Um Meta-Ambiente para Suporte a Construção Sistemática de Frameworks

Sergio Crespo[•]**, Carlos José P. de Lucena, and A. v. Staa**
Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225, 22453-900 Rio de Janeiro, Brazil
{crespo, lucena, arndt}@inf.puc-rio.br

PUC-Rio.Inf.MCC 39/98 Dezembro, 1998

ABSTRACT

In this paper we describe an environment that supports a design method for object oriented software in WEB, combining Frameworks and Design Patterns. This environment uses the concept of viewpoints and the views-a relation in object-oriented design to guide the designer when identifying hot-spots in the structure of the framework. Case studies have shown that high levels of software reuse can be achieved through the use of object-oriented frameworks. It is possible to capture the common aspects revealed through the design experience acquired while designing a family of application. Although analysis and design methods such as design patterns, meta-object protocol, refactoring, class reorganization, behavior of modification for framework, have been proposed to support frameworks there are still problems related to framework creation. The use of an environment that gives support to the view-point concept and the flexibility relationships, hot-spot and of reuse by view-a, providing a promising approach for this kind of problem, and an extension for the method in its applicability in order to accomplish reuse in large-scale.

Key Words: Abstract Data Views, Viewpoints, Hot-spot, Frameworks, Design Paterns, Metadata, XML, Views-a.

RESUMO

Este artigo apresenta um ambiente que dá suporte a um método de *design* para software orientado a objeto baseado na WEB, que combina *frameworks* e *viewpoints*. O uso do conceito de viewpoints e das relações hot-spot e views-a permitem ao projetista uma maior facilidade na identificação dos pontos de flexibilização (*hot-spots*) e do núcleo da estrutura do *framework*. Pesquisas recentes têm mostrado que altos níveis de reutilização de software podem ser atingidos através do uso de *frameworks* orientados a objetos, capturando os aspectos comuns de *design* de uma família de aplicações. Outras técnicas tais como *design patterns*, protocolos para meta-objetos, *refactoring*, reorganização de classes e modificação do comportamento de frameworks têm sido propostas para dar suporte ao desenvolvimento de *frameworks*. O uso de um ambiente que dá suporte ao conceito de *viewpoint* e às relações de flexibilização *hot-spot* e de reutilização *Views-a* proporciona uma promissora abordagem para este tipo de problema e uma extensão ao método em sua aplicabilidade para realizar reutilização em grande escala.

Palavras chaves: Abstract Data Views, Viewpoints, Hot-spot, Frameworks, Design Paterns, Meta-dados, XML, Views-a.

[•] Sérgio Crespo é Professor Assistente II do departamento de informática do Centro 6 na Universidade do Vale do Rio dos Sinos - Unisinos-RS.

1. INTRODUÇÃO

Pesquisas anteriores têm mostrado que altos níveis de reuso de software pode ser alcançado através do uso de frameworks orientados a objetos. Um framework orientado a objeto captura os aspectos comuns de uma família de aplicações. Ele também captura a experiência do projetista durante o processo de construção da aplicação. Isto possibilita aos projetistas e programadores, reutilizarem suas experiências tanto em design como programação[18].

Pesquisas atuais reconhecem a necessidade para integrar frameworks e patterns, uma vez que design patterns também promovem flexibilização e reutilização. Um design pattern descreve um problema que ocorre diversas vezes em um dado domínio e sua solução, permitindo que ela seja utilizada várias e várias vezes sem a necessidade de refazê-la novamente. Desta forma, podem ser usados para implementar as variações associadas com cada hot spots. Porém, não está claro como identificar os hot spots de framework, e fazer a conexão entre a seleção do design patterns e o implementação destas variações[5].

Esta proposta apresenta um ambiente que dá suporte a um método de design para software orientado a objeto, combinando frameworks e viewpoints. O ambiente utiliza os conceitos de viewpoints, design patterns e das relação Hot-spot[25][26] e Views-a[2] [21] [22] [23] como princípios básicos para a construção de frameworks.

2. CONCEITOS BÁSICOS E VISÃO GERAL DE UM MÉTODO DE DESIGN DE APLICAÇÕES BASEADO EM FRAMEWORKS

Um framework é definido como um software genérico para um determinado domínio de aplicação, que oferece uma arquitetura semi-acabada que permite tanto a construção de módulos simples, como o de sub-sistemas que podem ser amplamente reutilizados[5][7].

2.1 MÉTODOS HOOK E TEMPLATE

Segundo Pree[8], frameworks unificam sistemas de software para um domínio específico e constituem em uma construção semi-acabada de blocos de códigos prontos para uso; em associação com uma arquitetura global, obtida pela, composição e interação entre os blocos. Os aspectos de um framework que não são projetados para adaptação (*frozen spots*) são representados por meio de métodos template. O método template é exemplifica adiante.

Reutilizar frameworks muitas vezes significa adaptar alguns componentes e ou blocos para uma necessidade específica sobrepondo alguns dos métodos de suas subclasses. No projeto de frameworks vários aspectos que não podem ser antecipados, devem ser formulados genericamente de forma a permitir fácil adaptação posterior. Essa adaptação pode ser obtida pela correta inserção de *hooks*, ou *métodos hooks* no framework. Um método hook, é um método abstrato que geralmente não possui uma implementação específica associada (é vazio) e pode ser visto como um método virtual em C^{++}. O único propósito deste método é proporcionar maior flexibilidade posteriormente.

Dois tipos de flexibilização são possíveis: por herança e por composição. Nos dois casos veremos o papel executado pelos hooks.

3. UM MÉTODO DE DESIGN DE APLICAÇÕES BASEADO EM FRAMEWORKS

Um framework é definido como um software genérico para um determinado domínio de aplicação, oferecendo uma arquitetura semi-acabada que permite tanto a construção de simples blocos, como o de sub-sistemas que podem ser amplamente reusados[5][7].

A abordagem para o design de framework [2] apresentado aqui está baseado na idéia de que qualquer design pode ser dividido em duas partes: o design do núcleo (*kernel*) do sub-sistema e o design do sub-sistema específico dos pontos de flexibilização.

O núcleo do sub-sistema é comum a todas as aplicações que são geradas a partir deste framework, e o sub-sistema específico descreve as diferentes características para cada aplicação que o framework suporta ao ser estendido.A figuras 1 e 2 ilustram o método de desenvolvimento baseado em viewpoints, destacando seus processos e artefatos utilizados.

