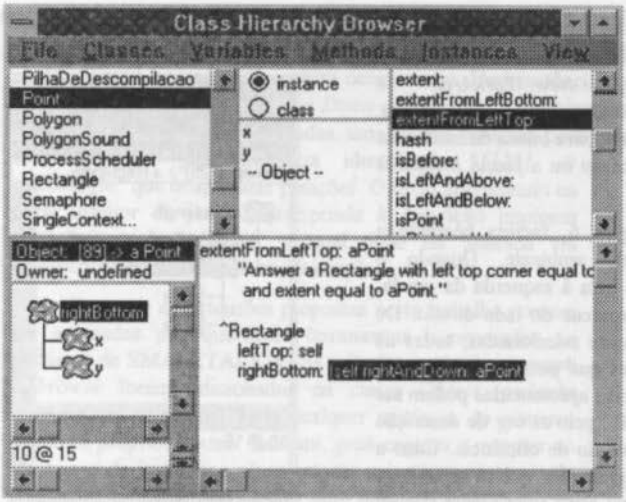

Figura 2. Uma janela *Class Hierarchy Browser* com extensões.

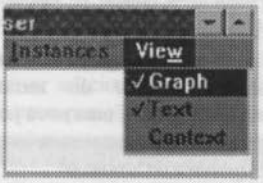

Figura 3. O menu *View* da janela mostrada na figura 2.

A visualização dos *panes* de contexto na figura 1, não encontrados na figura 2, é conseqüência das opções selecionadas pelo usuário no menu *View* existente em ambas as janelas. Este menu permite a escolha dinâmica dentre os panes 1) inspeção gráfica, 2) texto e/ou 3) contexto, aqueles que o usuário deseja visualizar no momento. A figura 3 mostra o estado do menu *View* da janela *Class Hierarchy Browser* da figura 2.

Quando uma classe ou método é selecionado na janela *Class Hierarchy Browser* estendida, o contexto classe ou método correspondente, caso existam, são automaticamente escolhidos, permitindo o entendimento, teste e modificação do método a partir da última alteração realizada pelo programador nestes contextos. Um contexto classe é normalmente usado para associar instâncias aos nomes de variáveis e parâmetros encontrados freqüentemente em vários metodos de uma mesma classe, enquanto que um contexto método geralmente associa instâncias aos nomes dos parâmetros e variáveis mais específicos a um dado método. O mapeamento de nomes a instâncias de um contexto classe é herdado por todos os seus contextos método, o mesmo ocorrendo com um contexto *Window* em relação aos contextos classe.

A figura 4 apresenta o menu *Instances*, presente em várias janelas incluídas ou estendidas por este trabalho. As opções *Graph_It* e *Browse_It* abrem uma janela de inspeção gráfica ou de edição de métodos, respectivamente, sobre o resultado de uma expressão avaliada na área de texto. Caso nenhuma expressão seja marcada para avaliação, estes comandos operam sobre o objeto *self* corrente, o qual tem *nil* como valor *default*. A janela de edição de métodos é um *Class Browser* do ambiente convencional que mostra os métodos existentes na classe do objeto retornado pela avaliação. A janela de inspeção gráfica é na verdade uma janela *WorkArea* que tem o menu *View*

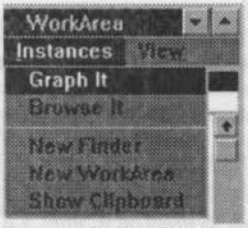

Figura 4. O menu *Instances*.

selecionado apenas na opção *graph* (figura 5). Além de inspecionar a estrutura de um objeto, o grafo tem uma opção no seu menu que permite capturar qualquer instância representada por um nó visualizado.

As opções *New_Finder*, *New_WorkArea* e *Show_Clipboard* do menu *Instances* (figura 4) abrem, respectivamente, uma nova janela *Finder* para busca de instâncias pelo ambiente, uma nova janela *WorkArea* ou a janela *Instances Clipboard* do ambiente.

