Um Ambiente para Geração de Múltiplas Visões de Requisitos de Software

Antônio Francisco do Prado
Marcelo Fernando Camargo Sirbone[1]

Universidade Federal de São Carlos - Departamento de Computação
Rodovia Washington Luiz, Km 235 - Caixa Postal 676
CEP 13565-905 - São Carlos - SP - Brasil
e-mail <prado, sirbone>@dc.ufscar.br

Resumo: Este artigo apresenta um ambiente, integrado a um Banco de Dados Orientado a Objetos, que suporta múltiplas visões de requisitos de software. Técnicas de especificação de requisitos de software são modeladas, com recursos gráficos e textuais, e persistidas no Banco de Dados Orientado a Objetos O2. O ambiente utiliza-se de uma Representação Canônica para geração de múltiplas visões dos requisitos.

Abstract: This paper presents an environment, integrated to an Object Oriented Database, that supports multiple views of software requirements. Software requirements specification techniques are modelled, using graphical and textual resources, and persisted in the Object Oriented Database O2. The environment uses a Canonical Representation for generating of multiple views of the specified requirements.

Palavras-chave: Engenharia de Software, Ferramentas Case, Base de Dados Orientada a Objetos e Representação Canônica.

1. Introdução

Para especificação de requisitos de software, tanto na fase de análise como de projeto, existem várias técnicas, linguagens e ferramentas [1, 2, 5, 7, 9, 14]. Os modelos obtidos a partir destas técnicas, linguagens e ferramentas são diferentes visões dos requisitos em cada método. A especificação de requisitos para um sistema é complexa e uma representação que permita múltiplas visões dos requisitos facilita o seu entendimento. Outra vantagem é que o desenvolvedor pode utilizar o método que lhe é mais familiar para a especificação de requisitos do sistema, facilitando o desenvolvimento de um sistema em equipe, pois seus membros não ficam restritos à utilização de uma mesma técnica de especificação dos requisitos.

Este artigo apresenta um ambiente, que possibilita a geração de múltiplas visões com o auxílio de uma Representação Canônica (RC) [8]. Este ambiente é composto de uma Ferramenta de Transformação, uma Ferramenta Gráfica e um Banco de Dados Orientado a Objetos (BDOO), conforme mostra a figura 1.

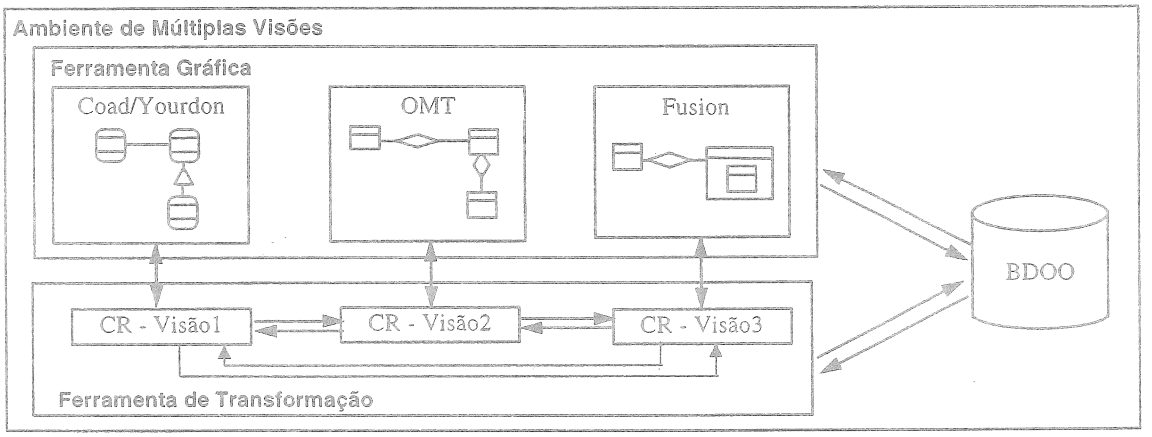

Figura 1 - Arquitetura geral do ambiente

[1] Bolsista da FAPESP

A Ferramenta Gráfica está representando um mesmo sistema modelado por diferentes técnicas, fornecendo várias visões dos requisitos do sistema. Para cada técnica visualizada na Ferramenta Gráfica existe uma descrição RC instanciada no BDOO. A Ferramenta de Transformação permite a transformação de técnicas de especificação de requisitos de sistemas para outras, como no exemplo da figura 1 na qual um Modelo de Classes&Objetos em Coad/Yourdon [1] é transformado para um Modelo de Objetos em OMT [9]. A ferramenta transforma a descrição RC (RC-Visão1) que representa o Modelo de Classes&Objetos do método Coad/Yourdon para outra descrição RC (RC-Visão2), representando o Modelo de Objetos do método OMT. Da mesma forma obtém-se a visão segundo técnicas do método Fusion [2].

Na próxima seção será apresentada a Representação Canônica. Na seção 3 aborda-se o ambiente para geração de múltiplas visões. Na seção 4 apresentam-se os trabalhos correlatos. Finalmente, na seção 5, apresentam-se as conclusões sobre o trabalho.