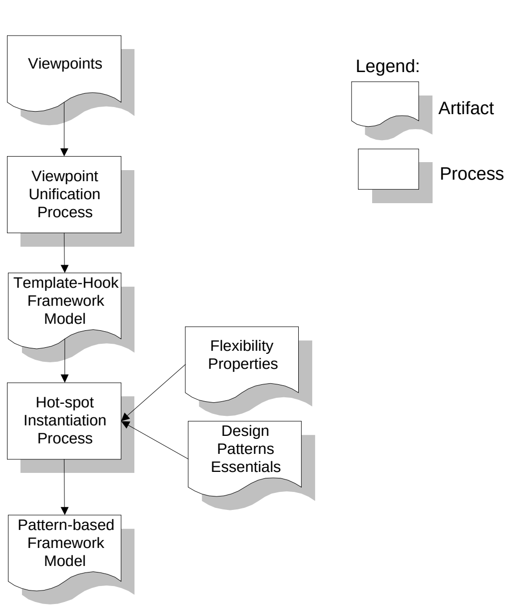

Figura 1. Processos do método de desenvolvimento baseado em Viewpoints.

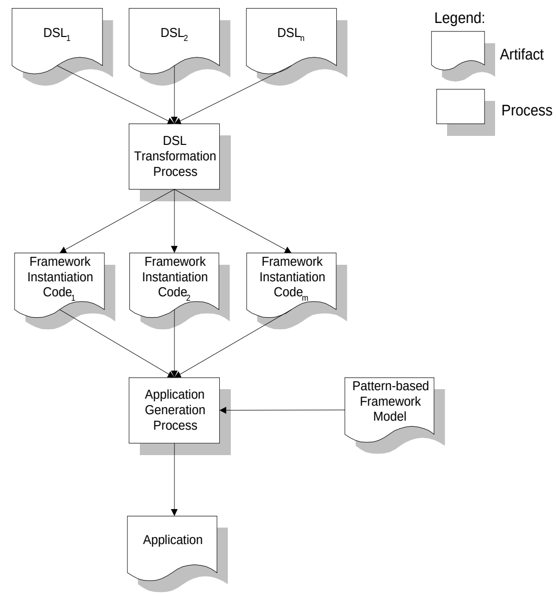

Figura 2. Processo de instanciação do framework

4. VIEWPOINTS

Segundo Filkelstein [12], um viewpoint pode ser compreendido como a perspectiva que um ator mantém no desenvolvimento de um processo combinando idéias, papéis, ou conhecimentos referentes a um determinado domínio. Isto nos permite que um problema possa ser visto por diferentes ângulos. Engenheiros de software reconhecem a necessidade de identificar e utilizar essa multiplicidade de visões na criação de um sistema de software[1] [2][6].

O primeiro artefato de entrada é um conjunto de diagramas que são desenvolvidos utilizando-se a abordagem de perspectivas. Uma perspectiva define os possíveis diferentes usos do framework. Desta forma, diferentes designs de sistemas associados às várias perspectivas serão produzidos.

Em nosso método um viewpoint é uma solução orientada a objetos completa. Por exemplo, vários sistemas geradores de relatórios poderiam ser vistos como diferentes viewpoints.

5. PROCESSO DE UNIFICAÇÃO DOS VIEWPOINTS

Uma vez que todos os viewpoints pertinentes estão definidos, a estrutura do núcleo do framework pode ser agora identificada, analisando-se as representações de todos os conjuntos de viewpoints para a identificação das estruturas e classes comuns a todos. Esta estrutura comum forma a unificação dos viewpoints. Essa unificação nos permite a obtenção das características comuns das classes e seus relacionamentos presentes nos vários conjuntos de viewpoints, assim como o núcleo do sub-sistema.

Os elementos que não estão presentes no núcleo são os únicos que variam e dependem do uso do framework. Esses elementos definem os pontos de flexibilização (hot-spots) do framework [8][11], que devem ser adaptados para a geração da aplicação. Cada hot-spot representa um aspecto de um framework que pode ter diferentes implementações para cada instanciação do mesmo.

Cada ponto de flexibilização representa um diferente aspecto do framework que pode resultar em várias aplicações dependendo de como o framework for instanciado. Um ponto de flexibilização pode também ser definido como o relacionamento entre um objeto específico da aplicação e o núcleo, podendo o objeto específico da aplicação utilizar os serviços do núcleo adaptando-os para uma visão particular de utilização deste serviço.

O processo de unificação não pode gerar automaticamente o design final do framework, pois os viewpoints representam aplicações concretas e quando unificados, produzem um framework que é uma abstração dos viewpoints, logo, por esta razão, a semântica de como definir as partes flexíveis do framework final não se encontram presentes neste design.

Um novo relacionamento em design orientados a objetos, chamado “relacionamento hot-spot”, é utilizado no processo de unificação para representar os relacionamentos entre o kernel e os objetos hot-spots. A semântica para esse novo relacionamento é obtida através dos design patterns essenciais. Isto implica que um relacionamento hot-spot é de fato um meta relacionamento que é implementado por um design pattern, que é gerado levando-se em conta as exigências de flexibilização do hot-spot[13][25][26].

Este processo impões algumas pré-condições sob os viewpoints:

- Os viewpoints necessitam ter consistência de nome, o que significa que classes com diferentes nomes representam diferentes conceitos e por esta razão não podem ser unificadas.
- As estruturas dos viewpoints devem ser consistentes, isto é, duas classes com os mesmos nomes e assinaturas não podem ter diferentes formas de relacionamentos em diferentes conjuntos de viewpoints.

Quando os viewpoints ficarem todos consistentes, eles poderão ser unificados seguindo as seguintes regras básicas:

1. P1) Para cada classe que pertence a um viewpoint, existe uma classe correspondente com o mesmo nome, dentro do modelo de framework baseado em hook[18].
2. P2) Se um método tem mesma assinatura e implementação em todos os viewpoints em que ele aparece, ele possui um método análogo no modelo de framework baseado em hook.
3. P3) Se um método existe em mais de um viewpoint com diferentes assinaturas, ele possui um método correspondente no modelo de framework baseado em hook, com o mesmo nome porém com assinatura e implementação indefinidas.
4. P4) Se um método existe em mais de um viewpoint com diferentes implementações, ele possui um método correspondente no modelo de framework baseado em hook, com o mesmo nome e assinatura, mas com implementação indefinida.
5. P5) Todos os métodos que utilizam métodos hook, são definidos através de modelos(templates). Existe sempre um relacionamento hot-spot entre um modelo de método e a classe que possui o respectivo método hook.
6. P6) Todo o relacionamento que não é do tipo hot-spot é preservado no modelo de framework baseado em hook.

