FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

EVERTON SIMÕES DA MOTTA

APRIMORAMENTO, VALIDAÇÃO DE UM PROCESSO DE DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS – PROFT/PU E CRIAÇÃO DE UMA FERRAMENTA DE APOIO

MARÍLIA
2008

EVERTON SIMÕES DA MOTTA

APRIMORAMENTO, VALIDAÇÃO DE UM PROCESSO DE DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS – PROFT/PU E CRIAÇÃO DE UMA FERRAMENTA DE APOIO

Trabalho de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação (Área de Concentração: Engenharia de Software).

Orientador: Profº. Drº. VALTER VIEIRA DE CAMARGO

MARÍLIA
2008

MOTTA, Everton Simões

Aprimoramento, validação de um processo de desenvolvimento de software orientado a aspectos – ProFT/PU e criação de uma ferramenta de apoio / Everton Simões da Motta; orientador: Valter Vieira de Camargo. Marília, SP: [s.n.], 2008.

53 f.

Trabalho de Curso (Graduação em Ciência da Computação) - Fundação de Ensino “Eurípides Soares da Rocha”, mantenedora do Centro Universitário Eurípides de Marília – UNIVEM.

1. Programação Orientado a Aspectos 2. Engenharia de Software 3. Processos de Desenvolvimento

CDD: 005.1

EVERTON SIMÕES DA MOTTA

APRIMORAMENTO, VALIDAÇÃO DE UM PROCESSO DE DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS – PROFT/PU E CRIAÇÃO DE UMA FERRAMENTA DE APOIO

Banca Examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da computação do UNIVEM/F.E.E.S.R, para obtenção do Grau de Bacharel em Ciência da Computação.

Resultado:

ORIENTADOR: _____

Profº. Drº. Valter Vieira de Camargo

1º EXAMINADOR: _____

2º EXAMINADOR: _____

Marília, ____ de _____ de 2008.

À minha família

AGRADECIMENTOS

Em primeiro lugar a Deus por me dar força e coragem para superar os desafios.

À meu orientador Valter por sempre me incentivar quando eu estava desanimando e acreditou em mim mesmo nas horas que nem eu acreditava.

À toda minha família, pela confiança que depositaram em mim, pois sem ela não estaria aqui hoje, principalmente a meus pais que com seus ensinamentos me passaram valores importantes que levarei por toda minha vida.

À todos os meus amigos; e não posso esquecer da minha namorada Flávia que teve paciência e me suportou todo esse tempo.

A alegria está na luta, na tentativa, no sofrimento envolvido
e não na vitória propriamente dita.

Mahatma Gandhi

MOTTA, Everton Simões. **Aprimoramento, validação de um processo de desenvolvimento de software orientado a aspectos – ProFT/PU e criação de uma ferramenta de apoio**. 2008. 53 f. Trabalho de Curso (Graduação em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2008.

RESUMO

A presente monografia tem como objetivo o estudo da programação orientada a aspectos e de processos de desenvolvimento de software orientado a aspectos existentes na literatura, conseqüentemente identificar suas qualidades e imperfeições, assim adquirindo conhecimento para aprimorar e validar o processo de desenvolvimento de software orientado a aspectos – ProFT/PU. Também criar uma ferramenta para auxiliar os desenvolvedores que utilizarão este processo. O ProFT/PU é um processo que visa orientar o desenvolvedor por todo o desenvolvimento, desde a fase inicial até o término do projeto. O principal objetivo do trabalho é facilitar a aplicação deste processo, visto que suas atividades, atualmente devem ser realizadas de forma manual. O fornecimento de uma ferramenta de apoio, além de agilizar o desenvolvimento também diminuir erros na inserção de dados nas tabelas sugeridas em fases do processo.

Palavras-chaves: Programação Orientado a Aspectos. Engenharia de Software. Processos de Desenvolvimento

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

INTRODUÇÃO

Contextualização

A Engenharia de Software vem evoluindo a cada dia, tendo como um de seus velhos objetivos a separação adequada de interesses existentes no código-fonte de um sistema (DIJKSTRA, 1976). Contribuindo com a separação de interesses transversais sugiram a Programação Orientada a Aspectos (POA) (KICZALES *et al*., 1997) e consequentemente novas linguagens de programação, como exemplo a linguagem AspectJ (KICZALES *et al*., 2001), tornando possível a implementação separando os interesses-base dos transversais, organizando-os em módulos independentes, o que era difícil usando apenas as técnicas tradicionais como as oferecidas pela Programação Orientada a Objetos que mantinham os interesses espalhados e misturados por todo o código-fonte do sistema.

Com esse avanço, surgiu a necessidade de revisar os processos, métodos e técnicas utilizadas até o momento, tendo que adaptá-las a essa nova maneira de "pensar". Então se inicio um novo processo de evolução na Engenharia de Software, trazendo novos conceitos, técnicas de modelagem de requisito (BANIASSAD *et al*., 2006; CLARKE e BANIASSAD, 2005; JACOBSON e Ng, 2004; ARAÚJO e MOREIRA, 2003; RASHID *et al*., 2002; ARAÚJO *et al*., 2002; GRUNDY, 1999), linguagens de modelagem para projeto (BERG *et al*., 2006; CLARKE e BANIASSAD, 2005; ALDAWUD *et al*., 2003; KATARA e KATZ, 2003; ZAKARIA *et al*., 2002; STEIN *et al*., 2002; PAWLAK *et al*., 2002; GRUNDY, 2000; SUZUKI e YAMAMOTO, 1999), técnicas de teste (ZHOU *et al*., 2004; ZHAO, 2003; LEMOS *et al*, 2004a; LEMOS *et al*., 2004b; ZHAO, 2002) e o processo de desenvolvimento de software orientado a aspectos chamado ProFT/PU (CARMARGO, 2006), que é estudado mais a fundo por este trabalho, esse processo visa conduzir o desenvolvedor de sistemas desde as fases iniciais até o término do projeto.

Motivação

Há uma carência na literatura de um processo de desenvolvimento de software orientado a aspectos maduro que auxilia o desenvolvedor por todo seu projeto. Além de

ser um campo atual também é um campo promissor para novas pesquisas. Destaca-se também a dificuldade em se conduzir um processo iterativo e incremental como o ProFT/PU, realizando as atividades manualmente, pois com a construção de uma ferramenta, não só agilizará a realização dessas atividades mas também diminuirá a probabilidade de ocorrer erros, que normalmente são cometidos quando tais atividades são realizadas de forma manual.

Objetivo

Este trabalho de pesquisa tem como objetivo principal a construção de uma ferramenta que apóia as fases iniciais do processo de desenvolvimento de software orientado a aspectos denominado ProFT/PU, permitindo facilitar e agilizar o condução desse processo e estudar alguns processos de desenvolvimento que abordam a programação orientada a aspectos, afim de validar e aprimorar o ProFT/PU.

CAPÍTULO 1 – PROGRAMAÇÃO ORIENTADA A ASPECTOS

1.1 Separação de Interesses

A separação de interesses (*separation of concerns*) na engenharia de software, refere-se à limitação da cognição humana em lidar com a complexidade de um software (DIJKSTRA, 1976). A proposta sugerida teve uma ampla aceitação pelas comunidades de Ciência da Computação e de Engenharia de Software, para resolver problemas complexos de desenvolvimento de sistemas mesmo Dijkstra não sendo um pesquisador de Ciência Cognitiva. O principal foco da proposta é dar maior atenção em uma porção ou interesse isoladamente, tendo-se ciência que está ignorando outras partes do problema temporariamente.

Existem constantes pesquisas na área de engenharia de software em relação à separação de interesses, desde a sua criação. Novas construções sintáticas eram fornecidas a cada novo paradigma de programação, buscando uma modularização mais adequada dos interesses com o objetivo de decompor grandes e complexos sistemas em módulos bem definidos altamente coesos. Assim gerando software com maior qualidade em relação ao entendimento e reuso de códigos e as inevitáveis futuras manutenções.

Os interesses foram basicamente divididos em interesses bases e interesses transversais: o que diz respeito à funcionalidade principal do sistema é chamado de interesse base, como exemplo de um sistema que realiza determinado cálculo, tem a funcionalidade principal o cálculo em si e não quem esta operando o sistema, onde será armazenado o resultado da operação entre outros. Já os interesses transversais podem ser tanto funcionais, como regras de negócio, quanto não-funcionais, como alguns exemplos clássicos encontrado na literatura: persistência, sincronização, interação de componentes, distribuição, gerenciamento de registros (*logging*) e controle de acesso (KICZALES *et al*., 1997). Foi adotado o termo “interesse transversal” fazendo analogia com a implementação com técnicas convencionais de programação que entrecorta transversalmente os módulos, afetando vários outros, causando o entrelaçamento (*tangling*) e espalhamento (*spreading*) de código de diferentes interesses. Quando se encontra em um mesmo módulo o código de um determinado interesse misturado com o código de um diferente interesse, ocorre o entrelaçamento. Quando se encontra em vários módulos do sistema código de um determinado interesse ocorre o espalhamento.

Na próxima sessão é apresentado um paradigma de programação criado com a evolução da engenharia de software, buscando uma separação de interesses mais adequada.

1.2 Paradigma de Programação Orientado a Aspectos

Em um artigo publicado por Kiczales em 1997 foi apresentada a Programação Orientada a Aspectos, com a idéia principal de melhorar o desenvolvimento de sistemas projetados de maneira convencional que causam problemas de entrelaçamento e espalhamento de código por todo o sistema, distinguindo os “interesses transversais” dos “interesses bases”, que em paradigmas de desenvolvimento de software, como o da programação orientado a objeto consiste na existência de apenas um tipo de módulo ou construção sintática que encapsula vários tipos de interesses (TARR *et al*., 2001), as classes da programação orientada a objeto são um exemplo, fazendo com que interesses que envolvem restrições não sejam adequadamente encapsulados e fiquem misturados ao interesses base. A POA foi criada como uma extensão da programação orientada a objetos, utilizando todos os seus conceitos e adicionando novas abstrações que contribuem para a separação de interesses (separation of concerns), por exemplo: aspectos (*aspects*), adendos (*advices*), pontos de junção (*join points*), conjunto de junção (*pointcut*) e declarações inter-tipos (*inter-type declarations/introductions*), possibilitando a implementação de interesses transversais em módulos aspectuais e os interesses base de forma convencional em classes normais.