2. Representação Canônica (RC)

A Representação Canônica (RC), proposta por Alan Davis [4], é uma forma de representar e armazenar requisitos de um sistema usando diferentes metodologias. Suas principais características são a eliminação de redundância entre linguagens e a possibilidade de gerar múltiplas visões dos requisitos. Ela é composta de um conjunto de elementos $E = \{e_1, e_2, \ldots, e_n\}$ e de um conjunto de relações $R = \{r_1, r_2, \ldots, r_n\}$, onde cada relação r conecta um par ordenado de elementos e, que não são necessariamente distintos. Além disso, cada $e_i \in E$ é uma *tupla*: $e_i = (et_i, el_i)$ onde et_i é o tipo do elemento, $et_i \in ET \cup FT$, onde $ET = \{entidade, processo, estado, mensagem, atributo, predicado, restrição, informação, transição\}$, com $FT = 2^{ET}$. Por sua vez, el_i é uma identificação única para o elemento. Também cada $r_i \in R$ é uma quádrupla: (rt_i, rl_i, se_i, te_i) onde rt_i é o tipo do relacionamento , $rt_i \in RT$, com RT = {*parte de, instanciação, tem valor, envia, recebe, estímulo, resposta, equivalência, associação, operando*}, rl_i é uma identificação única para a relação, se_i e te_i são os elementos inicial e final do relacionamento, $se_i \in E$ e $te_i \in E$.

As figuras 2 e 3 mostram respectivamente o Modelo de Classes&Objetos do método Coad/Yourdon para um sistema de Distribuidora de Produtos e a representação gráfica da descrição RC referente a este modelo.

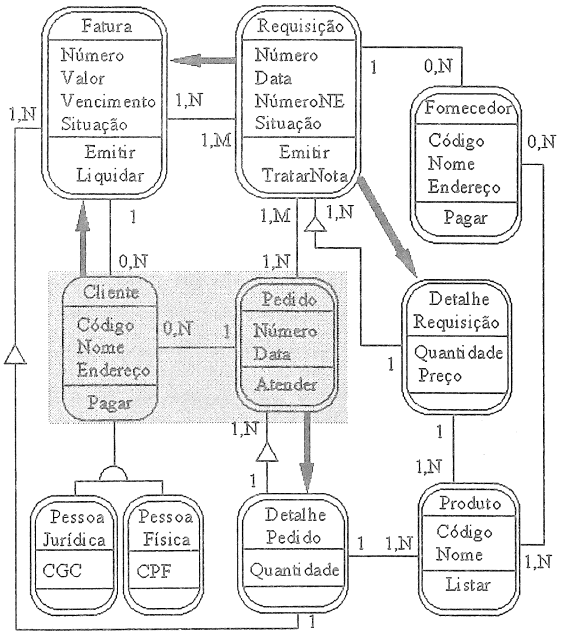

Figura 2 - Modelo de Classes&Objetos do método de Coad/Yourdon

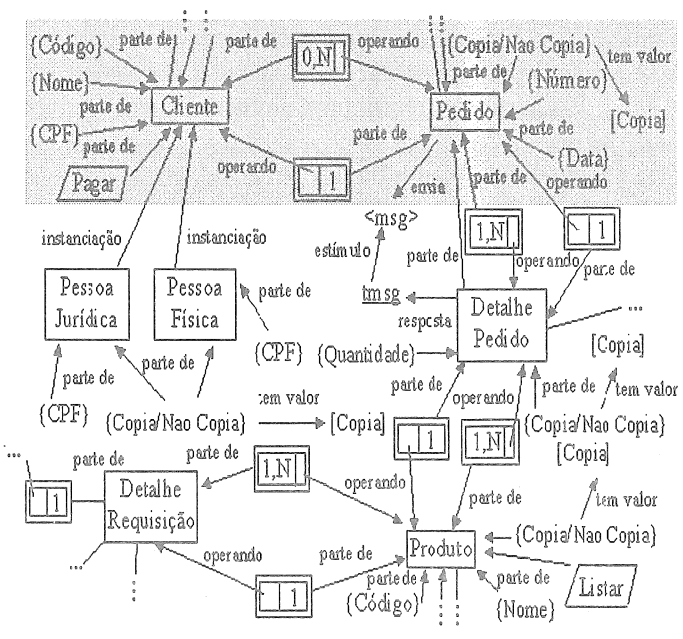

Figura 3 - Parte da descrição RC do Modelo Classes&Objetos do método Coad/Yourdon

O sistema de Distribuidora de Produtos, modelado na figura 2, é composto pelas classes: *Cliente* especializada em: *Pessoa Física* e *Pessoa Jurídica; Pedido* que tem *Detalhe Pedido; Fornecedor* que fornece *Produto; Fatura* que é paga pelo *Cliente, Requisição* que tem *Detalhe Requisição* e é enviada ao Fornecedor. Estas classes com seus atributos e serviços se relacionam através de estruturas de heranças, agregações, conexões de ocorrência e de mensagens.

Alguns dos elementos canônicos presentes na figura 3 são: os atributos "Código", "Nome", "Endereço", "CGC", "CPF", "Número", "Data", "Quantidade", as entidades "Cliente", "Pedido" e "Detalhe Pedido", o processo "Pagar" e as restrições "1,N", "1", "0,N" representando as cardinalidades dos relacionamentos de associação e de todo parte.

Os relacionamentos canônicos são sempre definidos entre dois elementos. Por exemplo, na figura 3 tem-se as relações canônicas "parte de" entre a entidade "Cliente" e o atributo "Código", "operando" entre a entidade "Cliente" e a restrição "1", "instanciação" entre as entidades "Cliente" e "Pessoa Física" , "tem valor" entre a informação "Cópia" e o atributo "Cópia/Não Cópia", "envia" entre a entidade "Pedido" e a mensagem "msg", "resposta" entre a entidade "Detalhe Pedido" e a transição "tmsg" e "estímulo" entre a transição "tmsg" e a mensagem "msg". A definição completa sobre a Representação Canônica e sua representação gráfica pode ser encontrada em "*A Canonical Representation for Requeriments*" [4].

