DOBuilder

Ambiente de Programação Visual com Objetos Distribuídos em Java

Juliano Malacarne – malacarn@inf.ufrgs.br
Cláudio Geyer – geyer@inf.ufrgs.br

Instituto de Informática – UFRGS
Av. Bento Gonçalves, 9500 – Campus do Vale - Bloco IV
Bairro Agronomia – Porto Alegre – Brasil
CEP 91501-970 - Caixa Postal 15064

Resumo

Este trabalho apresenta o modelo e a implementação de uma ferramenta visual para programação com objetos distribuídos em Java. Em vista da baixa quantidade e qualidade de ferramentas de programação para esses ambientes, o objetivo desta ferramenta é aplicar a programação visual e as principais tecnologias de engenharia de *software* (componentes, reutilização, encapsulamento, orientação a objeto) no desenvolvimento de aplicações com objetos distribuídos e assim facilitar a tarefa de programação desses tipos de aplicações. Com base nisso, o programador desenvolve sua aplicação combinando visualmente componentes de programação distribuída sem ter que se preocupar com os níveis mais baixos de implementação da distribuição. Na implementação, a distribuição é feita pelo ambiente de execução Voyager e a programação das aplicações é feita em Java. Usuários que utilizaram esta ferramenta concluíram que ela facilita a programação distribuída.

Palavras-chave: Objetos Distribuídos, Ambientes de Programação, Multicomputadores e Processamento Distribuído, Java, Programação Visual

Abstract

This works presents the model and the implementation of a visual tool for distributed objects programming in Java. Due to the lack of good programming environments for this kind of applications, the goal of this tool is to apply the visual programming technique and the main software engineering technologies (components, reuse, encapsulation, object orientation) to the distributed object application development, making simpler the programming task in this area. Based upon this, the programmer develops his/her application combining components for distributed programming visually without worrying about the lowest levels of distribution implementation. In the implementation, the distribution task is performed by the Voyager execution environment and the application programming is done in Java. Users who have used this tool have concluded that it makes the distributed programming easier.

Keywords: Distributed Objects, Programming Environments, Multicomputers and Distributed Processing, Java, Visual Programming

1 Introdução

As tecnologias de redes de computadores e orientação a objeto têm se tornado cada vez mais presentes em todas as áreas da informática. A maior parte das novas aplicações e ferramentas tenta se adaptar de alguma maneira a essas tecnologias, provendo suporte aos principais conceitos de orientação a objeto (encapsulamento, reuso, herança) e oferecendo meios para que executem num ambiente de rede. Não levou muito tempo para que essas duas tecnologias se integrassem, resultando no que hoje se chama de objeto distribuído [BRI98], que é um objeto tradicional com suas características tradicionais mas que existe em qualquer lugar da rede e se comporta de forma independente dessa localização.

Entretanto, por ser ainda relativamente recente, a tecnologia de objetos distribuídos sofre pela quantidade e qualidade insatisfatórias de ferramentas de desenvolvimento. A maioria dos programadores continua utilizando ferramentas mais antigas, como editores de textos e compiladores de linha de comando para essa tarefa. Tal problema se verifica também na programação paralela, que, entre outras soluções, faz uso da programação visual, onde são utilizadas representações gráficas para especificar as aplicações concorrentes. A razão de se utilizar esse tipo de abordagem na programação é que os gráficos são muito mais adequados à representação de estruturas bidimensionais (os programas concorrentes) do que as linguagens de programação textual, oferecendo ao programador um meio mais natural de representar e visualizar tais aplicações. As ferramentas HeNCE [BEG92], CODE [NEW92] e VisualProg [SCH96][MAL97] são exemplos de ferramentas desse tipo.

Essas novas ferramentas melhoram a especificação das aplicações com a programação visual, mas deixam de lados questões atualmente muito melhor tratadas em ferramentas de programação para aplicações *desktop*. Estas questões são principalmente a aplicação de conceitos de engenharia de *software*, como orientação a objeto e componentes de *software*, que são bastante empregados nas ferramentas de desenvolvimento centralizado como Delphi [BOR01a] e JBuilder [BOR01b], mas praticamente ignorados em ferramentas de programação visual paralela e distribuída.

Tendo em vista esses problemas, a ferramenta apresentada neste trabalho visa a atacar as principais deficiências das ferramentas atuais utilizadas na programação distribuída, que são a dificuldade de se especificar programas distribuídos por meio de linguagens textuais e a falta de suporte mais elaborado à engenharia de *software*. Procura-se com isso oferecer ao usuário um modelo de programação flexível que seja ao mesmo tempo poderoso e simples, permitindo que ele desenvolva aplicações com objetos distribuídos de maneira mais fácil do que com as ferramentas textuais. Isto tudo é buscado se utilizando como base a programação visual e a estruturação das aplicações segundo componentes de *software*.

Este artigo começará com um estudo sobre as principais tecnologias utilizadas neste trabalho: programação visual, programação com objetos distribuídos e componentes de *software*. A seguir, serão apresentados a estrutura e o modelo de programação da ferramenta, seguido por sua implementação. Nesta seção, está presente um exemplo a fim de facilitar a compreensão de como uma aplicação completa é representada nesta ferramenta. Por fim, há uma comparação com outros trabalhos e as conclusões, onde são colocadas as impressões tiradas de usuários que tiveram contato com essa ferramenta e onde também são apontados trabalhos futuros.

2 Ambientes de programação

Esta seção tratará das principais tecnologias empregadas no projeto e implementação desta ferramenta: programação visual, ferramentas de programação visual para ambientes paralelos e distribuídos, componentes de *software*, objetos distribuídos, programação orientada a eventos e ferramentas de desenvolvimento rápido.

2.1 Programação visual

Segundo [GLI84], uma linguagem ou sistema de programação é classificado como visual se uma ou ambas das seguintes condições for(em) satisfeita(s):

- entidades gráficas de nível mais alto estão disponíveis ao usuário como átomos que ele pode manipular;
- elementos gráficos (que podem conter textos e números como componentes) formam uma parte integral (não meramente decorativa) da exibição gerada pelo sistema para usuários que estejam programando ou executando uma aplicação.