O termo aspecto é usado para denotar a abstração da POA, é uma unidade modular que encapsula o interesse transversal isolando-o dos interesses base, projetada para afetar outros módulos do sistema.

Pontos de junção são locais bem definidos no programa onde os aspectos irão atuar, as chamadas e execuções de métodos são exemplos destes pontos. Assim estabelecendo o relacionamento entre os aspectos e as classes. Tais pontos também podem ser específicos em uma determinada linguagem orientada a aspectos, através de um conjunto de junção, que é o mecanismo que especifica os pontos de junção.

Adendo é um construtor semelhante ao método de uma classe, onde é definido o comportamento do aspecto, que será executado sempre que um ponto de junção associado é alcançado. Os três tipos de adendos existentes são: adendos anteriores

(*before*) que sua execução acontece antes do prosseguimento da computação; adendos posteriores (*after*) que sua execução acontece após a execução do ponto de junção e antes do retorno do controle para o chamador; adendos de substituição (*around*) podem tomar o controle do código base, podendo ou não devolve-lo.

Podem ser utilizadas as declarações de inter-tipos quando existe a necessidade de inserir novos métodos e atributos em classes base do sistema.

Os inter-tipos operam de forma estática, diferente dos adendos e conjuntos de junção, que operam de forma dinâmica. Assim possibilitando o aspecto afetar a estrutura estática ou dinâmica de classes e objetos. Também podem ser inseridos nos aspectos métodos e atributos internos, como classes O.O. e não somente especificações de elementos que entrecortam classes de um sistema.

A POA contém três propriedades básicas (KICZALES *et al*., 1997; ELRAD *et al*., 2001):

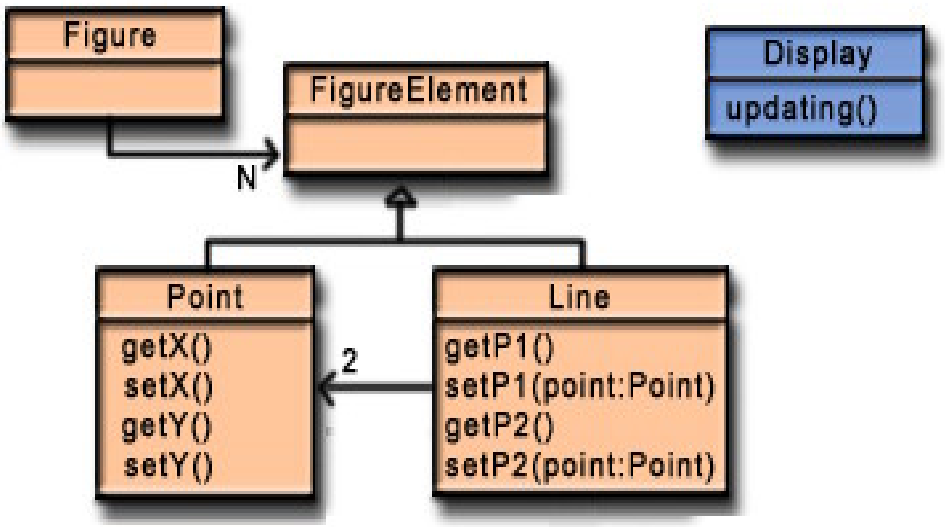
a-) dicotomia aspectos-base: refere-se à adoção de uma distinção clara entre classes e aspectos. Os sistemas orientados a aspectos são decompostos em classes e aspectos, onde são modularizados os interesses transversais nos aspectos e os interesses base nas classes;

b-) inconsciência: propriedade desejável da programação orientada a aspectos. É a idéia de que, para serem entrecortados por aspectos os componentes não precisam ser preparados (ELRAD *et al*., 2001). Por esta propriedade, os componentes não percebem quando poderão ser afetados pelos aspectos. Mas pesquisas atuais começam a apontar vantagens quando existe consistência da existência dos aspectos pelo código-base (KICZALES e MEZINI, 2005; GRISWOLD, *et al*., 2006);

c-) quantificação: é a capacidade de escrever declarações unitárias, que possam afetar vários pontos de um sistema (ELRAD *et al*., 2001). Por esta propriedade é possível fazer tipos de declarações como: “uma ação é executada sempre que determinada condição for verdadeira em um sistema”.

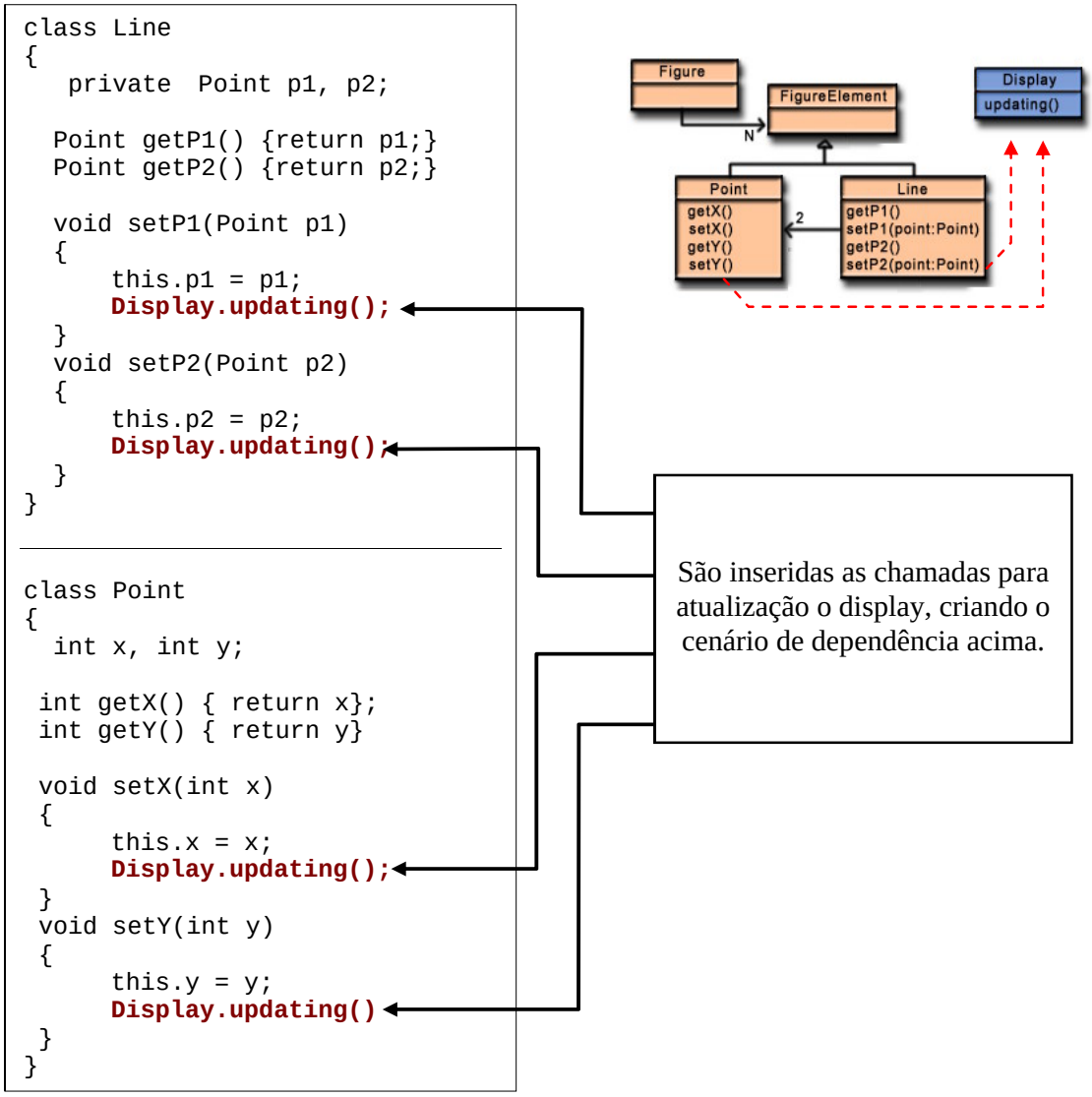
Na Figura 1 é ilustrado um diagrama de editor de gráfico, onde existe um interesse transversal ELRAD *et al*. (2001), que consiste na exibição da imagem (*classe Display*); será demonstrado um exemplo de implementação usando programação orientada a objeto, de maneira convencional, onde os interesses transversais não são devidamente separados do código base; e um outro exemplo de implementação usando programação orientada a aspecto demonstrando como deverá ser a separação dos interesses.

Figura 1 – Diagrama de Classes de um Editor Gráfico



Na Figura 2 é mostrado um exemplo usando programação orientada a objetos, resulta em inserir uma chamada ao método updating() da classe Display em todos os métodos que alteram o posicionamento de um ponto ou de uma linha (*métodos set*()*). Note que o código desse interesse ficou entrelaçado com o código de cada um dos métodos e espalhado por vários métodos do sistema.

Figura 2 – Implementação Orientada a Objetos do Interesse de Atualização do Display



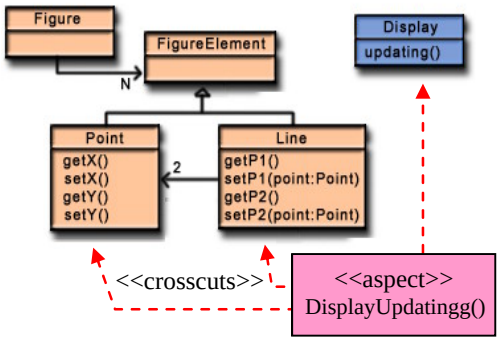
Na Figura 3 é mostrado um exemplo onde foram retiradas todas as chamadas do método *updating()* das classes Line e Point como havia no exemplo anterior, deixando o código base limpo, mantendo somente o que diz respeito a sua funcionalidade e criado um aspecto que entrecortará o código base no mesmo ponto onde foram retirados as chamadas do método, assim executando o mesmo comportamento de atualização do display.

