UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

FERRAMENTA DE APOIO AO MAPEAMENTO DE ESPECIFICAÇÃO ESTRUTURADA PARA ESPECIFICAÇÃO ORIENTADA A OBJETOS

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

MARILAN RICARDO TAGLIARI

BLUMENAU, JUNHO/2002

2002/1-55

FERRAMENTA DE APOIO AO MAPEAMENTO DE ESPECIFICAÇÃO ESTRUTURADA PARA ESPECIFICAÇÃO ORIENTADA A OBJETOS

MARILAN RICARDO TAGLIARI

ESTE TRABALHO DE CONCLUSÃO DE CURSO FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Everaldo Artur Grahl — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Everaldo Artur Grahl

_____

Prof. Marcel Hugo

_____

Prof. Jomi Fred Hübner

AGRADECIMENTOS

Gostaria de agradecer aos meus pais Heitor José Tagliari e Marilene Moschetta Tagliari pelo carinho que sempre tiveram por mim pelo incentivo ao estudo mesmo com a dificuldade da distância que nos separa, vocês foram fundamentais para a realização deste trabalho. Gostaria também de agradecer aos meus avós Nilo Moschetta e Otília Moschetta, minha irmã Mariléia, meu cunhado Flávio e principalmente minha sobrinha e afilhada Ana Flávia. Não poderia esquecer de citar meus tios Evandro e Claudia Moschetta juntamente com os primos Rodrigo e Mateus que sentiram muito minha falta e são pessoas muito especiais para mim.

Gostaria de fazer um agradecimento todo especial a minha noiva, amiga e companheira Ana Cristina que com paciência me auxiliou e acompanhou nas horas mais difíceis.

Também agradecer ao Prof. Everaldo Artur Grahl que com paciência e sabedoria me orientou neste trabalho de conclusão de curso. Ao Prof. Marcel Hugo pelo auxílio prestado para a realização deste trabalho e ao Prof. Jomi Fred Hübner pela presença na banca examinadora. Também a todos os demais professores e funcionários que prestaram auxílio direto ou indireto para a realização deste trabalho.

Gostaria também de lembrar dos colegas e amigos que passaram esses anos juntos em sala de aula e principalmente nesta etapa de conclusão pelo incentivo e apoio prestado. Alencar, Charles, Diener, Kuhnen, Edson, Velloso, obrigado por estarem ao meu lado.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS

CASE *Computer Aided Software Engineering* – Engenharia de Software Auxiliada por Computador

DDL *Definition Data Language* – Linguagem de definição de dados

DER Diagrama de Entidade Relacionamento

DFD Diagrama de Fluxo de Dados

OMT *Object Modeling Technique* – Técnica de Modelagem de Objetos

OO Orientação a objetos

SQL *Strutured Query Language* – Linguagem de consulta estruturada

UML *Unified Modeling Language* – Linguagem de Modelagem Unificada

XML *Extensive Markup Language* – Linguagem de Marcação Extensiva

RESUMO

O presente trabalho apresenta uma ferramenta de suporte ao engenheiro de software no mapeamento de especificações estruturadas para especificações orientadas a objetos. O mapeamento tem como origem diagramas de entidade relacionamento, diagramas de fluxos de dados e dicionário de dados de uma ferramenta CASE. Como resultado final do mapeamento são gerados o diagrama de casos de uso, diagrama de classes e diagrama de seqüência, representados através da UML (*Unified Modeling Language*). A estratégia de mapeamento utilizada foi baseada no trabalho desenvolvido por Joseph George.

ABSTRACT

The paper presents a tool to support the software engineer in the mapping from structured specifications to oriented objects specifications. The mapping has as origin entity relationship diagram, data flows diagram and data dictionary of a CASE tool. As final result of the mapping is generated the use-case diagram, class diagram and sequence diagram, represented through UML (Unified Modeling Language). The used strategy of mapping was based on the paper developed for Joseph George.