Resumindo, um programador de uma linguagem visual é capaz de expressar uma ação ou estado de uma aplicação através de gráficos. Embora essa não seja a única definição existente, pode-se aceitá-la como correta tendo em vista o caráter informal dessa definição. Uma definição ligeiramente diferente não alteraria o modo como o usuário enxerga um sistema visual.

2.2 Ferramentas visuais para ambientes paralelos e distribuídos

Linguagens de programação concorrente baseadas em representações textuais não são a melhor escolha para se descrever a concorrência. Isso se deve ao fato de que a ordem seqüencial do texto esconde a estrutura multidimensional dos programas [WIR94b]. Assim, utilizada na programação centralizada há mais tempo, a programação visual vem sendo pesquisada quanto ao seu uso na programação distribuída. Dessa pesquisa resultaram algumas ferramentas que procuram representar visualmente uma aplicação paralela ou distribuída: HeNCE [BEG94], CODE 2 [NEW92], VPE [NEW95], VisualProg [SCH96], GRADE [KAC97], Meander [WIR94a], P-RIO [LOQ98], Enterprise [WIL93], Tracs [BAR95] e PVMBuilder [PED99], entre outras. Estas ferramentas representam a aplicação como um grafo dirigido, que é considerada a forma mais adequada de se representar uma estrutura concorrente [BRO94]. Neste esquema, a aplicação é um conjunto de nodos e arcos, onde os nodos indicam os elementos distribuídos e os arcos dão informações sobre o relacionamento entre tais elementos.

2.3 Componentes de software

Um componente de *software* possui várias definições, mas todas elas giram em torno da interface padronizada que ele define e das capacidades de reutilização e de modularidade que possui. Essas características permitem a composição de componentes por terceiros na formação de estruturas de *software* mais complexas, eliminando muitas fases do desenvolvimento e assim aumentando a rapidez e a confiabilidade do desenvolvimento [BRO98].

O emprego desta tecnologia é motivado por várias razões: reutilização de projeto e implementação, aumento da confiabilidade (em geral o código já foi testado), ganho de tempo, diminuição do custo de manutenção, possibilidade de adoção de soluções

padronizadas de forma mais rápida e inclusão de novas funcionalidades a qualquer tempo ao adicionar novos componentes a uma solução já existente.

As desvantagens são a falta de garantia da qualidade dos componentes, a necessidade de um tempo extra para leitura da documentação e aprendizado do uso do componente, necessidade de novos testes em função de um novo contexto [WEY98], possibilidade de o desenvolvedor do componente cessar o suporte à sua manutenção e cavalos de Tróia. Além disso, atualizações no componente para adição de novas funcionalidades ou correção de *bugs* podem torná-lo incompatível com sistemas anteriormente suportados [VOA98].

Apesar dessas desvantagens, a existência de um sem número de ferramentas de desenvolvimento baseadas nessa tecnologia e o surgimento contínuo de novas arquiteturas de componentes atestam a sua validade. Um exemplo importante é a arquitetura de componentes da linguagem Java chamada JavaBeans [SUN01a], utilizada neste trabalho.

2.4 Objetos distribuídos

A combinação de duas tecnologias importantes que surgiram nos últimos tempos – redes de computadores e orientação a objeto – resultou no objeto distribuído. Um objeto distribuído possui todas as características de um objeto tradicional (modularidade, herança, encapsulamento, polimorfismo) com a diferença de que ele mantém essas características também num ambiente de rede, o que torna a localização do objeto um detalhe menos importante no momento de se fazer acessos a métodos ou dados desse objeto.

Para que um ambiente completo de objetos distribuídos funcione, com invocação e instanciação remota de métodos, é necessária a existência de um sistema agente que se responsabilize pelo gerenciamento dos objetos através dos diferentes ambientes locais que contêm objetos. As tecnologias mais conhecidas atualmente para resolver esse problema na arquitetura Java, utilizada neste trabalho, são Java RMI [SUN01b], Voyager [GLA99] e CORBA [OMG01].

2.5 Programação orientada a eventos e ferramentas RAD

Na programação tradicional, o fluxo de controle de execução é determinado por uma seqüência de comandos, executados um após o outro. Na programação orientada a eventos, o fluxo de execução dos objetos e por conseqüência de toda a aplicação é regido pelos eventos que cada objeto é capaz de gerar e de processar. Este tipo de abordagem é especialmente útil quando existem várias fontes para o controle de execução, como por exemplo uma interface gráfica, onde os comandos podem ser disparados de vários dispositivos simultaneamente, como mouse e teclado.

A aplicação inicial deste modelo de programação foram as interfaces gráficas. Após algum tempo, ele começou a ter papel importante também nas ferramentas RAD (*Rapid Application Development*). Ferramentas RAD [CAR95] possuem o objetivo principal de tentar reduzir o tempo de construção do *software*, utilizando-se para isso principalmente de orientação a objeto, componentes de *software*, geração automática de código (principalmente para interfaces gráficas), ferramentas de documentação e de depuração das aplicações. Dois representantes conhecidos dessa classe de ferramentas são os ambientes de programação Delphi e JBuilder, que utilizam esses conceitos para facilitar a geração de interfaces gráficas.

3 Modelo

Neste capítulo, será descrito o modelo existente por trás da ferramenta. Esta descrição começa com seus objetivos, passando pela sua estrutura geral, pelo modelo de programação e pela linguagem visual empregada na construção das aplicações.