Figura 3 – Implementação Orientada a Aspectos do Interesse de Atualização do Display

```
class Line
{
   private  Point p1, p2;

  Point getP1() {return p1;}
  Point getP2() {return p2;}

  void setP1(Point p1)
  {
      this.p1 = p1;
  }
  void setP2(Point p2)
  {
      this.p2 = p2;
  }
}
```

```
class Point
{
  int x, int y;

 int getX() { return x};
 int getY() { return y}

 void setX(int x)
 {
      this.x = x;
 }
 void setY(int y)
 {
      this.y = y;
 }
}
```

```
aspect DisplayUpdating
{
    pointcut move():

    call(void Line.set*(Point))
||
    call(void Point.set*(int))

   after() returning: move() {
    Display.updating();
  }
}
```



Foi criado um aspecto que entrecortará o código base nos mesmos pontos onde existia as chamadas do método *updating()* no exemplo anterior e mantendo o interesse transversal separado do código base; criando um novo cenário de dependência demonstrado na imagem acima.

Na próxima seção são apresentados linguagens de programação, que apóiam o paradigma de programação orientado a aspectos, pois não basta ter um novo paradigma, também é interessante tem uma linguagem onde seja possível implementar as abstrações que esse paradigma oferece.

1.3 Linguagem de Programação Orientada a Aspectos

Para a implementação de um sistema orientado a aspecto existem várias linguagens como: AspectC usadas para as linguagens C, AspectC# usadas para as linguagens C# e AspectJ usadas para as linguagens Java. Por adotar uma abordagem assimétrica, AspectJ (KICZALES *et al*., 2001) foi escolhida para ser utilizada em estudos deste trabalho.

Todos os conceitos da programação orientado a aspectos são implementadas com AspectJ, são modularizados com "aspectos" somente os interesses transversais existentes no sistema e as "classes" são usadas para a implementação do interesses-base.

No Quadro 1 é mostrada um exemplo da implementação de um aspecto que possui um conjunto de junção chamado move(), onde dois pontos de junção estão definidos; ambos são chamados do tipo (*call*), sendo o primeiro à todos os métodos que começam com a palavra "*set*" e estão na classe Line e o segundo somente com a diferença de que os métodos têm que estar na classe Point. Sempre que existir uma chamada a algum método set* das classes Line ou Point, o adendo do tipo after chama o método updating() da classe Display após a execução dos métodos definidos (set*). A declaração returning assegura que a execução do adendo after será efetuada somente se os pontos de junção, do conjunto de junção move(), retornarem de sua execução com sucesso. Dessa maneira, as chamadas ao método updating() da classe Display não existiriam nos métodos set*() das classes Line e Point.

Quadro 1 – Implementação do Mecanismo de Atualização do Display

```
aspect DisplayUpdating
{
    pointcut move():
    call(void Line.set*(Point)) || call(void Point.set*(int));

    after() returning: () {
      Display.updating();
  }
```

(CAMARGO, 2006)

Em AspectJ é possível criar aspectos abstratos, esses aspectos são semelhantemente as classes abstratas em Java, também podem ser criados conjuntos de junção abstratos onde pode-se implementar antecipadamente o comportamento

transversal de um aspecto abstrato sem ter conhecimento em quais pontos de junção e em qual código base irá atuar, é a idéia usada no desenvolvimento de frameworks orientados a aspectos, pensando no reuso de código. Um adendo pode ser definido sobre uma conjunto de pontos de junção abstratos, tais pontos são informados em um aspect concreto que especializa o abstrato.

No Quadro 2 é apresentado um aspecto abstrato que tem função de estabelecer conexões com o banco de dados, é definido um conjunto de junção abstrato nomeado doConnection() e um adendo do tipo after() que atua sobre ele. Sempre será chamado o método ConnectDB() quando algum ponto de junção informados no aspecto concreto que especializar o abstrato for alcançado. Assim, deve ser criado um aspecto concreto para especializar o aspecto abstrato criar algum ponto de junção onde a conexão será estabelecida, assim concretizando o conjunto de junção doConnection(), como mostra o Quadro 3, o aspecto concreto está indicando que em toda execução de um método init()a conexão deve ser estabelecida.

Quadro 2 – Aspecto Abstrato

```
public abstract aspect AbstractAspectConnection
{
       ...
      abstract pointcut doConnection();

      after() : doConnection()
      {
          ConnectDB();
      }
}
```

(CAMARGO, 2006)

Quadro 3 – Aspecto Concreto

```
public aspect AspectConnection extends AbstractAspectConnection
{
   pointcut doConnection(): execution (void *.init(..));
}
```

(CAMARGO, 2006)

Foram criadas diretrizes para a implementação de interfaces de entrecorte em AspectJ, fazendo com que sejam criadas separadas do comportamento transversal que irão ter atuação sobre elas (GRISWOLD *et al.*, 2005). É uma maneira de reorganizar o

código, criando aspectos que contém apenas conjuntos de junção e aspectos que contém apenas adendos que atuam nos conjuntos de junção definidos.

A interface de um sistema é um ponto importante em relação a seus módulos, interface são descrições ou receitas de como os módulos interagem com o resto do sistema (MEYER, 1997 apud (CHAVEZ *et al*., 2005)), (MEZINI e OSTERMANN, 2003 apud (CHAVEZ *et al*., 2005)). Existe um construtor específico na linguagem de programação Java para a implementação de interfaces, criando a possibilidade de expor um sub conjunto do comportamento de um módulo para outro. Para métodos aspectuais é aplicado o conceito de "interface de entrecorte" (*crosscutting interface*), que se resume em um conjunto conhecido como "detalhes de entrecorte" (*crosscutting features*) (CHAVEZ *et al*., 2005) que são comportamentos que podem afetar o código base de forma dinâmica ou estática(CHAVEZ, 2004).

Já na linguagem de programação AspectJ, não existe um construtor específico para a implementação de interfaces de entrecorte, desta forma as implementações deste tipo de interface são feitas no mesmo módulo do comportamento transversal (aspecto), como podemos observar na Quadro 1.

A interface de entrecorte consiste em um conjunto de junção move() com seus respectivos pontos de junção que devem ser expostos do código-base. A implementação dessa interface de entrecorte está no mesmo modulo que possui o comportamento transversal que atua sobre ela.

Existem propostas que sugerem diretrizes de como implementar interfaces de entrecorte em na linguagem AspectJ e o comportamento transversal que deve atuar sobre elas em módulos separados (GRISWOLD et al., 2005). Consiste em uma maneira reaorganização do código, encapsulando os conjuntos de junção em determinados aspectos e em outro encapsulado somente os adendos que atuam naqueles conjuntos de junção.

No próximo capítulo são apresentados processos que auxiliam no desenvolvimento de software orientados a aspectos, visando fornecer parâmetros e diretrizes que guiam os desenvolvedores.

CAPÍTULO 2 – PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS

2.1 Processos

Com a criação do paradigma POA, vários pesquisadores se dedicam aos estudos da influência deste paradigma em fases específicas de um processo de desenvolvimento.

Alguns destes pesquisadores foram estudados neste trabalho, na fase de projeto (BERG *et al*., 2006; CLARKE e BANIASSAD, 2005), AOCE (*Aspect-Oriented Component Engineering*), (GRUNDY, 2000; (*Architectural View of Aspects*) KATARA e KATZ, 2003; SUZUKI e YAMAMOTO, 1999; PAWLAK *et al*., 2002; STEIN *et al*., 2002; ZAKARIA *et al*., 2002; ALDAWUD *et al*., 2003), na fase de implementação (SOARES, 2004), na fase de análise e engenharia de requisitos (GRUNDY, 1999; CLARKE e BANIASSAD, 2005; RASHID *et al*., 2002; BANIASSAD *et al*., 2006) e também existem poucos que abordam todo o ciclo de vida (CLARKE e BANIASSAD, 2005; JACOBSON e NG, 2004).

Grundy (1999) tem seus estudos voltados para o desenvolvimento de software baseado em componentes utilizando métodos de engenharia de requisitos orientados a aspectos. Uma categorização de diversos aspectos de um sistema que um componente provê para os usuários finais ou para outros componentes.

Clarke e Baniassad (2005) focam a fase de análise da abordagem Tema, a partir de requisitos são identificados os temas. Antes é definida uma relação de um tema a cada requisito ou conjunto de requisito. Também são definidos critérios para a identificação de temas nos trabalhos de Clarke e Baniassad (2005), os temas encapsulam interesses transversais que são candidatos a aspectos assim denominados como temas transversais que em geral têm relacionamento com mais de um requisito, temas que encapsulam o interesse base, chamados de temas base, em geral têm relacionamento com apenas um requisito.

A proposta de Araújo e Moreira (2003) tem como objetivo identificar casos de uso aspectuais, sendo estes, funcionais ou não-funcionais. A realização dessa identificação inicia de um documento de requisitos, em que é elaborado um modelo inicial dos casos de uso, analisa-se esse modelo com o objetivo de fatorar as

funcionalidades, Isso é representado por relacionamentos com os estereótipos <<*includes*>>, <<*extend*>> e <<*constrain*>>, assim são identificados os requisitos não funcionais e modelados como casos de uso com esteréotipo <<*NF*>>, que são integrados com os casos de uso funcionais já modelados. O estereótipo <<*constrain*>> é usado na representação do relacionamento entre casos de uso não funcionais e funcionais. Por fim, os casos de uso que restringem mais de um caso de uso ou os que são incluídos ou estendidos por mais de um caso de uso, devem ser identificados como casos de uso candidatos a aspectos.

Rashid *et al*. (2002) propôs o modelo AORE, (*Aspect-Oriented Requirements Engineering*), que relaciona os requisitos a interesses com o objetivo de identificar os interesses candidatos a aspectos, que são os que entrecortam os requisitos. Este modelo também permite separar a especificação dos requisitos aspectuais, requisitos normais e regras de composição em módulos que seguem gabaritos bem definidos.

Muitos dos trabalhos que focam a fase de projeto visam estender técnicas de projeto orientado a objetos e UML, assim incluindo especialidades para permitir a modularização de interesses transversais.

O Tema/UML AOCE (GRUNDY, 2000) que é um destes trabalhos que focam a fase de projetos, tem como base a idéia de que cada componente de um sistema baseado em componentes provê ou usa serviços para/de outros componentes. São empregadas extensões da UML para tornar explícitos os detalhes aspectuais.

O Tema/UML (CLARKE E BANIASSAD, 2005) combina gabaritos da UML com o modelo de projeto orientado a assunto. São usadas algumas regras de composição para compor os projetos. Após serem identificados os temas transversais e bases. Também são seguidos gabaritos que representam genericamente as operações entrecortadas, em diagramas de interação onde é mostrado o comportamento padrão do tema quando combinado com algum tema base e são substituídas pelas operações dos temas bases na composição. Essa abordagem tem como ponto inicial, que padrões de comportamentos transversais existem conforme eles entrecortam os projetos base. Assim permitindo que o comportamento transversal seja projetado sem consciência do código base.

