FrameConstructor: uma ferramenta 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 42/98 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 reutilização de software pode ser alcançado através do uso de frameworks[8][12][4] orientados a objetos. Um framework orientado a objeto captura os aspectos comuns à 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 em programação[30][4]. Pesquisas atuais reconhecem a necessidade da integração de 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 vezes sem a necessidade de refazê-la novamente. Desta forma, podem ser usados para implementar as variações associadas com cada hot spot[33][34]. Um hot spot representa um aspecto de um framework que pode ter diferentes implementações para cada instanciação do mesmo. Porém, não está claro como identificar os hot spots de frameworks, e como fazer a conexão entre a seleção do design pattern e a implementação destas variações[5].

Este artigo apresenta um ambiente que dá suporte a um método de design[33][34] para software orientado a objetos, que combina frameworks e viewpoints[33][34][35]. O ambiente utiliza os conceitos de viewpoints, design patterns e as relações hot-spot e views-a como princípios básicos para a construção de frameworks.

O ambiente permite uma significativa ampliação das vantagens do método original[33][34] uma vez que permite a viabilização prática do método, a reutilização em grande escala, interoperabilidade com plataformas IMS[13], construção cooperativa de framework, na portabilidade do ambiente e na identificação do kernel e dos pontos de flexibilização do framework (tarefa nem sempre trivial, dependendo do número de classes envolvidas).

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 TEMPLATES

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[1]. 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.

[1] **Não confundir com template em C^{++} que possui significado totalmente diferente do que este que estamos utilizando.**

Dois tipos de flexibilização são possíveis: por herança e por composição. Nos dois casos veremos o papel executado pelos hooks.
No exemplo a seguir, queremos flexibilizar o tipo de algoritmo de classificação utilizado.
Será utilizado o conceito de pattern comportamental "Template Method" proposto em Gamma[5].
Considere-se a classe Sort que classifica vetores numéricos.
Sort somente inicia o processo de classificação após o correto recebimento do vetor de dados numéricos. Sort define duas operações concretas: RecebeVetor e ImprimeVetor. DoClassificaVetor não possui implementação, é um método vazio. As suas subclasses irão conter os detalhes de implementação, sobrepondo-se este método durante sua execução.

```
// exemplo de um código que contém um hook
Void Sort:: Classifica() {
RecebeVetor();
DoClassificaVetor();    // hook
ImprimeVetor();
}
//O cliente Sort, sempre chama Classifica, e  subclasses de Sort sempre
sobregravam DoClassificaVetor.

//DoClassificaVetor não faz nada em Sort.
void Sort::DoClassificaVetor() { }

//Subclasses sobregravam DoClassificaVetor para adicionar diferentes formas de
ordenação de vetores.

void MySort:: DoClassificaVetor() {
       // implementa um algoritmo de quick-sort
       ....
}
```

Um método hook pode ser visto, como os "locais" ou "pontos de flexibilização" que são chamados por métodos mais complexos que são comumente conhecidos como "métodos template" Wirf-Brock[12]; Gama[5] e Pree[8]. Métodos template definem um comportamento abstrato, um fluxo de controle genérico ou a interação entre objetos.
A idéia básica de um hook, é que a sobreposição permite mudanças no comportamento dos objetos sem a necessidade de alterar o código fonte das classes correspondentes. Neste caso tanto hook quanto template são métodos.
Em muitos casos pode-se ter o template e o hook em uma mesma classe e neste caso a adaptação só pode ser feita através de herança e sua aplicação requererá uma nova reinicialização.
Métodos hooks residem em classes (abstratamente) que são acopladas com a classe que contêm o método template correspondente, permitindo desta forma uma adaptação flexível pela ligação de componentes à implementações hooks específicas. Desta forma, pode-se ter adaptação em tempo de execução, sem a necessidade de um reinicialização do sistema.
Algumas vezes, métodos template e métodos hook são unificados em uma única classe. Este é o caso em algumas variações do pattern chamado de "*Flexible Object Creation*". Em muitas situações é melhor colocar as partes não flexíveis do framework e seus pontos de flexibilizações em classes separadas. Neste caso, a classe que contém o(s) método(s) hook pode ser consideradas como uma classe hook da classe que contém o correspondente métodos template. Chama-se a classe que contém os métodos template de classe template. Em outras palavras, a classe hook parametriza a correspondente classe template. Análogamente ao template e métodos hooks, uma situação particular pode determinar qual classe é um template e qual é uma classe hook. Se os métodos template de uma classe são utilizados como métodos hook por outra classe, a classe template torna-se uma classe hook[8].
A separação de template e classes hook proporciona a união abstrata de objetos destas classes de forma a que o comportamento de um objeto T(template) possa ser modificado através de

composição, ou seja, colando-o em um objeto H(hook) específico. Esta separação é uma pré-condição para que a adaptação possa ser feita em tempo de execução. A figura 1 abaixo, exemplifica este conceito.

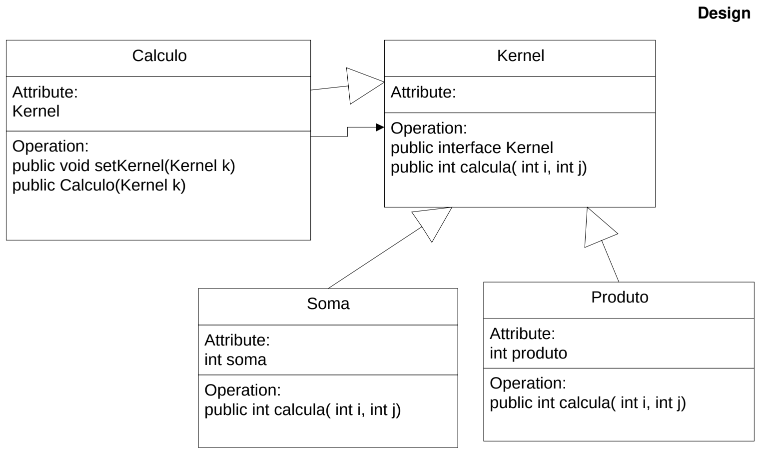

Figura 1. Sistema de cálculo

Este exemplo, mostra uma aplicação que utiliza um método abstrato chamado calcula da interface Kernel (Kernel.java). Este método recebe dois inteiros como parâmetros. A intenção é mostrar com se realiza uma flexibilização por Delegação sem a mecessidade de se recompilar todo o sistema (reinicializar). São implementadas duas formas de manipulação dos inteiros recebidos: Soma e Produto. Para isso existem duas classes concretas Soma.java e Produto.java que implementam o método calcula. Estas implementações sobrepõem o hook vazio em Kernel que é uma interface do exemplo. Abaixo pode-se ver o código fonte comentado deste exemplo. Observa-se a independência da implementação Produto.java. Se esta for alterada de produto para divisão, para sua implementação, será suficiente recompilar Produto.java para que todo o "Kernel do framework" se mantenha inalterado.
O exemplo seguinte implementa o Pattern “Strategy”de GAMMA[5] em JAVA.

```
//--------- Calculo.java------<**Delegações**>--- *** TEMPLATE ***
 public class Calculo implements Kernel {     // == [ OBS. cada arquivo.java é
                                              // um arquivo separado! ] == //

  // Gerência de delegação.
  private static Kernel kernel;

  public Calculo(Kernel k) {
    kernel = k;
  }
  public void setKernel(Kernel k) {
    kernel = k;
  }
  // Interface no núcleo para quem delegamos.
  public int calcula(int i,  int j) {
    return kernel.calcula(i,j);
  }
}

//--------Kernel.java-------------------------
public interface Kernel {                       //<------ é a CLASSE HOOK

       // Uma Interface é o equivalente a uma classe com todos os seus métodos
       // abstratos declarados assim como suas variáveis estáticas e finais.
       // Todo método deixado dentro da interface não tem implementaçãoe e toda
       // classe que implemente a interface é responsável pela codificação de
```

```
       // seus métodos. Uma interface é essencialmente, um modelo conceitual.

  public int calcula(int i,  int j);            //<------ é o MÉTODO HOOK
       // este método não sabe o que vai calcular!
       // NÃO tem Implementação alguma!
}

//-------Soma.java------------------------
public class Soma implements Kernel {
       // implements implementa uma interface em nosso caso chamada Kernel.
       // Neste caso, aqui deve estar o código de implementação da interface. Em
       // nosso exemplo a implementação restringi-se ao retorno de uma valor
       // int igual a SOMA de dois valores recebidos como parametros i e j.

  public int calcula(int i,  int j) {
     int soma=0;
     soma = i + j;
    return soma;
  }
}

//-------Produto.java---------------------
public class Produto implements Kernel {
       // implements implementa uma interface em nosso caso chamada Kernel.
       // Neste caso, aqui deve estar o código de implementação da interface. Em
       // nosso exemplo a implementação restringi-se ao retorno de uma valor
       // int igual ao PRODUTO de dois valores recebidos como parametros i e j.

  public int calcula(int i,  int j) {
     int produto=0;
     produto= i * j;
    return produto;
  }
}
//==================================================================
//-------Aplicacao.java--------< **** PROGRAMA PRINCIPAL **** >--------
class Aplicacao {
  public static void main(String[] args) {
    Calculo obj = new Calculo(new Soma());
       //-> O Fato de Aplicacao estar passando uma instância de Soma como
       // um parâmetro para o Construtor de Calculo faz com que Soma.java
       // seja compilado criando-se Soma.class. Ou seja, seu método
       // abstrato é sobreposto pelo template Soma.class,
       // desta forma a próxima linha que imprime o retorno da chamada
       // do método calcula realiza a soma dos inteiros!

    System.out.println("Primeira implementacao(soma) para o metodo:
calcula(9,3) = "+obj.calcula(9,3));
    Class produto = null;
    try {
      produto = Class.forName("Produto");
             // Neste caso, note-se que esta classe
             // é referenciada como uma string
             // Em tempo de compilação de
             // Produto.class, isto deve ser feito separadamente.

    } catch (ClassNotFoundException e1) { Calculo.java, não é gerada a classe
      System.out.println(e1); return;

    try {
      obj.setKernel((Kernel)produto.newInstance());
                    // Aqui, sobregrava-se o método calcula(em Kernel) com a
                    // implementação da classe Produto.class!
```

```
    } catch (IllegalAccessException e2) {
      System.out.println(e2);
      return;
    } catch (InstantiationException e3) {
      System.out.println(e3);
      return;
    }
    System.out.println("Segunda  implementacao(produto) para o metodo: 
calcula(9,3) = "+obj.calcula(9,3));
  }
}
```

Uma implementaçào semelhante em C++ pode ser vista em Gamma[5] na descrição do pattern *Strategy*.

2.2 VIEWPOINTS

Segundo Filkelstein [1][17][6], uma visão (viewpoint) pode ser compreendida como a perspectiva que um ator mantém no desenvolvimento de um processo ao combinar idéias, papéis, ou conhecimentos referentes a um determinado domínio. Isto 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][6].

O primeiro artefato de entrada é um conjunto de diagramas que são desenvolvidos utilizando-se a abordagem baseada em perspectivas. Uma perspectiva define os possíveis diferentes usos do framework. Desta forma, diferentes designs de sistemas associados às várias perspectivas serão produzidos (figura 2). Neste artigo, o termo viewpoints ou perspectivas será utilizado para representar um design orientado a objetos associado a uma perspectiva de um framework. O design é representado usando-se diagramas orientados a objetos. Em nossos exemplos, utilizamos a notação OMT[10] para os diagramas.

Segundo Roberts e Johnson [22],

"Desenvolvimento de frameworks reutilizáveis não pode acontecer simplesmente sentando-se e pensando-se sobre o domínio do problema. Ninguém tem perspicácia suficiente para propor tais abstrações formais".

Eles propõem o desenvolvimento de exemplos concretos para entender o domínio. Nossa estratégia é bastante semelhante: analisamos cada um dos pontos de vista com base em um exemplo concreto e derivamos o framework final a partir desta análise.

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.

A figura 2, apresenta um pequeno exemplo de dois viewpoints para um sistema gerador de relatórios.

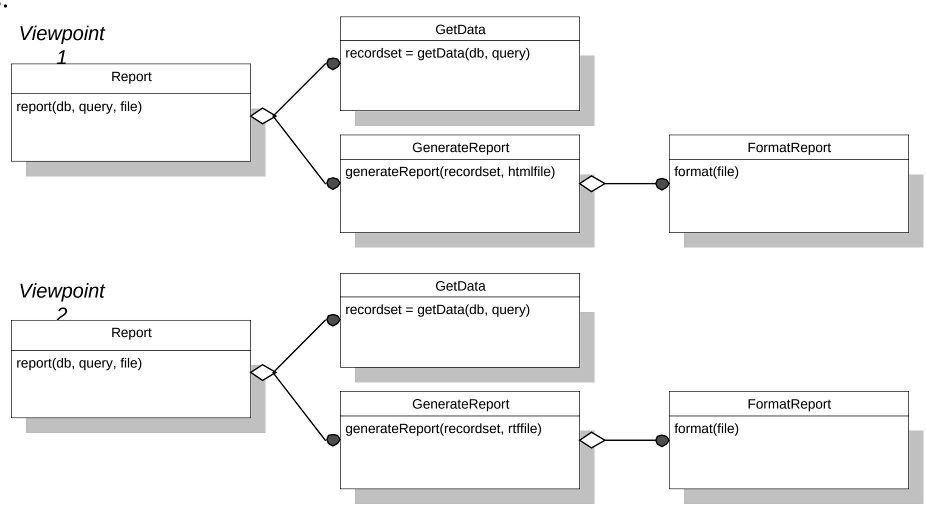

Figura 2. Viewpoints

Na figura 2 podemos ver que o viewpoint 1 apresenta um diagrama de um gerador de relatório simples. Sua classe principal é Report, que usa a classe GetData para permitir acesso à informação requisitada pelas classes Report e GenerateReport, para gerar um relatório de um arquivo em formato HTML.
O segundo viewpoint (2) segue basicamente a mesma estrutura do primeiro. A classe Report e GetData possuem os mesmos métodos com as mesmas assinaturas e implementações.
Estes dois conjunto de viewpoints deverão ser unificados segundo a proposta do método apresentada a seguir.

2.3 VISÃO GERAL DO MÉTODO

A abordagem para o design de framework [2] aqui apresentado baseia-se 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 com os 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 de cada aplicação à qual o framework dá suporte ao ser estendido.
A figura 3 ilustra o método de desenvolvimento baseado em viewpoints, destacando os processos e artefatos utilizados.

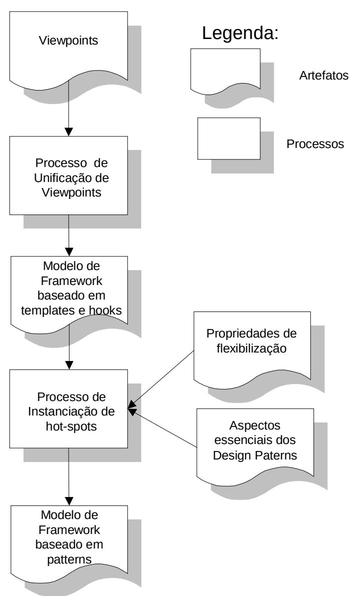

Figura 3. Processos do método de desenvolvimento baseado em Viewpoints.

onde:

Viewpoints	conjunto de classes de uma determinada aplicação, podem existir vários viewpoints.
Processo de Unificação dos Viewpoints	processa todos os viewpoints verificando classes, métodos e assinaturas para a geração de um novo modelo baseado em templates hook.
Modelo de Framework baseado em templates e hook	é um artefato gerado pelo processo de unificação onde aparecem todos os locais em que uma flexibilização é necessárias e o kernel do framework.
Processo de instanciação dos hot-spots	Permite que a relação hot-spot, que é representada pelos pontos a serem flexibilizados no Modelo de framework baseado em template hook, seja definida em termos de propriedades. Essas propriedades irão definir a forma de flexibilização.

