O Uso de Técnicas Visuais e Navegacionais para a Compreensão de *Frameworks* Orientados a Objetos

Marcelo Campo Ing. * R.T. Price Eng., M.Sc., D. Phil

Universidade Federal do Rio Grande do Sul - Instituto de Informática
Caixa Postal 15064 - Porto Alegre - RS- Brasil

* Universidad Nacional del Centro de la Provincia de Bs. As.
Fac. de Ciencias Exactas - ISISTAN
San Martín 57, (7000) Tandil, Bs. As., Argentina

email: {mcampo, tomprice}@inf.ufrgs.br

Resumo

Os *frameworks orientados a objetos* representam uma tecnologia de reutilização poderosa para construir, por especialização, aplicações dentro de um domínio. Entretanto, compreender as classes de um framework para especializá-las é uma tarefa que requer esforço e tempo consideráveis. Por isto, é muito importante dispor de ferramentas que permitam analisar aplicações construídas utilizando um framework através da visualização da estrutura estática e dinâmica das classes. Neste trabalho se descreve uma ferramenta que combina tecnologias visuais e navegacionais para ajudar na compreensão do funcionamento de um *framework*, através da análise dinâmica de aplicações construídas a partir dele. A análise dinâmica é realizada utilizando técnicas de reflexão computacional, as quais permitem monitorar transparentemente o comportamento da aplicação. A representação visual permite identificar rapidamente componentes abstratos e as hierarquias de classes que eles definem. O usuário pode interagir com a representação visual do framework, animar manualmente ou automaticamente as seqüências de passagem de mensagens e acessar o código dos métodos. A ferramenta tem demonstrado a sua eficácia para diminuir em muito o tempo necessário para compreender o funcionamento tanto de *frameworks* como de bibliotecas de classes existentes.

Abstract

Object-oriented frameworks represent a powerful reuse technology to build, by specialization, applications in a given domain. However, to understand the classes that compose a framework for their specialization is a hard and time consuming task. For this reason, tools that allow to analyze applications built using a framework, through the visualization of the static and dynamic structure of framework classes, are valuable. This work describes a tool that combines visual and navigational techniques to help the understanding of the dynamic behavior of a framework, through the dynamic analysis of applications built with it. This dynamic analysis is made using techniques of computational reflection, that allow the transparent monitoring of the application behavior. The visual representation allows to quickly identify abstract components and the class hierarchies that they define. The user can interact with the visual representation of the framework, manually or automatically animate the sequence of messages and access the code of the methods. The tool has shown to be very useful to reduce much of the time needed to understand the behavior of both frameworks and existing class libraries.

1. Introdução

Os *frameworks orientados a objetos* representam uma tecnologia de reutilização poderosa para a construção de aplicações através da especialização de arquiteturas genéricas de domínios de aplicação [Deutsch 89][Johnson 88]. Um *framework* é constituído por um conjunto de classes que abstraem as caraterísticas gerais de um domínio, e codificam uma estrutura de controle que invoca métodos que devem ser implementados por subclasses. Aplicações específicas são construídas especializando as classes do *framework* só para fornecer a implementação destes métodos, enquanto a estrutura de controle global da aplicação é herdada. Deste modo, um *framework* funciona como um molde para a construção de aplicações, ou subsistemas, dentro do domínio. Estas aplicações possuem todas a mesma estrutura, diferenciando-se no comportamento que implementam para os métodos que são invocados pelo framework.

A herança da estrutura de controle contrasta com a reutilização de componentes implementados por classes isoladas. Neste caso o usuário deve implementar o código que invoca os métodos definidos nessas classes. Para isto é suficiente conhecer a interface da classe, não importando seu projeto interno nem a sua implementação. No caso dos frameworks, entretanto, para completar o comportamento necessário um usuário precisa *compreender* o projeto interno das classes e a forma em que elas colaboram [Lajoie 94]. Estas classes se organizam em hierarquias estáticas que codificam o comportamento de um conjunto de instâncias que serão organizadas dinamicamente como uma rede, cuja topologia geralmente evolui nas diferentes etapas da execução da aplicação. Ainda que haja documentação disponível, compreender o comportamento destas aplicações é uma tarefa complexa que requer tempo considerável [Jonson 92][Buhr 92].

Examinar aplicações desenvolvidas utilizando um *framework* é, geralmente, um bom ponto de início para compreender como suas classes são instanciadas e relacionadas umas com outras. Por isto, são de grande utilidade ferramentas que possam inspecionar o comportamento dinâmico destas aplicações e coletar informação relevante acerca de como as classes estão relacionadas e colaboram entre si. Esta informação pode ser organizada e apresentada visualmente ao usuário, ajudando na compreensão do *framework*. A utilização de técnicas visuais permite concentrar maior densidade de informação numa apresentação e explorar a capacidade humana para análise de informação espacial e interpretação de cores. Somado a isto, as facilidades para construir animações que elas oferecem facilitam o processo de compreensão do comportamento dinâmico, o qual não pode ser convenientemente representado por notações textuais. Este potencial é ainda maior quando a visualização é complementada com técnicas que suportam a interação do usuário com a execução da aplicação e sua integração com o ambiente de desenvolvimento [Laffra 93].

Neste trabalho se apresenta MOViE (Meta-Object-based Visualization Environment), uma ferramenta projetada com o objetivo de apoiar na análise do comportamento de sistemas orientados a objetos e, em especial, de frameworks orientados a objetos. MOViE combina tecnologias visuais e navegacionais para ajudar na compreensão do funcionamento de um framework através da análise dinâmica de aplicações construídas a partir dele. A análise dinâmica é realizada utilizando técnicas de reflexão computacional, as quais permitem monitorar transparentemente o comportamento das aplicações, sem necessidade de realizar mudanças no seu código. O usuário pode navegar sobre a representação visual da arquitetura e reproduzir manualmente ou automaticamente as seqüências de passagem de mensagens, tanto no sentido normal como reverso.