Katara e Katz (2003) abordam a representação dos aspectos como pacotes estereotipados tendo artefatos da UML pertencente a um determinado interesse. A relação dos elementos existente em cada diagrama pode ser, *defines* ou *uses*, podendo ser mapeados para outros elementos em aspectos diferentes. É introduzida a notação de

diagrama de interesse (*concern diagram*), também é permitida a visualização de sobreposições de interesses.

O modelo de projeto *aSide* (CHAVEZ, 2004), é composta por três facetas, a estrutural, a dinâmica e a de composição. A faceta estrutural tem a função de modelar a estrutura dos aspectos e componentes e como se relacionam através de interceptações e mecanismos de composição tradicionais. A faceta dinâmica tem a função de modelar as interações das instâncias dos aspectos em relação aos componentes afetados por esses mesmos aspectos, por fim a faceta de composição busca facilitar a compreensão em um nível mais alto de abstração da combinação dos aspectos com os componentes.

Foi apresentado por Soares (2004) um método de implementação orientado a aspectos integrado com o (*Rational Unified Process* - RUP) (JACOBSON *et al.*, 1999). Algumas modificações na estrutura estática (disciplinas e atividades) e na estrutura dinâmica (fases) do RUP foram propostas pelo autor, para a utilização do método. Na parte dinâmica foi feita a principal modificação, para requisitos não funcionais são realizadas iterações específicas, deste modo, casos de uso que eram implementados em apenas uma iteração são implementados em iterações funcionais onde existe apenas requisitos funcionais como: gerenciamento de dados não persistentes e interfaces com o usuário, e somente após a validação dessas implementações, os requisitos não funcionais como o de persistência, distribuição e concorrência são incorporados em três iterações não funcionais.

A abordagem proposta por Jacobson e Ng (2004) apresenta o desenvolvimento de software orientado a aspectos baseado em casos de uso, visando identificar interesses transversais a partir dos casos de uso de um sistema, implementando-o em módulos separados. São identificados os casos de uso que implementam interesses transversais, principalmente pelos tipos de relacionamentos com outros casos de uso (*include ou extend*), para isso, critérios foram definidos afim de orientar os desenvolvedores na tomada de decisão dos tipos de relacionamentos estabelecidos entre os casos de uso. Um sistema é composto por casos de uso base e transversais, pois um caso de uso representa um interesse do sistema, ao implementar um caso de uso pode ser criada mais de uma classe com seus métodos e atributos relacionados ao interesse desejado. Também foi proposto pelo autor que a composição para a produção de um sistema seja feita através de entidades vazias criadas previamente, de declarações intertipo e em relação a codificação do sistema seja utilizado linguagem de programação orientada a aspectos.

Na próxima sessão é apresentado um novo processo chamado ProFT/PU que tem o objetivo de auxiliar o desenvolvedor de sistemas baseado em aspectos por todo o período do desenvolvimento.

2.2 O Processo ProFT/PU

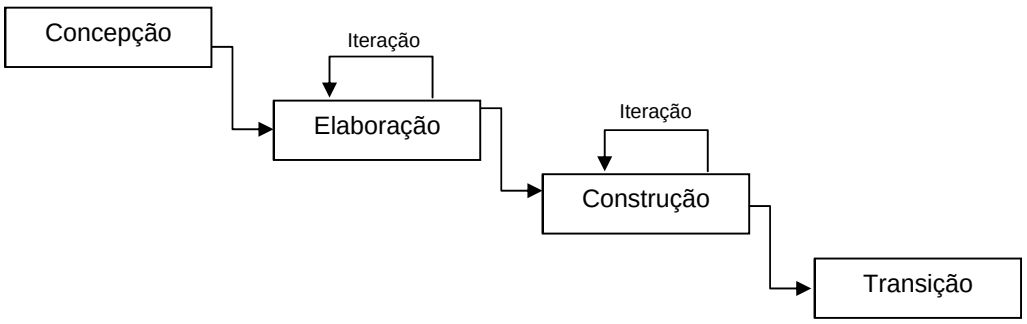
O processo de desenvolvimento de software orientado a aspecto denominado ProFT/PU foi criado em um trabalho apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP – São Carlos/SP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional (CAMARGO, 2006). O ProFT/PU visa auxiliar o desenvolvedor de sistema de software orientado a aspectos por meio de diretrizes e disciplinas por todo o período de desenvolvimento; em relação aos outros processos existentes como o de (CLARKE e BANIASSAD, 2005) e Jacobson e Ng, (2004) são destacados quatro pontos, sendo o primeiro que ele foi gerado da união dos conceitos, critérios e técnicas de modelagem de trabalhos consagrados da literatura e modelados em fases específicas. (CLARKE e BANIASSAD, 2005; JACOBSON e NG, 2004; BANIASSAD *et al*., 2006; KRECHETOV *et al*., 2006; ARAÚJO e MOREIRA, 2003; ARAÚJO *et al*., 2002). O segundo é que ele é baseado na identificação e acompanhamento de interesses Transversais durante todo o decorrer do processo. O terceiro é que ele é apoiado por Framworks transversais em determinadas atividades e o quarto é que ele é um processo iterativo e incremental.

O processo ProFT/PU acompanha a estrutura de fases e disciplinas do Processo Unificado (PU) (JACOBSON *et al*., 1999) apresentado por (LARMAN 2004), seguindo o conceito de ser iterativo e incremental, assim o desenvolvimento de um sistema é realizado em iterações, em que cada iteração é concentrada no desenvolvimento de um subconjunto de casos de uso, até que a implementação de todo o sistema seja finalizada. O ProFT/PU é ligeiramente influenciado pelos frameworks transversais, onde a existência de frameworks transversais que possam ser usados para facilitar a implementação de um interesse transversal são levadas em consideração em algumas atividades como a identificação de aspectos. Entretanto a implementação é independente dos frameworks transversais, assim possibilitando ser feita de forma convencional sem utilizar frameworks.

Como o PU, o ProFT/PU é iterativo e orientado pelos riscos. São acomodadas no interior das fases do PU as atividades que serão realizadas, que estão sendo mostradas na Figura 4. Na fase de concepção é gerada uma visão aproximada do sistema, casos de uso funcionais e não funcionais simples são definidos e algumas estimativas são esboçadas. Na fase de elaboração, são identificados os primeiros interesses transversais; é refinada a visão do sistema; a arquitetura central é implementada iterativamente, é identificado a maior parte dos requisitos do sistema e são selecionados os casos de uso que serão implementados como aspectos. Em cada uma das iterações desta fase, os casos de uso escolhidos para a iteração atual são implementados e inseridos aos casos de uso já desenvolvidos, assim obtendo uma versão executável. Na fase de construção são implementados de maneira iterativa os elementos restantes de baixo risco. Na fase de transição são feitas as implementações propriamente ditas e conduzidos os testes.

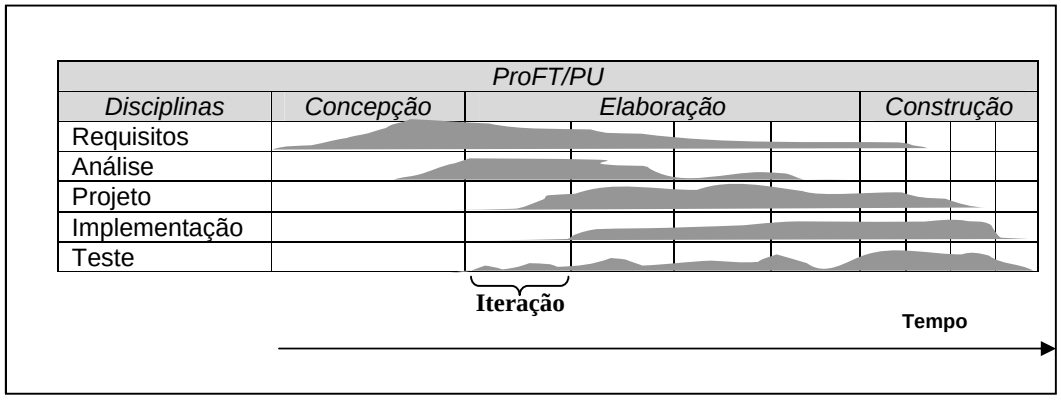
O PU e o ProFT/PU tem como base um conjunto de disciplinas que devem ser seguidas, como exemplo, requisitos, análise, projeto e implementação. São repetidas essas disciplinas em todas as fases, mas com ênfases diferentes conforme a progressão do processo, em cada fase a ênfase em determinada disciplina pode ser maior ou menor. Por exemplo, é dada uma ênfase maior na disciplina de requisitos na fase de concepção do que na fase de construção. Nas fases de elaboração e construção que são iterativas, tem nas primeiras iterações uma ênfase maior nas disciplinas de análise e projeto, e nas últimas iterações a ênfase maior é dada na disciplina de implementação. Na Figura 5 as disciplinas do ProFT/PU são mostradas e a ênfase que deve ser aplicada por todo o desenvolvimento.

Figura 4 – As Fases do PU



(CAMARGO, 2006)

Figura 5 – Visão Geral do ProFT/PU



(CAMARGO, 2006)

O ProFT/PU trabalha com a mesma estrutura de processo do PU, contudo foram criadas novas atividades específicas para a manipulação dos interesses transversais. Estão descritas no Quadro 4 as atividades do ProFT/PU em cada disciplina. As atividades particulares do ProFT/PU estão representadas em letras negritas, e as atividades encontradas em qualquer processo de desenvolvimento de software baseado no PU estão representadas em letras não negritas, mesmo sendo atividades comuns do PU possuem influência das novas atividades ou da existência de frameworks transversais ou até mesmo alguma alteração.

Em uma determinada fase pode ser executada qualquer atividade, porém é mais comum serem executadas as atividades da disciplina Requisitos na fase de Concepção do que na fase de Construção.