Modelo de Framework baseado em patterns	é um artefato gerado pelo processo anterior onde os relacionamentos tipo hot-spot são substituidos por apropriados Design Patterns de forma a melhor realizar a flexibilização exigida.

2.4 O PROCESSO DE UNIFICAÇÃO DOS VIEWPOINTS

Uma vez que todos os viewpoints pertinentes estão definidos, a estrutura do núcleo do framework pode ser agora identificada, a partir da análise das representações de todos os viewpoints visando 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 viewpoints, assim como a obtenção do 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. A figura 4 mostra a identificação do núcleo e dos pontos de flexibilização de uma aplicação abstrata.

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 do framework, 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. Por esta razão, a semântica da definição das partes flexíveis do framework final não se encontram presentes nesta fase do design. Design patterns essenciais propostos por Pree [8] oferecem uma semântica apropriada para flexibilizar frameworks, permitindo ao engenheiro de software, levar em consideração os requisitos de flexibilização ao selecionar o melhor construtor para cada hot-spot presente no framework.

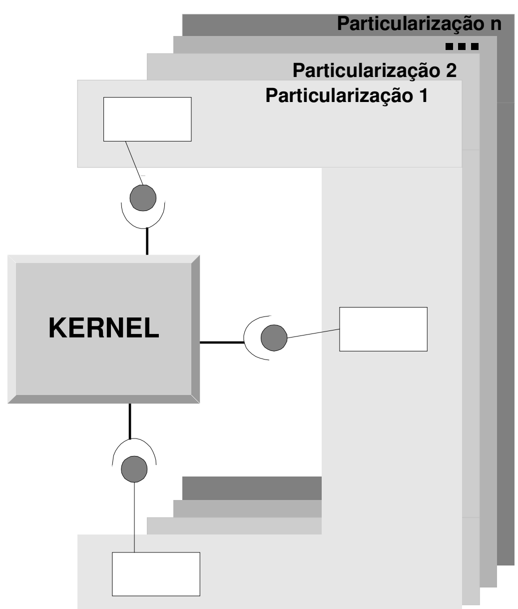

Figura 4. O Kernel e seus hot-spots

Um novo relacionamento em design orientados a objetos, chamado “relacionamento hot-spot”[33][34][35], 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 expressa em função 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[18].

O processo de unificação proposto impõe algumas pré-condições dos 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 viewpoints (Figura 5).

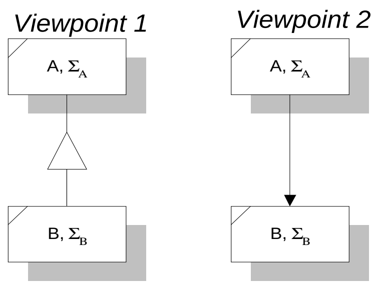

Figura 5. Inconsistência de Viewpoint

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.

Tomando-se o exemplo da figura 2, veremos como se produz o processo de unificação. O método GenerateReport da classe de mesmo nome, tem diferentes assinaturas nos dois conjuntos de viewpoints. No primeiro conjunto, GenerateReport demanda um arquivo em formato HTML, enquanto no segundo por um arquivo em formato RTF. Neste caso, pela regra número três, pode-se afirmar que esse é um método hook. O método format, que pertence a classe FormatReport, possui diferentes implementações nos dois viewpoints, visto que um configura arquivos tipo HTML e o outro arquivos tipo RTF. Pela regra número quatro, esse método é outro método Hook dentro do modelo de framework baseado em hook. Já pela regra número cinco, todos os métodos que utilizam métodos hooks são marcados como modelo(template) de método. Os métodos de templates neste caso são: report e generateReport. O diagrama apresentado na próxima figura, é o resultado da unificação dos viewpoints, que define o modelo de framework baseado em hook. As setas pontilhadas representam os relacionamentos hot-spot. Observe que a assinatura do método generateReport está indefinida, como estabelece a propriedade número três. A implementação dos métodos generateReport e format também estão indefinidos, desta vez regidas pelas propriedades de números três e quatro. As assinaturas e implementações que estiverem indefinidas, serão definidas no processo de instanciação dos pontos de flexibilização (hot-spot).

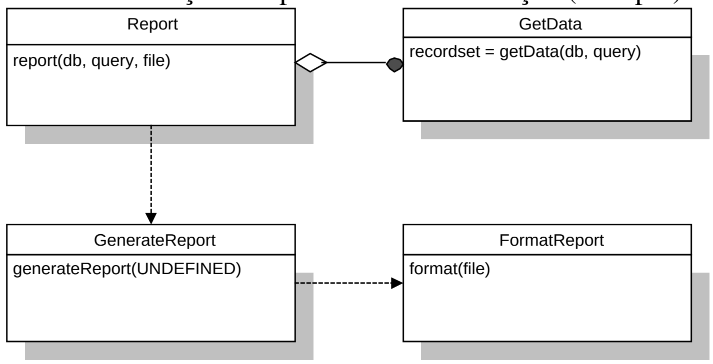

Figura 6 Modelo de framework baseado em Template-hook

2.5 MODELO DE FRAMEWORK BASEADO EM TEMPLATE-HOOK

A semântica deste modelo é dada pela análise dos 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 (figura 6).

O relacionamento hot-spot é um novo relacionamento em design orientados a objetos, que combina templates e classes tipo hook através da utilização de design patterns essenciais. Em [4] pode-se encontrar a formalização da semântica deste novo tipo de relacionamento utilizando-se teoria de categorias e cálculo de objetos.

2.6 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 7 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.

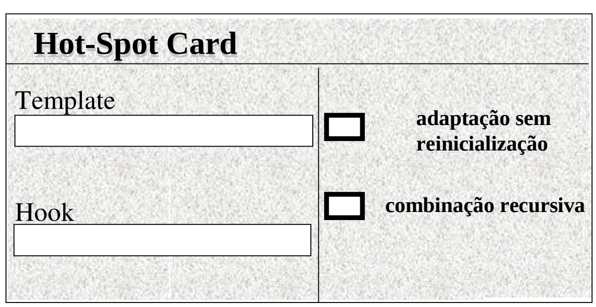

Figura 7. Hot-spot cards

Adaptação sem reinicialização do sistema:

Na unificação dos meta-patterns os métodos templates e hook tornam-se a mesma classe e existe a necessidade de se realizar uma adaptação por meio de herança. Isto implica uma reinicialização da aplicação para que esta mudança tenha efeito. Já na separação de meta-patterns, os métodos de template e hooks aparecem em diferentes classes. A adaptação necessária neste caso, pode ser realizada em tempo de execução através do uso de composição de objetos[8].

Combinação recursiva:

Segundo Pree[8], deve ser enfatizada a necessidade dos cartões Hot-spots no que se refere aos princípios de construção recursiva. Os aspectos de design abordados pelas combinações recursivas dos templates hooks não podem ser expressos no simples e reduzido vocabulário dos cartões Hot-spots. Este vocabulário tem seu foco deliberadamente na funcionalidade, excluindo conceitos tais como definição de interface e herança. Isto deixa o engenheiro de software livre para aplicar melhores técnicas de recursividade de forma a obter uma melhor e mais elegante flexibilização da arquitetura. Entretanto, é possível reconhecer muitos princípios de construção através da análise dos modelos de objetos. Relacionamentos tipo "parte de", "consiste de", "gerencia" e "possui" e similares, indicam o pattern Composite do catálogo GoF Gama[5]. Se uma classe abstrata se torna sobrecarregada, o engenheiro de software deveria optar por composição de comportamento através de cadeia de objetos: alguns destes comportamentos são colocados dentro de classes distintas, podendo ser adicionado por composição de objetos (Composite patterns). O pattern Composite, um pattern estrutural, têm como objetivo compor objetos dentro de uma estrutura de árvores para representar hierarquias tipo "todo-parte". Composite deixa o cliente manipular objetos individuais e compostos de forma uniforme. Este pattern é utilizado também pelo pattern "Chain of Responsibility"[5].

Desta forma, tendo a característica da hierarquia dos objetos e suas colaborações em mente, o engenheiro de software poderá intuitivamente determinar situações onde ele possa aplicar os princípios de construção [8].

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

Como exemplo, vamos considerar o relacionamento hot-spot presente na próxima figura. Suponha-se que seja necessário que um novo método generateReport seja definido em tempo de execução. Suponha-se também que o método format não tenha necessidade de ser redefinido em tempo de execução. Como nenhum dos relacionamentos requisita uma combinação recursiva, o Hot-spot cards que representa este exemplo de flexibilização pode ser representado como na figura 8.

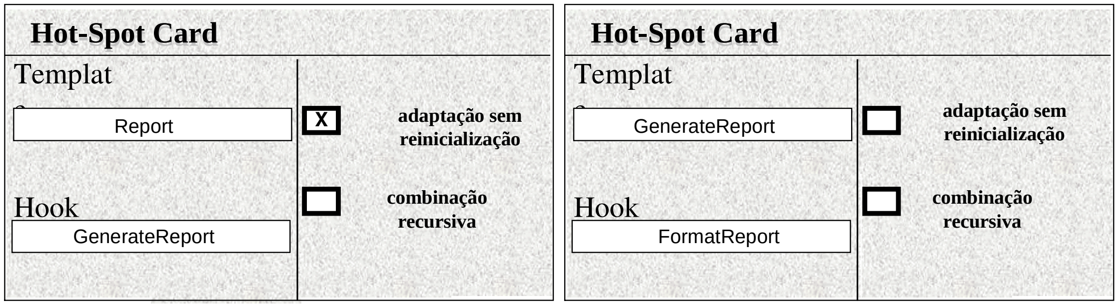

Figura 8. Hot-spot card: utilização

O diagrama resultante após a geração dos patterns apropriados para cada hot-spot pode ser visto na figura 9.

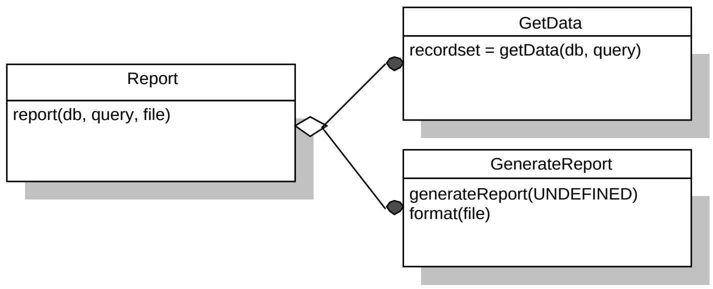

Figura 9. Modelo de framework baseado em pattern

2.7 MODELO DE FRAMEWORK BASEADO EM PATTERNS

O processo de instanciação dos hot-spots gera como resultado um diagrama de classes baseado em patterns. Esses patterns são representações de cada relacionamento tipo hot-spot presente até então no modelo de framework baseado em hot-spot. O framework baseado em patterns pode ser diretamente mapeado para alguma linguagem de programação [5]. Mecanismos de transformação para a geração de código em várias plataformas estão sendo estudados em [4]. O produto final deste módulo “framework baseado em patterns” é um modelo de dados completo para o processo de geração de uma aplicação em ambiente WEB.

2.8 PROCESSO DE INSTANCIAÇÃO DO FRAMEWORK

Este processo é responsável por gerar uma aplicação em ambiente WEB a partir do “framework baseado em patterns”.
A figura 10 mostra o processo de instanciação.

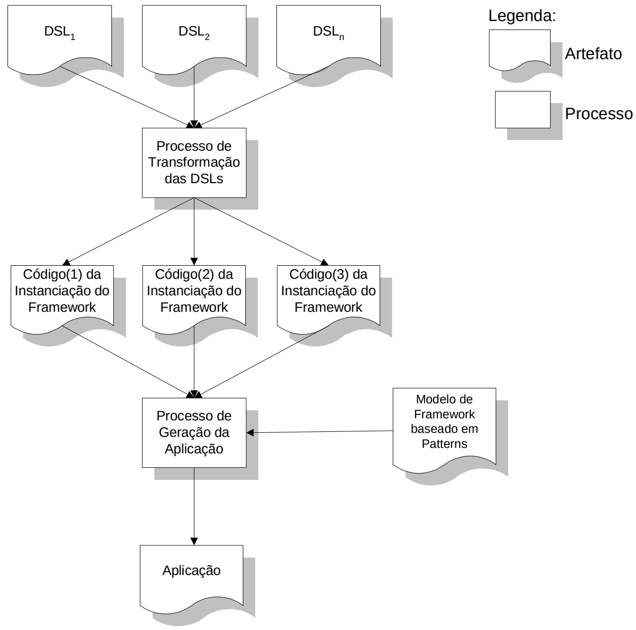

Figure 10. Processo de instanciação do framework

O processo de instanciação de frameworks requer os seguintes elementos: uma linguagem específica de domínio (DSL), um modelo de framework baseado em patterns e um sistema de transformação.

2.8.1 Linguagem Específica de Domínio

Uma linguagem específica de domínio (DSL) é utilizada para especificar formalmente design de software [23]. Isto é, uma linguagem formal que é representada com base em abstrações de um domínio de aplicação. Como exemplos de DSLs podemos citar:

- Tex and Latex, linguagens formatadoras de texto;
- YACC, um gerador de parser;
- Mathematica, uma linguagem extensível para modelagem matemática;
- Descrição de esquemas e linguagens de query para bancos de dados.

Uma vantagem de usar DSLs é que o especialista de domínio pode expressar diretamente conceitos específicos, em lugar de codificá-los. Isto permite ao especialista de domínio formalizar a especificação de uma solução de software imediatamente em vez de comunicar a especificação informalmente a um engenheiro de software que pode estar menos familiarizado com o domínio de aplicação. Quando projetamos uma DSL, dois importantes critérios devem estar em mente:

- A DSL deve ser inteligível para o especialista do domínio;
- Sua semântica deve permitir que uma especificação expressa em uma DSL seja traduzida em procedimentos efetivos que realizem as operações desejadas.

Tipicamente, tal especificação oferecerá restrições estáticas e dinâmicas sobre um artefato do domínio de aplicação, ou especificará seu comportamento dinâmico. Freqüentemente, um editor dirigido por sintaxe pode ser usado para ajudar o projetista da aplicação a formular uma aplicação usando uma nova DSL.

Considere-se o exemplo anterior de geração de relatórios, onde a instanciação de código necessita ser feita para os métodos: generateReport e format. Analisando-se os viewpoints iniciais podemos visualizar uma variação na assinatura de generateReport. Este método trabalha com arquivos no formato HTML e com o formato RTF. Logo uma variação na implementação deve ser também necessária para a correta configuração do layout da impressão. Vamos ver como seriam estes programas escritos em uma DSL.

```
ReportType := HTML
GenerateReportSpec := {ColMaxLength = 10; ColPerRow = 50}
FormatLayoutSpec := {UseFrames; Font = "Verdana"; Background = "White"}
```

Através de transformações deste código, todas as assinaturas e implementações que estão no Modelo de Framework Baseado em Patterns que ainda estão indefinidas, poderão ser instanciadas. O operador DSL ReportType é utilizado para completar a assinatura do método generateReport. A figura 11 mostra o design da estrutura para a aplicação gerada.