3. Ambiente para geração de múltiplas visões

O ambiente foi desenvolvido na plataforma SunOS/Unix e sistema de BDOO O2 [11, 12]. Para implementação da Ferramenta Gráfica foi utilizado o *toolkit* Xview[10]. Para implementação do Metamodelo e da Ferramenta de Transformação utilizou-se a linguagem de quarta geração O2C [12]. Baseado na Representação Canônica e com a Ferramenta de Transformação são geradas as múltiplas visões de requisitos de software, a partir de especificações modeladas na Ferramenta Gráfica, usando técnicas de um determinado método. Um metamodelo foi definido para organizar a persistência dos objetos gerados pelo ambiente.

3.1. Metamodelo para a persistência das visões

A figura 4 mostra o metamodelo utilizado para a persistência das visões descritas na Representação Canônica. Ao lado tem-se a notação básica do Modelo de Objetos do método Fusion utilizado pelo metamodelo. Instanciando-se o metamodelo tem-se que a classe *Projeto* representa o sistema a ser modelado, cujos principais atributos são:

- Nome - nome dado para o projeto.
- Descrição - resumo do projeto a ser especificado.

Especificações de requisitos de um mesmo sistema pertencem a um mesmo projeto. Um projeto pode ter uma ou mais visões, cada uma contendo técnicas de análise de requisitos de determinado método, como por exemplo, do método Fusion, Coad/Yourdon ou OMT. Portanto, cada abordagem é considerada uma visão e é representada por instâncias da classe *Visão*. Os atributos dessa classe são:

- Nome - nome da visão, ou seja, do método de especificação de requisitos adotado.
- Descrição - definição sobre o método utilizado.

Cada visão pode conter vários modelos ou técnicas, como por exemplo, no método Fusion temos o Modelo de Objetos e o Modelo de Interface. Cada modelo ou técnica é considerado um diagrama e é representado por instâncias da classe *Diagrama*, que possui os seguintes atributos:

- Nome - nome do modelo ou técnica adotada.
- Descrição - definição sobre o modelo ou técnica adotada.

Um diagrama possui um conjunto de descrições RC, que podem ser: Elementos ou Relações Canônicas. *ElementoCanônico* é uma classe de todos os objetos que aparecem na RC, exceto os

relacionamentos entre estes objetos. Nesta classe são persistidos os elementos canônicos (entidade, processo, estado, mensagem, atributo, predicado, restrição, informação e transição). A classe possui o seguinte atributo:

- Denominação - nome que o elemento canônico recebe na modelagem do sistema.

RelaçãoCanônica representa a conexão entre elementos canônicos e está relacionada ao *ElementoCanônico* através de sua Origem e Destino, mostrando o ponto inicial e final de um relacionamento. A obrigatoriedade (|) junto à *RelaçãoCanônica*, indica que esta somente existe se estiver associada aos elementos canônicos origem e destino. Nesta classe são persistidas as relações canônicas (parte de, instanciação, tem valor, envia, recebe, estímulo, resposta, equivalência, associação e operando).

As classes *ElementoCanônico* e *RelaçãoCanônica* herdam características da classe *RepresentaçãoCanônica*, que possui os seguintes atributos:

- Tipo - tipo do elemento ou relacionamento canônico.
- Descrição - contém um resumo sobre o elemento ou relacionamento canônico.