Quadro 4 – Atividades do ProFT/PU

Atividades da Disciplina Requisitos
Identificar Atores e Detalhar Casos de Uso Funcionais
Identificar e Especificar Casos de Uso Não-Funcionais
Planejar Iterações
Atividades da Disciplina Análise
Identificar e Registrar Casos de Uso Colaboradores
Identificar e Registrar Casos de Uso Candidatos a Aspectos
Criar Diagramas de Seqüência do Sistema
Definir Contratos das Operações do Sistema
Desenvolver Modelo Conceitual
Atividades da Disciplina Projeto
Identificar Aspectos
Selecionar Aspectos para a Iteração Atual
Projetar Aspectos
Acoplar Aspectos com a Base
Desenvolver Diagramas de Interação
Desenvolver Modelo de Classes de Projeto
Projetar Modelo de Dados
Atualizar Regras de Composição
Desenvolver Modelo de Projeto Composto
Desenvolver Visão da Arquitetura
Registrar Pontos de Junção
Atividades da Disciplina Implementação
Implementar Classes de Domínio
Implementar Banco de Dados
Atualizar Regras de Composição
Implementar Aspectos
Implementar Classes Controladoras
Implementar Interfaces
Testar

(CAMARGO, 2006)

Pode ser encontrado no trabalho de doutorado - Camargo (2006), o processo de desenvolvimento de software orientado a aspectos denominado ProFT/PU com uma grande riqueza de detalhes, ele esta descrito e demonstrado com um exemplo prático.

No próximo capítulo é apresentada com mais detalhes a linguagem XML, que é usada para organizar e armazenar dados gerados pelo FRIT Tool e pela ferramenta de modelagem MagicDraw.

CAPÍTULO 3 – FERRAMENTA DE APOIO AO PROCESSO PROFT/PU

3.1 Linguagem XML

O XML é uma linguagem extensível de formato, e esta sendo vista como uma evolução considerável na internet, desenvolvida pela W3C *(World Wide Web Consortium*), criada para exceder as limitações do HTML, que é padrão adotado nas páginas WEB. A linguagem XML foi definida como um formato universal para a estruturação de dados na Web, então são estabelecidas regras que permitem escrever esses arquivos para possibilitar sua identificação e visualização aos computadores.

Como no HTML também são usadas tags (parâmetros e palavras-chaves) no XML, uma tag é composta por um marcador de início de comando e outro de fim, com a diferença de que em alguns casos, quando é declarada uma tag no HTML e não é colocado seu marcador de fim de comando, mesmo assim a página pode ser exibida, já no XML não pode existir esse tipo de erro; no HTML as tags já estão definidas, por exemplo, para marcar onde inicia e termina um parágrafo é usada a tag , no XML as tags são usadas para demarcar blocos de dado, por exemplo, o programador pode definir que a tag   irá significar o nome, ou idade, ou telefone, enfim ele define o que determinada tag representará, assim possibilitando o programador criar um conjunto de tags para ser usada com determinados comandas funções e aplica-las em documentos Web que desejar.

Quando um programador cria suas próprias tags, é atribuído uma espécie de glossário no arquivo, como o DTD (*Document Type Definition*) e o XML Schema, tornando possível definir que uma determinada tag tenha em seu interior outras tags. No Quadro 5 é mostrado um exemplo prático de um bloco de dado escrito em XML, as tags estão sendo definidas no texto em negrito. Esta imagem ilustra uma pequena parte da capacidade do XML, pode ser encontrado facilmente muitas obras que abordam esse assunto com grande riqueza de detalhes.

Quadro 5 – exemplo de bloco escrito em linguagem XML.

```
<aluno>
    <ra> 35414-7 </ra>
    <nome> Everton Simões da Motta </nome>
    <curso> Bach. em Ciência da Computação </curso>
    <serie> 4º ano </serie>
</aluno>
```

A extensibilidade do XML foi tão bem aceita no mercado, que até mesmo as grandes corporações como a Microsoft, Google, IBM, Oracle, Sun, entre outras vêm incluindo módulos e funções em XML em seus produtos, assim tornando a linguagem cada vez mais difundida em todo o território mundial.

Na próxima seção é apresentada uma ferramenta de modelagem UML utilizada para a criação dos diagramas de caso de uso de um sistema computacional.

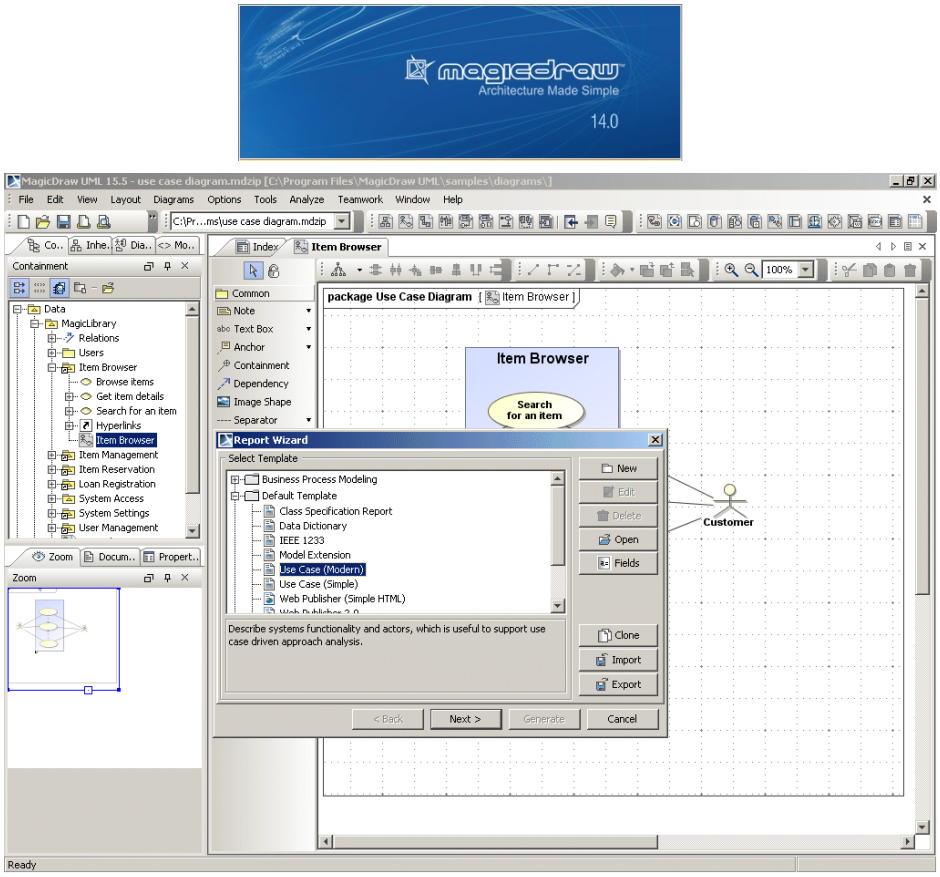
3.2 Ferramenta MagicDraw

MagicDraw é uma ferramenta de modelagem UML visual criada pela Nº Magic Inc, que em 1998 liberou MagicDraw UML Versão 1.0, a primeira aplicação em larga escala completamente desenvolvida em Java; a ferramenta foi projetada para desenvolvedores de software; funciona em conjunto com múltiplas linguagens de programação como (Java, C++, C#, entre outras); prevê a independência em muitos processos de desenvolvimento de software específicos, conforme a necessidade da empresa ou instituição. A facilidade de uso é uma forte característica do MagicDraw, com uma interface gráfica que permite que os usuários utilizem a ferramenta de maneira fácil.

MagicDraw iniciar a partir de qualquer ponto em sua arquitetura e processo de modelagem; não importando se o projeto encontra-se atualmente em uma fase de requisitos ou manutenção; perfis com UML e diagramas personalizados podem estender padrão UML 2; oferece também API aberta, onde pode-se estender funcionalidade para

incluir novos padrões de design, métricas, transformações e outros plugins; possibilita vários programadores trabalharem ao mesmo tempo em um mesmo modelo; e inclui módulos e funções XML em seus produtos.

Figura 6 – Tela do MagicDraw de geração de relatório



Na próxima seção é apresentada uma ferramenta que apóia o processo ProFT/PU auxiliando o desenvolvedor em sua condução.

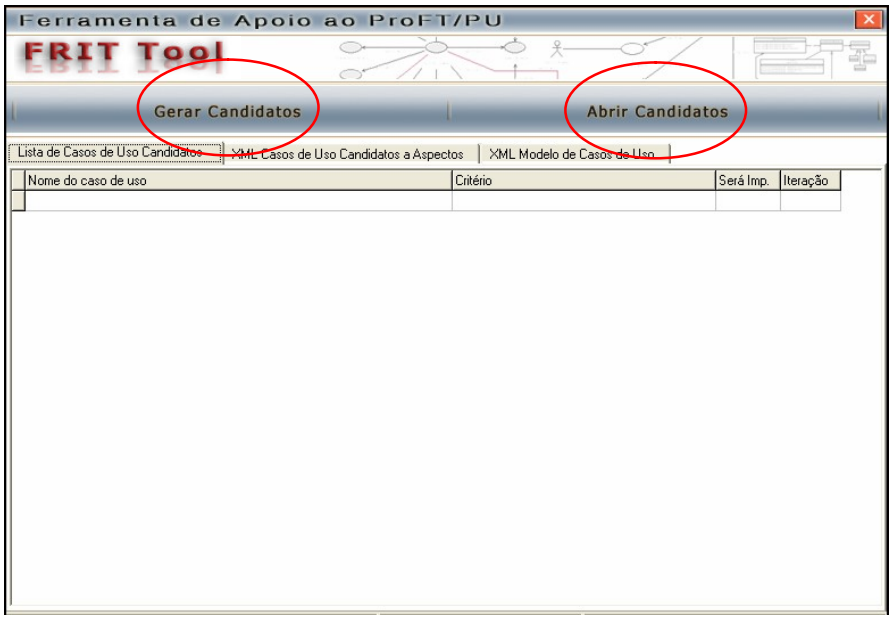
3.3 Ferramenta FRIT Tool

A Ferramenta FRIT Tool propõe apoiar o desenvolvedor de sistemas de software orientado a aspectos que utiliza o processo ProFT/PU. Esta versão enfatiza a atividade "**Identificar e Registrar Casos de Uso Candidatos a Aspectos**" da disciplina "Requisitos" do processo, aplicando as técnicas demonstradas no processo ProFT/PU.

A FRIT Tool identifica os candidatos a aspectos a partir de um arquivo XML que nele está contido o diagrama de caso de uso, gerado pela ferramenta de modelagem MagicDraw. A FRIT Tool abre e lê este arquivo XML, criando um outro arquivo XML contendo os casos de uso candidatos a aspectos e mostra em uma tabela, possibilitando o usuário especificar se determinado caso de uso será realmente implementado com aspectos e em que iteração ele será incluído ao projeto.