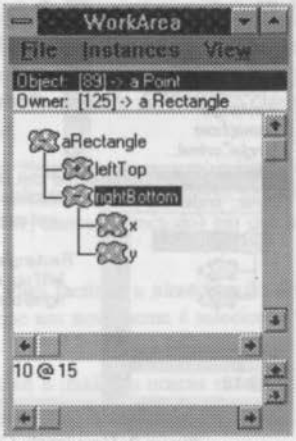

Figura 5. Inspeção gráfica usando uma simples janela *WorkArea*.

A janela *Finder*, vista na figura 6, facilita bastante a procura de qualquer instância pelo ambiente. Quando o programador seleciona uma classe na lista à esquerda da janela, uma lista de instâncias desta classe aparece do lado direito. De acordo com algumas opções previamente selecionadas, todas as instâncias da classe ou apenas aquelas que possuem número de identificação serão listadas. As instâncias apresentadas podem ser ordenadas pelo número de identificação, pelo *string* de descrição da instância ou aleatoriamente por questão de eficiência. Caso a classe não tenha instâncias ou o programador queira uma nova instância-exemplo, uma opção de menu ativa uma janela (figura 7) com o método *sample*, o qual cria este tipo de instância. O método de classe *sample* pode ser diferente para cada classe do sistema e foi adotado, neste trabalho, como uma maneira padronizada de se conseguir novas instâncias de uma classe. A definição de um método padrão para produzir amostras de instâncias de uma classe é muito importante porque o programador pode não estar suficientemente familiarizado com a classe para saber como criar e inicializar uma instância-exemplo necessária nas verificações e testes durante o desenvolvimento.

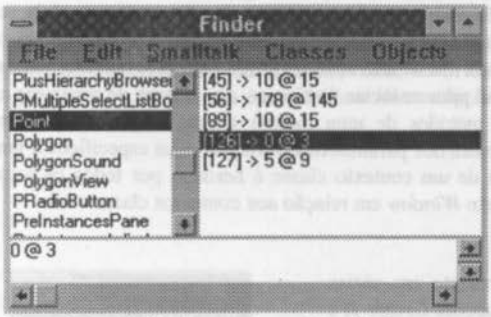

Figura 6. Janela *Finder* para busca de instâncias.

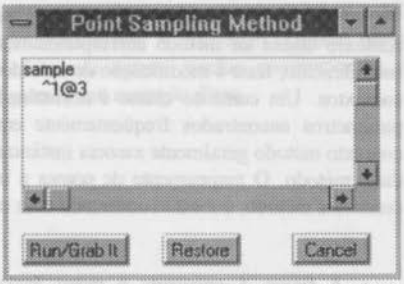

Figura 7. Janela de amostragem de instâncias a partir do método de classe *sample*.

Quase todos os objetos gráficos estendidos por este trabalho apresentam as opções *Grab* e *Drop* nos seus respectivos menus para implementar a captura e inserção das instâncias por eles representadas. Até o *Control Menu* (menu de controle das janelas com opções *move*, *size*, etc.) foi estendido com a opção *Grab_Window* para permitir a captura de qualquer objeto janela visualizado pelo programador. A janela *Instances Clipboard* (figura 8) mostra as últimas instâncias capturadas no ambiente. Este *clipboard* é responsável pelo gerenciamento das instâncias movimentadas pelas diversas janelas. Ele é único e global ao

ambiente, além de ser totalmente independente daquele utilizado pelo *MS-Windows*. A instância do topo da pilha mostrada no *clipboard* foi a última a ser capturada e será aquela utilizada numa inserção, caso nenhuma nova operação de captura ocorra antes. Uma instância pode ter sua posição deslocada pela pilha via o uso dos botões *Top*, *Up* ou *Down* que operam sobre a instância que estiver selecionada. Dependendo da seqüência de capturas realizadas, uma mesma instância pode estar em diferentes posições da pilha, como está mostrado pela instância identificada *"[125] -> aRectangle"* que ocupa duas posições. O texto apresentado na parte inferior da janela corresponde à descrição impressa (*printString*) da instância, a qual pode ser alterada via modificação e avaliação dessa descrição.