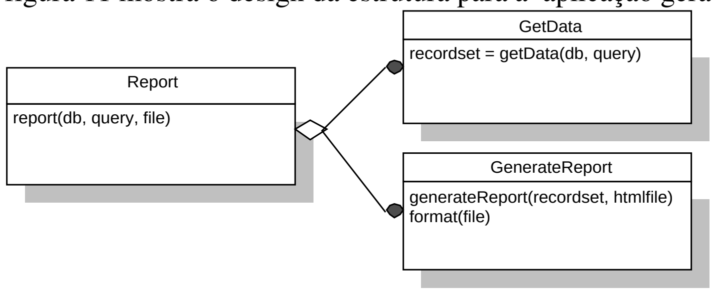

Figura 11. Aplicação instanciada

A funcionalidade fornecida por construtores da DSL, varia de acordo com o meta-pattern selecionado. A tabela abaixo ilustra estas variações.

Meta-pattern	DSL Semantics
Unification	Deve oferecer uma semântica apropriada para a geração do código do método hook.
Separation	Deve oferecer uma semântica apropriada para gerar o código do método hook e especificar como o método varia em tempo de execução. Este tipo de DSL tem que gerar o código que criará subclasses de classe concretas do modelo de framework baseado em pattern criando uma ligação para cada uma das variações do método, semelhante ao design pattern Strategy descrito no catálogo GoF [12].
Recursive Unification	Similar ao meta-pattern Unification, porém deve permitir composição recursiva.
Recursive Separation	Similar ao meta-pattern Separation, porém deve permitir composição recursiva.

Tabela 3. DSLs e meta-patterns

O produto final deste módulo é a geração de uma aplicação em um domínio específico para ambiente WEB.

A seguir vamos mostrar um ambiente que dará suporte ao método proposto.

3. 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[13] através dos conceitos de meta-dados e da utilização de XML[24] 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. Nesta descrição as informações que aparecem nas telas da ferramenta são do exemplo da figura 2. Posteriormente apresenta-se os passos necessários para a continuidade do ambiente e conclusões.

3.1 INTRODUÇÃO

Inicialmente vamos descrever dois conceitos utilizados no ambiente que nos permitirá uma grande taxa de reutilização de informação que são: meta-dados[16] e XML[24].
Estes dois conceitos foram propostos no projeto "EDUCON/NLII IMS" [13] que define uma arquitetura aberta para produção de material para Educação Baseada na WEB.
Um dos grande problemas hoje na WEB é o de interoperabilidade e reutilização de informação[16]. Resultante do aumento de aplicações tendo um browser como interface e a facilidade de se produzir material para este meio. Existe a necessidade de que os materiais produzidos possam ser reutilizados em grande escala e com um baixo custo de implementação. Este é um problema antigo da ciência da computação, porém agora com um enfoque modificado em virtude da nova plataforma para o desenvolvimento de aplicações, a WEB. A plataforma IMS propõe uma série de especificações para tentar solucionar os problemas citados 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. Isto nos permite uma otimização no processo de construção das perspectivas dado que todas as classes utilizadas em outras perspectivas só poderão ser alteradas pelos respectivos proprietários. 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, ou seja, várias pessoas trabalhando tanto de forma independente quanto ao mesmo tempo em grupo.

3.1.1 Meta-dados

Um meta-dado [24] 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[25]:
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 [27] 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.

3.1.2 XML

XML (eXtend Markup Language) [24] 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[28][29]. 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.

3.2 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[13].

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.

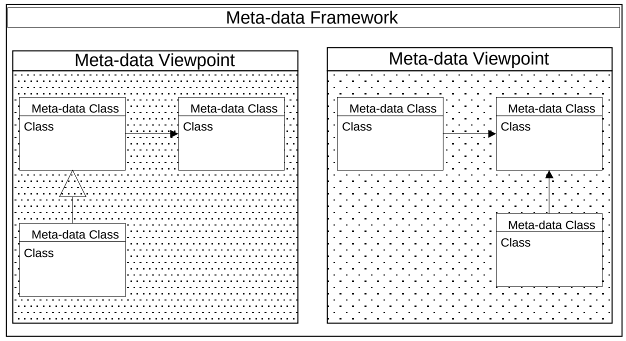

Figure 12. Meta-dados no ambiente

A figura 12, nos permite visualizar os vários tipos de meta-dados e o potencial de reutilização que isto nos oferece. 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.

3.3 ARQUITETURA DO AMBIENTE

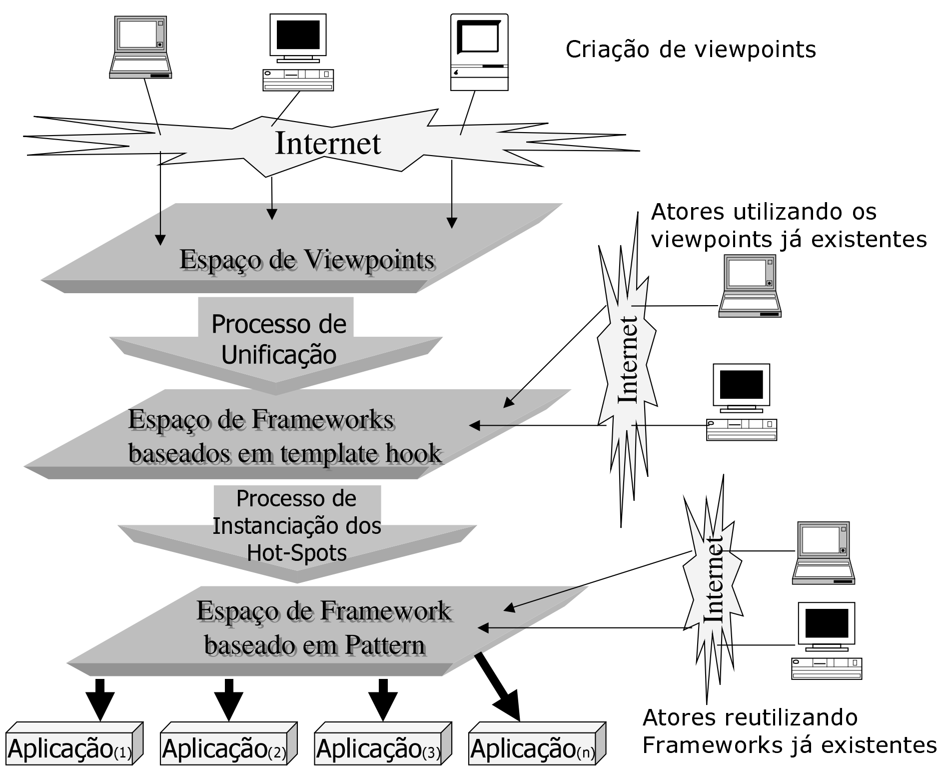

Figure 13. Arquitetura do Ambiente

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 13. A seguir vamos descrever os espaços e seus atores.

3.3.1 O espaço de viewpoint

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.

3.3.2 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*.

3.3.3 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.

3.4 AMBIENTE DE IMPLEMENTAÇÃO

O Ambiente está sendo desenvolvido na seguinte plataforma de hardware e software:
Sistema operacional: Windows NT Server 4.0
Linguagem de programação: JavaScript[20], CGILua[19], DHTML-HTML[21]
Banco de dados: Microsoft® Access 97
Browser para interação: Explorer e/ou Netscape.

Servidor WWW: Internet Information Server da Microsoft®.
O ambiente que dará suporte ao método descrito possui 5 módulos principais e ferramentas auxiliares.
A seguir, a figura 14 apresenta o diagrama E-R parcial do ambiente que permite o desenvolvimento das seguintes etapas:
Módulo de domínio, módulo de perspectiva, unificação de perspectivas e geração do framework template-hook.

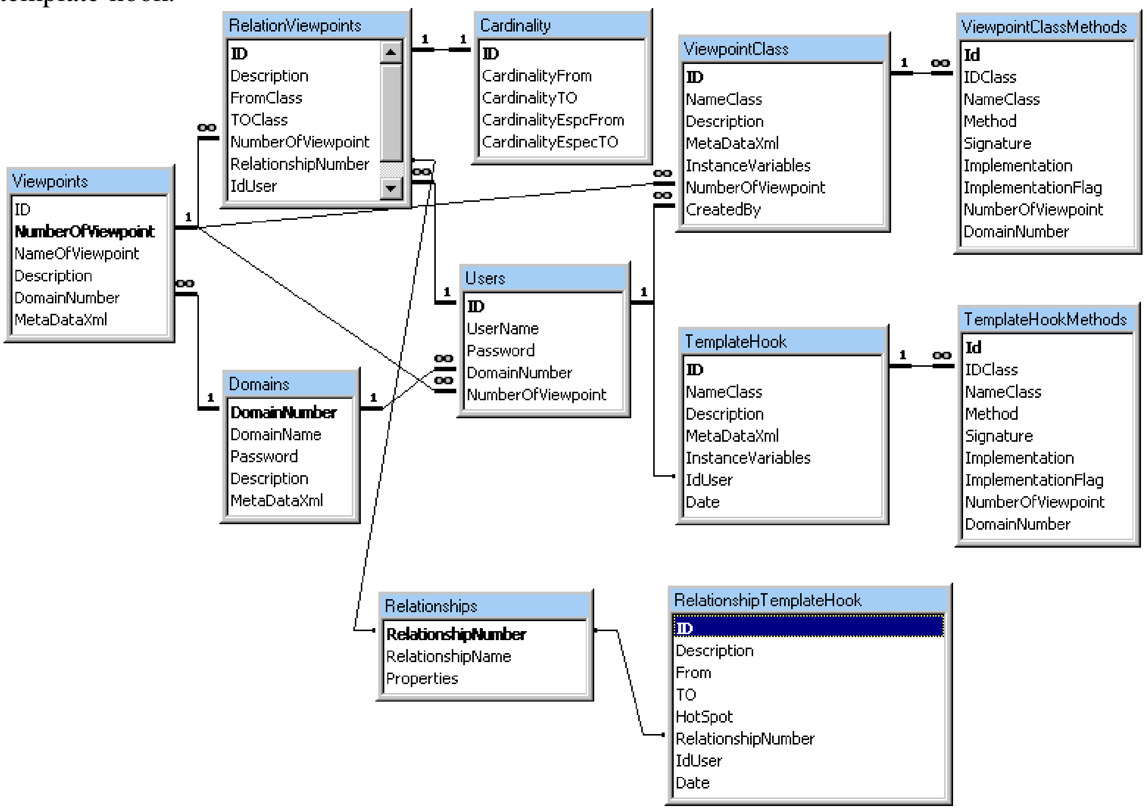

Figure 14. Diagrama E-R parcial do ambiente.

3.5 ADMINISTRAÇÃO DO AMBIENTE

O ambiente possui uma administração que funciona independentemente, permitindo que sejam gerenciados domínios, perspectivas e usuários.

3.5.1 Módulo de domínio

O diagrama E-R da figura 14, nos permite visualizar que um Domains possui uma relação de 1:muitos com Users, ou seja, existem vários usuários pertencente a um mesmo domínio de aplicação. Domínio possui uma relação de 1:muitos com Viewpoints, desta forma podemos ter várias perspectivas (ou viewpoints) dentro de um determinado domínio.
Com o objetivo de proporcionar uma separação de atividades por atores, este módulo permite a criação e a seleção de um determinado domínio de aplicação, como por exemplo, o domínio AulaNet[2][38][39]. O módulo possui uma ferramenta de gerência de usuário, possibilitando o acesso a todas as classes que pertencem a um determinado domínio. Isto pode ser visto na figura 15. A classe que implementa estas facilidades chama-se “Control Manager”. Como pode ser visto na figura 15, quando um usuário faz seu login, se este tiver permissão poderá ter a sua disposição uma janela onde poderá administrar o ambiente. Ao mesmo tempo é composto uma outra janela

dividida em 3 frames sobre a qual o ambiente opera. A qualquer momento o usuário pode trocar sua perspectiva ou o seu domínio. Para isso uma nova Janela chamada “New Login” é disponibilizada. A figura 15 mostra todo o fluxo de dados e controles desta fase.

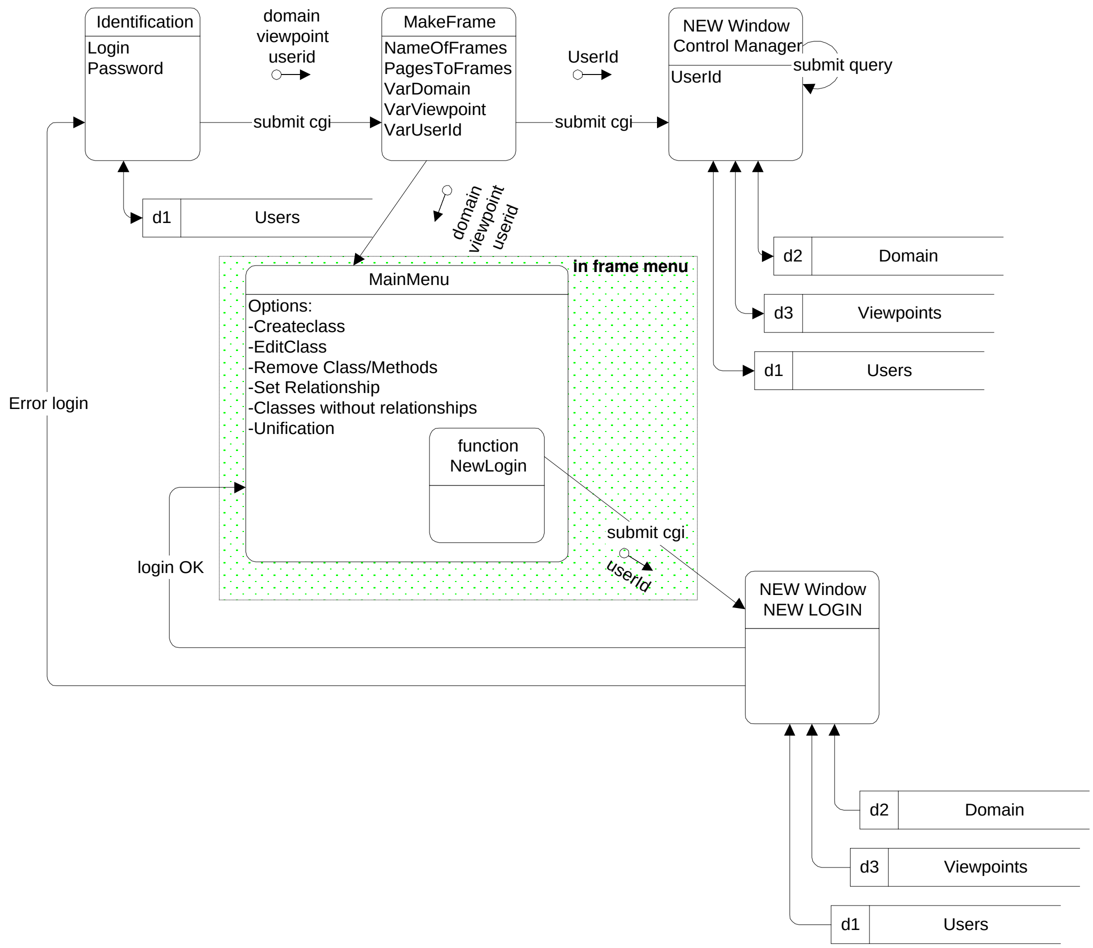

Figura 15. Processo de identificação – entrada no ambiente

Esse módulo (Control Manager) apresenta as seguintes funcionalidades:
Domínios:

- Visualizar as propriedades de um domínio
- Criar um novo domínio
- Remover um domínio já existente

Usuários:

- Definir usuários para um dado domínio
- Adicionar novo usuário
- Editar usuários
- Remover usuários

Perspectivas ou Viewpoints:

- Definir as perspectivas
- Editar
- Remover

O objetivo deste módulo é permitir que o ambiente possa ter um gerenciamento independente dos atores que irão construir as várias perspectivas de um framework. O ambiente possui dois níveis de senhas, uma que é utilizada para permitir acesso às tarefas administrativas e outra que permite aos atores desenvolverem as classes e relacionamentos em suas respectivas perspectivas.