Na primeira parte do artigo se realiza uma caraterização dos aspectos fundamentais envolvidos na visualização de *frameworks* e se definem os requisitos básicos para uma ferramenta que auxilie na compreensão da estrutura estática e dinâmica das classes. A seguir se descrevem as caraterísticas gerais de MOViE através de exemplos que ilustram as técnicas de visualização e animação arquitetônica utilizadas. Na seção seguinte se descreve sinteticamente o mecanismo reflexivo utilizado para a análise da execução de aplicações construídas com um framework. As figuras correspondentes aos exemplos são apresentadas no final do artigo.

2. Requisitos para Apoiar na Compreensão de Frameworks

Um framework orientado a objetos representa, em termos de classes, uma *arquitetura* genérica para um domínio de aplicação. Uma arquitetura define a divisão de um problema em componentes encarregados de implementar funcionalidades bem definidas. Também, a arquitetura define a forma em que estes componentes colaboram para realizar a funcionalidade do sistema. Isto é, quais componentes tem um conhecimento mútuo e como é o fluxo de controle permitido entre eles. No nível arquitetónico a ênfase principal é colocada nestes aspectos e não na forma em que os componentes são implementados. Assim, uma visualização adequada para compreender um framework deve permitir, em primeiro lugar, a visualização da estrutura e funcionamento prescrito pela arquitetura que ele codifica.

2.1 Caraterização de Frameworks

Para definir um sistema de visualização adequado é necessário caraterizar o que deve ser visualizado. Nesta seção se definem os aspectos essenciais que caraterizam a um framework e que, portanto, devem ser contemplados por uma ferramenta de visualização.

Do ponto de vista estrutural, em framework é composto por dois tipos de componentes [Wirf 90][Jonson 91]:

1. Introdução

Os *frameworks orientados a objetos* representam uma tecnologia de reutilização poderosa para a construção de aplicações através da especialização de arquiteturas genéricas de domínios de aplicação [Deutsch 89][Johnson 88]. Um *framework* é constituído por um conjunto de classes que abstraem as caraterísticas gerais de um domínio, e codificam uma estrutura de controle que invoca métodos que devem ser implementados por subclasses. Aplicações específicas são construídas especializando as classes do *framework* só para fornecer a implementação destes métodos, enquanto a estrutura de controle global da aplicação é herdada. Deste modo, um *framework* funciona como um molde para a construção de aplicações, ou subsistemas, dentro do domínio. Estas aplicações possuem todas a mesma estrutura, diferenciando-se no comportamento que implementam para os métodos que são invocados pelo framework.

A herança da estrutura de controle contrasta com a reutilização de componentes implementados por classes isoladas. Neste caso o usuário deve implementar o código que invoca os métodos definidos nessas classes. Para isto é suficiente conhecer a interface da classe, não importando seu projeto interno nem a sua implementação. No caso dos frameworks, entretanto, para completar o comportamento necessário um usuário precisa *compreender* o projeto interno das classes e a forma em que elas colaboram [Lajoie 94]. Estas classes se organizam em hierarquias estáticas que codificam o comportamento de um conjunto de instâncias que serão organizadas dinamicamente como uma rede, cuja topologia geralmente evolui nas diferentes etapas da execução da aplicação. Ainda que haja documentação disponível, compreender o comportamento destas aplicações é uma tarefa complexa que requer tempo considerável [Jonson 92][Buhr 92].

Examinar aplicações desenvolvidas utilizando um *framework* é, geralmente, um bom ponto de inicio para compreender como suas classes são instanciadas e relacionadas umas com outras. Por isto, são de grande utilidade ferramentas que possam inspecionar o comportamento dinâmico destas aplicações e coletar informação relevante acerca de como as classes estão relacionadas e colaboram entre si. Esta informação pode ser organizada e apresentada visualmente ao usuário, ajudando na compreensão do *framework*. A utilização de técnicas visuais permite concentrar maior densidade de informação numa apresentação e explorar a capacidade humana para análise de informação espacial e interpretação de cores. Somado a isto, as facilidades para construir animações que elas oferecem facilitam o processo de compreensão do comportamento dinâmico, o qual não pode ser convenientemente representado por notações textuais. Este potencial é ainda maior quando a visualização é complementada com técnicas que suportam a interação do usuário com a execução da aplicação e sua integração com o ambiente de desenvolvimento [Laffra 93].

Neste trabalho se apresenta MOViE (Meta-Object-based Visualization Environment), uma ferramenta projetada com o objetivo de apoiar na análise do comportamento de sistemas orientados a objetos e, em especial, de frameworks orientados a objetos. MOViE combina tecnologias visuais e navegacionais para ajudar na compreensão do funcionamento de um framework através da análise dinâmica de aplicações construídas a partir dele. A análise dinâmica é realizada utilizando técnicas de reflexão computacional, as quais permitem monitorar transparentemente o comportamento das aplicações, sem necessidade de realizar mudanças no seu código. O usuário pode navegar sobre a representação visual da arquitetura e reproduzir manualmente ou automaticamente as seqüências de passagem de mensagens, tanto no sentido normal como reverso.

Na primeira parte do artigo se realiza uma caraterização dos aspectos fundamentais envolvidos na visualização de *frameworks* e se definem os requisitos básicos para uma ferramenta que auxilie na compreensão da estrutura estática e dinâmica das classes. A seguir se descrevem as caraterísticas gerais de MOViE através de exemplos que ilustram as técnicas de visualização e animação arquitetônica utilizadas. Na seção seguinte se descreve sinteticamente o mecanismo reflexivo utilizado para a análise da execução de aplicações construídas com um framework. As figuras correspondentes aos exemplos são apresentadas no final do artigo.