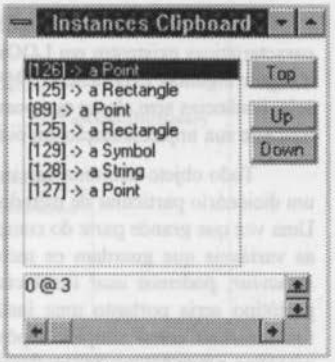

Figura 8. *Clipboard* de instâncias.

Para que as extensões propostas neste trabalho possam ser acessadas por quaisquer ferramentas incorporadas ao ambiente de SMALLTALK/V, os métodos *grab*, *drop*, *graph* e *browse* foram adicionados na classe *Object*, podendo portanto ser utilizados para qualquer instância do sistema. Como os próprios nomes indicam, *grab* captura o receptor no *clipboard* de instâncias, *drop* retorna a instância do topo do *Clipboard* de instâncias, *graph* abre uma janela de inspeção com o receptor sendo a raiz do grafo e *browse* abre uma janela *Class Browser* com os métodos do receptor.

5. Considerações sobre a Implementação

A flexibilidade existente em SMALLTALK para estender seu próprio ambiente facilitou bastante este trabalho. Tal flexibilidade decorre do fato da linguagem ser orientada a objetos e possuir uma arquitetura propícia ao uso de técnicas refletivas [12; 5; 14]. São essas técnicas que permitem ao ambiente inferir sobre seu próprio funcionamento, modificando seu processo interpretador em tempo de execução, caso seja necessário.

Os principais aspectos na implementação foram as facilidades de manipulação das estruturas das instâncias, realizado pela inspeção gráfica, e a avaliação de trechos de código que usam os mapeamentos de nomes a instâncias existentes nos contextos. A manipulação da estrutura das instâncias foi baseada no inspetor convencional de SMALLTALK, onde métodos do protocolo de determinadas classes, como *Object*, *Behavior* ou *Class*, por exemplo, são usados para refletir sobre a computação do sistema. Alguns dos principais dentre esses métodos são *instVarAt:* e *instVarAt:put:* em *Object*, usados para retornar ou modificar o objeto apontado por uma dada variável de instância do receptor, *instVarNames* em *Behavior*, que retorna os nomes das variáveis de instância do receptor, e *name* em *Class*, que retorna o nome da classe do receptor.

O entendimento da avaliação de trechos de código com contextos está ligado à compreensão do mecanismo de captura e inserção de instâncias nesses contextos. Esse funcionamento se beneficia do fato de SMALLTALK ser uma linguagem interpretada que permite a avaliação dinâmica de expressões. Em termos operacionais, quando um trecho de código é avaliado, um método contendo o *string* deste código, chamado *DoIt*, é compilado e inserido temporariamente na interface da classe do objeto *self* em questão (sendo *nil* o objeto *default*), de forma que o resultado deste trecho de código é o retorno da mensagem *DoIt* enviada a *self*. Essa compilação é um pouco diferente da normal para permitir que, na ausência de um comando explícito, o método retorne o resultado da sua última expressão. As opções *Do_It* e *Show_It* no menu de um *TextPane* fazem uso desse mecanismo no seu processamento. Pode-se conseqüentemente ampliar esse mecanismo para que o método *DoIt* aceite parâmetros que seriam, neste caso, os nomes existentes no

contexto do trecho avaliado. A ativação do *DoIt* seria então feita mediante a passagem das instâncias inseridas no contexto como parâmetros desse método. Esta técnica parece ser uma das mais simples para a avaliação de trechos de código com contextos associados.

6. Extensões para um Sistema Baseado em Instâncias

Como discutido em seções anteriores, a evolução deste trabalho para a incorporação de características existentes em LOOs baseadas em instâncias seria um passo bastante natural. Apresentamos a seguir algumas experiências [2] feitas em OBJECTWORKS[2] que permitem criar objetos protótipos, ou seja, instâncias sem classe que possuem seus próprios métodos. O conhecimento dessa técnica pode facilitar bastante sua implementação em outros compiladores, como SMALLTALK/V.