3.5.2 Criação de perspectivas

Acompanhando o diagrama E-R da figura 14, podemos verificar que: existe um relacionamento de muitos:1 de Viewpoints para Domínios, ou seja, pode-se ter vários conjuntos de perspectivas que pertencem a um dado domínio. Viewpoints possui um outro relacionamento de 1:muitos com ViewpointClass que são as instâncias da classe Viewpoint propriamente dita. Desta forma tem-se várias classes de objetos que pertencem a uma classe abstrata Viewpoint.

Este módulo, permite a criação de várias perspectivas em cima de um determinado domínio de aplicação. Uma perspectiva define a particularidade do uso de um framework.

As funcionalidades deste módulos são:

- Selecionar uma perspectiva
- Adicionar uma nova perspectiva
- Editar uma perspectiva
- Remover uma perspectiva

Por exemplo:

Domínio: AulaNet – Criar um ambiente para proporcionar educação baseada na Web.

Perspectivas:

Perspectiva1: do Web Master

O Web Master pode definir para cada instituição, departamento a interface das páginas variando as figuras, cores, fontes, botões, cores de fundo e a estrutura navegacional do site. Esta estrutura refere-se a seqüência de etapas para a completa criação de um curso. Por exemplo, podemos ter as seguintes etapas para a criação de um curso: a) dados do curso, b) dados do autor, c) serviços didáticos e d) entrada de conteúdo para os serviços já definidos. Esta seqüência pode ter mais ou menos passos dependendo do ponto de vista do Web Master.

Perspectiva2: do educador

O educador pode definir quais são as tarefas administrativas e ou didáticas que os autores terão disponíveis para a criação de cursos, tais como: utilização de slides, exercícios, sons, simulações, vídeo, ferramentas externas (como por exemplo: um link para simulação) e recursos Internet (como por exemplo: ftp, mail, etc.)

Perspectiva3: do autor

O autor pode facilmente criar um novo curso utilizando o material por ele já desenvolvido ou utilizar fragmentos de cursos já existentes no ambiente. Pode utilizar e configurar as formas de interação entre os participantes através da seleção de ferramentas próprias tais como: chats, news, listas, definição de grupos de estudantes para uma determinada tarefa ou conteúdo. Estas ferramentas foram previamente selecionadas pelo educador ficando a critério do autor a sua seleção e melhor forma de utilização.

Perspectiva4: do aluno

O aluno pode, sob sua ótica, selecionar um determinado curso, customizá-lo, buscar material do ambiente para uso local e disponibilizar material por ele criado dentro do ambiente.

Todo o processo é apoiado por uma ferramenta chamada *Ferramenta de Gerência de Perspectivas*.

3.6 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 16, 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.

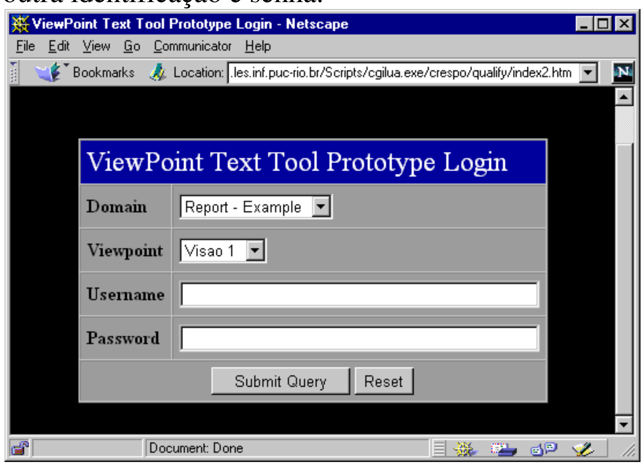

Figura 16. Viewpoint Text Tool

A figura 17, a seguir mostra o ambiente logo após o login do usuário. Pode-se ver no menu pop-up as opções de trabalho nesta etapa.

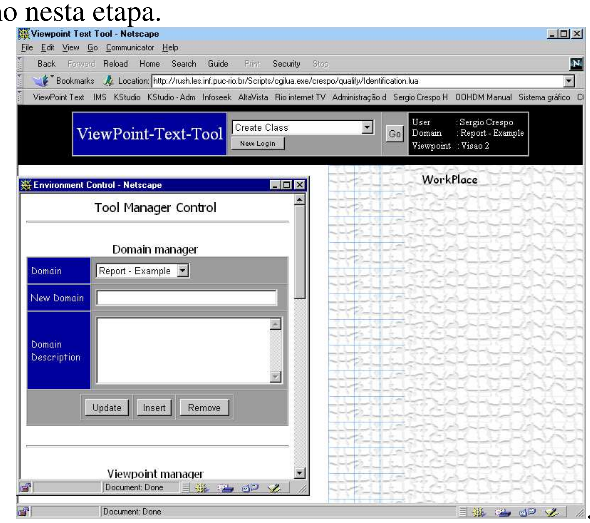

Figura 17. Perspectivas: Interface e seus menus.

A figura 17 nos permite ver que 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. A interface ,como pode ser vista na figura 17, apresenta um frame superior onde aparece um menu pop-up de opções. A área logo abaixo, foi dividida em dois frames: no frame a esquerda são apresentados os formulários para preenchimento e no frame a direita são mostrados informações adicionais de forma a permitir um melhor preenchimento dos formulários ou informação gerais aos atores sobre classes e suas propriedades, conexões, etc.

3.6.1 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 18 nos mostra a estrutura de dados e classes desta fase e a figura 19 a interface já implementada. 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.

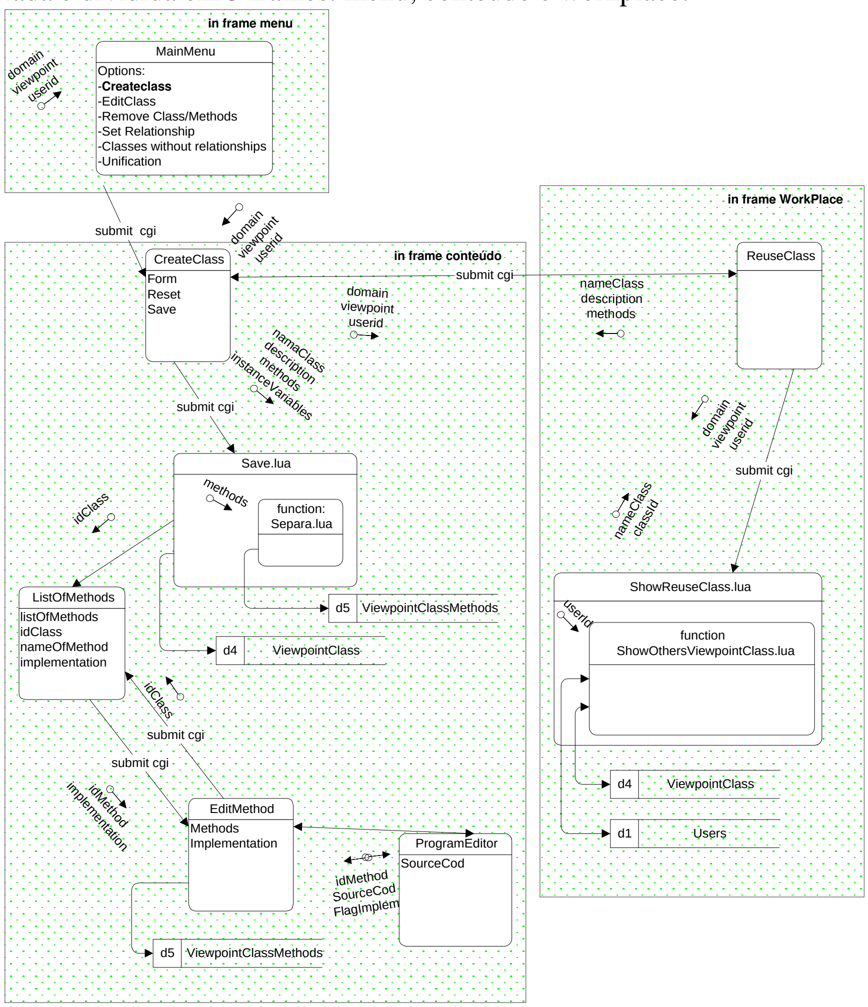

Figura 18. Diagrama de fluxo e classes da fase: Criando classes

A figura 19 nos mostra um exemplo desta interface.

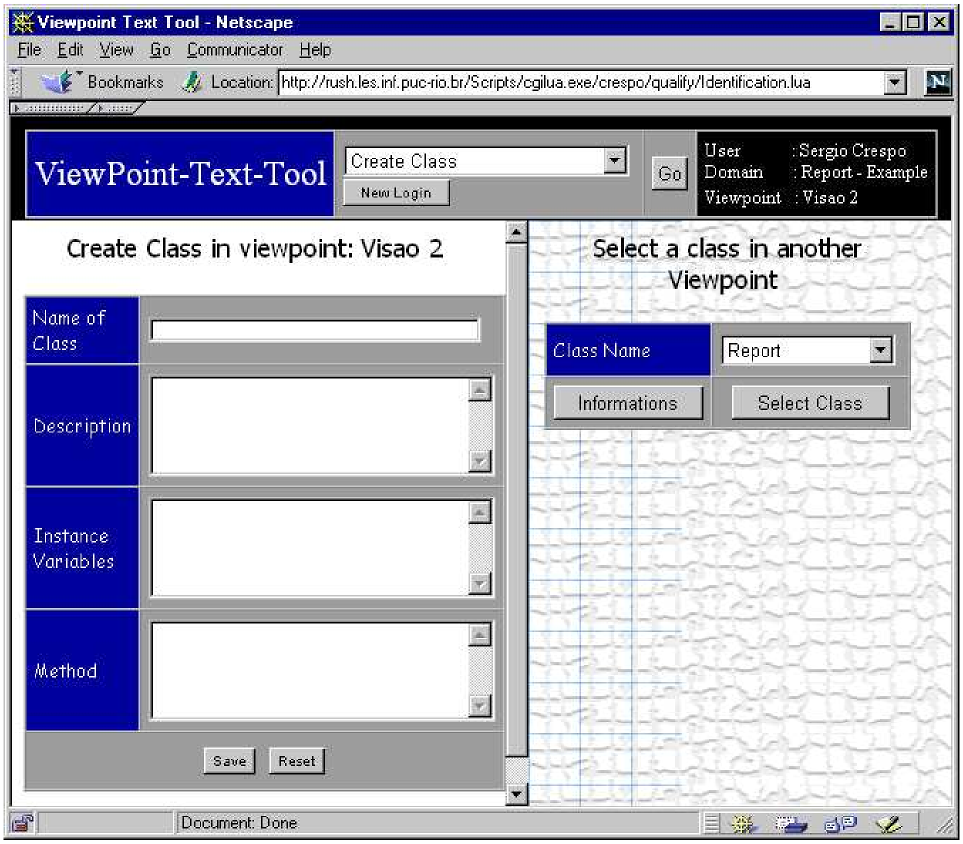

Figura 19. Criação de classe

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. Se for necessário utilizar as classes com outros relacionamentos, deve-se trocar os nomes das classes neste momento. Isto é necessários pois não se pode ter dois conjuntos de relacionamento entre classes distintos, pois isto acarretaria uma inconsistência entre viewpoints, como já foi visto anteriormente na descrição do método (seção 2.2).

Durante o processo de criação de uma classe, o usuário tem disponível um pequeno editor de programas para definir como será a implementação da classe. Ele poderá utilizar qualquer outro editor de programas para editar o programa, checar sintaxe, semântica, etc. e depois copiá-lo para dentro do editor do ambiente e salvá-lo sem problemas como pode ser visto na figura 20.

Quando a classe é criada, ela imediatamente torna-se disponível para todos os usuários que estejam trabalhando neste domínio, não importando se o usuário está remotamente conectado.

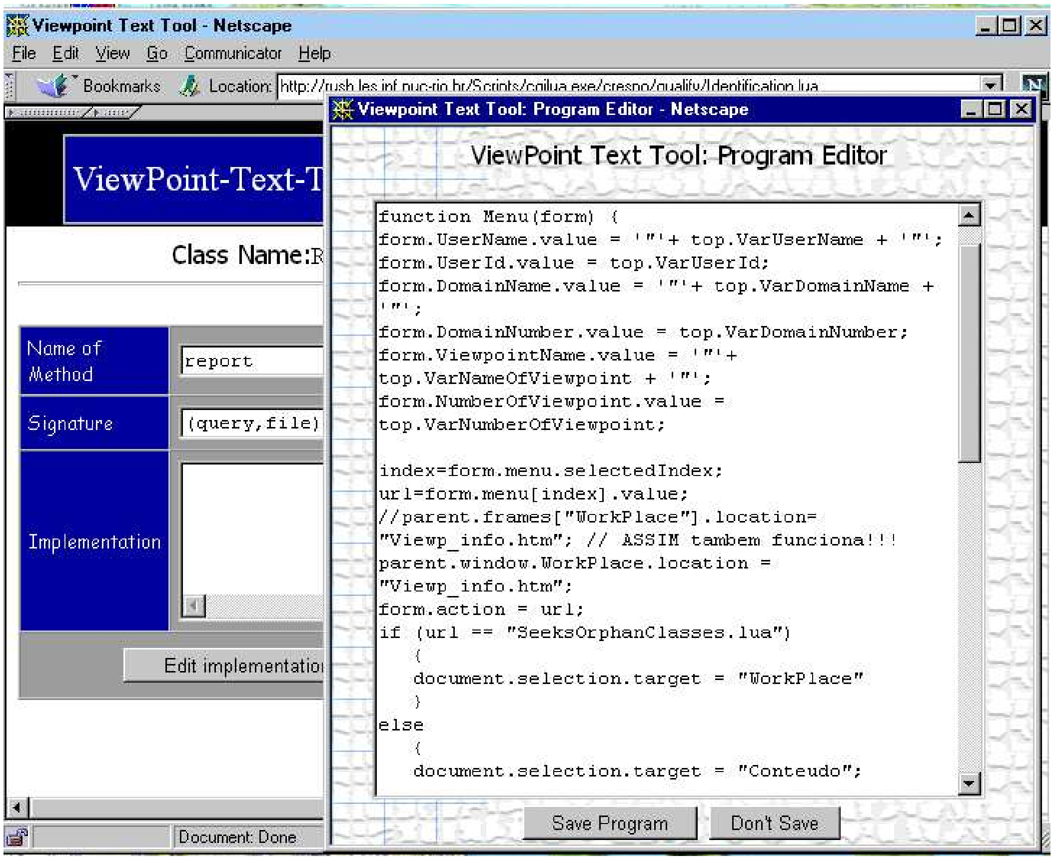

Figura 20. Usando o editor de programas do Ambiente

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..

É importante frisar que as classes criadas possuem um proprietário, tornando o ambiente mais seguro e protegido nas operações de atualização de dados. Isto fica mais claro no momento em que desejamos realizar alterações em algumas classes, como veremos a seguir.