2. Requisitos para Apoiar na Compreensão de Frameworks

Um framework orientado a objetos representa, em termos de classes, uma *arquitetura* genérica para um domínio de aplicação. Uma arquitetura define a divisão de um problema em componentes encarregados de implementar funcionalidades bem definidas. Também, a arquitetura define a forma em que estes componentes colaboram para realizar a funcionalidade do sistema. Isto é, quais componentes tem um conhecimento mútuo e como é o fluxo de controle permitido entre eles. No nível arquitetônico a ênfase principal é colocada nestes aspectos e não na forma em que os componentes são implementados. Assim, uma visualização adequada para compreender um framework deve permitir, em primeiro lugar, a visualização da estrutura e funcionamento prescrito pela arquitetura que ele codifica.

2.1 Caraterização de Frameworks

Para definir um sistema de visualização adequado é necessário caraterizar o que deve ser visualizado. Nesta seção se definem os aspectos essenciais que caraterizam a um framework e que, portanto, devem ser contemplados por uma ferramenta de visualização.

Do ponto de vista estrutural, em framework é composto por dois tipos de componentes [Wirf 90][Jonson 91]:

- ***Abstratos***: Representam as abstrações principais do domínio de aplicação. São a raiz de uma hierarquia de subclasses que implementam variantes específicas da abstração. Definem o protocolo geral ao qual respondem os componentes concretos que implementam a abstração. Também definem o fluxo de controle genérico de qualquer aplicação do domínio.
- ***Concretos***: Representam diferentes materializações dos componentes abstratos. Definem uma biblioteca de comportamentos específicos que podem ser trocados para implementar diferentes aplicações. Podem acrescentar novos métodos correspondentes à variante particular do componente abstrato que implementam.

O comportamento dos componentes é definido pelos métodos que eles fornecem. Estes métodos são categorizados de acordo com o papel que cumprem dentro do framework [Gamma 94]:

- ***Abstratos***: Definem a interface de uma operação, mas não fornece uma implementação. Deve ser redefinido em subclasses para implementar variantes específicas do comportamento esperado do método. O seu papel é identificar operações comuns para qualquer aplicação do domínio; sua implementação varia para cada aplicação específica.
- ***Base***: Fornecem uma implementação completa que *não deveria ser redefinida*. Implementam comportamento comum a qualquer aplicação do domínio.
- ***Hooks***: Fornecem uma implementação *default* que *poderia ser redefinida* por alguma subclasse. Implementam comportamento de utilidade que em certos casos precisa ser estendido. Isto é comum na seqüência de inicialização de componentes.
- ***Templates***: Implementam algoritmos genéricos que invocam pelo menos um método abstrato. O seu comportamento varia em função da implementação dos métodos abstratos que provê a classe na qual é invocado. Estes métodos implementam a estrutura de controle genérica de um framework, ou seja, eles representam o comportamento genérico do domínio de aplicação de aplicação do framework.

Num primeiro nível de abstração, uma visualização adequada para compreender um framework deve identificar os componentes abstratos, as categorias dos seus métodos e a forma em que colaboram para implementar a estrutura de controle genérica. A visualização destes aspectos oferece uma primeira aproximação para compreender como o framework é organizado, como se relacionam os seus componentes, quais deles devem ser especializados e quais métodos implementados.

Cada classe abstrata é a raiz de uma hierarquia de subclasses concretas, que fornecem a implementação dos serviços definidos, mas não implementados, nas classes abstratas. Freqüentemente estas subclasses implementam novos serviços que utilizam os serviços implementados nas superclasses.

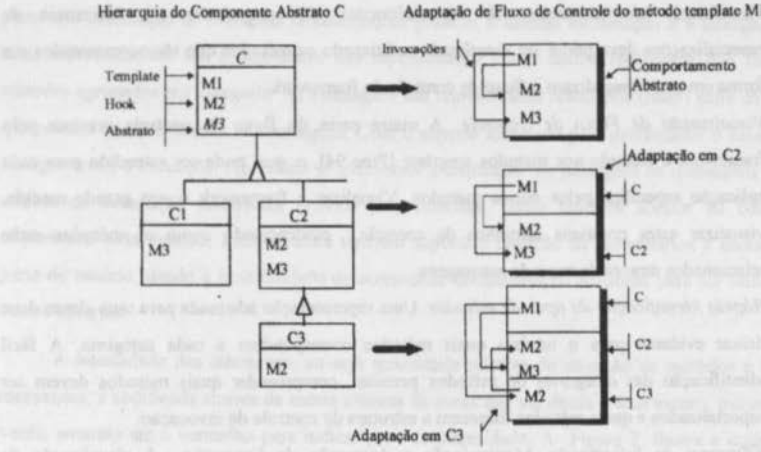

Figura 1. Adaptação do Fluxo de Controle de um Método Template

Isto conduz a complexas seqüências de passagem de mensagens que envolvem métodos implementados por varias subclasses em diferentes níveis da mesma hierarquia. Por exemplo, a Figura 1 ilustra a adaptação do fluxo de controle definido por um método *template* **M1**, de uma classe abstrata **C**. O método **M1** é um método template que invoca o método *hook* **M2** e o método abstrato **M3**. A subclasse concreta **C2** fornece uma implementação de **M3** e especializa **M2**. Nesta subclasse, o comportamento de **M1** é ajustado em função da implementação destes dois métodos. A subclasse **C3** redefine o método *hook* **M2**, o qual invoca a implementação do mesmo método definida na superclasse **C2**. Se o método **M1** é executado por uma instância de **C3** o seu comportamento final será diferente do comportamento correspondente para uma instância de **C2**.

Compreender este fluxo de controle é difícil através da análise do código dos métodos. Por este motivo, também é necessário visualizar, em alguns casos, a hierarquia de especializações enfatizando os métodos que são acrescentados e a forma na que especializam o fluxo de controle do framework.