Todo objeto tem uma variável de instância oculta que aponta para a sua classe. Cada classe contém um dicionário particular de métodos, o qual é usado quando uma mensagem é enviada a uma instância sua. Uma vez que grande parte do comportamento e estrutura de uma classe, definidos na sua metaclasse, como as variáveis que guardam os métodos, subclasses e superclasses, por exemplo, são heranças da classe *Behavior*, podemos usar *instâncias* desta última no lugar das classes de objetos protótipos. Um objeto protótipo seria portanto uma instância de alguma instância de *Behavior*. Tais instâncias de *Behavior* funcionariam como simples depósitos de métodos para objetos protótipos, permitindo que cada objeto protótipo possa ter um comportamento individual, sem a criação de novas classes. O uso das instâncias de *Behavior* em substituição às classes convencionais para os objetos protótipos têm várias razões:

i. Cada classe tem um nome global no sistema, o que gera problemas de nomeação, caso as classes sejam indiscriminadamente criadas para cada objeto protótipo do sistema;

ii. Como uma instância de *Behavior* é normalmente referenciada apenas pelo seu objeto protótipo (sua instância), ela também será liberada da memória quando deixarem de existir referências para o seu objeto protótipo;

iii. Classes são estruturas desnecessariamente complexas para serem usadas apenas como depósitos de métodos de objetos protótipos.

[2]SMALLTALK-80 da ParcPlace versão 4.1.

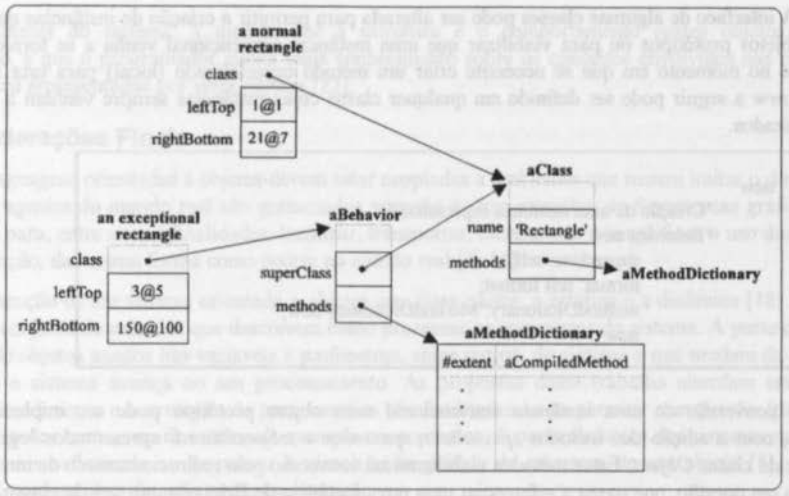

Figura 9. Visualização espacial da diferença entre instâncias normais e objetos protótipos.

Um objeto protótipo poderia ser usado, por exemplo, para criar uma instância excepcional de um retângulo (baseada na classe *Rectangle*) que, por ter características próprias, precisa redefinir uma versão otimizada de alguns métodos (ver figura 9). Este objeto retângulo representaria a localização de uma janela especial no ambiente que, apesar de mudar constantemente de posição, usa por toda sua existência a extensão determinada no momento da sua abertura. A avaliação do código a seguir cria o objeto proposto, redefine e otimiza o método *extent* apenas para este objeto, e mostra finalmente, numa janela de inspeção, o tal objeto.

```
| aBehavior prototype initPosition extent |
"Criação de uma nova instância de Behavior, baseada em Rectangle"
aBehavior := Behavior new
            superclass: Rectangle;
            methodDictionary: MethodDictionary new;
            setInstanceFormat: Rectangle format.

"Criação do objeto protótipo"
initPosition := 1@3.
extent := 10@15.
prototype := aBehavior new
            origin: initPosition;
            corner: initPosition + extent.

"Redefinição do método extent otimizado"
aBehavior compile: 'extent    ^', extent printString  notifying: nil

"Apresentação do objeto para testes"
prototype inspect.
```

A interface de algumas classes pode ser alterada para permitir a criação de instâncias que já nascem como objetos protótipos ou para viabilizar que uma instância convencional venha a se tornar um objeto protótipo no momento em que se necessite criar um método especializado (local) para esta instância. O método *new* a seguir pode ser definido em qualquer classe cujas instâncias sempre venham a ter métodos especializados.

```
new
        "Criação de uma instância especializada"
        ^Behavior new
                superclass: self;
                format: self format;
                methodDictionary: MethodDictionary new;
                new
```