3.6.2 A edição das classes

Quando o usuário deseja alterar alguma classe, o sistema apresenta uma relação de todas as classes existentes no domínio ao qual o ator pertence em ordem alfabética de atores. São apresentadas as classes de vários atores, porém o sistema disponibiliza a opção de alteração somente para as classes em que o ator é o autor. A figura 21 nos mostra o fluxo de dados e interação entre as classe desta fase.

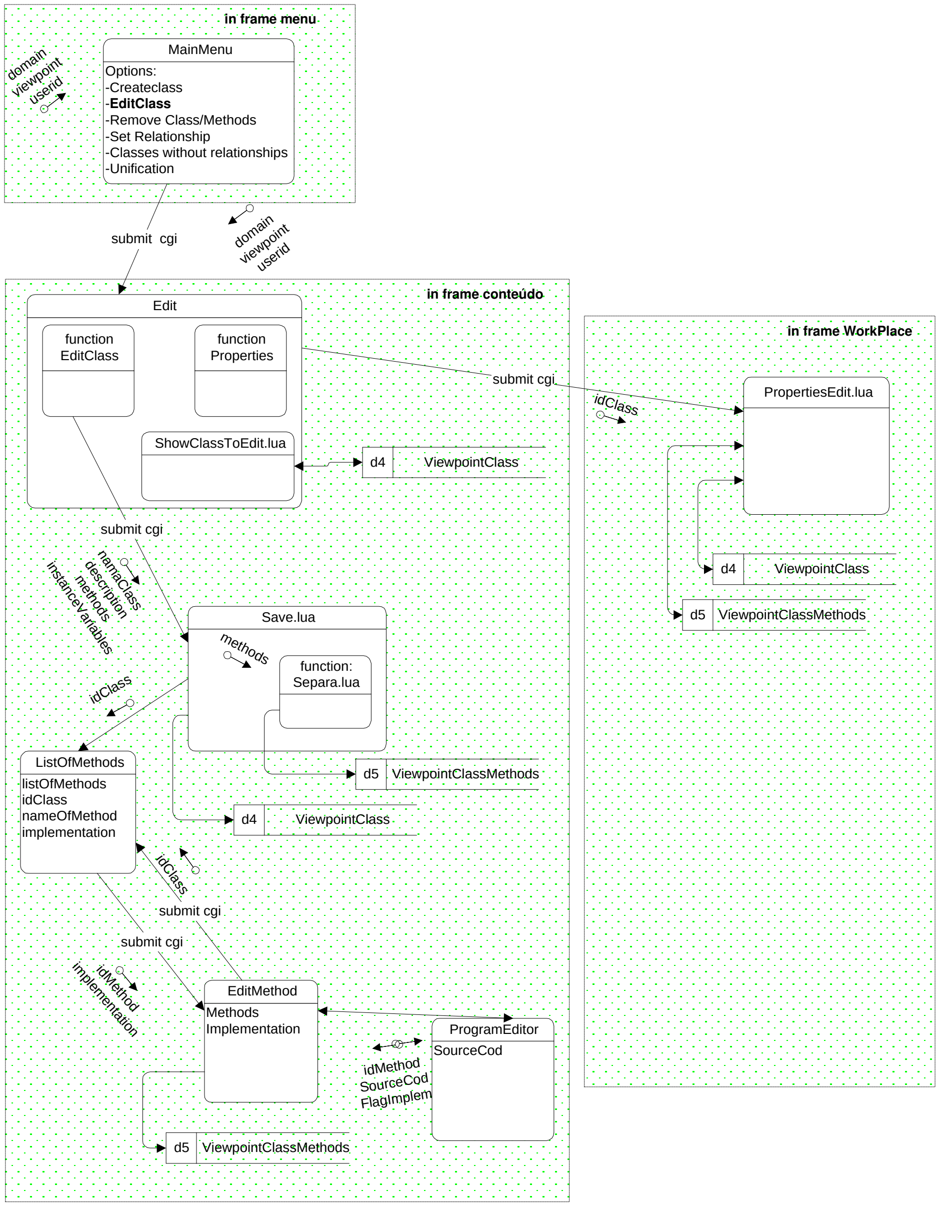

Figura 21. Diagrama de fluxo e classe para a fase: Editar classes

A seguir pode-se ver a tela em que aparecem as classes que pertencem ao domínio do usuário e as telas em que ele pode realizar alterações.

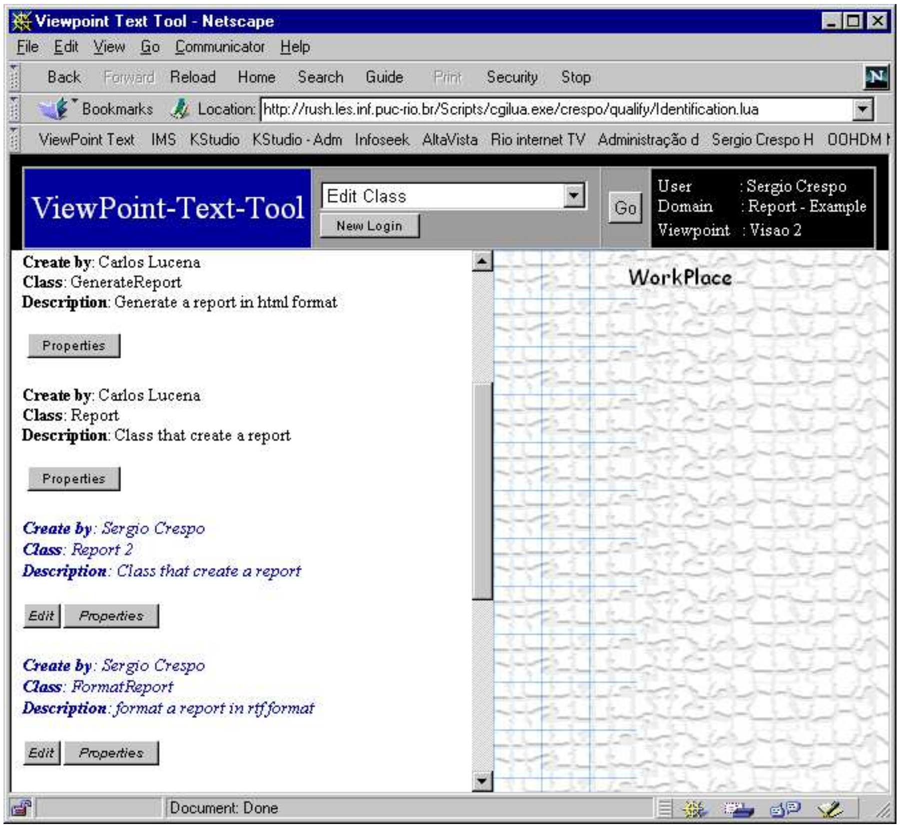

Figura 22. Edição de classes

A figura 22 nos mostra a relação de todas as classes que pertencem ao domínio no qual o usuário esta inserido. Note-se que as duas primeiras classes apresentadas são de um outro autor e logo, só é permitido que se vejam as suas propriedades. A terceira classe, de nome Report 2, apresenta agora dois botões: Edit e Properties. Neste caso o usuário que esta interagindo com o sistema é o próprio autor desta classe. Se ele optar por ver as propriedades, estas serão apresentadas no frame a direita (neste caso foram apresentadas as propriedades da classe FormatReport).
Na edição das classes, sempre que o autor fizer uma alteração da implementação será apresentado o fonte do programa no editor visto anteriormente com os botões "*Save*" e "*Don't Save*". Essa informação da mudança na implementação da classe se não registrada poderá acarretar problemas na flexibilização.
De forma análoga à criação de uma classe, o usuário poderá utilizar qualquer outro software para realizar a edição do programa e posteriormente copiá-lo para este editor do ambiente.

3.6.3 A remoção de classes e ou métodos

O ambiente permite que classes e métodos possam ser facilmente removidos. Quando uma classe é removida todos os relacionamentos que a referenciavam também são removidos. A figura 23 mostra o diagrama de fluxo e classes desta etapa.
A remoção de um ou mais métodos impacta diretamente no resultado da unificação dos viewpoints.

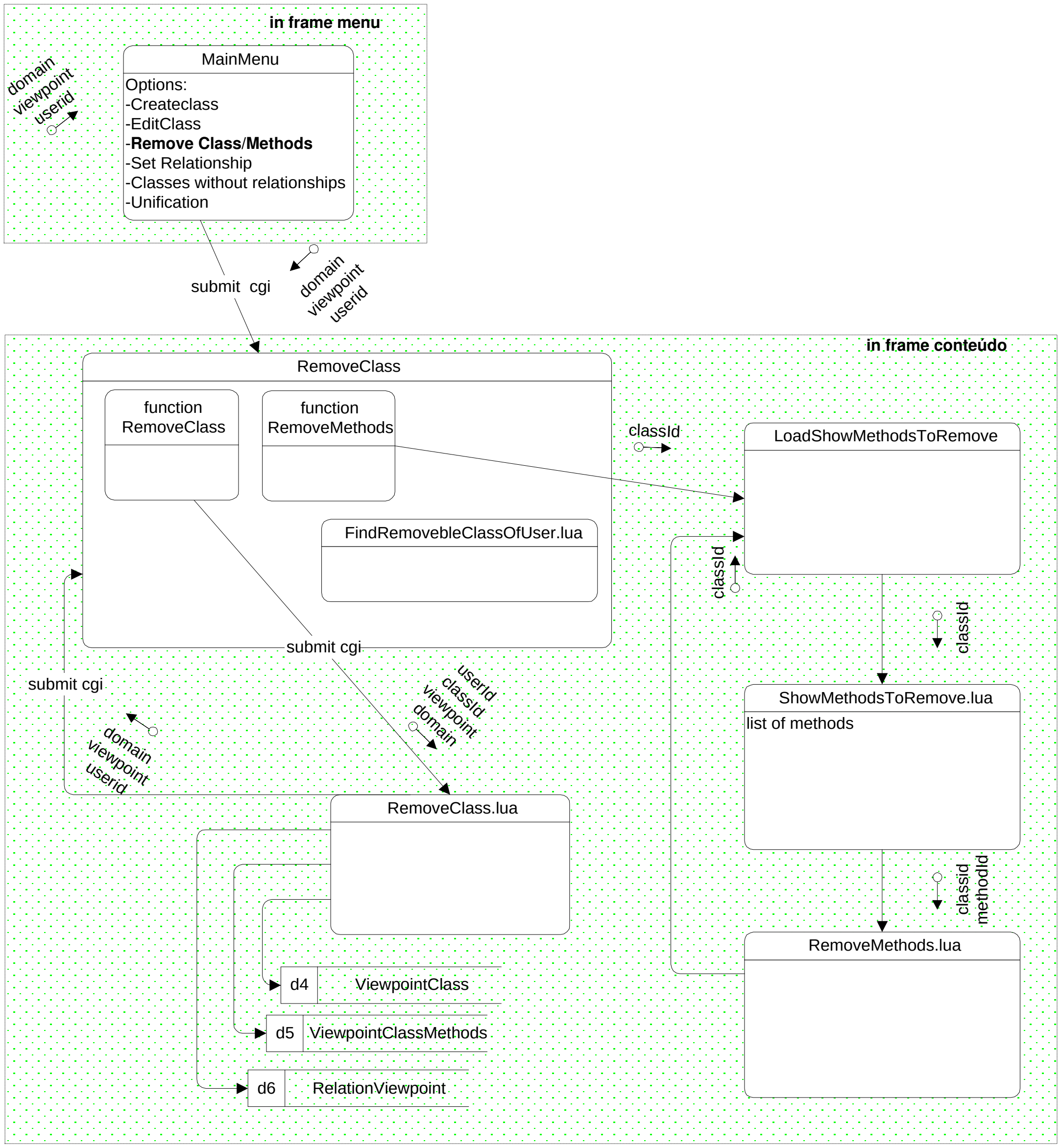

Figura 23. Diagrama de fluxo e classes da fase: Remover classes/métodos

3.6.4 O estabelecimento do relacionamento entre as classes

O relacionamento entre as classes, proporciona que as perspectivas comecem a ter semântica e também que já se possa pensar em obter o kernel de um framework. A obtenção do Kernel será vista no módulo de unificação.

As figuras 24 e 25 a seguir, mostram o diagrama de fluxo e classes desta fase e um exemplo desta interface, quando o usuário deseja configurar o relacionamento entre as classes.