O ambiente encontra-se em teste para as seguintes etapas:

- Gerência, onde se podem definir usuários, perspectivas e domínios;
- Manipulação de viewpoints, através da ferramenta “*ViewPoint-Text-Tool*” em que se pode: criar classes ou importá-las, editá-las, removê-las, definir seus relacionamentos e averiguar que classes estão sem qualquer relacionamento; e
- A unificação dos viewpoints.

6. MODELO DE FRAMEWORK BASEADO EM TEMPLATE-HOOK

A semântica deste modelo é dada pela análise dos conjuntos de viewpoints, ou seja, ocorre uma cópia de todos os requisitos especificados nos viewpoints originais para este novo modelo. Este novo modelo utiliza o relacionamento hot-spot para prover uma flexibilização que não foi prevista nos conjuntos iniciais de viewpoints.

7. PROCESSO DE INSTANCIAÇÃO DOS PONTOS DE FLEXIBILIZAÇÃO

Este processo usa os artefatos modelo de framework baseado em template-hook, cartões hot-spot e design patterns essenciais como entrada, gerando um modelo de framework baseado em patterns como saída. Ele é responsável por eliminar todos os relacionamentos tipo hot-spot do modelo de framework baseado em hook, substituindo-os por apropriados design patterns. A seleção do melhor pattern que implementa cada relacionamento hot-spot é guiada por um artefato chamado “Hot-spot cards”, que oferece uma maneira sistemática para selecionar o design pattern baseado nas propriedades de flexibilização presentes nos relacionamentos tipo hot-spot.

A figura 3 mostra o layout do artefato Hot-spot cards. Observe que duas propriedades são utilizadas aqui: Adaptação sem a necessidade de se realizar uma reinicialização do sistema e combinação recursiva.

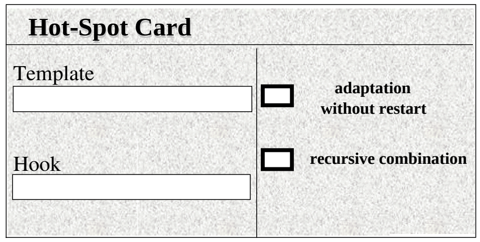

Figura 3. Hot-spot Card

A seguinte tabela mostra os relacionamentos entre as propriedades de flexibilização selecionadas do artefato Hot-spot cards e seus apropriados meta-pattern(patterns essenciais).

	Adaptação sem reinicialização	Combinação recursiva	Meta-pattern
1			Unification
2	☑		Separation
3		☑	Recursive Unification
4	☑	☑	Recursive Separation

Tabela 2. Relação entre propriedades e meta-patterns

8. O META-AMBIENTE

O ambiente encontra-se disponível para teste na seguinte URL:
http://rush.les.inf.puc-rio.br/crespo/qualify. Pode-se utilizar os Browsers da Microsoft ou da Netscape para interação.

A seguir será apresentado o meta-ambiente inicialmente introduzindo alguns conceitos básicos sobre em que termos o ambiente é compatível com plataformas IMS[15] através dos conceitos de meta-dados e da utilização de XML[14] para a representação destas meta-informações. Após esta introdução será apresentado alguns coneitos básicos de forma a deixar mais claro como se utiliza os mecanismos de flexibilização bem como exemplos. Na seção seguinte uma arquitetura do ambiente é apresentada focalizando os seus espaços e seus atores no processo de criação de frameworks. Segue-se, então, uma detalhada apresentação da ferramenta chamada “Viewpoint-Text-Tool” que possibilita toda a gerencia e unificação de viewpoints detalhando suas interfaces bem como suas estruturas de dados e posteriormente apresenta-se os passos necessários para a continuidade do ambiente e conclusões.

8.1 INTRODUÇÃO

Inicialmente vamos descrever dois conceitos utilizados no ambiente que nos permitirá uma grande taxa de reutilização de informação que são: meta-dados[4][16][19] e XML[14].

Estes dois conceitos foram propostos no projeto “EDUCON/NLII IMS” [15] que define uma arquitetura aberta para produção de material para Educação Baseada na WEB.

A plataforma IMS propõe uma série de especificações para tentar solucionar problemas de interoperabilidade e reutilização de informação[15][16] para o domínio da educação baseada na WEB. É proposta a utilização de meta-dados e o uso de XML para a representação desta meta-informação. A seguir vamos descrever o que são meta-dados e XML.

O ambiente a ser descrito, permite a possibilidade de se reutilizar classes definidas em outros viewpoints de forma a otimizar o processo de construção das perspectivas. Todo o domínio criado neste ambiente, possui um meta-dado que o descreve em termos de dados e comportamento. Uma vez definido o domínio da aplicação e seus usuários, pode-se iniciar a definição das perspectivas do framework. O ambiente permite o trabalho de forma compartilhada para construção de um determinado framework.

8.1.1 Meta-dados

Um meta-dado [4][16][19] é informação sobre informação, sobre recursos ou conteúdos. Uma meta-informação poderá ser muito relevante no processo de localização de recursos ou conteúdos. Em nosso caso estamos preocupados em buscas por conteúdo. Esse conteúdo será disponibilizado por um sistema gerado a partir de nosso Meta-Ambiente.

No projeto IMS a especificação de Meta-dados possui três partes essenciais:

Um dicionário de termos, tipos de recursos e um sistema para gerenciamento das especificações.

- *dicionário de termos* identifica os termos que constituem o meta-dado propriamente dito. Estes termos são utilizados como rótulos para a identificação dos recursos.
- Os *tipos de recursos*, descrevem os diferentes conjuntos de meta-dados, por exemplo: uma planilha possui uma descrição diferente do que a de um tutorial on-line.
- *sistema de gerenciamento de especificação*, oferece uma ferramenta de forma a obter um nível comum e estável nas especificações, de modo que se consiga uma linguagem comum entre contextos. Uma busca para ser efetiva necessita que tanto a pessoa que descreveu o recurso quanto a pessoa que procura por um recurso usem uma linguagem comum.