3.1 Objetivos, aplicações e usuários

Os objetivos e requisitos de uma ferramenta de programação para ambientes paralelos e distribuídos devem obviamente ser buscados nas necessidades de seus programadores. Muitas delas são necessidades comuns a quaisquer aplicações e são resultado dos princípios básicos de engenharia de *software*. Dentre os principais objetivos e requisitos, destacam-se:

- **programação orientada a objeto**: a possibilidade de empregar os conceitos de encapsulamento, modularização, herança e polimorfismo é importante no desenvolvimento de qualquer tipo de aplicação, seja ela concorrente ou não. Lado a lado com ela, seguem os componentes de *software*, que permitem que haja reutilização de código e extensão do sistema;
- **programação visual**: a visualização é muito importante em programação concorrente, pois facilita a interpretação do comportamento e da estrutura global da aplicação;
- **facilidade de uso e rapidez de desenvolvimento**: a programação concorrente é por si só mais complexa do que a programação centralizada, o que torna a facilidade de uso um aspecto mais crítico. Uma ferramenta mais fácil aumenta também a rapidez de desenvolvimento e por conseguinte a produtividade;
- **modelo de programação abstrato**: responsabilizando-se pela parte de programação de mais baixo nível e oferecendo ao usuário um modelo abstrato, a ferramenta livra o usuário de tarefas cuja execução é fonte provável de erros;
- **flexibilidade**: apesar de utilizar uma linguagem visual para a programação (o que costuma reduzir a flexibilidade), a geração de código fonte textual e o controle do usuário sobre ele mantêm a flexibilidade da programação textual tradicional;
- **portabilidade**: em sistemas distribuídos, a execução de programas em ambientes heterogêneos é uma necessidade, o que exige o suporte a diferentes plataformas de *hardware* e de *software*.

O alvo principal dessa ferramenta são as aplicações baseadas em objetos distribuídos, de âmbito local ou global, onde a concorrência seja definida explicitamente. Neste modelo, o programador é responsável pela decomposição, sincronização e comunicação distribuídas. Este é um dos motivos pelos quais se utiliza a programação visual, pois ela é ainda mais útil quando se deve realizar essas tarefas explicitamente.

Quantos aos usuários, procurou-se destinar a ferramenta aos programadores mais experientes, que já possuem conhecimento de programação distribuída. Como as tarefas de distribuição, comunicação e sincronização devem ser coordenadas pelo usuário, usuários sem conhecimento nessa área necessitarão de treinamento adicional para utilizarem a ferramenta plenamente. A flexibilidade oferecida pela programação na ferramenta é uma maneira de atender melhor a usuários mais experientes.

3.2 Estrutura e funcionamento da ferramenta

A estrutura da ferramenta é bastante semelhante à estrutura das ferramentas de desenvolvimento rápido de aplicações como Delphi e JBuilder. As principais partes da ferramenta, que podem ser visualizadas na Figura 3-1, são:

- **janela principal**: contém a barra de menus e de ferramentas, a paleta de componentes e uma área de texto contendo as mensagens da ferramenta e a saída das aplicações;
- **janela de edição do grafo**: formada pelo editor de grafos, pela visualização da estrutura do grafo e pelos editores de texto onde o código Java dos nodos é editado;
- **janela de propriedades e eventos**: apresenta editores para as propriedades dos objetos e possibilita a definição dos métodos tratadores dos eventos gerados pelos objetos.

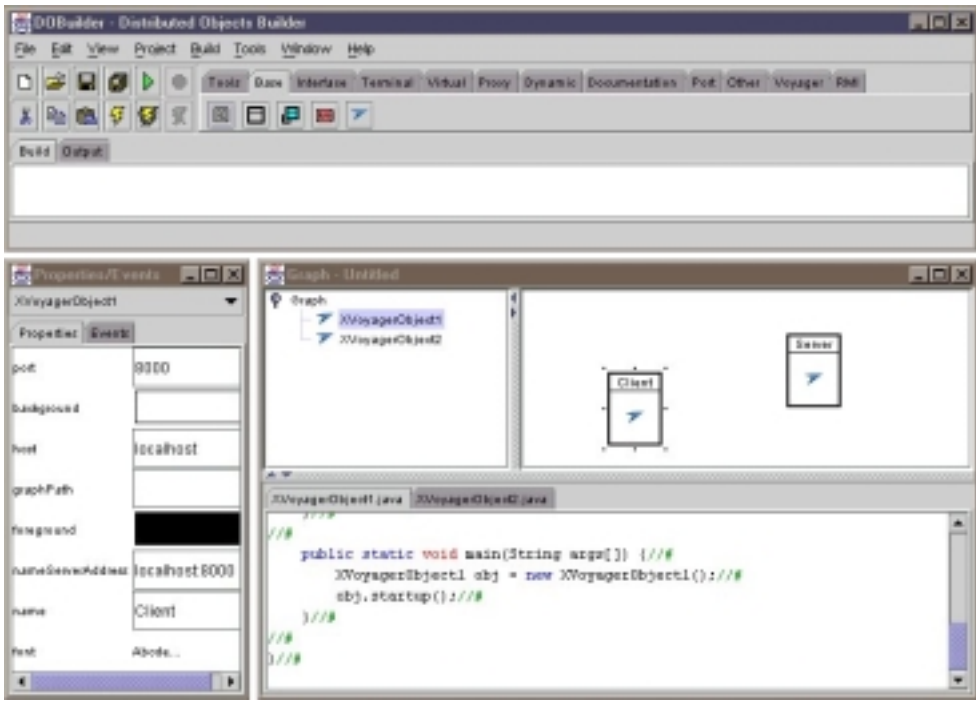

Figura 3-1. Interface gráfica da ferramenta.

O funcionamento da ferramenta, do ponto de vista do programador, é semelhante às ferramentas RAD. O processo se inicia pela definição da estrutura aplicação, por meio da manipulação visual dos componentes disponíveis na paleta de componentes. À medida que o programador insere componentes no grafo da aplicação, o código fonte textual é atualizado para que haja sincronismo entre as representações visual e textual. Enquanto o programador introduz os componentes, ele pode personalizar a aplicação inserindo código fonte Java diretamente em meio ao código gerado ou alterando as propriedades e métodos tratadores de eventos dos componentes inseridos no grafo.

Após inserir todos os objetos e personalizá-los de acordo com a sua aplicação, o programador comanda a geração de código para o controle da aplicação em ambiente distribuído (quando necessário), compila e executa o seu código.

3.3 Modelo de programação

O modelo de especificação das aplicações nessa ferramenta se baseia na construção de grafos dirigidos, empregados na maioria das ferramentas de programação visual para ambientes paralelos e distribuídos. O grafo é um meio natural de se especificar um cenário formado por diversos elementos que possuem relacionamentos entre si.