2.2 Requisitos Básicos

Tomando em consideração os aspectos citados acima, uma ferramenta visual que auxilie no processo de compreensão de um framework deverá satisfazer os requisitos seguintes:

1. *Visualização de Componentes Abstratos*: A visualização deve permitir identificar rapidamente os componentes essenciais da arquitetura, sua interface e a forma em que eles colaboram.

2. *Visualização da hierarquia de especializações de componentes*: A hierarquia de especializações deve poder ser visualizada enfatizando os métodos que são acrescentados e a forma em que especializam o fluxo de controle do framework.
3. *Visualização de Fluxo de Controle:* A maior parte do fluxo de controle previsto pelo framework é baseado nos métodos *template* [Pree 94], o qual pode ser estendido para cada aplicação específica pelos outros métodos. Visualizar o framework é, em grande medida, visualizar estes possíveis caminhos de controle, evidenciando como os métodos estão relacionados através de troca de mensagens.
4. *Rápida identificação de tipos de métodos*: Uma representação adequada para uma classe deve deixar evidente para o usuário quais métodos correspondem a cada categoria. A fácil identificação das categorias de métodos permite compreender quais métodos devem ser especializados e quais métodos fornecem a estrutura de controle de invocação.
5. *Filtragem de Informação, Manipulação e Animação da Execução*: A visualização da arquitetura deve ser complementada com técnicas que permitam ao usuário interagir e controlar o processo de análise. A complexidade das interações envolvidas num exemplo podem dificultar a visualização do comportamento de partes específicas do framework como também a compreensão da seqüência de controle. Por esta causa, uma ferramenta deve fornecer mecanismos para filtrar informação não desejada e realizar o seguimento passo a passo das interações entre os componentes da arquitetura tanto em forma automática como manual.
6. *Transparência das Aplicações*: A ferramenta deve suportar a análise de qualquer aplicação minimizando a intervenção do usuário e a intrusão no código das aplicações a serem analisadas.

3. Um protótipo de Visualização

Com base nos requisitos citados acima, foi desenvolvido MOViE, um protótipo Smalltalk que suporta a análise interativa de um framework através de uma representação visual da arquitetura. Esta representação é gerada incrementalmente por *introspeção*[1] da execução de aplicações desenvolvidas com um framework. A informação necessária para gerar a representação é obtida através da utilização de uma infra-estrutura de meta-objetos cuja função é descrita na seção 4.

A informação da execução é organizada utilizando um gerenciador de hiperdocumentos que

[1] Introspeção é a capacidade de um programa para inspecionar o estado dos objetos que definem a sua computação sem produzir modificações nesse estado [Foote 93].

permite a associação de múltiplas representações gráficas à mesma informação e a navegação entre estas representações. Os componentes são representados como nodos compostos que contém os métodos agrupados por categoria. As mensagens são representadas como elos (*links*) entre os métodos que enviam e implementam as mensagens. Com o suporte fornecido pelo gerenciador o usuário pode *navegar* sobre a execução, *reproduzir graficamente* a seqüência de passagem de mensagens desejada através da navegação automática provida pela interface, como também acessar ao código que implementa cada método. Esta estrutura também suporta a inserção de comentários e anotações por parte do usuário, dando a possibilidade de acrescentar documentação adicional para ser utilizada por outros usuários.

A intensidade das interações, ou seja quantidade relativa de ativação de métodos e envio de mensagens, é codificada através da escala clássica de cores que vai desde o azul escuro, passando pelo verde, amarelo até o vermelho para indicar grau de intensidade. A Figura 2. ilustra a utilização de MOViE para analisar um editor produzido com o framework para editores gráficos.

3.1 Visualização da Arquitetura Abstrata

Em função dos requisitos enunciados na seção anterior, a visualização da arquitetura foi projetada para destacar num alto nível de abstração os aspectos essenciais de um framework, ou seja, componentes abstratos, categorias de métodos e fluxo de controle de classes e instâncias.

3.1.1 Visualização de componentes abstratos e categorias de métodos

Uma classe é caraterizada por três partes: uma parte abstrata que agrupa os métodos abstratos; uma parte genérica que agrupa os métodos template e hook; e finalmente a parte que define o comportamento base. Esta representação permite identificar rapidamente quais métodos devem ser implementados por subclasses e quais os métodos que definem a estrutura de controle genérica. A representação também permite visualizar em conjunto o comportamento da classe (definido pelos métodos de classe) e o comportamento das instâncias (definido pelos métodos de instância). Os métodos de classe são mostrados sublinhados para diferenciá-los visualmente.

O componente *Figure* (Figura 2) ilustra a divisão dos três tipos de métodos: *displayOn:* é um método abstrato, *new* é um método de classe do tipo *template/hook* e *initialize* é um método básico. A cor com a qual é mostrado cada método indica a quantidade relativa de vezes que o método foi ativado. Por exemplo, no componente *DrawingView*, a cor vermelha de *step* indica que ele foi um método muito ativado, enquanto a cor roxa de *new* indica que o método foi pouco ativado. Num nível

que o método foi pouco ativado. Num nível intermediário a cor azul clara do método *preferredBounds* indica que o método foi mais ativado que *new* mas muito menos que *step:*.

A informação transmitida pela cor dos métodos de criação (*new* neste caso) serve para dar uma idéia aproximada da quantidade relativa de instâncias de cada classe que são criadas durante a execução de um exemplo. Esta informação é importante para compreender quais classes representam componentes implementados por uma única instância e quais representam componentes que representam múltiplas instâncias.

3.1.2 Visualização da hierarquia de componentes

A visualização dos componentes concretos é um requisito adicional para compreender o mecanismo de especialização do framework. Cada classe define um componente que agrupa visualmente a suas subclasses diretas. A apresentação de cada subclasse mostra somente aqueles métodos próprios já que os herdados são mostrados pela representação da sua superclasse. Deste modo é possível visualizar simultaneamente o comportamento genérico do framework, o comportamento próprio das especializações e o fluxo de controle dentro da hierarquia[2].