Os meta-dados são organizados [26] dentro de categorias ou campos. Cada categoria representa uma característica de um dado recurso, como, por exemplo, o título ou o resumo de um artigo.

A representação de meta-dados deverá utilizar um formato XML/RDF [17] para a especificação dos termos utilizados.

A localização do meta-dado poderá estar em dois locais: em um arquivo externo ou dentro do próprio arquivo HTML. A definição de sua localização dependerá do tipo de projeto a ser realizado. Em nosso caso, os meta-dados serão parte integrante de todas as classes e objetos.

8.1.2 XML

XML (eXtend Markup Language) [14] é uma linguagem de marcação para a definição de formatação de páginas HTML e a descrição de recursos.

XML é um subconjunto menos formal da especificação SGML. Esta linguagem de marcação nasceu por volta de novembro de 1996 durante a *SGML 96 Conference* em Boston, tendo sido formalizada em Julho de 1997.

XML retém os principais benefícios de SGML[http://www.sgmlsource.com/]. Com XML têm-se uma generalização no processo de descrição e um determinado autor pode agora criar suas próprias Tags, permitindo que seus documentos sejam auto-descritivos e formatados.

XML hoje é muito utilizado para a definição de conteúdo de servidores PUSH na WEB, nas aplicações voltadas para comércio eletrônico e em livrarias eletrônicas para a localização de títulos ou volumes, etc.

Também está se estudando uma melhor forma de modelar os meta-dados do domínio da aplicação, de todos os vários tipos de patterns e das perspectivas, de forma a proporcionar uma maior reutilização dos componentes.

8.2 REUTILIZAÇÃO DE COMPONENTES

A utilização de meta-dados juntamente com o uso de XML como formato padrão para representar o comportamento e funcionalidades de classes, nos permite a reutilização de vários de seus componentes. A adoção destes formatos leva em consideração os padrões já definidos no Projeto EDUCON/NL II IMS[15].

Podemos pensar em reutilização da seguinte forma:

Reutilização de conteúdo criado: Para cada classe instanciada em um framework para educaçào na WEB todo o conteúdo disponibilizado tanto por autores quanto por alunos segue o mesmo padrão. Ou seja, para cada conteúdo gerado, um meta-dado é especificado automaticamente em formato XML. Desta forma pode-se obter que todos os dados gerados sejam facilmente interoperáveis por qualquer servidor que utilize o padrão IMS.

Reutilização de especificação: Conforme foi dito anteriormente, toda a classe criada possui um meta-dado associado em formato XML que a descreve através de seus atributos, relacionamentos, métodos e operadores, dando suporte a representação de herança e especialização de classes e objetos. A utilização de meta-dados junto com a utilização da linguagem XML para representação de dados nos permite agora uma reutilização de especificações em 3 níveis:

Reutilização de classes: todas as classes geradas pelo ambiente proposto podem ser reutilizadas inter e intra domínios, perspectivas e viewpoints. É possível também que estas classes possam ser utilizadas por outros ambientes.
Reutilização de viewpoints: cada viewpoint, que é um conjunto de classes, possue seus meta-dados correspondente. Logo, de forma análoga, a reutilização neste caso não é de uma simples classe, mas de um ou vários viewpoints.
Reutilização de frameworks: como um framework é composto por vários viewpoints, pode-se também neste caso pensar em uma reutilização de mais alto nível, como a de frameworks.
A reutilização se torna mais intensiva dependendo de sua própria granularidade. Ou seja, a reutilização de uma simples classe é menos intenso do que o a de um viewpoint e este por sua vez menos efetivo do que a reutilização de um framework.

8.3 ARQUITETURA DO AMBIENTE

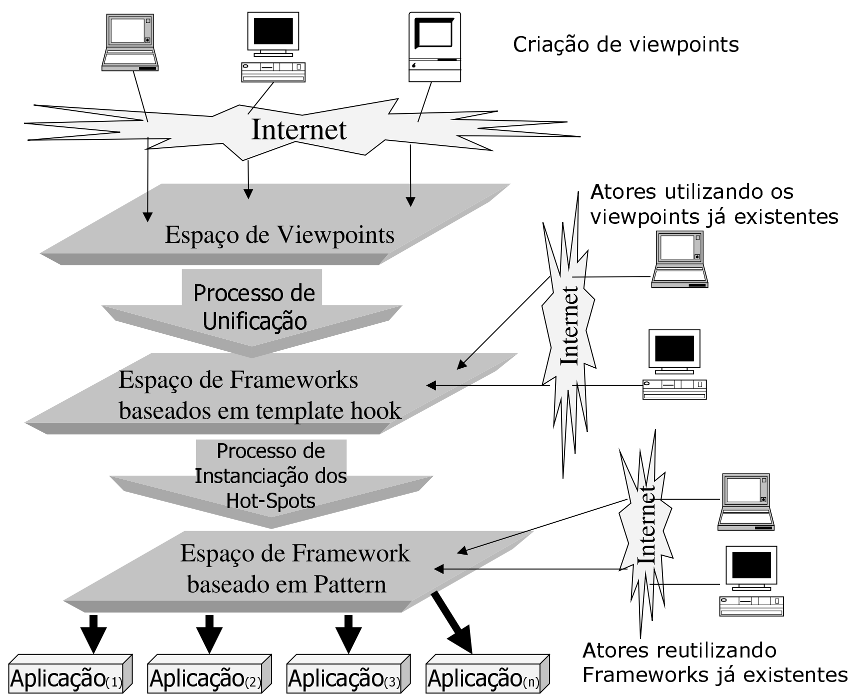

Figura 4. Arquitetura do Ambiente

A arquitetura do ambiente compreende três espaços principais:

- Espaço de Viewpoints
- Espaço de Frameworks baseados em template hook e
- Espaço de Framework baseado em Pattern.

Esses espaços são utilizados por diferentes atores que podem ter acesso ao sistema conforme esta representado na figura 4. A seguir vamos descrever os espaços e seus atores.