De maneira geral, nessas ferramentas os nodos se referem aos elementos que executam concorrentemente. A semântica referente aos arcos não é rígida e depende tanto do modelo geral de programação escolhido como dos tipos de objetos envolvidos no relacionamento. Há linguagens visuais de programação paralela onde o arco indica fluxo de dados (CODE 2, VPE, VisualProg, GRADE, P-RIO, Tracs) e outras onde a mesma representação se traduz por fluxo de controle (HeNCE, Meander, Enterprise). Neste trabalho, o significado do relacionamento dependerá dos tipos de objetos envolvidos. Por exemplo, numa relação entre uma porta de comunicação de entrada e uma de saída, o arco indica comunicação por troca explícita de mensagem. Em outro caso, quando o arco conecta um objeto cliente a um objeto que exporta um método remoto, a interação indicará que o objeto origem está chamando um método executado remotamente no objeto designado pelo nodo destino do arco. Para identificar com clareza o tipo de interação, a representação visual do arco também varia.

3.4 Nodos

O nodo, juntamente com o arco, corresponde à parte mais importante da linguagem visual. Os tipos de nodos existentes nessa linguagem visual são os seguintes (Figura 3-2):

- **nodo básico**: representa a unidade básica de distribuição personalizada pelo usuário. O objeto designado por este nodo terá o seu código em Java editado diretamente pelo programador com a lógica da aplicação para tal objeto. Este tipo de objeto será instanciado remotamente e poderá disponibilizar alguns de seus métodos para que sejam chamados de locais remotos;
- **nodo de interface**: nodo que descreve a interface de acesso a um objeto distribuído, contendo os métodos que podem ser chamados remotamente. Corresponde à interface da linguagem de programação Java, o que significa que são objetos que apenas representam uma interface de acesso ao objeto, sem conter a implementação deles;
- **nodo virtual**: representa um objeto externo ao sistema e acessado pela aplicação, como por exemplo um servidor de envio de mensagens de correio eletrônico. A ferramenta não gera código para este objeto, mas através do nodo virtual os objetos da aplicação tomam informações sobre como ter acesso ao objeto real;
- **nodo terminal**: é um nodo básico ou de interface em formato de código objeto. Não pode, portanto, ter seu código editado. Serve para a formação de uma biblioteca de componentes para esta ferramenta (o nodo terminal está para a linguagem desta ferramenta do mesmo modo que os componentes JavaBeans estão para a linguagem Java);
- **componente**: módulo de *software* adicionado a um nodo básico com o objetivo de executar alguma função específica. A ligação entre um componente e o nodo básico a que ele pertence é feita através do registro e notificação de eventos;
- **componente procurador**: tipo especial de componente acoplado a um nodo básico para que este possa invocar métodos em objetos remotos;

- **componente para criação dinâmica**: também é um tipo especial de componente agregado a um nodo básico e tem a finalidade de prover a esse nodo básico a capacidade de criar dinamicamente outros objetos distribuídos;
- **método**: objeto adicionado a nodos básicos e nodos de interface com o intuito de representar visualmente porções do código desses objetos. Corresponde aos métodos de classes e de interfaces da linguagem Java;
- **nodo de interface de grafo**: quando conectado a um objeto do grafo, define que este objeto do grafo será visível fora dele nas chamadas a grafo. A interface do grafo será visível no nodo de chamada a grafo;
- **chamada a grafo**: a fim de facilitar a visualização de aplicações com um grande número de objetos, um objeto de chamada a grafo encapsula uma estrutura composta por vários objetos num único objeto no programa visual. Além de facilitar a visualização de aplicações complexas, permite que haja reutilização de conjuntos de objetos.

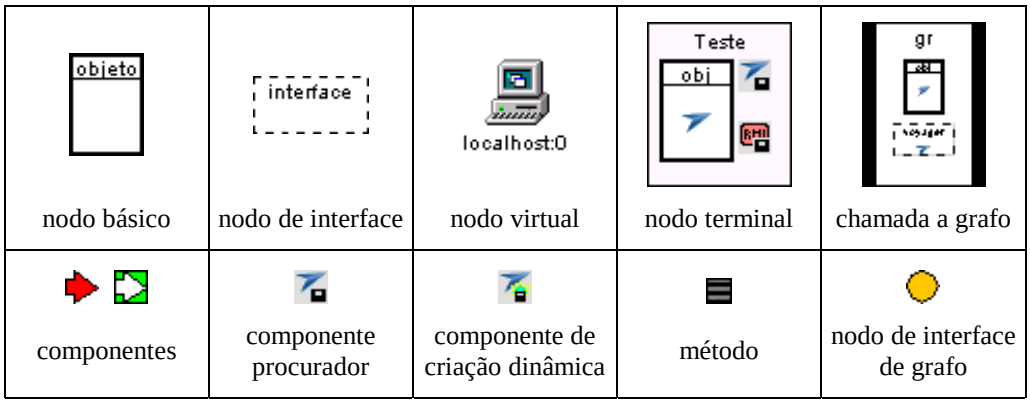

Figura 3-2. Tipos de objetos da linguagem visual.

Os objetos apresentados até agora e que podem ser vistos na Figura 3-2 são usados diretamente na especificação do programa. Existem ainda os objetos campo de texto e figura que são utilizados na documentação e apresentação visual do programa (Figura 3-3).

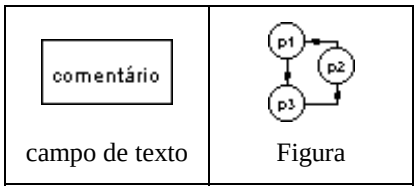

Figura 3-3. Objetos para documentação e apresentação visual.

A escolha de todos esses tipos de objetos foi baseada no estudo e análise de aplicações distribuídas que revelaram a sua necessidade. Não se pretende, portanto, afirmar que esta linguagem seja capaz de expressar qualquer programa distribuído. Seu objetivo é especificar as questões mais importantes nessa área, como invocação de método, criação de objetos e definição de interfaces. Além disso, uma estruturação modular da linguagem que permitisse uma implementação baseada em componentes foi outro objetivo perseguido neste projeto, pois com isso determinadas características que não estivessem presentes na sua versão inicial pudessem ser acrescentadas por meio de componentes.