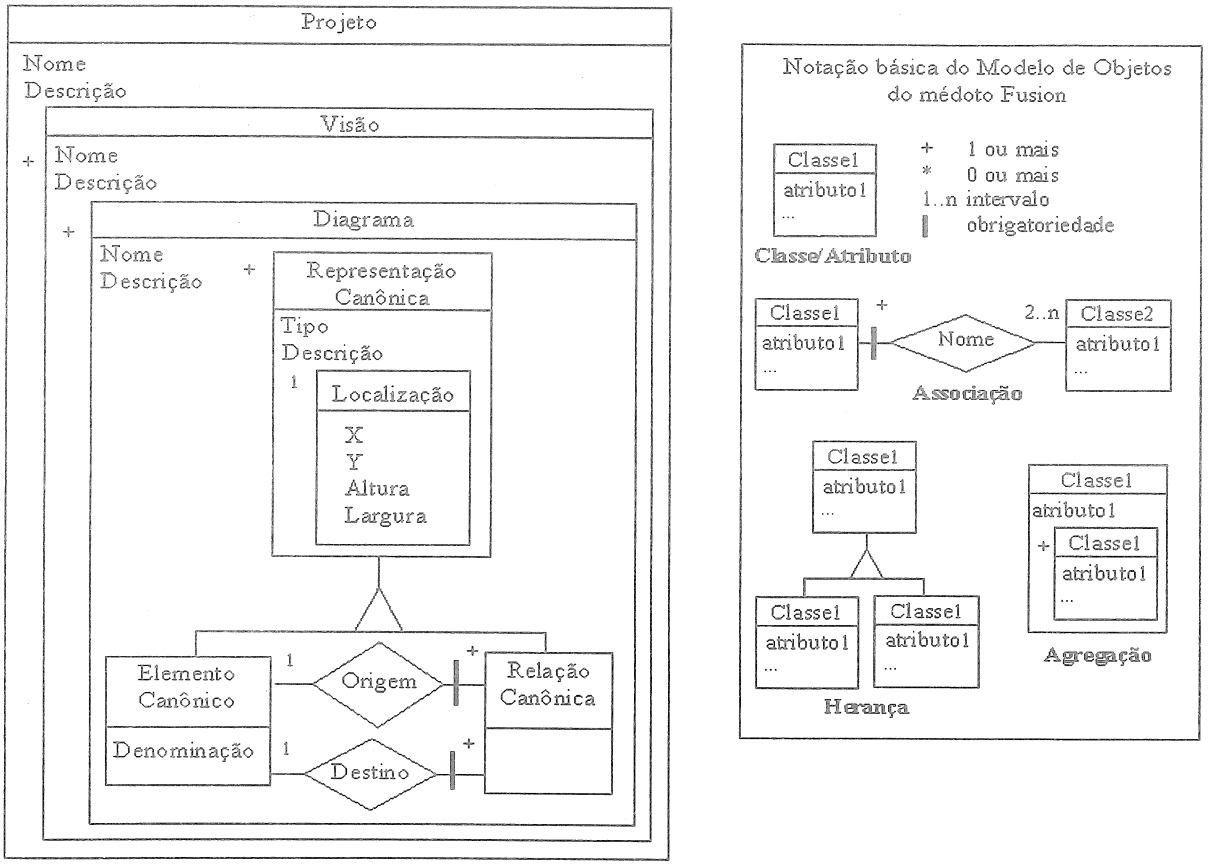

Figura 4 - Metamodelo para persistência das visões

A RC proposta por Davis [4] foi estendida com a classe *Localização*, agregada à classe *RepresentaçãoCanônica*, para permitir a recuperação da representação gráfica de cada técnica descrita pela RC. A classe *Localização* possui os seguintes atributos:

- X, Y - atributos que representam o ponto central do elemento gráfico.
- Largura - largura do elemento gráfico.
- Altura - altura do elemento gráfico.

Baseado neste metamodelo, são persistidos os objetos usados na especificação de requisitos e posteriormente são efetuadas consultas nestes objetos.

3.2. Implementação do Metamodelo no Banco de Dados Orientado a Objetos

A figura 5 mostra parte do código das classes *Diagrama* e *RepresentaçãoCanônica* do Metamodelo da figura 4. Para se obter um melhor desempenho na recuperação dos objetos, optou-se por implementar a agregação da classe *Diagrama* com as classes *ElementoCanônico* e *RelaçãoCanônica*. Portanto, o atributo **elemento** representa um conjunto de objetos do tipo *ElementoCanônico*, modelando a agregação entre as classes *Diagrama* e *ElementoCanônico*. Já o atributo **relacao** representa um conjunto de objetos do tipo *RelaçãoCanônica*, modelando a agregação entre as classes *Diagrama* e *RelaçãoCanônica*. Cada classe é responsável pela persistência de seus objetos. Por exemplo o serviço Persiste_ElemCanonico da classe Diagrama persiste o elemento canônico no BDOO.

```
class Diagrama inherit Object public type
tuple(nome: string,
      descricao: string,
      elemento: set(ElementoCanonico),
      relacao: set(RelacaoCanonica))
method
  public Persiste_ElemCanonico(elemca: ElementoCanonico),
  public Persiste_RelaCanonica(relaca: RelacaoCanonica),
  ...
end;
```

```
class RepresentacaoCanonica inherit Object public type
tuple(tipo: string,
      descricao: string,
      localizacao: Localizacao)
method
  public init(n: string, d: string, l: Localizacao)
  ...
end;
```

Figura 5 - Implementação das classes Diagrama e RepresentaçãoCanônica

A classe *RepresentaçãoCanônica* mostra, além dos atributos tipo e descrição, o atributo **localizacao**, que implementa a agregação das classes *RepresentaçãoCanônica* e *Localização*.

A figura 6 mostra parte da implementação das classes *ElementoCanônico* e *RelaçãoCanônica*. Estas classes herdam características da classe *RepresentaçãoCanônica*. As associações entre objetos são implementadas da mesma forma que na agregação, exceto que, nas associações define-se o relacionamento em ambas as classes. Assim tanto, na classe *ElementoCanônico* como na classe *RelaçãoCanônica* tem-se a definição das associações, através dos atributos **origem** e **destino**.

```
class ElementoCanonico inherit RepresentacaoCanonica public
type
tuple(denominacao: string,
      origem: set(RelacaoCanonica),
      destino: set(RelacaoCanonica)
method
    public Atualiza_Origem(relaca: RelacaoCanonica),
    public Atualiza_Destino(relaca: RelacaoCanonica)
    ...
end;
```

```
class RelacaoCanonica inherit RepresentacaoCanonica public
type
tuple(origem: ElementoCanonico
      destino: ElementoCanonico
method
    public init(n: string, d: string, l: Localizacao,
                elemca1: ElementoCanonico,
                elemca2: ElementoCanonico)
    ...
end;
```

Figura 6 - Implementação das classes ElementoCanônico e RelaçãoCanônica

3.3. Ferramenta Gráfica

A figura 7 mostra janelas da Ferramenta Gráfica responsável pelo desenvolvimento e visualização gráfica dos requisitos de um sistema. Uma das janelas mostra o Modelo de Classes&Objetos do método Coad/Yourdon e a outra o Modelo de Objetos do método Fusion. São duas visões do sistema de Distribuidora de Produtos, mostrado na figura 2.