8.3.1 O espaço de viewpoints

Neste espaço são criadas todas as classes para um determinado domínio de aplicação. É utilizada uma ferramenta "Viewpoint Text Tool" para permitir que classes sejam criadas de maneira fácil, ou reutilizadas e permitir a definição de relacionamentos entre elas. Neste espaço podem estar atuando vários atores através de diferentes perspectivas através da WEB, utilizando um browser como sua interface. Note-se que o trabalho deste atores não é dependente de tempo ou qualquer sincronismo entre eles. O ambiente permite que o trabalho seja realizado de forma assíncrona. Os atores para obterem uma aplicação executável deverão ainda passar pelos outros dois espaços. A interação entre os atores neste espaço é bem forte, pois todas as classes criadas são compartilhadas por todos. Uma vez definidas as perspectivas os atores estão aptos a passarem para outro espaço, o *espaço de frameworks baseados em template hook* que é gerado pelo processo de unificação de viewpoints.

8.3.2 O espaço de frameworks baseados em template hook

Este espaço é resultante do processo de unificação dos viewpoints e nele podemos encontrar o kernel do framework e seus hot-spots. Esse espaço pode receber dois tipos de atores: os atores que definiram os conjuntos iniciais de viewpoints e um outro grupo de atores que não participou desta fase, mas pode aproveitar todo o trabalho realizado anteriormente. A partir deste ponto inicia-se a definição das propriedades de flexibilização, utilizando uma ferramenta chamada Hot-Spot Card Tool. O ambiente separa os contextos destes atores de forma a não permitir que atores interfiram no trabalho de outros atores. Podem-se definir vários tipos de propriedades de relacionamentos para a mesma relação. Isto provocará, no final, uma aplicação diferente para cada tipo de ator. Note-se que neste espaço, a reutilização agora possui uma granularidade maior do que no primeiro. Uma vez definidos todos os relacionamentos ocorre uma instanciação dos hot-spot gerando-se o próximo espaço chamado de *espaço de framework baseado em pattern*.

8.3.3 O espaço de framework baseado em Pattern

O *espaço de framework baseado em pattern*, é um espaço que pode contar agora, com três tipos de atores: os atores que são provenientes do primeiro espaço, atores provenientes do segundo espaço e atores que somente atuam no terceiro espaço de forma a gerar aplicações a partir do framework existente aqui e que está composto agora, não mais por relacionamentos tipo hot-spot mas por patterns apropriados que implementam tais relacionamentos. Aqui todos possuem um framework já flexibilizado, sendo possível ainda uma maior customização para a geração de vários tipos de aplicativos do framework inicial. O grau de reutilização (em grande escala) nesta etapa é bem maior do que nas anteriores, pois agora estamos reusando frameworks. O ambiente permite ainda que os diversos atores façam uma navegação procurando por classes, templates, relacionamentos e frameworks durante o processo de criação de suas aplicações.
Veremos adiante o ambiente propriamente dito. Não serão apresentados todas as facilidades acima descritas. Será apresentado o espaço de Viewpoint, como se dá o processo de unificação e como a ferramenta que dá suporte ao ambiente tratará os dados para as demais fases.
A seguir será apresentada a ferramenta que faz o gerenciament e unificação dos viewpoints, chamada Viewpoint-Text-Tool.

8.4 FERRAMENTA DE GERÊNCIA DE PERSPECTIVAS

Esta ferramenta permite o gerenciamento das classes que pertencem as diversas perspectivas do domínio da aplicação bem como a definição de seus relacionamentos. Ela apresenta as seguintes funcionalidades:

- Selecionar uma classe
- Criar uma nova classe
- Editar uma classe
- Remover uma classe
- Definir o tipo do relacionamento.
- Classes sem relacionamentos

No momento do Login do usuário, como pode ser visto na figura 5, um conjunto de informações torna-se persistente durante toda a interação com o ambiente. Essas informações são: identificação do domínio a que ele pertence, identificação das perspectiva a que ele pertence e a sua própria identificação como usuário. Note-se que para o usuário ter uma nova perspectiva de uso, ele deve ter uma outra identificação e senha.

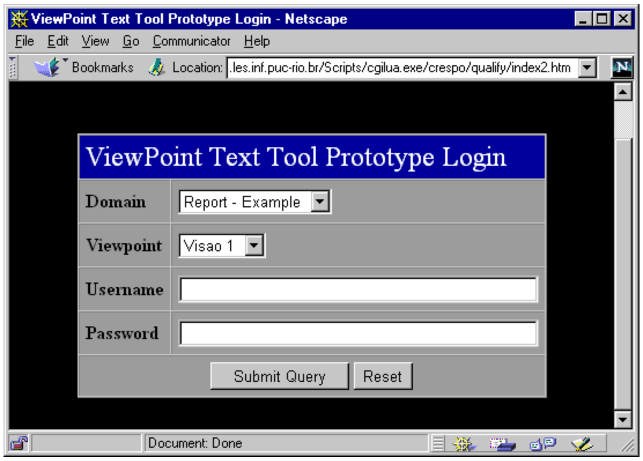

Figura 5. Viewpoint Text Tool

Uma vez feito o login ao ambiente, o usuário já identificado pode dar início às seguintes atividades: definir os relacionamentos entre as várias classe que fazem parte do domínio, editar as propriedades de uma classe, alterar os métodos de uma classe, variáveis e implementações, remover classes, criar uma nova classe em uma perspectiva e consultar por todas as classes que não possuem nenhum relacionamento dentro de uma perspectiva.

8.5 CRIAÇÃO DE UMA NOVA CLASSE

Durante o processo de criação de novas classes, o usuário deve definir: o nome da classe, uma descrição desta classe, suas variáveis de instâncias e os métodos que ela implementa e seu código fonte. A figura 6 nos mostra a estrutura de dados e classes desta fase. Como pode ser visto nesta figura, após o login do usuário, uma nova instância do browser é criada e dividida em 3 frames: menu, conteúdo e workplace.