3.5 Arcos

O arco é o elemento da linguagem que explicita graficamente os relacionamentos entre os diferentes objetos da aplicação. É ele que faz a diferença entre a programação visual e a programação textual, tornando a expressão de relacionamentos e ligações mais significativa e por isso muito importante em certos contextos, tais como a programação concorrente. Na programação textual não existe uma entidade desse tipo que se destine essencialmente a expressar relacionamentos e ligações. É por isso que relacionamentos como trocas de mensagens e chamadas remotas de método são difíceis de se perceber nesses programas. Para um ser humano, a representação visual de tais interações é consideravelmente mais expressiva do que a representação textual da mesma situação.

A semântica do arco varia conforme os tipos de objetos que estão sendo relacionados. Assim, combinando todos os tipos de objetos da linguagem, obtêm-se os relacionamentos constantes da Tabela 3-1.

Tabela 3-1. Relacionamentos.

Destino / Fonte	Nodo básico	Nodo de interface	Nodo virtual	Nodo terminal	Componente	Procurador	Criação dinâmica	Método	Interface de grafo	Chamada a grafo	Documentação
Nodo básico	3	2	1	-	1	1	1	1	8	7	9
Nodo de interface	-	-	-	-	-	-	-	-	8	-	9
Nodo virtual	-	-	-	-	-	-	-	-	8	-	9
Nodo terminal	-	-	-	-	-	-	-	-	8	-	9
Componente	1	1	1	-	1	1	1	1	8	-	9
Procurador	4	6	1	-	1	1	1	1	8	-	9
Criação dinâmica	5	6	1	-	1	1	1	1	8	-	9
Método	-	-	-	-	-	-	-	-	-	-	9
Interface de grafo	8	8	8	8	8	8	8	-	-	-	9
Chamada a grafo	-	-	-	-	-	-	-	-	-	-	9
Documentação	9	9	9	9	9	9	9	9	9	9	9

A coluna da esquerda da Tabela 3-1 indica o nodo fonte do relacionamento e a linha de cima, o destino. Isso significa que, por exemplo, um arco partindo de um nodo básico e chegando num nodo de interface estabelece o relacionamento 2, ou seja, uma implementação de interface. A tabela exibe ao todo nove tipos de relacionamentos:

- **(1) normal**: é um relacionamento onde há apenas amarração de propriedades entre os objetos fonte e destino. Por exemplo, entre uma porta de comunicação de entrada e uma de saída, há a amarração da propriedade “endereço de destino” da porta de saída com a propriedade “endereço local” da porta de entrada;

- **(2) implementação de interface**: o nodo origem implementa a interface representada pelo nodo destino;
- **(3) compartilhamento de memória**: os objetos fonte e destino compartilham o mesmo espaço de endereçamento. Como acontece entre nodos básicos, na prática este relacionamento permite que um nodo básico faça chamadas de métodos diretamente a outro nodo básico, pois os dois possuem a mesma localização física;
- **(4) referência a objeto remoto**: a origem do arco está chamando um método remoto que é executado no objeto destino do arco;
- **(5) criação de objeto remoto**: o nodo fonte do arco cria o objeto destino dinamicamente;
- **(6) interface de objeto remoto**: o objeto fonte do relacionamento conhece a interface implementada pelo objeto remoto;
- **(7) criação de grafo**: o nodo origem cria o grafo destino dinamicamente;
- **(8) interface de grafo**: o objeto conectado ao nodo de interface de grafo será visível externamente ao grafo;
- **(9) documentação**: indica apenas a que objeto se refere determinado objeto do tipo documentação.

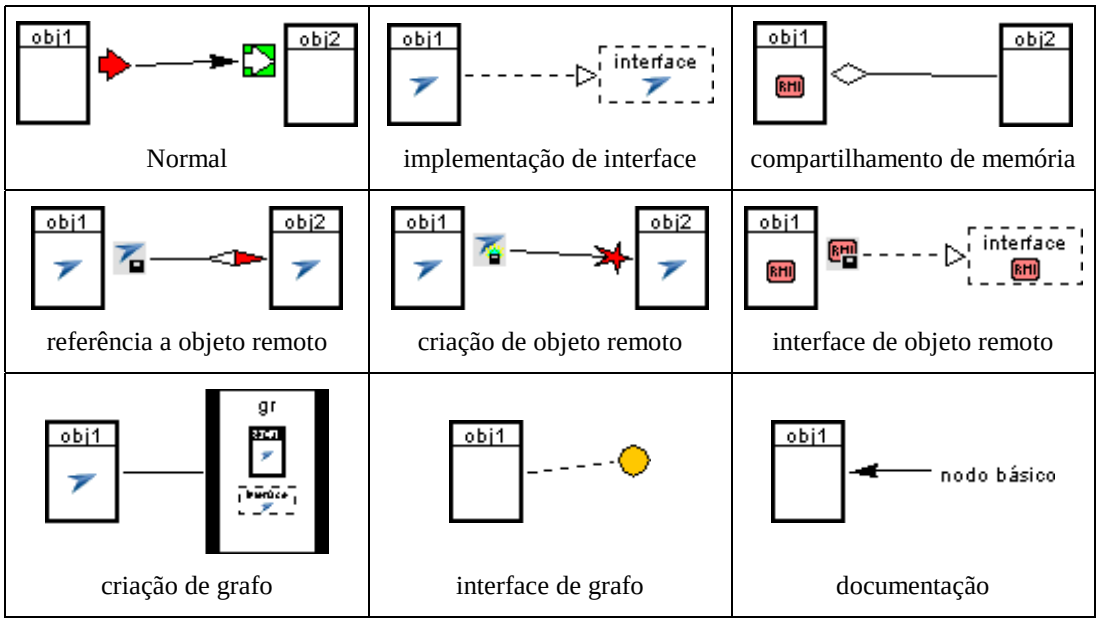

Figura 3-4. Diferentes tipos de arcos.

A Figura 3-4 mostra uma representação para os diferentes tipos de arcos. A linguagem visual não define como deve ser a aparência do arco, mas a implementação deve fazer com que os diferentes tipos de relacionamentos sejam representados visualmente de maneira diversa entre si e com a forma apropriada ao caso a que se referem. Isto permite flexibilidade na visualização e também que novos relacionamentos adicionados pelo usuário sejam configurados conforme a sua vontade.