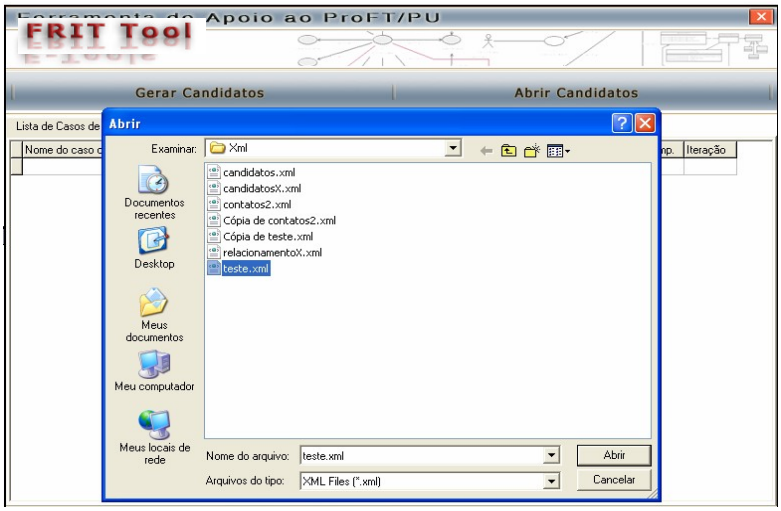
Na Figura 7 é mostrada a tela inicial da FRIT Tool onde percebe-se dois botões; o 'Gerar Candidatos' e o 'Abrir Candidatos' que serão mais bem detalhados no decorrer dessa sessão.

Figura 7 – Tela inicial da Ferramenta FRIT Tool



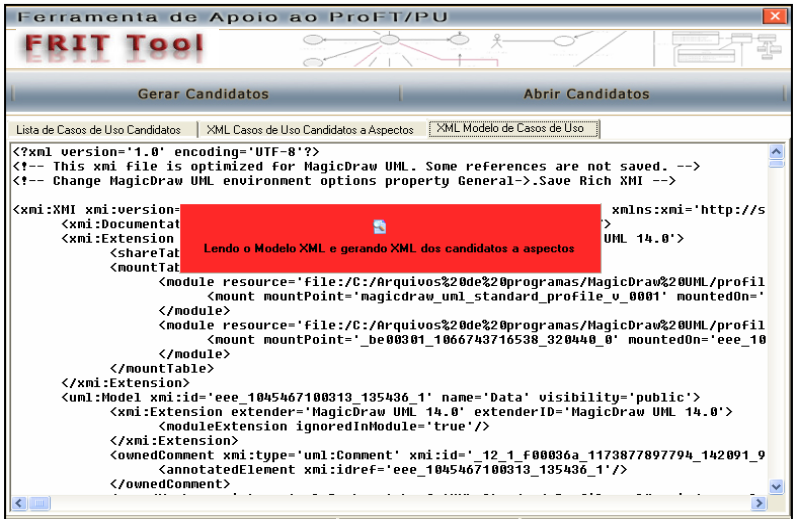
Ao *clicar* no botão ‘Gerar Candidatos’ o sistema abrirá uma nova janela onde o usuário deverá indicar o arquivo XML que contem o diagrama de caso de uso criado pela ferramenta de modelagem MagicDraw. Como mostrada na Figura 8.

Figura 8 – Janela usada para abrir o arquivo XML gerado pelo MagicDraw.



A FRIT Tool abre e percorre todo o arquivo, afim de localizar todos os casos de uso existentes em seu interior, logo após aplica as técnicas propostas no ProFT/PU para definir quais destes casos de uso serão candidatos a aspectos. Como mostrada na Figura 9.

Figura 9 – Tela do FRIT Tool abrindo e percorrendo todo código em XML como o modelo de caso de uso e classificando os casos de uso candidatos a aspectos.



Em seguida é criado um outro arquivo escrito em XML contendo somente os casos de uso que foram considerados candidatos a aspectos que são apresentados em uma tabela ao usuário. A tabela possui quatro colunas, sendo a primeira, os nomes dos casos de uso; a segunda, os critérios aplicados que fizeram com que esse caso de uso fosse incluído à Tabela; a terceira, possibilita o usuário indica quais casos de uso serão realmente implementados com aspectos; e a quarta e última coluna, o usuário indica em que iteração o caso de uso será implementado. Como mostra a Figura 10.

Figura 10 – Tela do FRIT Tool com a tabela dos candidatos a aspectos classificados.

O botão ‘Abrir Candidatos’ é usado para abrir o arquivo XML com os casos de uso candidatos a aspectos já gerados anteriormente e suas respectivas configurações.

CONSIDERAÇÕES FINAIS

Ao decorrer dos estudos conduzidos no presente trabalho destacou-se a dificuldade em determinar quais casos de uso deveriam ser realmente implementados com aspectos, então as pesquisas foram direcionadas para a identificação desses casos de uso.

Foi estuda a eficácia de se aplicar ao ProFT/PU uma matriz para organizar os relacionamentos entre os casos de uso, pois assim possibilita uma melhor visualização que no próprio diagrama.

Com o uso da matriz, foi obtida uma melhor visualização dos relacionamentos entre os casos de uso, assim facilitando a aplicação dos critérios para a escolha dos candidatos a aspectos quando realizada manualmente. Mesmo assim são necessários mais testes para saber se realmente o uso da matriz quando o processo é realizado de forma manual traria mais benefícios que deficiências.

Já com a criação de uma ferramenta em que o desenvolvedor não realizará a aplicação das técnicas de identificação dos candidatos de forma manual, a matriz não faria muito sentido, pois os dados coletados no diagrama para alimentar a matriz seriam suficiente para identificar os candidatos a aspectos, também não usando a matriz o código-fonte da própria ferramenta seria mais simples de ser implementados.

Foi criada uma ferramenta que identifica os casos de uso candidatos a aspectos por meio de um arquivo XML gerado pela ferramenta de modelagem MagicDraw, assim agilizando o processo e diminuindo a margem de erro na aplicação dos critérios de identificação dos candidatos.

Trabalhos futuros deverão ser realizados buscando ampliar a capacidades da ferramenta criada, fazendo com que atue também em outras fases do ProFT/PU. Mais teste para o aprimoramento e validação do ProFT/PU deverão ser feitos.

REFERÊNCIAS

ALDAWUD, O., ELRAD, T., BADER, A. **UML Profile for Aspect-Oriented Software Development.** In: Proceedings of Workshop of Aspect Oriented Modeling with UML of Aspect Oriented Software Development Conference (AOSD), 2003.

ARAÚJO, J., MOREIRA, A. M. D. **An Aspectual Use-case Driven Approach.** In: JISBD, pp. 463–468, 2003.

BANIASSAD, E., CLEMENTS, P.C., MOREIRA, A., ARAÚJO, J., RASHID, A., TEKINERDOGAN, B. **Discovering Early Aspects.** IEEE Software, pp. 61-70, 2006.

BERG, K. VAN DEN., CONEJERO, J.M., HERNÁNDEZ, J. **Identification of Crosscutting In Software Design.** In: Proceedings of the Aspect-Oriented Modelling Workshop (AOM'06) in conjunction with Aspect-Oriented Software Development (AOSD'06), Bonn, Alemanha, 2006.

CAMARGO, V.V. **Frameworks transversais: definições, classificações, arquitetura e utilização em um processo de desenvolvimento de software.** Tese de Doutorado. Instituto de Ciências Matemáticas e de Computação, ICMC-USP – São Carlos/SP, Novembro, 2006.

CHAVEZ, C. V. G., GARCIA, A., KULESZA, U.,SANT'ANNA C., LUCENA, C. **Taming Heterogenerous Aspects with Crosscutting Interfaces**. In: Proceedings of Simpósio Brasileiro de Engenharia de Software (SBES'05), pp 216-231, Uberlândia, MG, Brasil, 2005.

CHAVEZ, C.V.G. A **Model-Driven Approach for Aspect-Oriented Design**. Tese de Doutorado. Departamento de Informática da PUC-Rio, 2004.

CLARKE, S., BANIASSAD, E. **Aspect-Oriented Analysis and Design: The Theme Approach.** Addison-Wesley, 2005.

DEITEL, H. M.; DEITEL, P. J.. **. Java: como programar**. 3ª ed. Porto Alegre: Bookman, 2001. 1201p.

DIJKSTRA, E. W. **A Discipline of Programming.** Prentice-Hall, 1976

ELRAD, T., FILMAN R., BADER A. **Aspect-Oriented Programming**. Communications of the ACM, vol 44, pp 29-32, 2001.

GRISWOLD, W.G., SHONLE, M., SULLIVAN, K., SONG, Y., CAI, Y., RAJAN, H. **Modular Software Design with Crosscutting Interfaces**, IEEE Software, pp 51-60, 2006.

GRUNDY, J. **Aspect-Oriented Requirements Engineering for Component-Based Software Systems.** In: 4th Symposium on Requirements Engineering, Limerick, Ireland, IEEE, pp. 84-91, 1999.

GRUNDY, J. **Multi-Perspective Specification, Design and Implementation of Software Components Using Aspects.** In: Journal of Software Engineering and Knowledge Engineering, 20, 6, pp. 713-724, 2000.

JACOBSON, I., BOOCH, G., RUMBAUGH, J. **The Unified Software Development Process.** Reading, MA. Addison-Wesley, 1999.

JACOBSON, I., NG, P. **Aspect-Oriented Software Development With Use Cases.** Addison-Wesley, 2004.

KATARA, M., KATZ, S. **Architectural View of Aspects.** In: 2nd Conference on Aspect-Oriented Software Development (AOSD), Boston, ACM, pp. 1-10, 2003.

KICZALES, G., LAMPING, J., MENDHEKAR, A., MAEDA, C., LOPES, C., LOINGTIER, J., IRVING, J. **Aspect Oriented Programming.** In: Proceedings of 11 ECOOP. pp. 220-242, 1997.

KICZALES, G., HILSDALE, E., HUGUNIN, J., KERSTEN, M., PALM, J., GRISWOLD, G. **Getting Started With AspectJ**, Communications of the ACM, vol 44, No. 10, pp.59-65, 2001.