Na janela principal do ambiente é possível selecionar, armazenar, remover ou criar um novo projeto, através do botão *"Projeto"*. Com o botão *"Visao"* escolhe-se o método de desenvolvimento de software orientado a objetos e em seguida define-se o modelo ou técnica de especificação de requisitos que se deseja trabalhar. As diferentes visões são apresentadas em janelas com opções para desenhar e transformar as

especificações de uma visão para outra. Para se obter uma nova visão em técnicas de outros métodos usa-se a opção *"Transformar"*, associada à Ferramenta de Transformação.

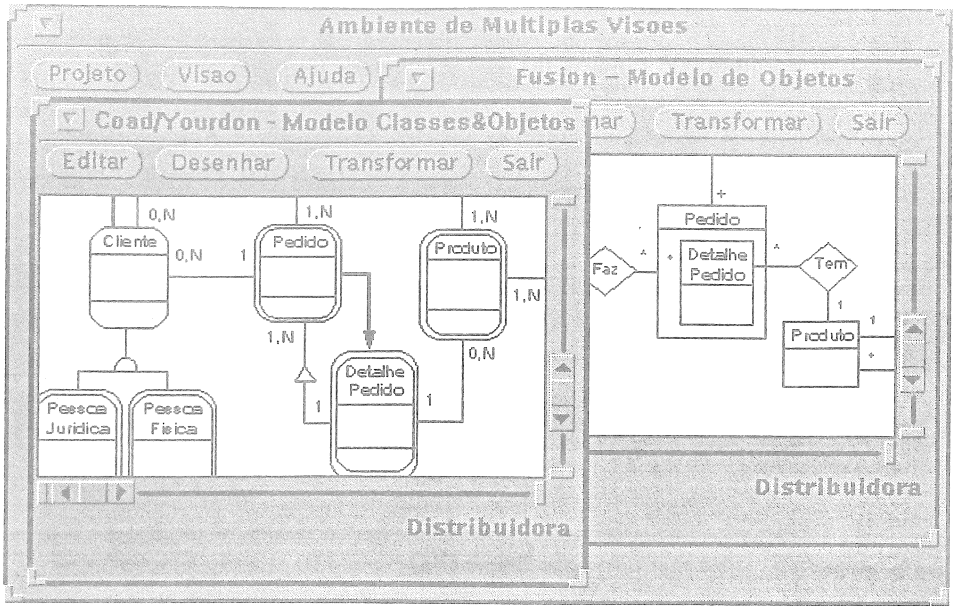

Figura 7 - Visão geral da Ferramenta Gráfica

As visões são persistidas no BDOO O2 através de uma descrição RC, composta dos elementos e relações canônicos. Um elemento canônico pode ser composto por outros elementos canônicos, como por exemplo, o elemento que representa uma classe em Coad/Yourdon possui os elementos Atributo e Serviço (ver figura 8) e, em Fusion somente Atributo.

Cada serviço de uma classe possui sua definição completa com parâmetros, tipo de retorno e a respectiva mini-especificação, conforme mostra a figura 9, para o serviço *Pagar* da classe *Cliente*.