4 Implementação

A implementação foi desenvolvida inteiramente em Java, utilizando-se o Sun JDK 1.2.2 como apoio ao desenvolvimento. Java possui muitas características importantes para a implementação deste modelo, como a existência de uma arquitetura padrão de componentes (JavaBeans), classes para geração de interfaces gráficas, orientação a objeto, portabilidade de código executável e suporte à programação de eventos e a objetos distribuídos.

Para o ambiente de execução dos objetos distribuídos foi escolhida a ferramenta Voyager, que entre outras coisas oferece invocação remota de métodos, criação remota de objetos e coleta de lixo distribuída. Além disso, outro ponto forte de Voyager é a sua simplicidade, que permite inclusive que objetos tradicionais desenvolvidos para sistemas centralizados sejam portados automaticamente para um ambiente distribuído.

4.1 Nodos e arcos

Os nodos são implementados através de interfaces Java. Isto significa que cada tipo de objeto deve implementar uma determinada interface para ser tratado pela ferramenta como um objeto de tal tipo. Por exemplo, o nodo básico deve implementar a interface XBaseNode, que contém métodos referentes ao comportamento e funcionamento de um nodo básico, como o método que define o gerador de código utilizado para responder à inserção de componentes e métodos ao objeto. Para facilitar a implementação por parte do usuário, são fornecidas implementações padronizadas para cada tipo de objeto. Assim, no caso do nodo básico padrão, há um objeto padronizado com a implementação da interface que pode servir de base para a implementação de qualquer outro nodo básico.

Os objetos oferecidos com a ferramenta são implementados como componentes JavaBeans. Dessa forma, cada objeto possui suas propriedades, métodos e eventos. As propriedades e eventos de um objeto são lidos dinamicamente pela ferramenta, que então oferece editores para introduzir e alterar seus valores durante a manipulação visual dos grafos. Esta análise é feita através da introspecção de Java, que oferece métodos para ler o código objeto dos componentes JavaBeans.

Os arcos não são implementados como objetos. Os arcos são relacionamentos estabelecidos de forma implícita e dependente dos tipos de objetos envolvidos. Num arco do tipo normal, o relacionamento é feito combinando-se propriedades dos objetos fonte e destino (as propriedades relacionadas possuirão o mesmo valor em tempo de projeto). O relacionamento de implementação de interface é concretizado pela adição da declaração da implementação da interface no código do nodo básico participante do arco. O mesmo acontece com os outros arcos, ou seja, não há objeto específico que explicite o arco, mas apenas um relacionamento estabelecido de forma implícita, assim como acontece em relacionamentos de bases de dados relacionais.

Os relacionamentos de referência a objeto remoto, criação de objeto e criação de grafo são ligeiramente mais complexos, pois adicionam código fonte específico para a atividade correspondente. Assim, por exemplo, no relacionamento de criação de objeto remoto, é inserido no objeto fonte do relacionamento uma porção de código para criar objetos remotos. Assim, no local onde o usuário deseja criar remotamente um objeto ou invocar um método remoto, ele insere uma chamada ao código criado para o relacionamento do arco.

4.2 Geração de código

Uma propriedade importante dos nodos básicos é o seu gerador de código. Quando um nodo básico é inserido no grafo da aplicação, é criada uma janela de edição de código fonte Java onde inicialmente é colocado um esqueleto do código fonte do objeto. Esse esqueleto é a declaração de uma classe que deriva da classe do tipo de objeto inserido.

À medida que componentes são adicionados ao nodo básico e suas propriedades são alteradas, o gerador de código é invocado para processar as alterações no código fonte. Quando isso acontece, é passado como parâmetro o código fonte original, a operação (e seus parâmetros) e é retornado o código fonte modificado. Para permitir que o usuário processe suas alterações no código, o gerador diferencia as linhas de código geradas e as linhas inseridas pelo usuário (as linhas geradas são anotadas com uma seqüência especial).

4.3 Grafos

Outra característica importante da ferramenta é a hierarquização da aplicação em grafos. Uma aplicação pode ser formada por vários grafos, cada um contendo o seu próprio domínio de nomeação, o que permite que se tenham várias instâncias de uma mesma classe de grafo interagindo entre si na aplicação.

Para que isto seja possível, quando a identificação de um objeto é inserida no servidor de nomes, é adicionada a ela a identificação do grafo ao qual ela pertence (caso não haja nenhuma identificação, o objeto pertence ao grafo principal do programa). Assim, se um grafo é criado dinamicamente a partir de outro, à identificação de seus objetos é prefixada a identificação do grafo. Esta estrutura de representação permite que se façam chamadas recursivas de grafos e se implemente algoritmos como o *quicksort* de forma simples.

4.4 Exemplo

O exemplo a seguir oferece uma visão de como uma aplicação final é representada nesta ferramenta. O exemplo da Figura 4-1 mostra o grafo de uma aplicação de bate-papo implementada com chamada remota de métodos.

Figura 4-1. Bate-papo implementado com chamadas de métodos remotos.

Neste exemplo, existem dois objetos principais: o nodo básico *client* e o nodo básico *server*. Este último, implementa a interface *interface* que contém três métodos na sua definição. O cliente possui dois componentes: um componente de interface gráfica e um componente procurador. Este último se conecta visualmente à interface implementada pelo servidor, permitindo que o objeto cliente conheça os métodos disponibilizados no servidor e ao próprio servidor. O objeto *client* contém também um componente de interface gráfica que faz a interação com o usuário.

5 Trabalhos relacionados

Ao longo deste trabalho foram citados diferentes tipos de ferramentas que podem ser comparadas com a que é proposta neste trabalho. Cada tipo possui suas virtudes e deficiências, cuja análise serviu de base para a produção deste trabalho. Procurou-se tomar de cada ferramenta as características que seriam úteis no desenvolvimento de aplicações com objetos distribuídos e assim chegar a uma ferramenta melhor para o programador destes tipos de aplicações. Segue então uma breve análise sobre as ferramentas de programação textual para ambientes distribuídos, ferramentas de programação visual para esses mesmos ambientes e ferramentas RAD. Além disso, é feita uma comparação com relação aos ambientes de programação em geral, onde se buscou atingir objetivos comuns a qualquer um deles.