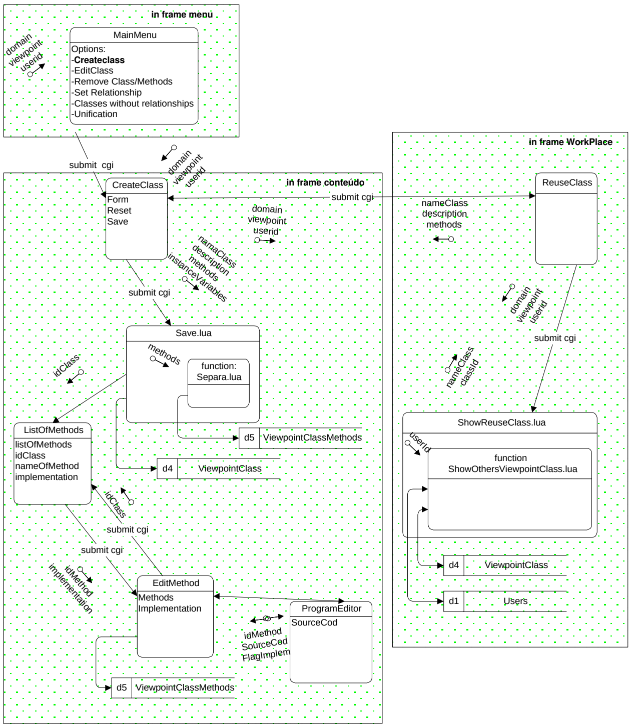

Figura 6. Diagrama de fluxo e classes da fase: Criando classes

Quando o usuário inicia o processo de criação de uma nova classe, é apresentado no frame a direita uma lista de todas as classes que pertencem ao domínio ao qual ele pertence. Porém estas classes não são necessariamente de sua autoria. São classes que foram definidas por outros atores que podem fazer parte de sua mesma perspectiva ou de outras perspectivas. Se a classe interessa ao ator ele simplesmente clica sobre o nome da classe e depois no botão select. Desta forma todos os dados desta classe serão repassados a este ator. O ator neste caso, poderá alterar os seguintes dados desta classe: variáveis de instâncias, métodos e a sua implementação. Quando se utilizam classes que estão presentes em outros viewpoints, deve-se preservar os relacionamentos que, por ventura, existam entre essas classes. Uma classe criada torna-se imediatamente torna-se disponível para todos os usuários que estejam trabalhando neste domínio, não importando se o usuário está remotamente conectado. Quando o usuário esta editando a implementação de uma classe que foi importada de outro viewpoint, o editor apresenta uma interface um pouco distinta de forma a que o autor, neste momento, informe ao ambiente se ele alterou a implementação. Esta informação é importante futuramente para o processo de flexibilização. O restante da especificação mais detalhada da interface desta ferramenta pode ser visto em [20].

8.6 VERIFICADOR DE INTEGRIDADE LÓGICA E DE IMPLEMENTAÇÃO

O ambiente que dá suporta ao método, encontra-se em fase de desenvolvimento e conforme pode-se observar na figura 1, os próximos passos serão o estabelecimento dos requisitos/propriedades de flexibilização, utilizando a ferramenta Hot-spot Tool de forma a encontrar o Design Pattern mais apropriado para cada relacionamento hot-spot. Para isso esta sendo definido uma base de dados para armazenamento e catalogação dos Design Patterns. Estes design patterns terão a princípio uma implementação padrão para cada tipo de pattern utilizando-se uma linguagem de alto nível, própria para o domínio da aplicação ou simplesmente Java. A primeira versão do ambiente gerará somente aplicações instanciadas do framework em linguagem Java. Porém, antes de se iniciar a geração do código propriamente dito, será realizado uma checagem de forma a se identificar se o que o usuário especificou inicialmente na definição das classes (atributos, métodos, implementação) tem correspondência com o que foi implementado nos métodos que necessitavam de flexibilização.

Implementar um módulo (VILI) que realize uma consistência sobre Design e implementação. Está consistência comparará se todos os métodos com suas respectivas assinaturas especificados durante o processo de criação das classes que fazem parte dos vários conjuntos de viewpoints possuem uma correspondência na implementação e vice-versa. Este módulo será utilizado após a etapa de unificação, mostrando ao usuário possíveis inconsistências entre a especificação e a implementação dos métodos.

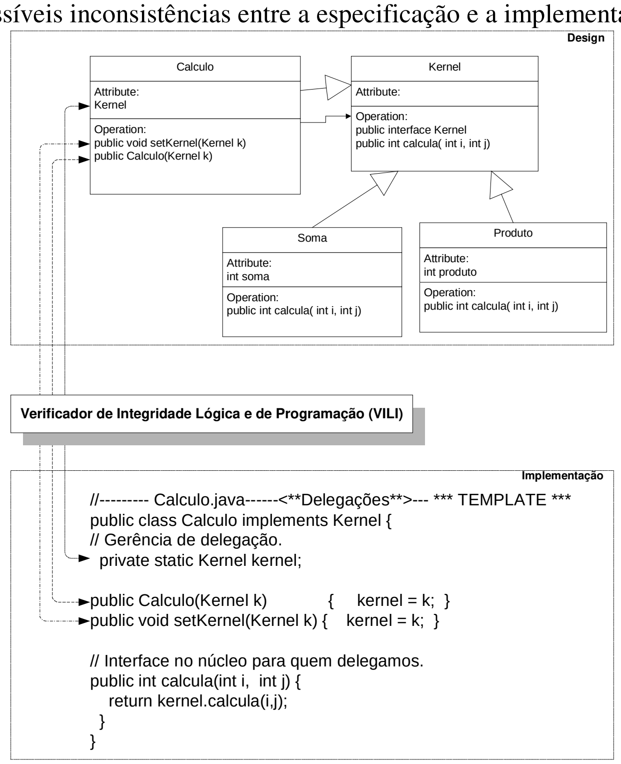

Figura 7. Verificação de inconsistência

Analisando a implementação, procura-se extrair as declarações de métodos com as suas respectivas assinaturas e compará-las com a correspondente especificação da classe que consta no banco de dados. Desta forma pode-se obter uma análise nas duas direções, identificando prováveis inconsistências nas implementações dos métodos.

Esta ferramenta será utilizada nas seguintes etapas:

Logo após a unificação já dito acima, e durante o processo de instanciação do framework. Nesta etapa utilizando-se uma DSL, o framework gerado poderá criar vários tipos de aplicações dependendo da forma com que seus hot-spots forem flexibilizados. Em virtude de se necessitar a participação do usuário nesta etapa, deve-se fazer uma checagem se o que ele esta implementado possui uma correta descrição em termos de design, pois não temos como garantir que o usuário em questão tenha participado de todas as etapas do processo de desenvolvimento. Este usuário poderia somente esta reutilizando um framework já gerado sem ter o conhecimento das interfaces de flexibilização dos métodos.