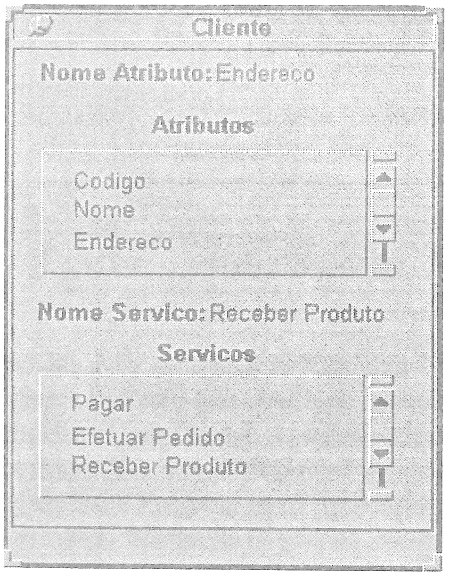

Figura 8 - Janela de elementos para a classe Cliente

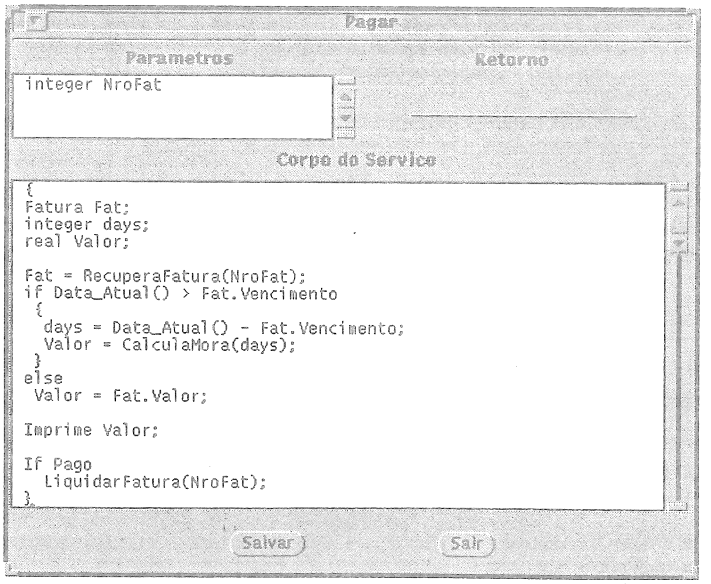

Figura 9 - Janela do serviço Pagar

A comunicação da Ferramenta Gráfica com a Ferramenta de Transformação ocorre através de eventos gerados a partir da Ferramenta Gráfica. Cada evento dispara uma mensagem que é tratada por uma função específica dependendo do seu tipo. A figura 10 mostra o tratador de mensagens "**MessageLoop**" que identifica e despacha cada mensagem. Por exemplo, para a mensagem **VISAO_COAD_MCO** é executada a função **Gerar_Visao**, que mostra a janela com recursos para modelar classes e objetos no método Coad/Yourdon.

```
transaction body MessageLoop in application AMV_Tool
{...
 extern message;
 Janela_Principal();
 ...
 switch(message)
 {
  case PROJETO_ABRIR:
      Montar_Projeto_Abrir ();
      break;
  case PROJETO_SALVAR:
      Montar_Projeto_Salvar();
      break;
  case VISAO_COAD_MCO:
      Gerar_Visao();
      break;
  ...
  case TRANSFORMAR:
      Transformar(visao1,visao2,d);
      break;
  ...
  }
...
xv_destroy_safe(fprincipal);
};
```

Figura 10 - Implementação do MessageLoop

3.4. Ferramenta de Transformação

A Ferramenta de Transformação [8] é responsável pelas transformações entre as descrições RCs para a obtenção de múltiplas visões. Embora os elementos e relacionamentos utilizados nas representações de diferentes técnicas sejam os mesmos, estes são combinados de forma diferente em cada representação. Assim uma descrição RC (RC-Visão1) poderá exigir transformações, que compatibilizem as diferenças entre as visões, produzindo uma nova descrição RC (RC-Visão2). Dependendo da semelhança entre as visões estas mudanças podem ser simples ou complexas, ou mesmo não existir.

O evento de seleção do botão Transformar, mostrado na Ferramenta Gráfica da figura 7, gera a mensagem **TRANSFORMAR**, onde o tratador de mensagem direciona a execução para a função Transformar (ver figura 10). A figura 11 mostra parte da implementação desta função Transformar, que recebe como parâmetros o nome da visão **fonte** da transformação (visao1), o nome da visão **destino** da transformação (visao2) e o diagrama (d) que contém o conjunto de relacionamentos a serem transformados. Esta função chama outras funções específicas, como ***TransRC_CoadFusion***, que realiza a transformação do método Coad/Yourdon (visão fonte) para o método Fusion (visão destino).

```
Function body Transformar (visao1: string, visao2: string, d: Diagrama)
{
 ...
  if (visao1 == "Coad/Yourdon")
   if (visao2 == "Fusion")
     {
     v = new Visao("Fusion", "Método orientado a Objetos";
     daux = new Diagrama("Modelo de Objetos", "Modelo estático do Fusion");
     for (relaca in d->repres.relacao)
       TransRC_CoadFusion(relaca,daux);
     }
   ...
};
```

Figura 11 - Função Transformar

O comando *for*, da função transformar, percorre todo o conjunto de relacionamentos para aplicação das transformações. De acordo com o metamodelo da figura 4, estes relacionamentos, obrigatoriamente, possuem

os elementos origem e destino. Portanto, ao se buscar um relacionamento no BDOO, busca-se também os elementos origem e destino associados ao relacionamento, produzindo um ganho na recuperação de objetos.

As classes Cliente e Pedido, seus atributos e a conexão de ocorrência entre estas classes pertencentes ao modelo de Classes&Objetos do método de Coad/Yourdon do sistema de Distribuidora de Produtos, será utilizado para explicar o uso das transformações. Uma nova visão deste modelo será obtida no método Fusion.

Para transformar a descrição RC mostrada na figura 3 para a descrição RC Fusion, executa-se a transformação ***TransRC_CoadFusion***. A figura 12 mostra parte da implementação desta transformação, que recebe como parâmetros cada relacionamento a ser transformado e o diagrama que conterá os novos elementos e relacionamentos canônicos gerados.

```
Function body TransRC_CoadFusion( relaca: RelacaoCanonica, daux: Diagrama)
{
o2 tuple(elem:ElementoCanonico,existe:integer) tipo1,tipo2;

tipo1=TransRC_elem_CoadFusion(relaca->origem,daux);        /* retorna um elemento canônico
tipo2=TransRC_elem_CoadFusion(relaca->destino,daux);       /* retorna um elemento canônico
TransRC_rela_CoadFusion(relaca,tipo1.elem,tipo2.elem,daux); /* transforma um relacionamento
transaction;
if (tipo1.existe==0)
  daux->Persiste_ElemCanonico(tipo1.elem);                 /* armazena um elemento canônico
if (tipo2.existe==0)
  daux->Persiste_ElemCanonico(tipo2.elem);                 /* armazena um elemento canônico
...
};
```

Figura 12 - Transformação TransRC_CoadFusion

A função ***Trans_elem_CoadFusion*** transforma os elementos canônicos associados ao relacionamento canônico e retorna o elemento canônico gerado. A seguir é transformado o relacionamento canônico através da função ***Trans_rela_CoadFusion***, que recebe como parâmetros: o relacionamento canônico a ser transformado, os elementos canônicos retornados pela função ***Trans_elem_CoadFusion*** e o diagrama que conterá os novos elementos e relacionamentos canônicos. A função ***Persiste_ElemCanonico*** torna o elemento canônico persistente no BDOO.