KICZALES, G., MEZINI, M. **Aspect-Oriented Programming and Modular Reasoning**. In: Proceedings of International Conference on Software Engineering (ICSE'05), St. Louis, Missouri, USA, pp. 49-58, 2005.

KRECHETOV, I., TEKINERDOGAN, B., GARCIA, A., CHAVEZ, C., KULESZA, U. **Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design.** In: Aspect-Oriented Modelling Workshop (AOM'06), in

conjunction with Aspect-Oriented Software Development Conference (AOSD'06), Bonn, Alemanha, 2006.

LARMAN, Craig. . **Utilizando UML e padrões : uma introdução à análise e ao projeto orientado a objetos.** Porto Alegre: Bookman, 2002. 492p.

LEMOS, O.A., VINCENZI, A.M.R., MALDONADO, J.C., MASIERO, P.C. **Teste de Unidade de Programas Orientados a Aspectos.** In Anais do 18º. Simpósio Brasileiro de Engenharia de Software (SBES'04) (indicado para best-paper) Brasília, DF, Brasil, 2004.

LEMOS, O.A.L., MALDONADO, J.C., MASIERO, P.C. **Data-Flow Integration Testing criteria for Aspect-Oriented Programs.** In: Anais do 1o. Workshop de Desenvolvimento de Software Orientado a Aspectos (WASP'04) – realizado em conjunto com o SBES'04, Brasília, DF, Brasil, 2004a.

MARTIN, James; ODELL, James J.. **. Análise e projeto orientados a objeto**. São Paulo: Makron Books, 1996. 639p.

PAWLAK, R., DUCHIEN, L., FLORIN G., LEGONG-AUBRY, F., SEINTURIER, L, MARTELLI, L. **A UML Notation for Aspect-Oriented Software Design.** In: Proceedings of Workshop of Aspect Oriented Modeling with UML of Proceedings of Aspect Oriented Software Development Conference (AOSD), 2002.

RASHID, A., SAWYER, P., MOREIRA, A., ARAUJO, J. **Early Aspects: A Model for Aspect-Oriented requirements engineering.** In: Joint Int'l Conference Requirements Engineering (RE), Essen, Germany, IEEE, pp 199-202, 2002.

RASHID, A., CHITCHYAN, R. **Persistence as an Aspect. In: Proc.** of the 2nd International Conference on Aspect Oriented Software Development (AOSD), Boston–USA, March, 2003.

SOARES, S. **An Aspect-Oriented Implementation Method**. Dissertação de Mestrado. Centro de Informática (CI), Universidade Federal de Pernambuco (UFPe), Setembro, 2004.

SOARES, S., LAUREANO, E., BORBA, P. **Implementing Distribution and Persistence Aspects with AspectJ**. In: Proc. the 17th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), pp 174-190, November, 2002.

STEIN, D., HANENBERB, S., UNLAND, R. **Designing Aspect-Oriented Crosscutting in UML.** In: Workshop Aspect-Oriented Modeling with UML, AOSD, Enschede, April, 2002.

SUZUKI, J., YAMAMOTO, Y. **Extending UML With Aspects: Aspect Support in the Design Phase.** In: Workshop on Aspect-Oriented Programming (ECOOP), Lisboa, 1999.

TARR, P., OSSHER, H., SUTTON, S. Hyper/JTM : **Multi-dimensional Separation of Concerns for Java.** In: Proc. of the 24th International Conference on Software Engineering. Orlando, Florida, 2002.

ZAKARIA, A.A., HOSNY, H., ZEID, A. **A UML Extension for Modelling Aspect-Oriented Systems.** In: Proceedings of Workshop of Aspect Oriented Modeling with UML of Aspect Oriented Software Development Conference (AOSD), 2002.

ZHAO, J. **Tool Support for Unit Testing of Aspect-Oriented Software**. In: OOPSLA 2002 Workshop on Tools for Aspect-Oriented Software Development, Seatle, WA, 2002.