A ferramenta trabalha da seguinte forma: na base de dados, uma classe possui vários atributos e métodos e cada método tem uma implementação. A solução a ser adotada consiste em fazer uma checagem partindo da declaração de métodos que consta da base de dados (fase de Design) e procurar em todas as linhas da implementação por uma chamada deste método com a correspondente assinatura. Caso este método declarado na criação da classe não seja encontrado na implementação do próprio método, gera-se uma comunicação com o usuário informando em que classe e método esta inconsistência foi apurada.

8.7 UNIFICAÇÃO EM DIFERENTES PLATAFORMAS

Estamos analisando o suporte que o ambiente irá proporcionar para a unificação de viewpoints em diferentes plataformas, conforme pode ser visto na figura 8.

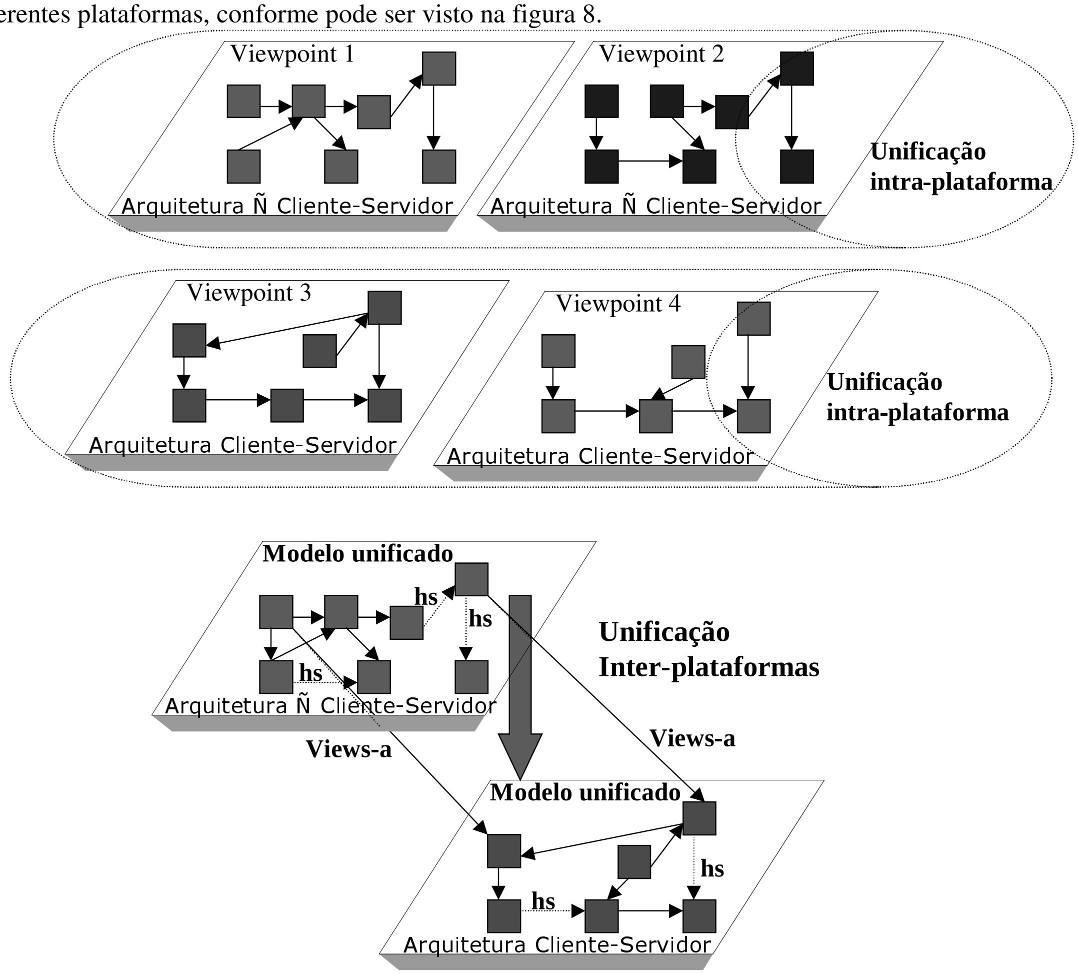

Figura 8. Unificação em várias plataformas

Para a unificação de plataformas diferentes, estamos utilizando o relacionamento Views-a [2][21] [22] [23] [24], que nos permitirá “colar” um Design Patterns para que as duas plataformas possam ser unificadas.
Após unificados todas as plataformas, inicia-se o processo de geração de código propriamente dito. Para este processo, será utilizada a máquina DRACO-PUC[9] para as transformações necessárias para a geração final do código executável. Como entrada para a máquina DRACO, o ambiente normalizará sua base de dados de forma a poder transformá-la em cláusulas e fatos PROLOG. Esta etapa do trabalho utiliza a tese de mestrado de Jing Dong , “A Transformational Process-Based Approach to Object-Oriented Design”, da Universidade de Waterloo – Canadá[10]. A abordagem desta tese, utiliza uma linguagem de processo que nos permite representar as modificações de modelos de objetos através do processo de design e, em particular, as mudanças induzidas pela aplicação de design patterns. O ponto forte da associação dos resultados da tese ao ambiente é que muitos dos Design Patterns do Gamma[5] já se encontram modelados em clausula e fatos PROLOG, bem como a notação de modelos de objetos. Também estamos desenvolvendo uma ferramenta de busca, que atuará de forma a poder-se construir frameworks utilizando-se pedaços de especificações já existentes tais como: classes, perspectivas e até mesmo de frameworks.

9. CONCLUSÃO

A utilização de viewpoints e da relação de flexibilização hot-spot, nos permite uma rápida e sistemática construção de frameworks. A identificação das classes comuns aos viewpoints formam o kernel e os pontos de flexibilização ficam facilmente visíveis. O método proposto utiliza um ambiente baseado na WEB que facilita a cooperação entre designers para a construção de frameworks, permitido-se que usuários localizados em locais remotos trabalharem de forma harmônica sem que isso impacte em suas especificações. O ambiente utiliza o conceito de meta-dados para armazenar informações sobre todas as suas classes e objetos em formato XML, tornando desta maneira toda a sua estrutura de armazenamento interoperável e compatível com o padrão IMS. Esta característica de armazenamento de classes e objetos permite que se tenha um alto grau de reutilização não só em termos de viewpoints e suas classes, mas também de conteúdo gerado após a aplicação já ter sido implementada em alguma linguagem de programação. É possível também realizar a reutilização de viewpoints e de frameworks, pois estes possuem também uma meta-informação em formato XML.
O ambiente encontra-se em teste para as seguintes etapas:

- Gerência, onde se podem definir usuários, perspectivas e domínios;
- Manipulação de viewpoints, através da ferramenta “*ViewPoint-Text-Tool*” em que se pode: criar classes ou importá-las, editá-las, removê-las, definir seus relacionamentos e averiguar que classes estão sem qualquer relacionamento; e
- A unificação dos viewpoints.