1 INTRODUÇÃO

Uma das preocupações da indústria de software é a necessidade de criar softwares e sistemas mais complexos, exigindo menos tempo de desenvolvimento, softwares de fácil manutenção e com um custo de desenvolvimento mais baixo. Nos últimos anos surgiram novas técnicas de desenvolvimento de software, entre elas a orientação a objetos, que alcançaram bons resultados, chamando a atenção dos desenvolvedores de softwares. A técnica de orientação a objetos permite que o software seja construído de objetos que tenham um comportamento específico. Os próprios objetos podem ser construídos a partir de outros, os quais podem ainda ser construídos de outros. A análise de sistemas no mundo orientado a objetos é feita analisando os objetos e os eventos que interagem com esses objetos. O projeto de software é feito reusando-se classes de objetos existentes e, quando necessário, construindo-se novas classes. Um problema encontrado nas empresas de softwares é a dificuldade de se reutilizar e migrar boa parte dos projetos de antigos sistemas, desenvolvidos de forma estruturada, em projetos orientados a objetos. Uma solução para este problema seria a adoção de um processo de mapeamento de modelos estruturados para modelos orientados a objetos.

Segundo George (1996), a ausência de uma estratégia de mapeamento faz com que muitos ambientes de desenvolvimento de softwares utilizem tanto características do modelo estruturado, quantas características do modelo orientado a objetos, em seus processos de desenvolvimento de software. George (1996) apresentou uma proposta para o mapeamento que consiste na obtenção do modelo estruturado, que segundo Pressman (1995), retrata o fluxo e o conteúdo das informações. O resultado da operação de mapeamento é um diagrama de relacionamentos de objetos. Zibell (1996) e Saldanha (1999), concluíram em seus trabalhos, que os resultados alcançados com a utilização da proposta de George (1996), foram satisfatórios e que demonstraram a viabilidade do mapeamento das especificações.

Existem muitas ferramentas que auxiliam os engenheiros de software no processo de desenvolvimento de especificações estruturadas, entre elas a ferramenta CASE *Power Designer* da empresa *Sybase*, que permite ao usuário criar diagramas de entidade relacionamento, dicionário de dados e diagramas fluxo de dados (Sybase, 2002). Este trabalho foca esta ferramenta por ser muito utilizada no meio acadêmico na FURB.

O modelo orientado a objetos deve ser especificado em uma linguagem de modelagem de objetos. Segundo Furlan (1998), a UML (*Unified Modeling Language*) é uma linguagem padrão para especificar, visualizar, documentar e construir artefatos de um sistema que pode ser utilizado com todos os processos ao longo do ciclo de desenvolvimento e através de diferentes tecnologias de implementação. Esta linguagem é a sucessora da onda de métodos de análise e projeto orientado a objetos, que surgiu no final dos anos oitenta e no início dos anos noventa. Mais especificamente, ela unifica os métodos de Booch, Rumbaugh e Jacobsen (Fowler, 2001). Este trabalho trata da orientação a objetos e mais especificamente da UML.

As principais justificativas para a realização deste trabalho são a carência de ferramentas que possibilitem o mapeamento de especificação estruturada para especificação orientada a objetos e a necessidade de se continuar e aprimorar os trabalhos iniciados por Zibell (1996) e Saldanha (1999) que tratam deste problema.

1.1 OBJETIVOS

O objetivo principal deste trabalho é a construção de uma ferramenta que apóie o engenheiro de software no processo de mapeamento de modelos estruturados para modelos orientados a objetos.

Os objetivos específicos do trabalho são:

a) construir um protótipo de ferramenta CASE que permita a editoração gráfica de diagramas de entidade relacionamento e diagramas de fluxo de dados;
b) construir um protótipo de ferramenta CASE que permita a editoração gráfica de diagramas de caso de uso, diagramas de classes e diagramas de seqüência da UML;
c) permitir a leitura do repositório da ferramenta CASE *Power Designer 6*.

1.2 ESTRUTURA DO TRABALHO

A seguir será apresentada uma breve descrição de cada capítulo do trabalho.

O primeiro capítulo apresentou uma introdução ao assunto estudado neste trabalho, bem como a justificativa para realização do mesmo e seus objetivos.

O segundo capítulo apresenta uma breve revisão bibliográfica sobre especificação estruturada e especificação orientada a objetos, abordando conceitos e diagramas.

O terceiro capítulo apresenta a proposta de mapeamento de especificação estruturada para especificação orientada a objetos utilizada no trabalho e na ferramenta.