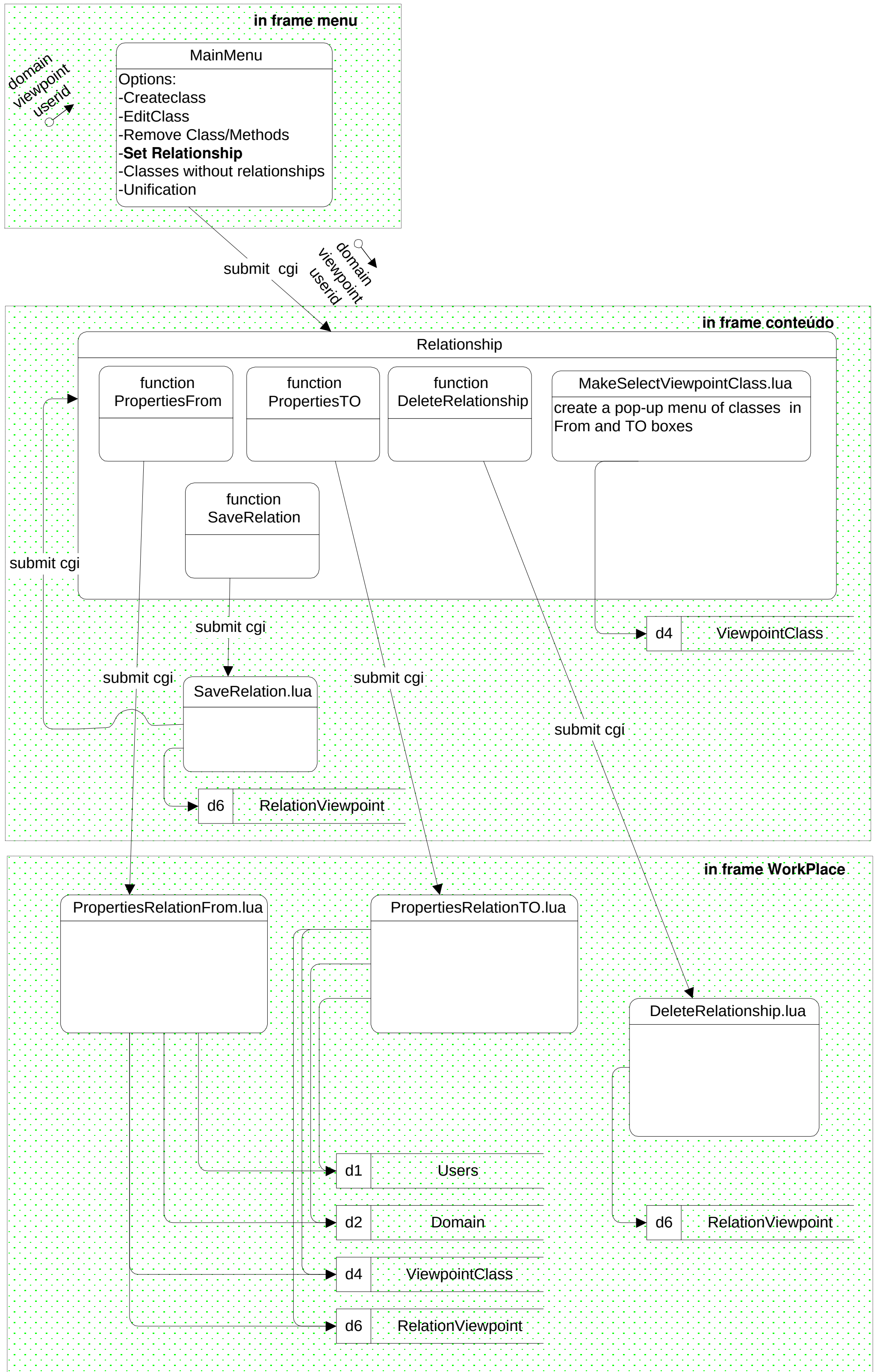

Figura 24. Diagrama de fluxo e classes da fase: Criação de relacionamentos

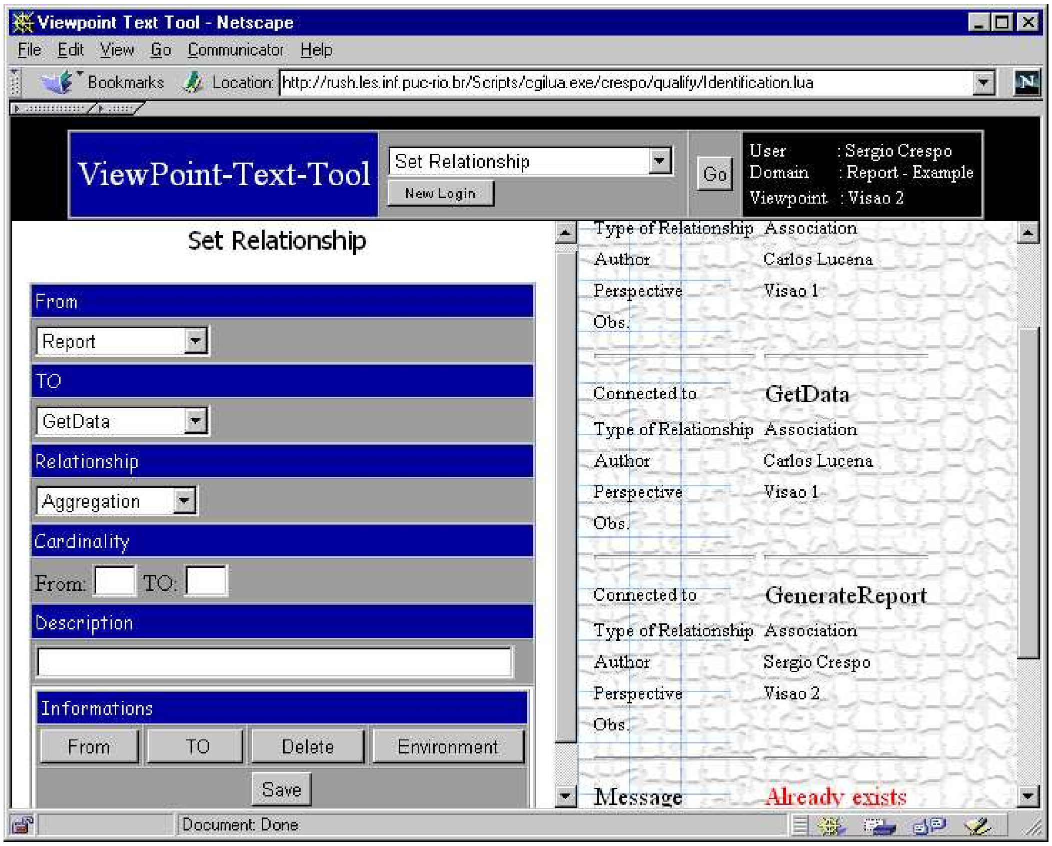

Figura 25. Relacionamento entre classes

A figura 25, apresenta a interface que possibilita ao usuário realizar o relacionamento entre as diversas classes por ele definidas. Inicialmente o frame a direita não apresenta qualquer informação, estas serão vistas caso algum botão no frame a esquerda seja pressionado.

A interface apresenta as seguintes informações: classe origem (From) classe destino (TO), tipo do relacionamento a ser utilizado que são: *Gen-Spec*, *Whole part*, *Instance* e *Message*, cardinalidade entre os relacionamentos para origem e destino e uma descrição do relacionamento. Esta descrição é meramente para efeito de documentação, de forma a facilitar um pouco mais o entendimento do relacionamento criado por outro usuário. Na área de informações, constam três botões: From, TO e Environment.

O usuário tem as seguintes possibilidades de informação a partir desta interface:

♦ Saber que classes são chamadas pela classe selecionada em From. Para isto basta selecionar uma classe no menu pop-up que contém o nome de todas as classes que o usuário pode selecionar para um relacionamento e clicar no botão From. A figura acima apresenta um exemplo desta operação. Neste caso a classe de nome *Report* já possui relacionamentos com as seguintes classes:

GetData onde o autor é Sergio Crespo e o relacionamento utilizado é *Gen-Spec* na perspectiva *WebMaster* e *GeneratedReport* onde o relacionamento utilizado é *Gen-Spec*, o autor é Sergio Crespo na perspectiva *WebMaster*. Note que neste caso, aparece uma mensagem informando ao usuário, neste caso Sergio Crespo, que este relacionamento já existe. Desta forma o sistema não permite que outro relacionamento seja criado acidentalmente. Para a criação de outro relacionamento o usuário deverá remover este relacionamento e após isto feito criar um novo.

♦ Saber quem são as classes que chamam ou estão relacionados com a classe selecionada em TO. Para isto basta clicar no botão TO e desta forma poder-se-a saber quais as classes que possuem algum relacionamento com a classe selecionada.

Sempre que um relacionamento é estabelecido o ambiente procura em todo o domínio se não existe um outro anterior em que as classes selecionadas em From e TO sejam as mesmas e identifica o tipo de relacionamento utilizado. Se este novo relacionamento em outra perspectiva tentar atribuir um outro tipo de relacionamento, por exemplo: trocar de Gen-Spec para Instance, é exibida uma mensagem de alerta para o usuário informando que ele não poderá alterar este tipo de relacionamento. Isto indica que as duas classes por ele utilizadas foram importadas de outra perspectiva e desta forma deve-se preservar o tipo de relacionamento de elas tinham entre si.

3.6.5 A Procura de classes sem relacionamentos

O ambiente oferece ainda a possibilidade do usuário saber quais são as classes que não possuem qualquer tipo de relacionamento. Isto impede que existam classes mortas dentro do ambiente, uma vez que o processo de criação de classes é dissociado do processo de relacionamento entre elas. O usuário pode criar uma série de classes e posteriormente realizar os relacionamentos.
As figuras 26 e 27 mostram respectivamente o diagrama de fluxo e classes para essa fase, bem como a interface já implementada.

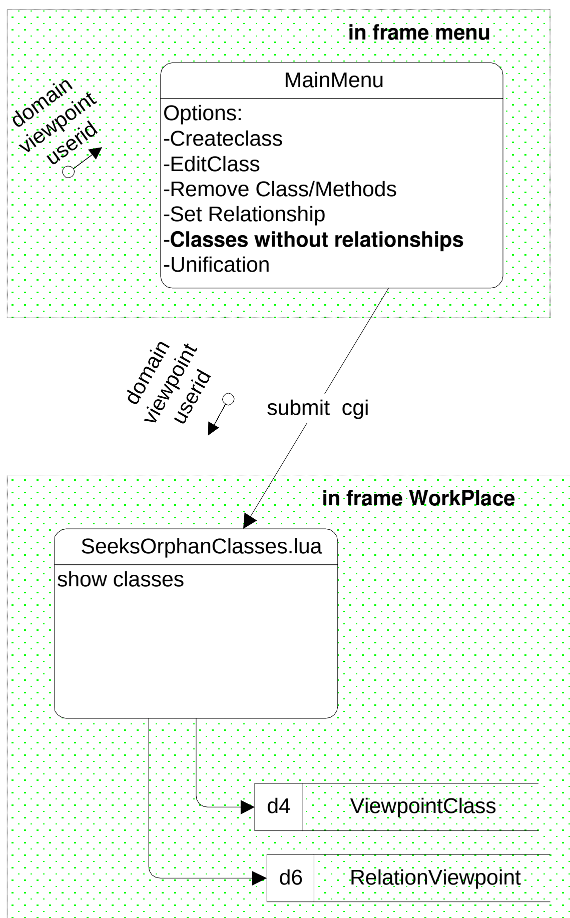

Figura 26. Diagrama de fluxo e classes da fase: Classes sem relacionamentos.

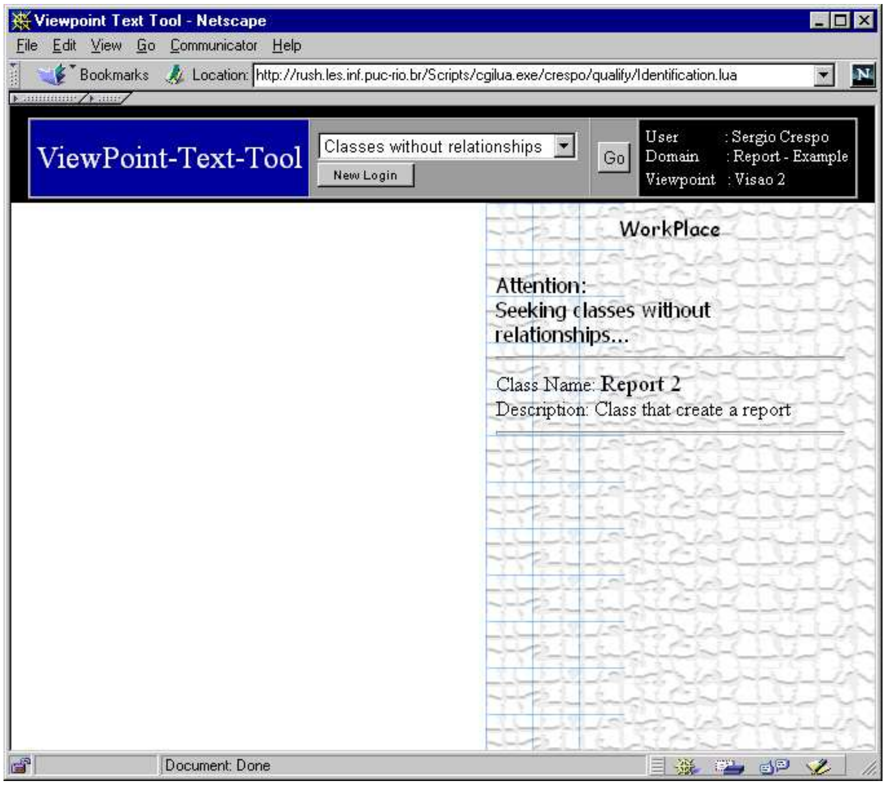

Figura 27. Classes sem relacionamentos

Com a informação apresentada na figura 27 o usuário poderá dirigir-se ao módulo de relacionamento e criar novos relacionamentos de forma a que este problema não mais ocorra.

O resultado da utilização da ferramenta chamada "Gerência de Perspectivas" nos permite que tenhamos a criação de várias perspectivas com suas respectivas classes sem que nenhuma violação ao método tenha sido cometida e para isso as pré-condições abaixo são implementadas pela ferramenta.
Pré-condições dos 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.

Este conjunto de classes será utilizado no processo de unificação, que será detalhado a seguir.

3.7 MÓDULO DE UNIFICAÇÃO

Esse módulo é responsável por realizar a unificação de todos os viewpoints gerando as seguintes informações:

- Kernel da aplicação
- os relacionamentos Hot-spots e
- os dados para a visualização do "Template-hook Framework Model".

O Kernel da aplicação é obtido através da análise das estruturas comuns encontradas em todos os viewpoints, gerando uma representação em uma base de dados relacional apropriada.

Os hot-spots são os elementos não comuns em todos os conjuntos de viewpoints. Estes elementos representam aspectos distintos de um framework e são representados pelo ambiente através de uma relação chamada hot-spot.
O Template-hook Framework Model é gerado a partir das informações obtidas do Kernel da aplicação e de todas as propriedades presentes nos vários Hot-spots obtidos dos conjuntos de viewpoints. Para a geração deste modelo são utilizadas as seis propriedades já vistas em 2.4. A seguir é mostrada uma representação utilizando tabelas de um banco de dados relacional para expressar o diagrama da figura 5.

FROM	**Method**	**vp**	**TO**	**Method**
Report	report(db,query,file)	1	GetData	recordset=getData(db,query)
Report	report(db,query,file)	2	GetData	recordset=getData(db,query)
Report	report(db,query,file)	1	GenerateReport	generateReport(recordset, **htmlfile**)
Report	report(db,query,file)	2	GenerateReport	generateReport(recordset, **rtffile**)
GenerateReport	GenerateReport(recordse, **htmlfile**)	1	FormatReport	format(file)
GenerateReport	GenerateReport(recordse, **rftfile**)	2	FormatReport	format(file)

Tabela 3. Tabela com os viewpoints (visão parcial)

Na tabela acima podemos fazer as seguintes leituras:
A primeira linha nos diz que existe uma classe de nome Report que pertence ao primeiro conjunto de viewpoints com method "report(db,query,file)" e que possui uma ligação com outra classe de nome GetData que pertence ao mesmo conjunto de viewpoint com métodos "recordset=getData(db,query)".
A segunda linha nos mostra a mesma informação porém com classes pertencentes a outro viewpoint, neste caso o segundo viewpoint. Note-se que em ambos os conjuntos tem-se a mesma classe com os mesmos métodos. Isto nos proporciona a informação de que são objetos comuns, e que por isso deverão fazer parte do Kernel do framework.
Nas linhas três e quatro podemos notar que as informações dos viewpoints são as mesmas da coluna "From", porém os objetos conectados possuem diferentes métodos. Isto nos permite concluir que haverá a necessidade de se flexibilizar e utilizar um relacionamento hot-spot e um hook na classe que mudou de método.
A seguir, utilizando-se as propriedades necessárias para se realizar a unificação descrita em 2.2, pode-se ver a representação em tabelas dos objetos transformados.

FROM	**Method**	**hs**	**TO**	**Method**
Report	Report(db,query,file)	0	GetData	Recordset=getData(db,query)
Report	Report(db,query,file)	1	GenerateReport	GenerateReport(indefinido)
GenerateReport	GenerateReport(indefinido)	1	FormatReport	format(file)

Tabela 4. Tabela de unificação – visão parcial

A tabela acima nos informa o seguinte:
Na primeira linha, temos um objeto Report conectado a outro objeto GetData que não implementam nenhum relacionamento hot-spot (hs=0).
Na segunda linha encontramos o mesmo objeto chamado Report conectado a outro objeto chamado GenerateReport que implementa um relacionamento hot-spot (hs=1).
Como podemos ver todo o processo de geração do framework é desencadeado a partir dos conjuntos iniciais de viewpoints. E mais, o módulo gera como artefato de saída um modelo de um framework baseado em template-Hook, que será a entrada para o módulo de Instanciação de perspectivas.
A figura 28 a seguir, mostra de forma esquemática o diagrama de fluxo e classes da unificação dos viewpoints.