A conversão de uma instância convencional num objeto protótipo pode ser implementada, por exemplo, com a adição dos métodos *specialize:*, *specialize* e *isSpecialized*, apresentados logo abaixo, na interface da classe *Object*. Estes métodos viabilizam tal conversão pelo redirecionamento de um ponteiro, na instância em questão, que passa a referenciar uma nova instância de Behavior, em vez da classe original.

```
specialize: aString
        "Compila aString como um método especializado para esta instância apenas"
        self specialize.
        self class compile: aString notifying: nil
```

```
specialize
        self isSpecialized ifTrue:[ ^self ].
        class := Behavior new
                superclass: self class;
                format: self class format;
                methodDictionary: MethodDictionary new.
        self changeClassToThatOf: class basicNew
```

```
isSpecialized
        ^self class shouldBeRegistered not
```

Apesar de uma instância da classe *Behavior* não poder dinamicamente estender a estrutura de seu objeto protótipo, assim como o faz com os métodos, várias técnicas podem ser usadas para contornar esta situação. Uma idéia simples seria a criação de uma subclasse de *Behavior*, chamada de *ProtoBehavior*, a qual substituiria a classe *Behavior* na criação de objetos protótipos. *ProtoBehavior* introduziria a variável de instância *protoState* que seria usada para conter um dicionário com os novos nomes e valores usados por um objeto protótipo na extensão da sua estrutura.

O conhecimento dessas técnicas permite a produção de ferramentas que facilitem a criação e edição da estrutura e comportamento de objetos protótipos. Estes podem vir a ser usados como bases na definição

de novas classes do sistema, à medida que a estrutura e o comportamento destes objetos vão se estabilizando, e que o programador ganha mais conhecimento sobre os conceitos envolvidos nas entidades do mundo real representadas por estes objetos [22].

7. Considerações Finais

As linguagens orientadas a objetos devem estar acopladas a ambientes que tentem imitar o dinamismo com que os agentes do mundo real são gerenciados, através do uso intensivo de ferramentas gráficas com dispositivos para, entre outras finalidades, localizar, transportar, reestruturar e simplificar o uso dos objetos da programação, da mesma forma como ocorre no mundo real [4; 16].

A execução de um sistema orientado a objetos tem duas partes: a estática e a dinâmica [18]. A parte estática é o conjunto de métodos que descrevem como processar as mensagens do sistema. A parte dinâmica é conjunto de objetos usados nas variáveis e parâmetros, entre outros, do sistema e que mudam de estado à medida que o sistema avança no seu processamento. As propostas desse trabalho abordam essas duas partes. O entendimento e a evolução da parte estática são facilitados ao se permitir a avaliação (parcial ou total) dos métodos com o uso de instâncias-exemplo em contextos. A parte dinâmica dos sistemas recebe um tratamento mais sofisticado ao se prover a visualização gráfica da estrutura de um objeto [8], além da simplificação na sua identificação, acesso e alteração.

O enriquecimento do ambiente SMALLTALK com características existentes na programação centrada em instâncias permite um desenvolvimento mais interativo de sistemas, onde os riscos de falhas numa versão concluída diminuem pelo fato das entidades do sistema estarem constantemente sendo inspecionadas, avaliadas e testadas. Isto torna possível que os componentes (código e variáveis) de um método sejam testados ao mesmo tempo em que ele vai sendo desenvolvido, e não somente na conclusão de sua classe.

SMALLTALK apresenta um ambiente propício à exploração de tais técnicas de programação, sendo a sua arquitetura refletiva o principal responsável por isto. Essa constatação ditou sua escolha na elaboração deste trabalho.

Uma comparação do conceito e do ambiente para programação centrada em instâncias aqui discutidos pode ser feita com o PORTIA, mencionado anteriormente. Acreditamos ter expandido as idéias em torno desse tema, ao mesmo tempo que contribuído para elucidar dúvidas sobre o funcionamento e a implementação desse tipo de sistema.