A Figura 3 ilustra a hierarquia do componente abstrato Figure (num estado da execução do exemplo), na qual aparece o comportamento próprio da classe e um sub-componente *ContainerFigure*. Este, por sua vez, tem um sub-componente *LayeredContainerFigure* do qual herdam *Drawing* e *ConstraintDrawing*. Também é possível observar o fluxo de controle interno da hierarquia, o qual indica mensagens enviados a *self* ou *super* (em terminologia Smalltalk).

3.1.3 Visualização do fluxo de controle

O fluxo de mensagens *inter* e *intra* componentes é representado por setas que unem o método que envia a mensagem e o método ativado pela mensagem. A cor da seta indica a quantidade relativa de vezes que a mensagem foi enviada. Esta informação, combinada com a fornecida pela cor dos métodos, permite visualizar rapidamente quando um método cicla enviando a mesma mensagem a um componente. A diferenciação entre métodos de classe e instância é importante para identificar a seqüência de criação de instâncias de cada componente. A criação de instâncias geralmente envolve seqüências de inicialização que devem ser respeitadas pelas subclasses.

A informação das mensagens, combinada com os tipos de métodos, permite condensar na apresentação do componente abstrato uma grande quantidade de informação, relativa à adaptação de

[2] Esta estratégia de visualização pode ser modificada para mostrar todos os métodos definidos em cada componente, caso seja isto necessário.

métodos *hook* e à forma em que as instâncias são organizadas. No caso do método abstrato *displayOn:* mostrado na Figura 3, a seta recursiva indica que subclasses do componente abstrato estão enviando a mesma mensagem a componentes do mesmo tipo. Isto é uma forte sugestão de um objeto composto recursivamente por objetos pertencentes a uma hierarquia comum. Entretanto, a seta recursiva no método *new* indica que subclasses estão redefinindo o método e invocando o método herdado, típico comportamento de métodos *hook*.

3.2 Análise Detalhado da Estrutura de Controle

A representação gráfica apresenta o problema de saturação visual de elos, quando as interações são numerosas, porém é de utilidade para visualizar a complexidade global da arquitetura e os pontos nos quais existe maior densidade de troca de mensagens. Assim, para realizar a análise detalhada das interações a ferramenta provê mecanismos para filtrar informação e visualizar só aqueles aspectos de interesse para o usuário.

A representação de rede hipermedia utilizada para o modelo de execução permite suportar diferentes alternativas de visualização e a navegação da estrutura de controle. Cada passo de controle no framework é representado como um elo que une o método que envia a mensagem com o método destino. A natureza bidirecional dos elos permite reproduzir a seqüência de passagem de mensagens partindo de um método determinado. Esta funcionalidade é de grande utilidade para determinar qual foi o fluxo de mensagens que determinou a ativação de um dado método. Além disso, é possível acessar o código que implementa um método e a todas suas redefinições na hierarquia, permitindo assim determinar *por que* um método foi ativado.

A Figura 4 apresenta um exemplo das capacidades fornecidas pela interface para visualizar as mensagens recebidas e enviadas por um método. O usuário pode solicitar visualizar quais são as mensagens que ativam um método e realizar o seguimento destas mensagens em sentido inverso. Também é possível visualizar as mensagens enviadas por um método e realizar o seguimento normal do fluxo das mensagens. Isto permite realizar uma análise detalhada de como os componentes interagem, focalizando a atenção nos pontos de maior interesse.

Com a visualização utilizada pela ferramenta a navegação é realizada dentro do mesmo diagrama, mas se outras representações visuais fossem utilizadas (visões separadas para cada classe por exemplo) o acesso aos métodos invocados poderia envolver a navegação entre diferentes visões.

3.2.1 Animação arquitetônica

A seqüência de envio de mensagens é importante para compreender a dinâmica das interações, e fundamentalmente os caminhos de controle codificados pelo framework. Para isto, a interface com o usuário suporta a animação da seqüência de passagem de mensagens. O seguimento pode ser global, ou seja, o seguimento de toda execução, ou local mostrando só a seqüência de mensagens enviadas por um método determinado. Nesta animação o cursor se desloca na tela indicando o método invocado.

A Figura 5 apresenta uma imagem da ferramenta na qual o usuário está realizando a animação da seqüência de mensagens do exemplo da Figura 4. Nesse ponto da execução a classe *ContainerFigure* é marcada como a receptora da mensagem *extent* (o qual é um método abstrato definido por *Figure*) e o usuário está visualizando a especialização do método ativado na classe receptora. Deste modo é possível visualizar simultaneamente a estrutura de controle genérica codificada pelo framework e a as especializações do comportamento abstrato. Também é possível visualizar quando métodos específicos de classes concretas invocam a serviços definidos pelo framework, como é o caso do método *noSelections* da classe *Drawing*. Este método invoca o método template *displayBox* implementado pela superclasse abstrata *Figure*. Este método template invoca, por sua vez, o método *extent*, sendo executada a redefinição implementada na superclasse *ContainerFigure*.

A representação de passos de controle, e não de mensagens, permite realizar o seguimento do comportamento da arquitetura sem redundância. Cada passo é visualizado só uma vez permitindo ao usuário concentrar-se no funcionamento global da arquitetura, já que evita o seguimento de ciclos desnecessários (os quais podem se deduzir do código de cores).

A visualização única dos métodos de uma hierarquia em conjunto com a capacidade de animação facilita o reconhecimento visual do fluxo de controle próprio do framework. A seqüência de mensagens que tem um caminho de controle comum permanece inalterada na tela, enquanto se animam os caminhos de controle que cada especialização implementa.

4. Recuperação da Estrutura de Frameworks

Uns dos problema centrais na visualização de comportamento dinâmico é como coletar a informação relevante a ser visualizada. Esta informação habitualmente é obtida anotando o código da aplicação para anunciar eventos de interesse (envio de mensagens, criação de instâncias, etc.), os quais são interpretados pelo sistema de visualização [De Paw 93] [Stasko 94]. Estes mecanismos de instrumentação, porém, requerem a modificação do código da aplicação e informação estática

relevante (herança ou redefinição de métodos, por exemplo) deve ser recuperada através de outras técnicas.