As figuras 13 e 14 mostram respectivamente a descrição RC gerada pela transformação TransRC_CoadFusion e o Modelo de Objetos (nova visão) no método Fusion correspondente à descrição RC. Na figura 13 tem-se os seguintes elementos canônicos: os atributos *"Código"*, *"Nome"*, *"Endereço"*, *"Número"* e *"Data"*, as entidades *"Cliente"* e *"Pedido"* três restrições representando o nome a as cardinalidades do relacionamento. Já os relacionamentos canônicos são: os relacionamentos *"parte de"* e *"operando"*.

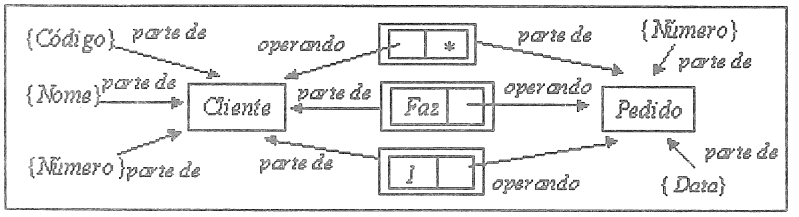

Figura 13 - Descrição RC do Modelo de Objetos do método Fusion

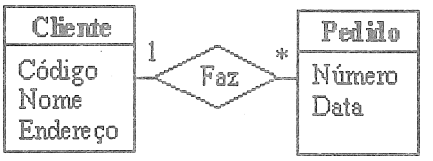

Figura 14 - Modelo de Objetos do método Fusion

Uma visão em determinado método contém técnicas que ao serem transformadas para técnicas de outros métodos podem ser agrupadas ou separadas em diferentes representações. Por exemplo, o nome do serviço "*Pagar*" está presente apenas na descrição RC da Figura 3, pois na fase de análise do método Fusion os serviços aparecem no Modelo de Operação e não no Modelo de Objetos.

Transformações também podem ser necessárias para compatibilizar pequenas diferenças entre uma técnica de um método para outro. Por exemplo, uma das diferenças entre a descrição RC do método de Coad/Yourdon mostrada na Figura 3 e a descrição RC do método de Fusion mostrada na figura 13, é com relação à mudança na notação das cardinalidades. A descrição RC do método de Coad/Yourdon tem a restrição "*0,N*" como "*parte de*" da entidade "*Cliente*" e "*operando*" da entidade "*Pedido*". Enquanto que a descrição RC do método de Fusion tem esta restrição como "*parte de*" da entidade "*Pedido*" e "*operando*" da entidade "Cliente" e ainda com a denominação trocada para "*". O mesmo ocorre para o restante dos elementos canônicos que representam cardinalidades nos Modelos de Objetos.

4. Trabalhos correlatos

Pesquisas tem sido realizadas para construir ferramentas que automatizam o processo de desenvolvimento de software, desde a análise até a implementação [13, 15, 16, 17, 18, 19]. Segundo Armbruster [13] uma Ferramenta Case ideal deveria conter as seguintes características: oferecer várias metodologias para o desenvolvimento de software, ser capaz de realizar transformações entre as técnicas de desenvolvimento de software, realizar verificação de erros lógicos, ser de fácil utilização, suportar trabalho em grupo e gerar código executável, entre outras.

Armbruster, comparando diferentes Ferramentas Case [13] existentes no mercado, entre elas o Paradigm Plus 2.0 [13,15], Together C++ [13, 19] e Select OMT [13], chegou à conclusão que não existe uma Ferramenta Case perfeita. A composição ideal deveria conter as melhores características de cada Ferramenta Case disponível no mercado. O Paradigm Plus 2.0 é a Ferramenta Case mais completa, suportando vários métodos de especificação de requisitos de software, geração de código executável para diversas linguagens (C/C++, Ada e Smalltalk), entre outras características. O Together C++ possui como característica marcante um excelente analisador sintático. Já o Select OMT possui facilidade na utilização de grandes modelagens.

Ted R. Crouch especificou o ambiente TINA [3], através do qual supõe-se ser possível modelar sistemas sobre diferentes enfoques utilizando a Representação Canônica como uma linguagem intermediária. A base de dados deste ambiente foi dividido em três componentes. O componente base de dados da RC responsável por representar a "*estrutura de*", os "*relacionamentos entre*" elementos e os próprios relacionamentos canônicos. O componente RC gráfico que contém os dados necessários para representar os requisitos de software sobre diferentes visões. Finalmente, o componente RC restrições responsável pela validação dos relacionamentos canônicos entre os elementos.

Não se tem notícias sobre a implementação do ambiente TINA, inclusive da interface gráfica, pela qual seria possível desenvolver e visualizar os requisitos de software, e de um mecanismo que realizasse as transformações entre as técnicas de especificações de requisitos.

Nosso ambiente procurou incorporar não apenas as características mencionadas por Armbruster e Crouch, mas também uma tecnologia nova baseada na orientação a objetos.

5. Conclusões

Aspectos relacionados com uma representação única para os requisitos foram investigados e um protótipo foi construído para validar as idéias de se ter as múltiplas visões.

Um ambiente de múltiplas visões produz uma especificação mais completa dos requisitos do software. O sistema pode ser visualizado por diferentes métodos e técnicas de especificação de requisitos com cada visão enfocando características distintas, que ajudarão a compreender melhor o sistema a ser desenvolvido. Com isso o desenvolvedor não fica preso a um determinado método de especificação de requisitos, pois ele pode escolher o método que lhe é mais familiar. A semântica de diferentes modelos dos vários métodos orientados a objetos poderão ser integradas, facilitando a compreensão do sistema.