5.1 Ambientes de programação em geral

Como objetivos de vários ambientes de programação, esta ferramenta apresenta uma linguagem de programação orientada a objeto, com suporte a componentes reutilizáveis de *software*. O código é portável entre diferentes arquiteturas e em boa parte gerado automaticamente, diminuindo assim o número de erros. Como deficiências, estão o menor desempenho de Java e a necessidade de treinamento do usuário.

5.2 Ferramentas de programação textual

Com relação às ferramentas tradicionais de programação textual para objetos distribuídos, onde a distribuição é feita através de bibliotecas de comunicação ou de uma sintaxe específica da própria linguagem (programação distribuída direta com Java, por exemplo) as vantagens são a visualização gráfica dos relacionamentos e da estrutura da aplicação, o uso da linguagem Java (orientada a objeto e já conhecida), emprego de componentes de *software* e eficiência com relação à explicitação da concorrência. Entretanto, as linguagens textuais já possuem bastante código desenvolvido e se baseiam em ambientes de execução mais eficientes (Java é bastante ineficiente).

5.3 Ferramentas de programação visual

As vantagens em relação à maioria das ferramentas atuais de programação visual para ambientes paralelos e distribuídos como HeNCE, CODE, VisualProg e VPE são a programação de objetos distribuídos, programação orientada a eventos, melhor suporte à engenharia do *software* com o uso de componentes, portabilidade da ferramenta e do código gerado, programação cliente-servidor e flexibilidade de codificação originada da geração de código textual.

Como desvantagens, consta a falta de visualização da execução e depuração das aplicações, ineficiência de Java para programação paralela, falta de suporte direto para paralelismo (de dados, por exemplo) e de determinadas características especiais que existem em algumas ferramentas, como por exemplo tolerância a falhas.

5.4 Ferramentas RAD

Com relação às ferramentas de programação visual em Java (JBuilder, por exemplo), as principais vantagens são a visualização global da aplicação distribuída, o desenvolvimento hierárquico das aplicações e o desenvolvimento integrado de clientes e servidores. As desvantagens são que não há suporte à geração de interfaces gráficas, não há depuração do código seqüencial e a geração de código não é totalmente transparente (ainda que poucas, há algumas regras que devem ser seguidas pelo programador).

6 Conclusões

Este trabalho apresentou o projeto e a implementação de uma ferramenta de programação visual para o desenvolvimento de aplicações que usam objetos distribuídos. As principais motivações vieram do uso da programação visual na programação concorrente e da utilização de conceitos de engenharia de *software* na programação de objetos distribuídos, amenizando assim deficiências das ferramentas atuais de programação paralela e distribuída. Procurando facilitar a tarefa dos programadores dessas aplicações, a ferramenta apresenta como características principais a programação visual com objetos distribuídos, a programação orientada a eventos, o suporte à reutilização de componentes e a geração automática parcial de código para a linguagem Java.

A fim de verificar se os objetivos da ferramenta foram atingidos, deve-se avaliar a ferramenta conforme o ponto de vista do usuário, o que não pode ser feito simplesmente de maneira teórica. Uma das dificuldades da validação da usabilidade dos sistemas é o fato de que não há estudo teórico que possa ser usado para avaliar a facilidade de uso de um ambiente ou metodologia de programação. A conseqüência disso é que se deve realmente implementar e usar os sistemas para compreender os seus méritos e limitações [BRO94]. A constatação acaba sendo feita de forma natural através da aceitação ou não dos usuários, com conclusões freqüentemente subjetivas e dependentes do contexto.

Sendo assim, sem uma implementação, pouco poderia se concluir sobre a validação deste trabalho, o que torna a sua implementação uma atividade obrigatória. Depois disso, para ajudar na validação, esta ferramenta foi utilizada por alunos experientes em programação concorrente e em Java numa disciplina de objetos distribuídos no Curso de Pós-Graduação do Instituto de Informática da UFRGS. Como principais pontos positivos, foram citados pelos seus usuários os seguintes [POR01]: existência de componentes para programação distribuída, programação através do modelo de eventos que evita que programadores Java necessitem conhecer os mecanismos de passagem de mensagens, documentação gráfica do sistema e uma interface amigável que respeita os padrões dos ambientes de desenvolvimento mais conhecidos atualmente. Tais opiniões vieram ao encontro dos objetivos da ferramenta, que eram enfatizar a programação distribuída, procurando facilitá-la através de componentes e da programação visual, juntamente com uma interface de programação adequada ao programador. As desvantagens são principalmente o baixo desempenho de Java e a necessidade de adaptação dos usuários antigos à nova ferramenta.

Como principais trabalhos futuros, citam-se o acréscimo de módulos de monitoração e visualização, a implementação da depuração tanto seqüencial para o código interno do objeto como distribuída para análise da comunicação, implementação de novos componentes, incorporação de um escalonador para a escolha da máquina mais adequada à instanciação de um novo objeto e adição de novos recursos à linguagem visual para aumentar o seu poder de expressão. Alguns trabalhos já estão sendo desenvolvidos e integrados ao DOBuilder, como o ReMMoS [FER00] (ambiente de controle transparente de réplicas de objetos), o DOVisualizer [ARA00] (visualização de objetos distribuídos) e o DEPAnalyzer [AZE00] (analisador estático de dependências entre objetos).

Resumindo, este trabalho apresentou o projeto e a implementação de uma ferramenta de programação visual para objetos distribuídos em Java que, além de auxiliar no desenvolvimento através da programação visual, procura também oferecer um ambiente que aplique os principais conceitos de engenharia de *software*. Com base no uso e nos testes feitos, concluiu-se que a ferramenta facilita o desenvolvimento das aplicações, pois oferece ao usuário um modelo de programação flexível e uma interface visual amigável.

7 Referências

[ARA00] ARAÚJO, E. B. **Uma Ferramenta de Visualização para Aplicações Distribuídas em Java**. V Semana Acadêmica do PPGC, II-UFRGS. Disponível em http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana2000/EdvarAraujo (25 Nov. 2000).

[AZE00] AZEVEDO, S. C. **Análise Estática de Programas Orientados a Objetos**. V Semana Acadêmica do PPGC, II-UFRGS. Disponível em http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana2000/SilvanaAzevedo (25 Nov. 2000).