Uma alternativa para evitar estas limitações é interceptar o fluxo normal das mensagens entre objetos e derivar o controle para outros objetos encarregados de monitorar a execução. Este objetos, denominados *meta-objetos* [Maes 87], monitoram a execução do programa e geram a representação que é utilizada pelo sistema de visualização. Esta capacidade para alterar o fluxo normal de execução é uma forma de comportamento reflexivo, na qual os objetos *reflitem* seu comportamento nos meta-objetos. A implementação de meta-objetos depende em grande medida das facilidades providas pela linguagem de programação. No caso de Smalltalk, um suporte para meta-objetos pode ser implementado com relativo pouco esforço devido a que, tanto classes como métodos, são representados como objetos normais dentro do ambiente. Em linguagens estáticas como C++, meta-objetos podem ser implementados através de um pre-processador que altere o código das classes para derivar as mensagens aos meta-objetos especificados [Chiba 93].

Os meta-objetos podem ser associados com qualquer objeto da aplicação de forma não intrusiva, permitindo assim analisar o comportamento dinâmico de qualquer aplicação. No caso da análise de frameworks, é necessário capturar o comportamento dos componentes da arquitetura, ou seja classes e suas especializações. Por isto os meta-objetos são associados com as classes, isto é, todas as instâncias de uma classe reflitem no mesmo meta-objeto.

MOViE opera num processo incremental composto de duas fases. Primeiro as classes que serão inspecionadas são refletidas sobre um conjunto predeterminado de meta-objetos. Depois deste processo, a aplicação é executada e as interações (determinadas pelo fluxo de mensagens entre objetos) detectadas pelos meta-objetos são mostradas graficamente para o usuário. Novas classes podem ser refletidas para obter uma visão mais completa das interações entre classes, e o processo é repetido pelo analista.

MOViE utiliza um conjunto de meta-objetos projetados para trabalhar juntos na análise das mensagens recebidas por instâncias e classes. Estes meta-objetos são encarregados de reconhecer os tipos de métodos e de manter a contagem de invocações de cada um deles. Também, eles são encarregados de determinar qual classe implementa cada método invocado e de construir a representação da hierarquia de classes dos componentes abstratos.

Esta solução tem a vantagem de permitir estender e especializar os meta-objetos para acrescentar novas funções de análise com mínimo esforço. Alem disso, é possível controlar interativamente que função especifica deseja-se analisar. A ferramenta permite ao usuário ativar e desativar interativamente

a reflexão nos meta-objetos com o objetivo de visualizar só as classes envolvidas numa função específica. Por exemplo, é possível só visualizar as interações envolvidas na criação de uma figura com o editor do exemplo, ou visualizar as interações envolvidas na seleção de um ícone da paleta de interface.

Para associar os meta-objetos cada classe a ser refletida é submetida a um processo que troca os métodos compilados por métodos que transferem o controle para um objeto gerenciador de meta-objetos. O gerenciador é o encarregado de derivar as mensagens ao(s) meta-objeto(s) associados com o receptor e estes são os encarregados de executar o método original. Este mecanismo é suportado por um framework para meta-objetos desenvolvido como resultado do trabalho na ferramenta [Campo 95]. Este framework provê o suporte para a implementação de diferentes estilos de gerência de meta-objetos como por exemplo, múltiplos meta-objetos associados com instâncias ou classes, prioridades de ativação, etc.

5. Trabalho Relacionado

Nos últimos anos a utilização de técnicas visuais começou a captar o interesse dos pesquisadores da área de orientação a objetos. Múltiplas representações gráficas foram desenvolvidas para diferentes metodologias propostas [Wirf 90][Booch 91][Rumbaugh 91][Jacobson 92], *tracers* gráficos [Bocker 90], linguagens visuais [Gutfreund 90][Hirakawa 90] e ferramentas visuais para composição de aplicações [de Mey 92]. Contudo, a utilização de tecnologia visual tem sido pouco explorada para apoiar no processo de reutilização de frameworks.

DePaw, Helm, Kimelman e Vlissides [De Paw 93] desenvolveram técnicas para a visualização de comportamento dinâmico de programas C++, através de múltiplas visões que apresentam informação resumida dos resultados de uma execução. As visões são representadas como matrizes cujas entradas são preenchidas com cores para visualizar a freqüência de criação e destruição de instâncias, invocações *inter* e *intra* classes, historia de atribuição de instâncias, etc. A informação por elas aportadas é muito valiosa para compreender o funcionamento global de um programa, mas não evidenciam com clareza os componentes abstratos, nem o fluxo de controle entre estes componentes, nem como eles são especializados. Também não suportam a interação do usuário durante o processo de análise, dando uma visão *post-mortem* da execução de uma aplicação.

VizBug++ é um sistema de visualização para depuração de programas C++ [Stasko 94] que integra visões diagramáticas da hierarquia de classes, instâncias e fluxo de controle. VizBug++ se baseia na interpretação de eventos para gerar uma animação da execução com setas que representam a invocação de funções e métodos. O sistema suporta funções de navegação semelhantes às descritas

neste artigo, mas é pensado para análise de programas e não de frameworks, razão pela qual não enfatiza os aspectos próprios dos frameworks discutidos acima.