O protótipo também implementa uma interface gráfica para desenvolvimento e visualização dos requisitos e dispõe de mecanismos para realizar transformações entre estes requisitos.

A utilização da Ferramenta de Transformação faz a geração de múltiplas visões de requisitos de forma direta e transparente ao desenvolvedor. A utilização da Representação Canônica como uma forma intermediária

reduziu as transformações necessárias entre métodos de especificação de requisitos de software. Estas são utilizadas para compatibilizar as conversões entre técnicas de diferentes métodos.

Os recursos disponíveis pelas Ferramentas Gráficas e de Transformação já atendem às principais técnicas do desenvolvimento de software orientado a objetos para os métodos presentes no ambiente.

Os resultados já alcançados são bastante significativos e mostraram a viabilidade de se ter um ambiente de múltiplas visões dos requisitos de software. Estudos de casos envolvendo Modelo de Objetos com várias classes estruturadas por herança, agregação e conexões de ocorrência já foram construídos e testados. Estes primeiros testes permitiram concluir que a geração de múltiplas visões será bastante útil, principalmente porque facilitará a especificação, a documentação, a manutenção e a reutilização dos requisitos modelados em outros sistemas similares. Um reuso da fase de análise do ciclo de vida do software é obtido em diferentes técnicas de modelagem orientadas a objetos.

A utilização de um BDOO como plataforma de implementação do ambiente é outro ponto relevante desta pesquisa, pois facilita a ampliação e melhoria do ambiente. Como trabalhos futuros temos a:

- extensão do ambiente para suportar outros métodos e técnicas de especificação de requisitos de software;
- adição de novas funcionalidades no ambiente para permitir a geração automática: do Banco de Dados e do código em linguagem executável do sistema, a partir dos modelos especificados; e
- Uso da ferramenta Draco, objeto de pesquisa de um dos autores, Prado [20], nas transformações.

Referências Bibliográficas

[1] **Coad, P.; Yourdon, E.;** *Análise Baseada em Objetos,* Editora Campus, Yourdon Press, 1991;

[2] **Coleman, D.; Arnold, P.; Bodoof, S.; Dollin, C.; Gilchrist, H.; Hayes, F.; Jeremaes, P.;** *Object-Oriented Development, The Fusion Method,* Prentice Hall Object Series, 1994;

[3] **Crouch, T.R.;** *A Database Design for Representing Requirements in their Canonical Form,* Tese de Mestrado da University of Colorado, 1994.

[4] **Davis, A.M.; Jordan, K.; Nakajima, T.;** *A Canonical Representation for Requeriments,* Technical Report, University of Colorado Springs, 1995, 30p.

[5] **Jacobson, I.;** *Object-Oriented Software Engineering*, Addison-Wesley, Reading, MA, 1992;

[6] **Kirner, T. et alli.;** *Ambiente para Representação de Múltiplas Visões de Requisitos: O Metamodelo e uma Linguagem de Transformação*, Anais do X Simpósio Brasileiro de Engenharia de Software - SBES, 1996.

[7] **Martin, J.; Odell, J. O.;** *Análise e Projeto Orientados a Objeto*, Makron Books, 1996.

[8] **Prado, A.F.; Vieira, M.T.P.; Sirbone, M.F.C.;** *Um Ambiente de Suporte a Múltiplas Representações de Requisitos de Software,* 1996, VII Semana de Informática, Universidade Estadual de Maringá.

[9] **Rumbaugh, J., et alli;** *Modelagem e Projetos Baseados em Objetos*, Editora Campus, 1994.

[10] **Heller, D.,** *XView Programming Manual for Version 11 of the X Window System*, Volume Seven, July 1990.

[11] **Deux, O.;** *The O2 System*, Communications of the ACM, Vol. 34, N°. 10, pp. 34-48, Oct. 1991.

[12] ***O2C Reference Manual, Release 4.6***, O2 Technology, Setembro 1995.

[13] **Armbruster, J. L.;** *Comparing CASE Tools*; Dr. Dobb's Journal, Junho 1995, p.76-86.

[14] **Booch, G.;** *Object Oriented Analysis and Design with Applications*, The Benjamin/Cummings Publishing Company, Inc., Redwood City (CA), USA, 1994.

[15] ***Paradigm Plus Release 2.0. Reference Manual***, Protosoft, 1994.

[16] ***FusionCASE - SPV, Version 1.3.1***, SoftCASE Consulting, 1995.

[17] **Bergmann, U.;** *Construção de um Domínio de Desenvolvimento de Software Orientado a Objetos Segundo o Paradigma Draco*, Dissertação (Mestrado), Instituto Militar de Engenharia, Rio de Janeiro, 1996.

[18] **Lucena, C.J.P.; Leite, J.C.; Fernandes, J.R.; Gheiner, M.; Prado, A. F.;** *JSP/PUC: Um Ambiente de Software Experimental para Estudo do Processo de Automação de Desenvolvimento de Software,* SBA Controle & Automação, Vol.7 N°.3, p. 126-146, Dez. 1996.

[19] **Together/C++ Professional 2.0**, Object International Software Ltd. Stuttgart, Alemanha, 1996.

[20] **Prado, A. F.,** *Estratégia de Reengenharia de Software Orientada a Domínios*, Tese de Doutorado, PUC_RJ, 1992.