[BAR95] BARTOLI, A.; CORSINI, P.; DINI, G.; PRETE, C. A. Graphical Design of Distributed Applications Through Reusable Components. IEEE Parallel & Distributed Technology, Spring 1995, p.37-50.

[BEG92] BEGUELIN, A.; DONGARRA, J.; GEIST, A.; SUNDERAM, V. **Visualizing and Debugging in a Heterogeneous Environment.** IEEE Computer, v.26, n.6, Jun. 1993.

[BEG94] BEGUELIN, A.; DONGARRA, J.; GEIST, A.; MANCHEK, R.; MOORE, K.; NEWTON, P.; SUNDERAM, V. **HeNCE: A Users' Guide Version 2.0.** Junho, 1994. Disponível em ftp://netlib2.cs.utk.edu/hence/HeNCE-2.0-doc.ps.gz (23 Jan. 2001).

[BOR01a] BORLAND. **Delphi**. Disponível em http://www.borland.com/delphi. (23 Jan. 2001).

[BOR01b] BORLAND. **Borland JBuilder**. Disponível em http://www.borland.com/jbuilder. (23 Jan. 2001).

[BRI98] BRIOT, J.-P.; GUERRAOUI, R.; LÖHR, K.-P. **Concurrency and Distribution in Object-Oriented Programming**. ACM Computing Surveys, New York, Setembro, 1998.

[BRO94] BROWNE, J. C.; DONGARRA, J.; HYDER, S. I.; MOORE, K.; NEWTON, P. **Visual Programming and Parallel Computing.** Dept. of Computer Sciences, Univ. of Texas at Austin, Relatório Técnico TR94-229, 1994. Disponível em ftp://ftp.cs.utexas.edu/pub/code2/ut-cs-94-229.ps.Z (25 Nov. 2000).

[BRO98] BROWN, A. W.; WALLNAU, K. C. **The Current State of CBSE.** IEEE Software, vol. 15(5):37-46. Setembro/Outubro 1998.

[CAR95] CARD, D. N. **The RAD Fad: Is Timing Really Everything?** IEEE Software, vol. 12(5):19-22. Setembro, 1995.

[FER00] FERRARI, D. N; VARGAS, P. K.; GEYER, C. F. R. **ReMMoS - um modelo de replicação em ambientes que permitem mobilidade de objetos.** WSCAD' 00 - Workshop de Sistemas Computacionais de Alto Desempenho - SBAC-PAD'00, São Pedro, SP, Brasil. Outubro, 2000.

[GLA99] GLASS, G. **Overview of Voyager: ObjectSpace's Product Family for State-of-the-Art Distributed Computing.** ObjectSpace, 1999.

[GLI84] GLINERT, E. P.; TANIMOTO, S. L. **PICT: An interactive graphical programming environment.** IEEE Computer, vol. 17(11):7-28. Nov., 1984.

[KAC97] KACSUK, P.; CUNHA, J. C.; LOURENÇO, J.; DÓZSA, G.; FADGYAS, T.; ANTAO, T. **A graphical development and debugging environment for parallel programs.** Parallel Computing, Parallel Computing Journal, Elsevier, vol. 22(13):1747-1770. Fevereiro, 1997.

[LOQ98] LOQUES, O.; LEITE, J.; CARRERA, E. V. **P-RIO: A Modular Parallel-Programming Environment.** IEEE Concurrency, p.37-50, Jan.-Mar., 1998.

[MAL97] MALACARNE, J. **Implementação de um Ambiente Gráfico para o Desenvolvimento de Aplicações Distribuídas.** Projeto de Diplomação, CIC-UFRGS, 91 pp., Porto Alegre, 1997.

[NEW92] NEWTON, P.; BROWNE, J. C. **The CODE 2.0 Graphical Parallel Programming Language.** Proc. ACM Int. Conf. on Supercomp. Julho, 1992.

[NEW95] NEWTON, P.; DONGARRA, J. **Overview of VPE: A Visual Environment for Message-Passing.** 1995. Disponível em ftp://cs.utk.edu/pub/newton/vpe/docs/hcw95.ps.Z (23 Jan. 2001).

[PED99] PEDERSEN, J. B.; WAGNER, A. **PVMbuilder - A Tool for Parallel Programming**. P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 108-112, 1999. Springer-Verlag Berlin Heidelberg 1999.

[OMG01] OMG - Object Management Group. **CORBA - Common Object Request Broker Architecture.** Disponível em http://www.omg.org/ (7 Fev. 2000).

[POR01] PORTO, E. K. K.; PILLA, M. L. **DOBuilder**. Disponível em http://www.inf.ufrgs.br/procpar/disc/cmp167/trabalhos/sem2000-1/T2/kenzo_pilla/ (23 Jan. 2001).

[SCH96] SCHRAMM, J. F. L. **Ambiente Gráfico para o Desenvolvimento de Aplicações Distribuídas**. CPGCC-UFRGS, 1996, 78p.

[SUN01a] SUN MICROSYSTEMS. **JavaBeans.** Disponível em http://java.sun.com/products/javabeans/ (23 Jan. 2001).

[SUN01b] SUN MICROSYSTEMS. **RMI.** Disponível em http://java.sun.com/products/jdk/rmi/ (23 Jan. 2001).

[VOA98] VOAS, J. **Maintaining Component-Based Systems**. IEEE Software, vol. 15(4):22-27. Julho/Agosto, 1998.

[WEY98] WEYUKER, E. J. **Testing Component-Based Software: A Cautionary Tale**. IEEE Software, vol. 15(5):54-59, Setembro/Outubro 1998.

[WIL93] WILSON, G. V.; SCHAEFFER, J.; SZAFRON, D. **Enterprise in Context: Assessing the Usability of Parallel Programming Environments**. Department of Computer Science. The University of Alberta, Edmonton, Alberta, Canada, Relatório Técnico TR 93-09. Junho, 1993.

[WIR94a] WIRTZ, G. **The Meander Language and Programming Environment.** in Proceedings of the 2nd SMS-TPE, Moscow, Rússia. Setembro, 1994.

[WIR94b] WIRTZ, G. **Modularization and Process Replication in a Visual Parallel Programming Language.** in Proceedings of the IEEE Visual Languages 1994, St. Louis, USA. Setembro, 1994.