6. Conclusões e Trabalho Futuro

Nas seções precedentes se descreveram sinteticamente as caraterísticas principais de uma ferramenta pensada para apoiar na compreensão da estrutura e funcionamento de frameworks orientados a objetos. MOViE é atualmente um protótipo que tem demonstrado o grande potencial que a combinação de técnicas visuais, navegacionais e reflexivas oferecem para a construção de ferramentas para a compreensão tanto de frameworks quanto programas específicos. Uma primeira versão foi utilizada com sucesso pelos autores para compreender e especializar, em dois dias, o framework do compilador Smalltalk, para interceptar os acessos a variáveis de instância. Atualmente, MOViE está sendo utilizado por programadores com pouca experiência em Smalltalk para compreender a biblioteca de classes do ambiente. Utilizando a ferramenta eles foram capazes de utilizar e especializar rapidamente classes complexas, como por exemplo a biblioteca de conexão como processos externos. Neste sentido MOViE poderia ser um excelente veículo para o ensino de como programas orientados a objetos são estruturados e como suas classes colaboram entre si.

A utilização de meta-objetos facilita a implementação de um processo de análise iterativo no qual o usuário pode começar analisando um conjunto reduzido de classes e avançar gradualmente na compreensão de aplicação completa. Também os meta-objetos permitem focalizar a análise em grupos específicos de classes que implementam funções determinadas do framework. No estado atual do desenvolvimento de MOViE, entretanto, ainda falta uma interface visual que suporte a identificação das classes a ser refletidas. Este processo atualmente deve ser realizado textualmente pelo usuário desde o ambiente.

Apesar de ter sido inicialmente pensada para compreender frameworks, MOViE apresenta muitas possibilidades de ser estendida para suportar funcionalidades de desenvolvimento e manutenção de programas. A representação da arquitetura como um hiperdocumento facilita a extensão da ferramenta para suportar funções de documentação da arquitetura, como também a introdução de anotações ou comentários acerca do funcionamento da mesma. O suporte reflexivo pode ser estendido, por exemplo, para implementar métricas de avaliação de qualidade de projeto e de reutilização. As técnicas visuais são de grande utilidade para a instrumentação e sintonia de programas, permitindo identificar rapidamente os pontos nos quais se concentra a maior densidade de colaborações entre classes. Atualmente está sendo estudada a utilização de técnicas de visualização 3D como um meio de diminuir a complexidade visual da representação gráfica. A utilização de graficas 3D oferece uma alternativa de grande interesse para suportar a navegação dentro de espaços virtuais que representem a

arquitetura do framework, como também para condensar numa mesma apresentação visões estáticas e dinâmicas da arquitetura.

7. Referências

[Booch 91] Booch G. Object-Oriented Design with Aplications, The Benjamin/Cummings Publishing Co., Reading, Mass., 1991.

[Bocker 90] Bocker, H.; Herczeg, J. *What tracers are made of*, Proceedings of the ECOOP/OOPSLA'90 Conference, Ottawa, Canada, October 1990.

[Burh 92] Buhr R.; Casselman R. *Architectures with Pictures*, Proceedings OOPSLA'92, Vancouver, Canadá, 1992.

[Campo 95] Campo, M.; Price, R.T. *Meta-Object Support for Framework Understanding Tools*, submitted to the ECOOP'95 Workshop on Reflection and Meta-Object Protocol., May 1995.

[Chiba 93b] Chiba, S.; Masuda, T. *Designing an Extensible Distributed Language with a Meta-Level Architecture*. Proceedings of the ECOOP'93 Conference, July 1993.

[De Paw 93] De Paw, W.; Helm, R.; Kimelman, D.; Vlissides, J. Visualizing the Behavior of Object-Oriented Programms, Procs. OOPSLA'93, Washington, D.C., October 1993.

[Deutsch 89] Deutsch P. *Frameworks and reuse in the Smalltalk-80 system*, In: Software Reusability Vol II: Systems and Applications, ACM Press, 1989.

[Foote 93] Foote, B. *Architectural Balkanization in the Post-Linguistic Era*, OOPSLA '93 Workshop on Object-Oriented Reflection and Meta-level Architectures. Washington, D.C., October 1993.

[Gamma 94] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Micro-Architectures for Reusable Object-Oriented Design*, Addison-Wesley, 1994.

[Hirikawa 90] Hirikawa, M.; Tanaka, M.; Ichikawa, T. *An Iconic Programming System, HI-VISUAL*, IEEE Transactions on Software Engineering 16(10), October 1990.

[Jacobson 92] Jacobson, I.; Christeron, M.; Overgaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, ACM Press, 1992.

[Johnson 88] Johnson, R.; Foote, B. *Designing Reusable Classes*, Journal of Object Oriented Progamming, June/July 1988.

[Johnson 91] Johnson, R.;Russo, V. *Reusing Object Oriented Designs*, Univ. Illinois Tech Rep. UIUCDCS91-1696, 1991.

[Johnson 92] Johnson, R. Documenting Frameworks Using Patterns, Procs. OOPSLA'92, Vancouver, Canada,1992.

[Laffra 93] Laffra, C. *Advanced Techniques for Understanding, Profiling and Debugging OO Systems*, Workshop Report, OOPSLA'93. Washington DC, October 1993.

[Lajoie 94] Lajoie, R.; Keller, R. *Design and Reuse in Object-Oriented Frameworks: Patterns, Contracts, and Motifs in Concert*, Procs. of the 62nd Congress of the Association Canadienne Française pour l' Avancement des Sciences, Montreal, Canada, May 1994.

[Maes 87] Maes P.; *Concepts and Experiments in Computational Reflection*, OOPSLA '87 Proceedings ACM SIGPLAN Notices Vol. 22, Nr. 12, December 1987.

[de Mey 92] de Mey V. *VISTA Implementation*, ITHACA Report, ITHACA. CUI.92.E#4, Centre Universitaire d'Informatique,University of Geneva, December 1992.

[Pree 94] Pree. W. Meta-Patterns: Abstracting the Essentials of Object-Oriented Frameowrks, In: Object-Oriented Programming, Springer-Verlag, 1994. Proceedings of ECOOP '94.

[Rumbaugh 91] Rumbaugh, J.; *et al.* Object Oriented Modeling and Design, Prentice-Hall, 1991.