APÊNDICE A – Código-fonte da Ferramenta FRIT Tool

```
unit Main;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
 Menus,  ComCtrls, ExtCtrls, StdCtrls, ImgList, ShellApi,
Variants,
 LibXmlParser, ToolWin, Buttons, xmldom, DB, DBClient,
Provider,
 Xmlxform,  Grids, DBGrids, jpeg;

type
  TFrmMain = class(TForm)
    DlgOpen: TOpenDialog;
    ImageTopo: TImage;
    PanelCorpoSystem: TPanel;
    PageControl: TPageControl;
    Aba_Xml_ModeloCasoUso: TTabSheet;
    MemoSource: TRichEdit;
    Aba_Xml_CasosUsoCandidato: TTabSheet;
    MemoContents: TRichEdit;
    Bt_sair: TImage;
    Bt_abrir: TImage;
    Bt_Gerar: TImage;
    Timer1: TTimer;
    Panel1: TPanel;
    Animate1: TAnimate;
    Label1: TLabel;
    Aba_Frm_CasoUsoCandidato: TTabSheet;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    DBGrid1: TDBGrid;
    cds_candidato: TClientDataSet;
    cds_candidatoNOME: TStringField;
```

```
  cds_candidatoCRITERIO: TStringField;
  cds_candidatoSERAIMPL: TStringField;
  cds_candidatoITERACAO: TStringField;
  cds_candidatoID: TStringField;
  DataSource1: TDataSource;
  cds_relacionamento: TClientDataSet;
  cds_relacionamentoTIPO: TStringField;
  cds_relacionamentoORIGEM: TStringField;
  cds_relacionamentoDESTINO: TStringField;
  StatusBar1: TStatusBar;
  procedure FormShow(Sender: TObject);
  procedure FormHide(Sender: TObject);
  procedure MemoSourceChange(Sender: TObject);
  procedure PageControlChange(Sender: TObject);
  procedure Bt_sairClick(Sender: TObject);
  procedure Bt_GerarClick(Sender: TObject);
  procedure Bt_abrirClick(Sender: TObject);
  procedure Timer1Timer(Sender: TObject);
  procedure Add_criterios;
  procedure Criterio_qtde;
  procedure Criterio_extd;
  procedure Criterio_nf;
  procedure Del_nao_candidatos;

  procedure DBGrid1DblClick(Sender: TObject);

private
  XmlParser               : TXmlParser;

  file_caso_uso,
  file_relacionamento     : TEXTFILE;

  Nome_caso_uso_candidato,
  Nome_relacionamento     : String;

public
  Elements   : TObjectList;
  CreateTree : BOOLEAN;
  Changed    : BOOLEAN;
```

```
    PROCEDURE FillContent;
    PROCEDURE FillTree;
    PROCEDURE FillDtdTree;
    PROCEDURE ReadFile (Filename : STRING);
    PROCEDURE ApplySource;
  end;

var
  FrmMain: TFrmMain;
  tempo:integer;

IMPLEMENTATION

uses UFormCandidatos, UFormNumIteracoes;

{$R *.DFM}

TYPE
  TElementNode = CLASS
                   Content : STRING;
                   Attr    : TStringList;
                   CONSTRUCTOR Create (TheContent : STRING;
TheAttr : TNvpList);
                   DESTRUCTOR Destroy; OVERRIDE;
                 END;

CONSTRUCTOR  TElementNode.Create  (TheContent  :  STRING;
TheAttr : TNvpList);
VAR
  I : INTEGER;
BEGIN
  INHERITED Create;
  Content := TheContent;
  Attr    := TStringList.Create;
  IF TheAttr <> NIL THEN
    FOR I := 0 TO TheAttr.Count-1 DO
      Attr.Add  (TNvpNode  (TheAttr  [I]).Name  +  '='  +
TNvpNode (TheAttr [I]).Value);
END;
```

```
DESTRUCTOR TElementNode.Destroy;
BEGIN
  Attr.Free;
  INHERITED Destroy;
END;

procedure TFrmMain.FormShow(Sender: TObject);
begin
  XmlParser := TXmlParser.Create;
  Changed := FALSE;
  Elements := TObjectList.Create;
  PageControl.ActivePage := PageControl.Pages [0];
  if ParamCount > 0 then
    ReadFile (ParamStr (1));
end;

procedure TFrmMain.FormHide(Sender: TObject);
begin
  Elements.Free;
  XmlParser.Free;
end;

procedure TFrmMain.FillContent;
var
  id_caso_uso : STRING;
begin
  Nome_caso_uso_candidato        :=        ExtractFilePath
(Application.ExeName)+'candidatos.xml';
  Nome_relacionamento          :=          ExtractFilePath
(Application.ExeName)+'relacionamento.xml';

  AssignFile (file_caso_uso, Nome_caso_uso_candidato);
  AssignFile (file_relacionamento, Nome_relacionamento);

  try
    Rewrite (file_caso_uso);
    Rewrite (file_relacionamento);
```

```
    Writeln (file_caso_uso,
            ' <?xml version="1.0" standalone="yes"?> ');
    Writeln (file_caso_uso,
            ' <DATAPACKET Version="2.0">');
    Writeln (file_caso_uso,
            ' <METADATA> <FIELDS> ');
    Writeln (file_caso_uso,
            '   <FIELD   attrname="ID"   fieldtype="string"
WIDTH="50"/> ');
    Writeln (file_caso_uso,
            '  <FIELD  attrname="NOME"  fieldtype="string"
WIDTH="50"/>  ');
    Writeln (file_caso_uso,
            ' <FIELD attrname="CRITERIO" fieldtype="string"
WIDTH="50"/>  ');
    Writeln (file_caso_uso,
            ' <FIELD attrname="SERAIMPL" fieldtype="string"
WIDTH="50"/>  ');
    Writeln (file_caso_uso,
            ' <FIELD attrname="ITERACAO" fieldtype="string"
WIDTH="50"/>  ');
    Writeln (file_caso_uso,
            ' </FIELDS> </METADATA> ');
    Writeln (file_caso_uso,
            ' <ROWDATA> ');

    Writeln (file_relacionamento,
            ' <?xml version="1.0" standalone="yes"?> ');
    Writeln (file_relacionamento,
            ' <DATAPACKET Version="2.0">');
    Writeln (file_relacionamento,
            ' <METADATA> <FIELDS> ');
    Writeln (file_relacionamento,
            '  <FIELD  attrname="TIPO"  fieldtype="string"
WIDTH="50"/> ');
    Writeln (file_relacionamento,
            '  <FIELD  attrname="ORIGEM"  fieldtype="string"
WIDTH="50"/>  ');
    Writeln (file_relacionamento,
```

```
            ' <FIELD attrname="DESTINO" fieldtype="string"
WIDTH="50"/>  ');
    Writeln (file_relacionamento,
            ' </FIELDS> </METADATA> ');
    Writeln (file_relacionamento,
            ' <ROWDATA> ');

    try
      XmlParser.StartScan;
      XmlParser.Normalize := FALSE;
      id_caso_uso:='';

      while XmlParser.Scan do
        if (XmlParser.CurPartType = ptContent) then
         begin
           if  (Pos('ownedMember',XmlParser.CurName) > 0 )
then
            begin
              if
(XmlParser.CurAttr.Value('xmi:type')='uml:UseCase') then
               begin
                 Writeln (file_caso_uso,
                       '                                    <ROW
ID="'+XmlParser.CurAttr.Value('xmi:id')+'"'
                     +                                        '
NOME="'+XmlParser.CurAttr.Value('name')+'"'
                     + ' CRITERIO=""'
                     + ' SERAIMPL="Sim'+'"'
                     + ' ITERACAO=""'
                     + ' />');

id_caso_uso:=XmlParser.CurAttr.Value('xmi:id');
               end
              else
                id_caso_uso:='';
            end;

           if  (Pos('include',XmlParser.CurName) > 0 ) then
            begin
```

```
              if
(XmlParser.CurAttr.Value('xmi:type')='uml:Include')
                  and (id_caso_uso<>'') then
                begin
                  Writeln   (file_relacionamento,   '   <ROW
TIPO="include"'
                      + ' ORIGEM="'+id_caso_uso+'"'
                      +                                    '
DESTINO="'+XmlParser.CurAttr.Value('addition')+'"'
                      + ' />');
                end;
            end;

           if  (Pos('extend',XmlParser.CurName) > 0 ) then
            begin
              if
(XmlParser.CurAttr.Value('xmi:type')='uml:Extend')      and
(id_caso_uso<>'') then
                begin
                  Writeln   (file_relacionamento,   '   <ROW
TIPO="extend"'
                      + ' ORIGEM="'+id_caso_uso+'"'
                      +                                      '
DESTINO="'+XmlParser.CurAttr.Value('extendedCase')+'"'
                      + ' />');
                end;
            end;

           if   (Pos('nf:NF',XmlParser.CurName)  >  0  )  then
// rever
            begin // (bloco 8)
              Writeln    (file_relacionamento,    '    <ROW
TIPO="NF"'
                  +                                        '
ORIGEM="'+XmlParser.CurAttr.Value('base_UseCase')+'"'
                  + ' DESTINO=""'
                  + ' />');
            end;
         end;
```

```
     Writeln (file_caso_uso, ' </ROWDATA> </DATAPACKET>');
     Writeln     (file_relacionamento,     '     </ROWDATA>
</DATAPACKET>');
    finally
      CloseFile (file_caso_uso);
      CloseFile (file_relacionamento);
    end;

    MemoContents.PlainText := TRUE;
    MemoContents.Lines.LoadFromFile
(Nome_caso_uso_candidato);

  except
    MemoContents.Lines.Clear;
  end;
end;

procedure TFrmMain.FillTree;
begin
  XmlParser.Normalize := TRUE;
  XmlParser.StartScan;
end;

procedure TFrmMain.FillDtdTree;
begin

end;

procedure TFrmMain.ReadFile (Filename : STRING);
begin
  Screen.Cursor := crHourGlass;
  Elements.Clear;
  MemoSource.PlainText := TRUE;
  MemoSource.Lines.LoadFromFile (Filename);
  ApplySource;
end;

procedure TFrmMain.ApplySource;
begin
```

```
  Screen.Cursor := crHourGlass;   // muda cursor
  XmlParser.LoadFromBuffer (MemoSource.Lines.GetText);
  FillContent;
  Changed       := FALSE;
  Screen.Cursor := crDefault;
end;

procedure TFrmMain.MemoSourceChange(Sender: TObject);
begin
  Changed := TRUE;
end;

procedure TFrmMain.PageControlChange(Sender: TObject);
begin
  if PageControl.ActivePage <> Aba_Xml_ModeloCasoUso then
    if Changed then
      ApplySource;
end;

procedure TFrmMain.Bt_sairClick(Sender: TObject);
begin
   FrmMain.Close;
end;

procedure TFrmMain.Bt_GerarClick(Sender: TObject);
begin
  if DlgOpen.Execute then
    begin
      tempo:=0;
      Timer1.Enabled:=true;
    end;
end;

procedure TFrmMain.Bt_abrirClick(Sender: TObject);
begin
   Nome_caso_uso_candidato        :=        ExtractFilePath
(Application.ExeName)
                              +'candidatos.xml';
   cds_candidato.FileName:= Nome_caso_uso_candidato;
```

```
   cds_candidato.Open;
end;

procedure TFrmMain.Timer1Timer(Sender: TObject);
var
  filename: string;
  i: integer;
begin
  PageControl.TabIndex:=2;
  FrmMain.Enabled:=false;
  Screen.Cursor := crHourGlass;
  tempo:=tempo+1;
  Panel1.Visible:=true;
  Animate1.Active:=true;

  if tempo>3 then
   begin
     Timer1.Enabled:=false;
     filename:=DlgOpen.Filename;
     ReadFile (filename);
     FillContent;

     i:=Length(filename);
     while (filename[i]<>'\') and (i>0) do
       i:=i-1;

     cds_candidato.FileName:= Nome_caso_uso_candidato;
     cds_candidato.Open;
     Add_criterios;
     Animate1.Active:=false;
     Panel1.Visible:=false;
     FrmMain.Enabled:=true;
     Screen.Cursor := crDefault;
     PageControl.TabIndex:=0;
   end;
end;

procedure TFrmMain.Criterio_qtde;
var
```

```
  qtde : integer;
begin
  qtde:=0;
  cds_relacionamento.First;
  while (not cds_relacionamento.Eof) do
   begin
     if
cds_candidatoID.Value=cds_relacionamentoORIGEM.Value then
       inc(qtde);
     cds_relacionamento.Next;
   end;
  if qtde>1 then
   begin
     cds_candidato.Edit;
     if Length(cds_candidatoCRITERIO.Value)>0 then

cds_candidatoCRITERIO.Value:=cds_candidatoCRITERIO.Value+',
QTDE'
     else
       cds_candidatoCRITERIO.Value:='QTDE';
     cds_candidato.Post;
    end;
end;

procedure TFrmMain.Criterio_extd;
var
  extd : boolean;
begin
  extd:=false;
  cds_relacionamento.First;
  while (not cds_relacionamento.Eof) and (not extd) do
   begin
     if
(cds_candidatoID.Value=cds_relacionamentoORIGEM.Value)
         and (cds_relacionamentoTIPO.Value='extend') then
           extd:=true;
     cds_relacionamento.Next;
   end;
  if extd then
```

```
   begin
     cds_candidato.Edit;
     if Length(cds_candidatoCRITERIO.Value)>0 then

cds_candidatoCRITERIO.Value:=cds_candidatoCRITERIO.Value+',
EXTEND'
     else
       cds_candidatoCRITERIO.Value:='EXTEND';
     cds_candidato.Post;
   end;
end;

procedure TFrmMain.Criterio_nf;
var
  est_nf : boolean;
begin
  est_nf:=false;
  cds_relacionamento.First;
  while (not cds_relacionamento.Eof) and (not est_nf) do
   begin
     if
(cds_candidatoID.Value=cds_relacionamentoORIGEM.Value)
         and (cds_relacionamentoTIPO.Value='NF') then
           est_nf:=true;
     cds_relacionamento.Next;
   end;
  if est_nf then
   begin
     cds_candidato.Edit;
     if Length(cds_candidatoCRITERIO.Value)>0 then

cds_candidatoCRITERIO.Value:=cds_candidatoCRITERIO.Value+',
<<NF>>'
     else
       cds_candidatoCRITERIO.Value:='<<NF>>';
     cds_candidato.Post;
   end;
end;
```

```
procedure TFrmMain.Del_nao_candidatos;
begin
  cds_candidato.First;
  while (not cds_candidato.Eof) do
   begin
     if  cds_candidatoCRITERIO.Value='' then
       cds_candidato.Delete;
     cds_candidato.Next;
   end;
  DeleteFile(Nome_relacionamento);
end;

procedure TFrmMain.Add_criterios;
var
 extd : integer;
begin
  cds_candidato.Close;
  cds_candidato.Open;
  cds_candidato.First;
  while (not cds_candidato.Eof) do
    begin
      extd:=0;
      cds_relacionamento.Close;
      cds_relacionamento.FileName:=Nome_relacionamento;
      cds_relacionamento.Open;
      Criterio_qtde;
      Criterio_extd;
      Criterio_nf;
      cds_candidato.Next;
    end;
    cds_relacionamento.ApplyRange;
    cds_relacionamento.Close;
    Del_nao_candidatos;
    cds_candidato.First;
end;

procedure TFrmMain.DBGrid1DblClick(Sender: TObject);
begin
  if DBGrid1.SelectedIndex=2 then
```

```
   begin
     cds_candidato.Edit;
     if DBGrid1.SelectedField.AsString='Sim' then
       cds_candidatoSERAIMPL.asstring:='Não'
     else
       cds_candidatoSERAIMPL.asstring:='Sim';
     cds_candidato.Post;
   end
  else
   if DBGrid1.SelectedIndex=3 then
    begin
      FrmNumIteracoes.ShowModal;
    end;
end;

end.
```

```
unit UFormNumIteracoes;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, StdCtrls, Buttons;

type
  TFrmNumIteracoes = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
```

```
  public
    { Public declarations }
  end;

var
  FrmNumIteracoes: TFrmNumIteracoes;

implementation

uses UFormCandidatos, Main;

procedure TFrmNumIteracoes.BitBtn1Click(Sender: TObject);
begin
  FrmMain.cds_candidato.Edit;
  FrmMain.cds_candidatoITERACAO.Value:=Edit1.Text;
  FrmMain.cds_candidato.Post;
  FrmNumIteracoes.Close;
end;

procedure TFrmNumIteracoes.BitBtn2Click(Sender: TObject);
begin
  FrmNumIteracoes.Close;
end;

procedure TFrmNumIteracoes.FormShow(Sender: TObject);
begin
  Edit1.SetFocus;
end;

end.
```