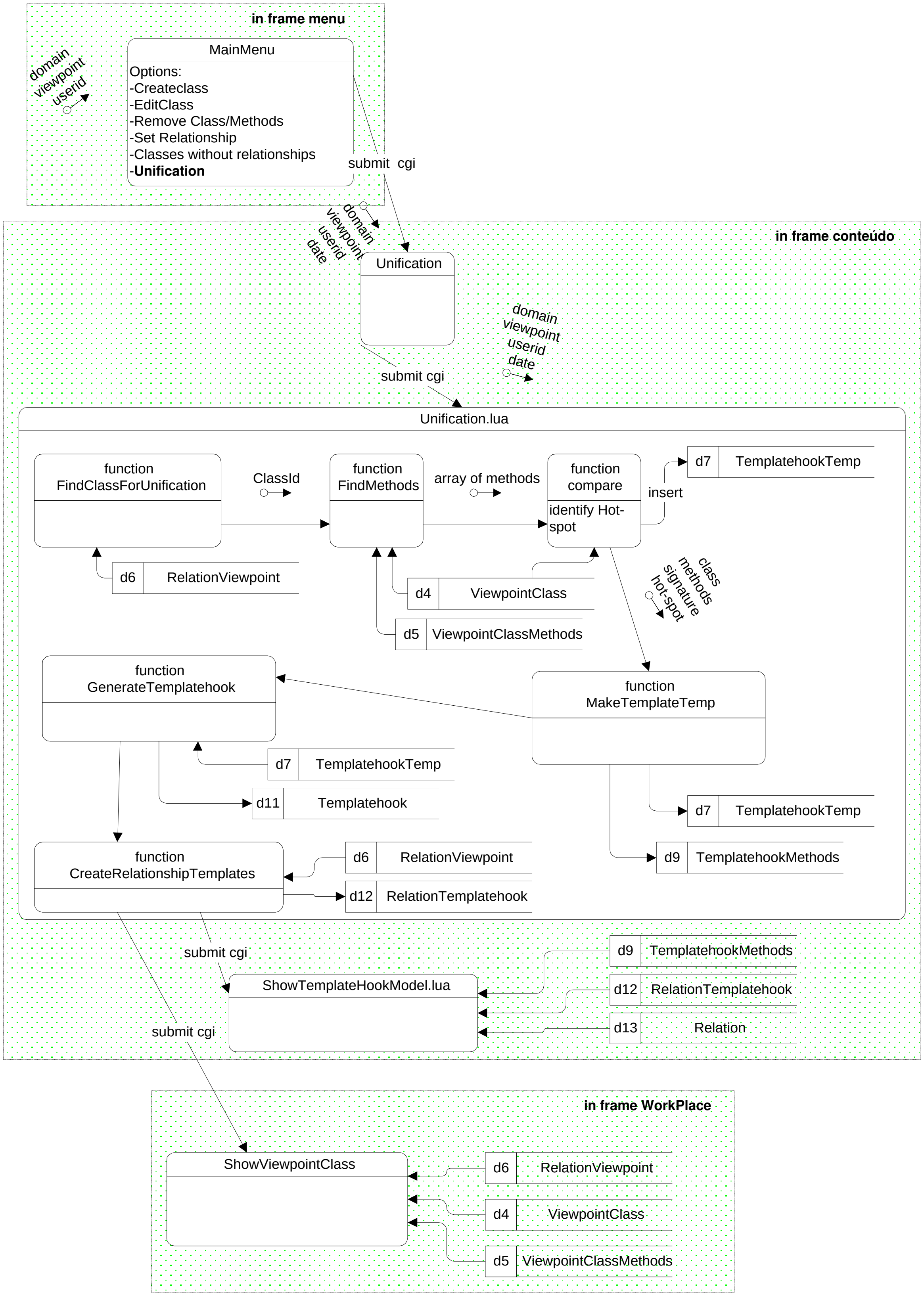

Figura 28. Diagrama de fluxo e classes da unificação dos viewpoints

Abaixo pode-se ver o algoritmo para unificação.

```
// Algoritmo de unificação
// recebe como parâmetros:
// * ID do Usuário = IdUser

// realiza o join entre tabelas
juncao = agrupa("RelationViewpoints", ViewpointClass")
if (juncao ==0) "erro juncao" fim.


totalrecord = recordcoutn("RelationViewpoint")

// guarda dados processados para posterior inclusão no template
define array vet[totalrecord]

tabela = LerLinhadaTabela()
if tabela == Nil then exit
user = tabela[IdUser]

while (LerLinhadaTabela() != Nil)
   F1nan = tabela[ClassNameFrom]
   F1met = tabela[MethoFrom]
   F1var = tabela[VariablesFrom]
   F1imp = tabela[ImplementationFrom]
   F1id  = tabela[IDFrom]
   F1des = tabela[DescriptionFrom]
   //
   T1nan = tabela[ClassNameTO]
   T1met = tabela[MethoTO]
   T1var = tabela[VariablesTO]
   T1imp = tabela[ImplementationTO]
   T1id  = tabela[IDTO]
   T1des = tabela[DescriptionTO]
   //
   tabela = LerLinhadaTabela()  // pega próxima linha da tabela
   //
   F2nan = tabela[ClassNameFrom]
   F2met = tabela[MethoFrom]
   F2var = tabela[VariablesFrom]
   F2imp = tabela[ImplementationFrom]
   F2id  = tabela[IDFrom]
   F2des = tabela[DescriptionFrom]
   //
   T2nan = tabela[ClassNameTO]
   T2met = tabela[MethoTO]
   T2var = tabela[VariablesTO]
   T2imp = tabela[ImplementationTO]
   T2id  = tabela[IDTO]
   T21des = tabela[DescriptionTO]
   //
   // verifica se as classes são as mesmas
   //
   if (F1nam == F2nam and T1nam == T2nam) then
      //inicia a comparação de métodos, variáveis e implementações
      // as funções retornam as string:
      // "undefined" se houve alguma mudança ou
      // a própria string se nada mudou

      compmetFrom = ComparaMetodo(F1met, F2met)
      compmetTO   = ComparaMetodo(T1met, T2met)
```

```
      compimpFrom = ComparaImplementation(F1imp, F2imp)
      compimpTO   = ComparaImplementation(T1imp, T2imp)
      compvarFrom = ComparaVariables(F1var, F2var)
      compvarTO   = ComparaVariables(T1var, T2var)

      if (compmetFrom == "undefined" or compimpFrom =="undefine" or compvarFrom
=="undefined" or compmetTO ==
          "undefined" or compimpTO =="undefine" or compvarTO =="undefined")
then

         HS = 1 a relacao eh hotspot
      else
         HS = 0
      end
      // inclui os dados processados em um vetor auxiliar para depois de
terminado o processamento, os dados do
      // vetor sejam incluídos em Template_Hook_Framework_model.

      incluiVet(F1nam, F2nam, compmetFrom, compmetTO, compimpFrom, compimpTO,
compvarFrom, compvarTO, user, date())

   else
      tabela = LerLinhadaTabela()
   end
end
// monta Template_Hook_Framework_model.
// nada mais eh do quer ler uma linha do veto e inseri-la em uma tabela na
mesma ordem em que aparece no vetor
// o vetor tem o mesmo formato da tabela Template_Hook_Framework_model.

for (i=1 to totalrecord)
   mota(vet[1])   // somente indicado o parâmetro, em realidade são mais
parâmetros
end
//
return
//===================== enf of program=====================//
```

O algoritmo em linhas gerais funciona da seguinte forma:

Faz-se uma junção de duas tabelas *RelationViewpoints* e *ViewpointClass* de forma a obter uma tabela resultante conforme aparece na figura 25 chamada "*Unification_Process_temp*". Para cada usuário é criada a tabela "*Unification_Process_temp*" seguida de seu ID. Isto impede que se outro usuário ao mesmo tempo disparar o processo de unificação, não ocorra um sobreposição de dados em uma única tabela. Nota-se que um campo data esta presente em ambas as tabelas "*Unification_Process_temp*" e "*Template_hook_Framework_Model*". Desta forma, sempre que o usuário desejar uma nova unificação, o sistema irá procurar remover todos os dados criados para essa data para depois voltar a incluí-los já alterados. Ainda é possível obter os dados de dias anteriores, visto que o sistema armazena essas interações. Caso o usuário deseje, no modo de Administração, poderá remover os dados de dias anteriores.

Uma representação parcial desta tabela pode ser vista na tabela 5:

FROM	**Method**	**vp**	**TO**	**Method**
(i) **Report**	report(db,query,file)	1	GetData	recordset=getData(db,query)
(i+1) **Report**	report(db,query,file)	2	GetData	recordset=getData(db,query)
Report	report(db,query,file)	1	GenerateReport	generateReport(recordset, htmlfile)
......	......	...	.....	......

Tabela 5. "Unification_Process_temp" dados parciais

Após a junção, comparam-se os métodos, variáveis de instâncias e as implementações das classes de From e TO de duas linhas consecutivas da tabela.
Se houver alguma diferença, então os valores de retorno dos dados comparados nas funções serão "undefined" e HS=1. Estes dados são inseridos em um vetor e compara-se de novo as próximas linhas.
Quando toda a tabela for comparada, passa-se à fase de leitura de todo o vetor e realiza-se a inclusão de seus dados em uma tabela chamada "Template_hook_Framework_Model". Esta tabela tem todas as informações necessárias para que se gere o artefato "Template hook Framework Model". Neste caso também se realiza um busca por dados com data igual a data em que esta sendo feita a operação, removendo-os antes de incluí-los. Também é possível obter-se os dados de interações passadas.
Uma representação parcial desta tabela pode ser vista na tabela 6:

FROM	Method	hs	TO	Method
Report	Report(db,query,file)	0	GetData	Recordset=getData(db,query)
Report	Report(db,query,file)	1	GenerateReport	GenerateReport(indefinido)
....	.....	...	....	.....

Tabela 6. "Template Hook Framework Model" dados parciais

O produto final desta etapa são os dados de entrada para que se possa realizar o processo de instanciação de perspectivas, que será detalhado a seguir. A figura 29 apresenta o resultado da unificação do exemplo da figura 3.

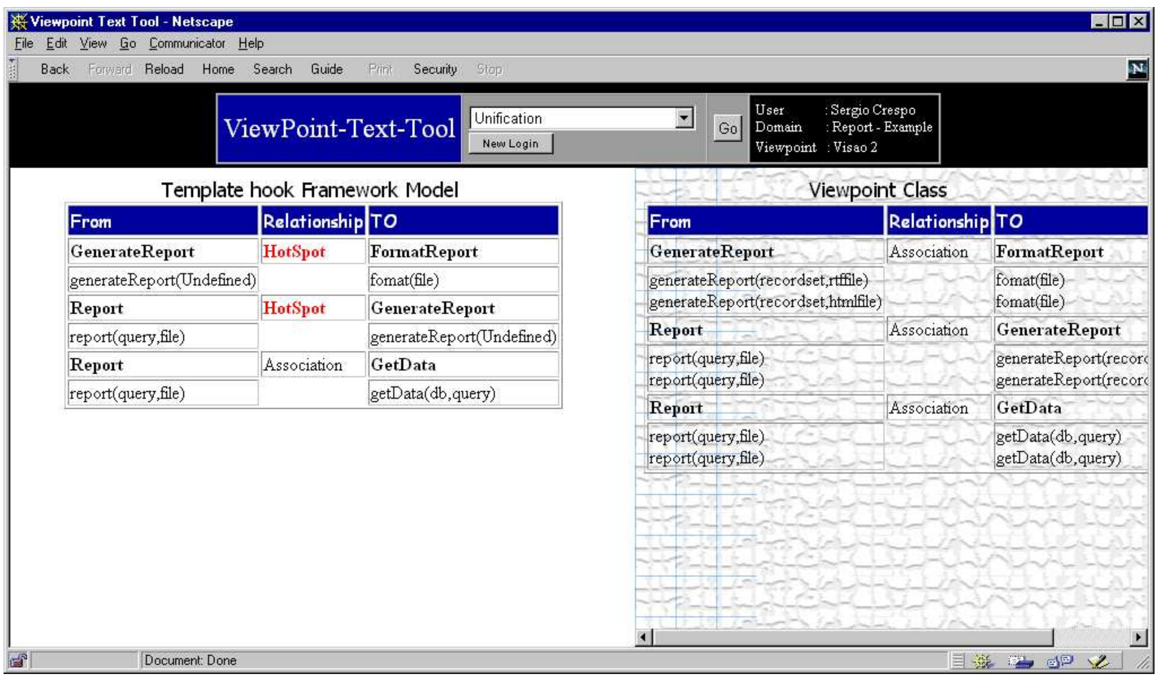

Figura 29. Unificação dos viewpoints

3.8 INSTANCIAÇÃO DE PERSPECTIVAS

Esse módulo utiliza como input um modelo de framework baseado em template-hook. O principal objetivo aqui é eliminar os relacionamentos tipo hot-spot trocando-os por um pattern apropriado. Para isto o módulo utiliza duas ferramentas:

- Hot-spot Card Tool e

* Design Pattern Essential Catalog Tool.

A ferramenta Hot-spot Card Tool é responsável por permitir que o usuário defina que propriedades de flexibilização escolhendo entre as duas classes para que estas sejam implementadas da melhor maneira. Para isso a ferramenta utiliza um catalogo de patterns de forma a poder localizar e substituir o relacionamento hot-spot e suas classes conectadas por um pattern apropriado. A figura 30 nos mostra um exemplo da ferramenta.

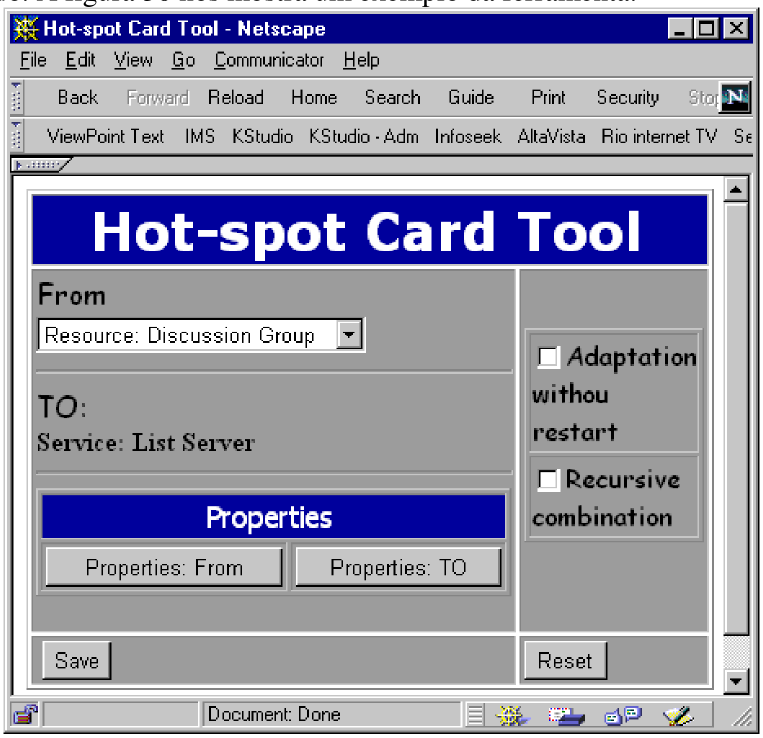

Figura 30. Hot-spot Card Tool

Note que os templates ficarão disponíveis para que se possa definir propriedades para o relacionamento hot-spot entre eles. Caso já exista o relacionamento, este já aparecerá marcado e o sistema não permitira que ele seja redefinido para uma mesma perspectiva.

A ferramenta Hot-spot Card Tool possui as seguintes funcionalidades:

* Seleção do template apresentado após o objeto hook ao qual está relacionado.
* Permite definir as propriedade de flexibilização que são necessárias para este relacionamento.
* Mostra as propriedades dos objetos template e hook.