A continuação deste trabalho prevê uma ampliação das idéias propostas para que todas as ferramentas do ambiente padrão de SMALLTALK passem a contar com as extensões aqui apresentadas, de forma tal que o transporte dessas extensões para as diversas implementações de SMALLTALK seja algo de fácil realização.

Referências

[1] Jay Almarode: "Rule-Based Delegation for Prototypes", ACM - OOPSLA'89 Proceedings, October, 1989.

[2] Kent Beck: "Instance Specific Behavior: How and Why", The Smalltalk Report, SIGS publications, Volume 2 Number 6, March-April, 1993.

[3] Craig Chambers, David Ungar, Bay-Wei Chang & Urs Hölzle: "Parents are Shared Parts of Objects: Inheritance and Encapsulation in SELF", Lisp and Symbolic

Computation: An International Journal, 4, 3, 1991.

[4] Jay Fenton & Kent Beck: "Playground: An Object Oriented Simulation System with Agent Rules for Children of All Ages", ACM - OOPSLA'89 Proceedings, October, 1989.

[5] Brian Foote & Ralph E. Johnson: "Reflective Facilities in Smalltalk-80", ACM - OOPSLA'89 Proceedings, October, 1989.

[6] Eric Gold & Mary Beth Rosson: "Portia: An Instance-Centered Environment for Smalltalk", ACM - OOPSLA'91 Proceedings, 1991.

[7] Adele Goldberg & David Robson: "SMALLTALK-80: The Language", Addison-Wesley, 1989.

[8] Dan Ingalls, Scott Wallace, Yu-Ying Chow, Frank Ludolph & Ken Doyle: "Fabrik: A Visual Programming Environment", ACM - OOPSLA'88 Proceedings, September, 1988.

[9] Wilf LaLonde & John Pugh: "Instance-Based Programming with PARTS", Journal of Object-Oriented Programming, SIGS publications, March-April, 1993.

[10] Wilf R. LaLonde, Dave A. Thomas & John R. Pugh: "An Exemplar Based Smalltalk", ACM - OOPSLA'86 Proceedings, September, 1986.

[11] Henry Lieberman: "Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems", ACM - OOPSLA'86 Proceedings, September, 1986.

[12] Paulo H. C. Lisboa, José Fernando Tepedino & Silvio Lemos Meira: "Reflexão Computacional em SMALLTALK", SEMISH'93, Florianópolis, SC, setembro,1993.

[13] Paulo H. C. Lisboa: "Reflexão em SMALLTALK: Ambientes e Extensões para Sistemas Reais", Tese de Mestrado em elaboração, DI-UFPE, 1993.

[14] Pattie Maes: "Computational Reflection", Technical Report, Artificial Inteligence Laboratory, Vrije Universiteit Brussel, Belgica, Bruxelas, 1987.

[15] Bertrand Meyer: "Object-Oriented Software Construction", Prentice Hall International, Series in Computer Science, 1988.

[16] Michael F. Kleyn & Paul C. Gingrisch: "GraphTrace - Understanding Object-Oriented Systems Using Concurrently Animated Views", ACM - OOPSLA'88 Proceedings, September, 1988.

[17] Michael L. Nelson: "An Object-Oriented Tower of Babel", ACM - OOPS Messenger, volume 2, number 3, July, 1991.

[18] Trygve Reenskaug & Anne Lise Skaar: "An Environment for Literate Smalltalk Programming", ACM - OOPSLA'89 Proceedings, October, 1989.

[19] Lynn Andrea Stein: "Delegation is Inheritance" ACM - OOPSLA'87 Proceedings, October, 1987.

[20] B. Stroustrup: "The C++ Programming Language", Addison-Wesley, 1986.

[21] David Ungar & Randall B. Smith: "Self: The Power of Simplicity", ACM - OOPSLA'87 Proceedings, October, 1987.

[22] David Ungar, Craig Chambers, Bay-Wei Chang & Urs Hölzle:"Organizing Programs Without Classes", Lisp and Symbolic Computation: An International Journal, 4, 3, 1991.