As contribuições deste trabalho compreendem:

- A viabilização prática do método proposto[23][24][25][26] e na sua extensão através da definição de meta-dados para as classes, viewpoints e frameworks desenvolvidos;
- O trabalho compartilhado na definição de frameworks que o ambiente oferece;
- A portabilidade do ambiente, visto que sua interface única com o usuário é um browser;
- O suporte à identificação do kernel e dos pontos de flexibilização do framework (tarefa nem sempre trivial, dependendo do número de classes envolvidas);
- A definição e criação de uma ferramenta que faça a verificação de design e de implementação (Verificador de Integridade Lógica e de Programação), verificando design e implementação;
- A flexibilização por delegação e não somente por herança. Usando herança para flexibilizar (em muitos métodos isto é a única forma de flexibilização), sobrecarrega-se muito a aplicação e os desenvolvedores, além de criar um forte acoplamento entre as classes quando se utiliza herança múltipla;
- A existência de um dicionário de métodos, informando para o usuário: onde o método é utilizado, que classes o utilizam, sua interface, autor, implementação;
- A unificação de design com plataformas diferentes através do uso da relação Views-a;
- A definição de meta-dados para as classes, viewpoints e frameworks desenvolvidos. Isto proporciona a reutilização em grande escala de componentes em determinados viewpoints, ou mesmo de viewpoints inteiros (tipo pacote) e em última instância de frameworks completos.

- A possibilidade de uma instanciação direta de um modelo baseado em Patterns sem que o usuário neste caso tenha participado dos processos anteriores de criação/definição. O ambiente oferece uma ferramente que informa ao usuário que pontos do framework pode ocorrer flexibilizações e qual a interface dos hot-spots.

10. REFERÊNCIAS

1. M. Ainsworth, A. H. Cruickshank, L. J. Groves, and P. J. L. Wallis, "Viewpoint specification and Z", Information and Software Technology, 36(1), pages 43-51, February 1994.

2. P.S.C. Alencar, D.D. Cowan, S. Crespo, M. F. Fontoura, and C.J.P. Lucena, "A Viewpoint-based Design Method: A Case Study in Frameworks for Web-based Education", CS-98, Computer Science Department, University of Waterloo, 1998.

3. P. Alencar, D. Cowan, C. J. Lucena, and L.C. Nova, "The Views Relationship in Object-Oriented Analysis and Design", Technical Report, CS-97-13, University of Waterloo, Ontario, Canada, 1997.

4. Dublin Core Metadata Element Set: Resource Page, http://purl.oclc.org/metadata/dublin_core/

5. E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, "Design Patterns, Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

6. ITU Recommendation X.901-904 – ISO/IEC 10746 1-4, "Open Distributed Processing – Referring Model Parts 1-4", July 1995.

7. R. Johnson, "Frameworks = (Components + Patterns)", Communications of the ACM, Volume 40, Number 10, October 1997.

8. W. Pree, "Design Patterns for Object-Oriented Software Development", Addison-Wesley, 1995.

9. J. C. S. P. Leite, M. Sant'anna, and F. G. Freitas, "Draco-PUC: a Technology Assembly for Domain Oriented Software Development"; Proceedings of the 3rd IEEE International Conference of Software Reuse, 1994.

10. Jing Dong, "A Transformational Process-Based Approach to Object-Oriented Design", Master thesis, University of Waterloo – Canada.

11. H. A. Schmid, "Systematic Framework Design by Generalization", Communications of the ACM, Volume 40, Number 10, October 1997.

12. Filkelstein,A. , Kramer, J., Nuseibeh, B. , Filkelstein,L. and Goedicke, M., "Viewpoints: A Framework for Integrating Multiple Perspectives in System Development", Internation Journal of Software Engineering and Knowledge Engineering 2(1):31-58, March 1992, World Scientific Publishing Co.

13. W. Pree, "Framework Patterns", OOPLSA'97 ,Sigs Books and Multimedia, 1997.

14. Light, R., "Presenting XML", Sams.net publishing, 1997.

15. http://www.imsproject.org/

16.http://www.imsproject.org/metadata/MDintro.html

17. http://www.w3.org/TR/WD-rdf-syntax/

18.Douglas C. Schmidt, Mohamed Faya, and Ralf Johnson, "Software Patterns", Communications of the ACM, 39(10), October 1997.

19. Metadata at W3C, http://www.w3.org/Metadata/.

20. S. Crespo, C. J. Lucena e Arndt v. Staa, " FrameConstructor: uma ferramenta para Suporte a Construção Sistemática de Frameworks ", MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, September 1998.

21. D. Cowan and C. J. P. Lucena, "Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse", IEEE Transactions on Software Engineering, March 1995.

22. P. Alencar, D. Cowan, S. Crespo, M. F. Fontoura, and C. J. Lucena, "Pragmatic Issues Behind Framework Design", Workshop #12 - Pragmatic Issues in Using Frameworks Implications for Framework Design, ACM Object-oriented Programming, Systems, Languages, and Applications (OOPSLA'98), Vancouver, Canada, 1998 (Web publication).

23. S. Crespo, M. F. Fontoura,and C. J. Lucena, "A Framework Development Method Using Viewpoints and the Views-a Relationship in the OO Design", MCC13/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, April 1998.

24. P. Alencar, D. Cowan, T. Nelson, M.F. Fontoura, and C. J. Lucena, "Viewpoints and Frameworks in Component-Based Design", in book "Object-Oriented Application Frameworks", M. Fayad, D. Schmidt (editors), three volumes, John-Wiley, 1998.

25. M. F. Fontoura, E. H. Haeusler, and C. J. Lucena, "A Framework Design and Instantiation Method Based on Viewpoints", MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, August 1998.

26. M. F. Fontoura, E. H.Haeusler, and C. J. Lucena, "The Hot-Spot Relationship in OO Framework Design", MCC33/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, September 1998.