O produto final deste módulo é a geração dos dados para que se possa obter como saída um artefato tipo Modelo de Framework Baseado em Pattern. Este artefato apresenta agora um diagrama OMT com vários patterns conectados gerados a partir das definições das propriedades de flexibilização presentes nos relacionamentos hot-spot. Em realidade realiza-se a troca dos hot-spots por patterns apropriados. Esta ferramenta trabalha em conjunto com a ferramenta Design Pattern Essential Catalogo Tool.

A ferramenta Design Pattern Essential Catalogo Tool permite que se localize um apropriado pattern em função de um conjunto das propriedades de flexibilização presentes nos relacionamentos hot-spot.

Esta ferramenta apresenta as seguintes funcionalidades:

* Adicionar um novo item
* Editar um item
* Localizar um item

- Remover um item
- Adicionar o código fontes da implementação do Pattern
- Editar o código fonte do Pattern.

Cada Pattern que compõe o catálogo possui um meta-dado em formato XML, com as seguintes informações:

```
<?XML version=’1.0’?>
<s:schema>
<elementType id=”domain”>
        <string/>
<elementType id=”viewpoint”>
        <string/>
<elementType id=”className”>
        <string/>
<elementTtype id=”author”>
        <any/>
<elementType id=”pre-condition”>
        <any/>
<elementType id=”Pattern”>
        <element type=”#domain”/>
        <element type=”#viewpoint”/>
        <element type=”#className”/>
        <element type=”#author”/>
        <element type=”#pre-condition”/>
        <element type=”#pos-condition”/>
        <attribute name=”properties” type=”ENUMERATION”
        values=” ’Adaptation without restarting’ ,’ Recursive combination’ ”>
</s:schema>
```

A definição de um Meta-dado para cada item do catálogo permitirá que outros ambientes utilizem este catálogo. Veremos agora como se dá a autoria compartilhada de frameworks dentro deste ambiente.

3.9 DESIGN COOPERATIVO COM O APOIO DO AMBIENTE

As possibilidades de design cooperativo com o apoio do ambiente são bem fortes pelos seguintes motivos:

O ambiente não utiliza nenhum valor já gerado para um novo processamento.

Cada nova classe incluída em um determinado viewpoints torna-se imediatamente disponível para todos os usuários deste domínio de aplicação.

O ambiente baseia-se sempre nos viewpoints como ponto de partida para qualquer processamento solicitado pelo usuário.

O ambiente utiliza rotinas em CGI-BIN para disparar os processos dos diversos usuários que pertencem às várias perspectivas de um framework. Isto permite gerar diferentes códigos executáveis em função do número de classes que compõem os diversos viewpoints. Por exemplo: se em um dado momento *T1* o ambiente contabiliza X usuários e Y classes no total e algum usuário gera uma aplicação executável a partir destes viewpoints esta aplicação poderá ser diferente em um momento *T2* com $Y + n$ classes no total. Isto é possível pois o conjunto de viewpoints variou.

Desta forma, partindo sempre dos viewpoints pode-se ter n versões de protótipos sem que isso impacte no desenvolvimento da aplicação.

O próximo passo a seguir é a geração de código em virtude da existência modelo de framework baseado em patterns. O módulo que realiza esta tarefa é chamado módulo de framework baseado em patterns e utiliza para isso uma ferramenta chamada *Pattern-based Framework Tool*.

3.10 MÓDULO DE FRAMEWORK BASEADO EM PATTERNS

Esta ferramenta possibilita a geração de modelos baseados em patterns através do modelo de framework baseado em template-hook através de versões.

Todos os templates-hooks são armazenado e um dos atributos destes objetos é a data em que ele foi gerado. Isto nos possibilita a geração de diversas instâncias de frameworks a partir de uma determinada data. Isto, nos permite também realizar uma comparação entre os diversos modelos com base na história.

Este módulo permite, através da utilização de uma uma ferramenta *chamada Pattern-based Framework Tool*, a edição e ou inclusão dos métodos e operadores das classes envolvidas no framework utilizando-se uma linguagem de domínio que permitirá a realização das transformações necessárias para a geração da aplicação.

Uma configuração padrão já consta do modelo obtida a partir dos patterns utilizados na composição do framework. Essa ferramenta é necessária para que se possam fazer os ajustes necessários que irão permitir uma maior flexibilização do ambiente gerado. A ferramenta encontra-se em fase de desenvolvimento.

3.11 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. A figura 31 representa o que foi dito.

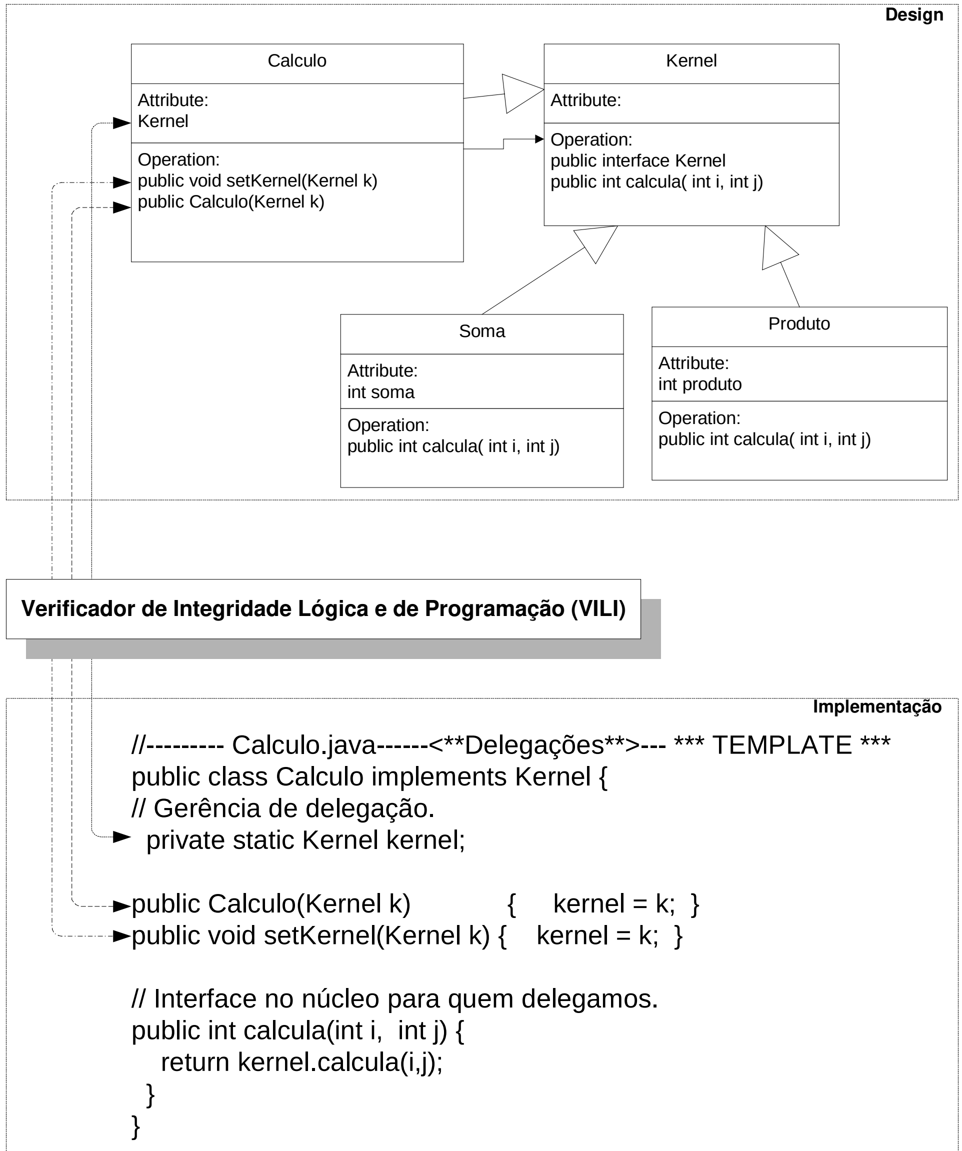

Figura 31. Verificação de inconsistência

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.

3.12 UNIFICAÇÃO EM DIFERENTES PLAFAFORMAS

Estamos analisando o suporte que o ambiente irá proporcionar para a unificação de viewpoints em diferentes plataformas, conforme pode ser visto na figura 32.

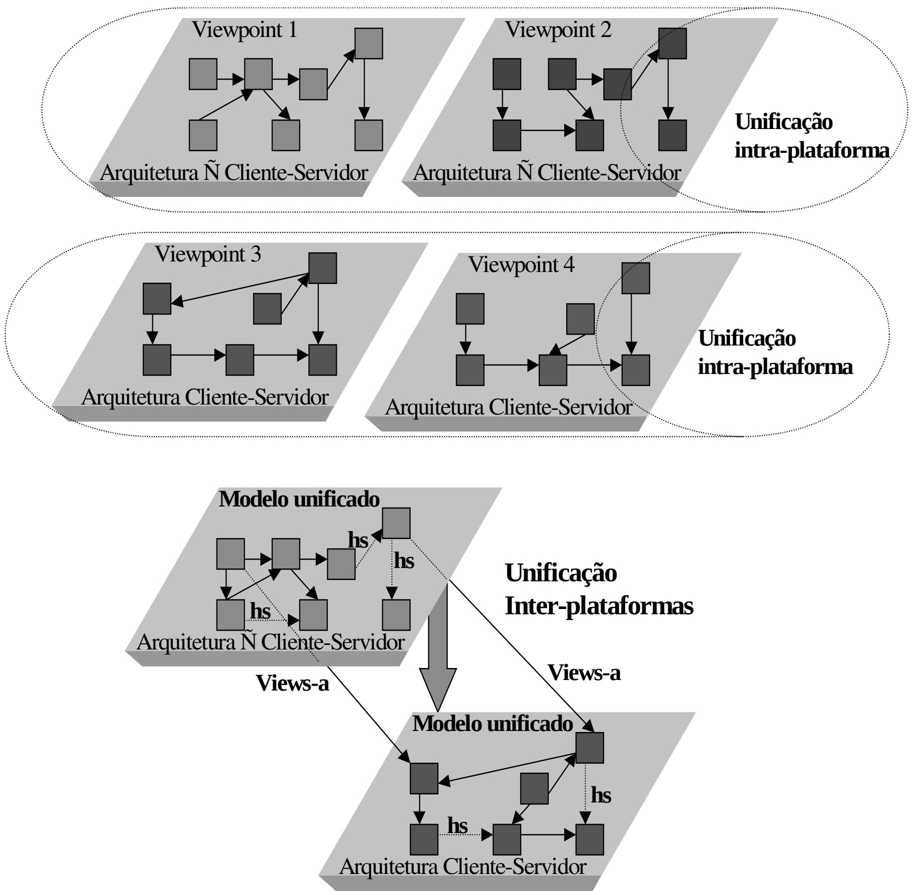

Figura 32. Unificação em várias plataformas

Para a unificação de plataformas diferentes, estamos utilizando o relacionamento Views-a [2][15][31][36][37][40], 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[32] 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. 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.

4. 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.

A contribuição deste trabalho concentra-se em:

- Na viabilização prática do método proposto[33][34][35] e na sua extensão através da definição de meta-dados para as classes, viewpoints e frameworks desenvolvidos;
- No trabalho compartilhado na definição de frameworks que o ambiente oferece;
- Na portabilidade do ambiente, visto que sua interface única com o usuário é um browser;
- Na identificação do kernel e dos pontos de flexibilização do framework (tarefa nem sempre trivial, dependendo do número de classes envolvidas);
- 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;
- Na 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;
- Na existência de um dicionário de métodos, informando para o usuário: onde o método é utilizado, que classes o utilizam, sua interface, autor, implementação;
- Na unificação de design com plataformas diferentes através do uso da relação Views-a;
- Na 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.
- Na 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.

5. REFERÊNCIAS BIBLIOGRÁFICAS

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. Alencar, D. Cowan, S. Crespo, M. F. Fontoura, and C. J. Lucena, "A Viewpoint-based Design Method: A Case Study in Frameworks for Web-based Education", Technical Report, CS-98, University of Waterloo, Ontario, Canada, 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. M. F. Fontoura, E. H. Hermann, 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, 1998.
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. James Rumbaugh, "Relational Database Design Using an Object-Oriented Methodology", Communications of the ACM, 31(4): 417, April 1988.
10. James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, "Object-Oriented Modeling and Design", Prentice Hall, Englewood Clifs, NJ, 1991.
11. H. A. Schmid, "Systematic Framework Design by Generalization", Communications of the ACM, Volume 40, Number 10, October 1997.
12. R. Wirfs-Brock, B. Wilkerson, and L. Wiener, "Designing Object-Oriented Software", Englewood Cliffs, NJ, Prentice-Hall, 1990.
13. EDUCON/NLII - IMS: http://www.imsproject.com
14. Bent Bruun Kristensen. Roles: Conceptual abstration theory and practical language issues. Special Issue of theory and Practice of Object Systems (TAPOS) on Subjectivity on Object-Oriented systems, 1996.
15. 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.
16. Meta Content Framework Using XML, http://www.w3.org/TR/NOTE-MCF-XML/
17. 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.

18. W. Pree, “Framework Patterns”, OOPLSA’97 ,Sigs Books and Multimedia, 1997.
19. Anna M. Hester, Renato Borges and Roberto Ierusalimschy. Cgilua: A multi-paradigmatic tool for creating dynamic WWW pages. In XI Simpósio Brasileiro de Engenharia de Software, 1997.
20. David Flanagan, “ JavaScript: The Definitive Guide, 3rd Edition 3rd Edition”, June 1998, O'Reilly & Associates, Inc.
21. Scott Isaacs ,"Inside Dynamic Html (Microsoft Programming Series)", Paperback, Microsoft Press, 1998.
22. D. Roberts and R. Johnson, “Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks” in “Pattern Languages of Program Design 3”, Addison-Wesley, 1997.
23. P. Hudak, “Building Domain-Specific Embedded Languages”, Computing Surveys, 28A(4), ACM, 1996.
24. Light, R., “Presenting XML”, Sams.net publishing, 1997.
25. http://www.imsproject.org/metadata/MDexec.html
26. http://www.imsproject.org/metadata/MDintro.html
27. http://www.w3.org/TR/WD-rdf-syntax/
28. http://www.sgmlsource.com/
29. http://www.w3.org/Archives/Public/w3c-sgml-wg/
30. Douglas C. Schmidt, Mohamed Faya, and Ralf Johnson, “Software Patterns”, Communications of the ACM, 39(10), October 1997.
31. 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.
32. 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.
33. Crespo, S.; Da Fontoura, M.F.; P. de Lucena, C.J., "Using Viewpoints, frameworks, and domain-specific languages to enhance software reuse", European Reuse Workshop - ERW'98, Madrid, Spain, 1998.
34. 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.
35. 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.
36. 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).

37. 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.

38. S. Crespo, M.F. Fontoura, and C. J. Lucena, “AulaNet - An Object-Oriented Environment for Web-based Education", International Conference of the Learning Sciences (ICLS'98), Atlanta, USA, 1998.

39. C. J. Lucena, H. Fuks, R. Milidiú, L. Macedo, N. Santos, C. Laufer, M. Ribeiro, M. F. Fontoura, R. Noya, S. Crespo, V. Torres, L. Daflon, and L. Lukowiecki, “AulaNet - An Environment for the Development and Maintenance of Courses on the Web", International Conference on Engineering Education 98, ICEE'98, Rio de Janeiro, Brazil, August 1998.

40. 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.