[Stasko 94] Stasko J.; Jerding, F. *Using Visualization to Foster Object-Oriented Program Understanding*. Georgia Institute of Technology, Tech. Rep. GIT-GVU-94-33, 1994.

[Gutfreund 90] Gutfreund, S. ManiplIcons in ThinkerToy, in Visual Programming Environments: Applciations and Issues, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[Wirf 90] Wirfs-Brooks R.; Johnson R. *Surveying Current Research in Object Design*, Communications of the ACM, September 1990, Vol 33, Nº 9.

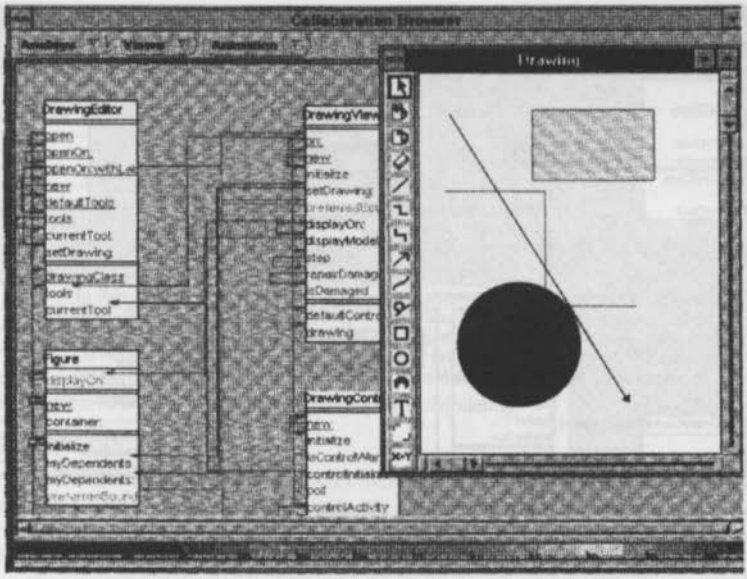

Figura 2. Análise de un editor construído com o framework HotDraw

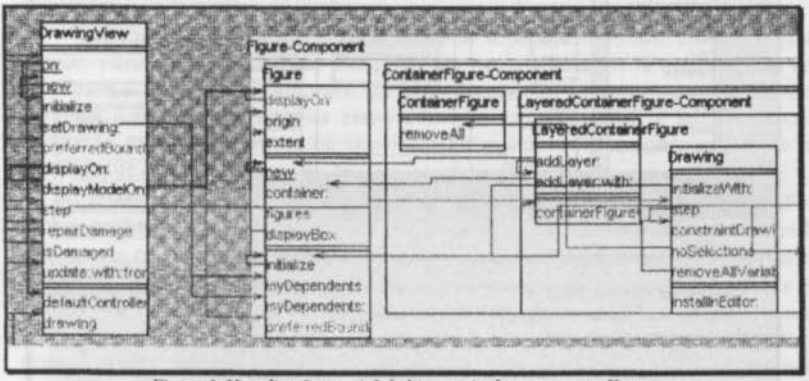

Figura 3. Visualização parcial da hierarquia do componente Figure

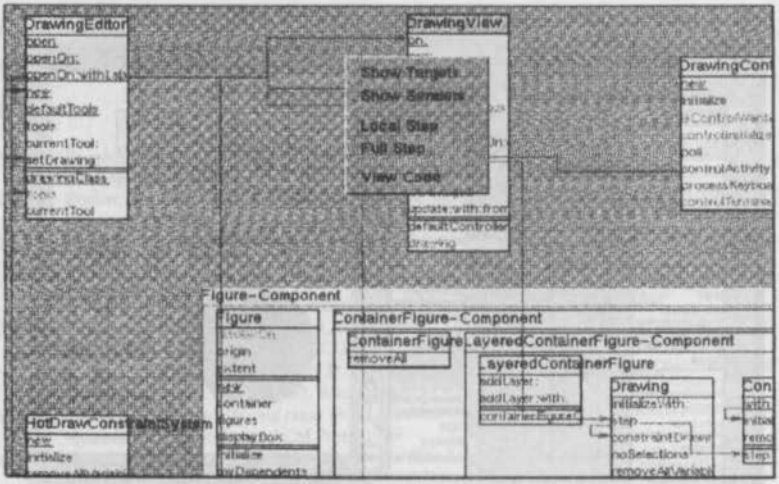

Figura 4. Funções de navegação partindo de um método

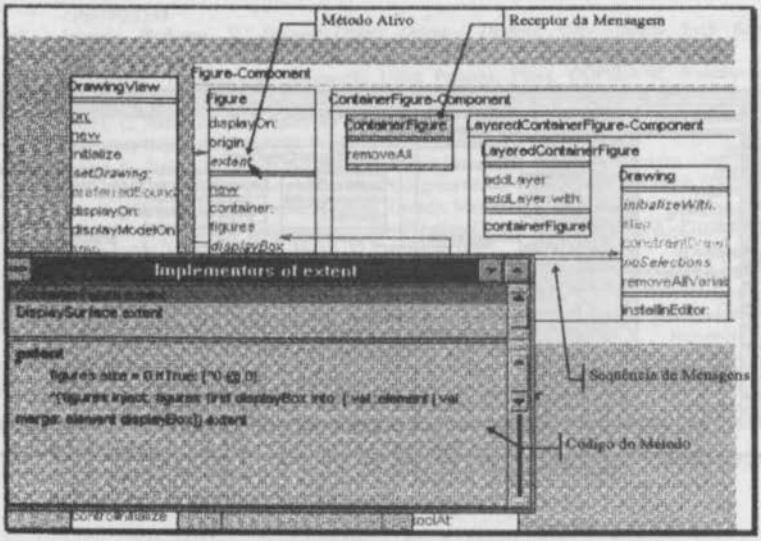

Figura 5. Exemplo de animação da arquitetura e inspeção de código