O quarto capítulo apresenta a descrição da ferramenta de apoio ao mapeamento e detalhes de sua implementação.

O quinto capítulo apresenta as conclusões obtidas através da realização deste trabalho e também algumas sugestões para futuros trabalhos.

2 ESPECIFICAÇÃO DE SISTEMAS

Segundo Martin (1991), a especificação do sistema é um elo entre a análise e o projeto. Ela fornece uma descrição dos requisitos do sistema a ser construído. O principal objetivo da análise é produzir uma especificação do sistema que defina a estrutura do problema a ser resolvido de acordo com a visão do usuário.

2.1 ESPECIFICAÇÃO ESTRUTURADA

A análise estruturada foi projetada para ser compatível com o projeto estruturado e fornecer a melhor entrada possível para ele. Quando a análise estruturada antecede o projeto, a especificação está na forma desejada por ele, e assim o projeto pode começar imediatamente. A especificação do sistema produzida pelo processo de análise estruturada é chamada de especificação estruturada e tem as seguintes características (Martin, 1991):

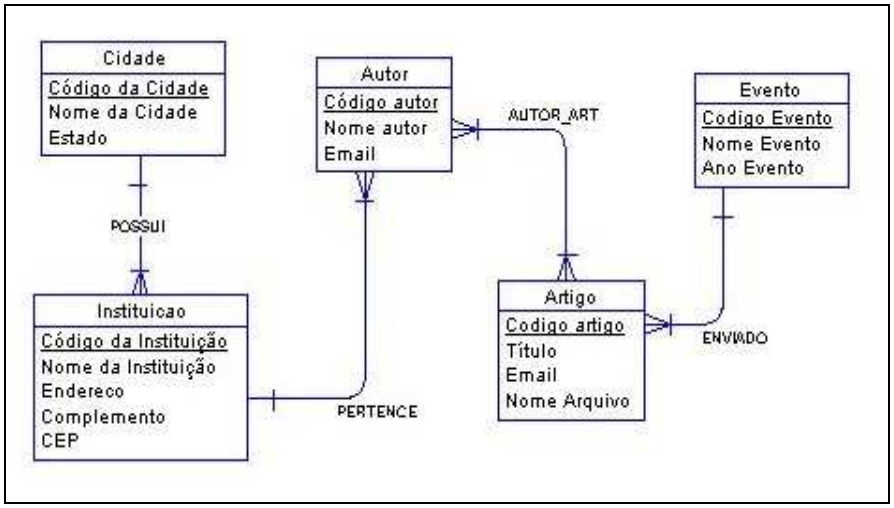
a) é gráfica – um modelo gráfico dos processos ou procedimentos do sistema;
b) é decomposta – capaz de mostrar os componentes do sistema em diversos níveis de detalhes;
c) é um modelo *top-down* hierárquico – desenvolvido pelo uso da decomposição funcional;
d) é lógica – em contrapartida ao modelo físico dos processos ou procedimentos do sistema.

Entre os principais componentes das especificações estruturadas encontram-se o diagrama de entidade relacionamento, diagrama de fluxo de dados e o dicionário de dados que são descritos a seguir.

2.1.1 DIAGRAMA ENTIDADE RELACIONAMENTO (DER)

Yourdon (1990) descreve DER como um modelo em rede que descreve a diagramação dos dados armazenados de um sistema em alto nível de abstração (Figura 1).

FIGURA 1 - DIAGRAMA DE ENTIDADE RELACIONAMENTO



O DER é formado pelos seguintes componentes (Pompilho, 1994): entidades, classe de entidades, relacionamentos, atributos, domínios e generalização.

2.1.1.1 ENTIDADE

Pompilho (1994) define entidade como alguma coisa de desempenha um papel específico no sistema que está sendo modelado. Algo sobre o qual se deseja guardar informações. Uma entidade pode ser:

a) um objeto real, como um livro, um lugar, um avião;
b) uma pessoa, como um empregado, um contribuinte, um aluno, um cidadão;
c) um conceito abstrato, como um curso, uma cor, uma empresa;
d) um acontecimento, isto é, uma situação em que algo está ocorrendo ou está planejado, como o fornecimento de uma encomenda, a inscrição de um aluno em um curso, o recebimento de uma encomenda, um casamento.

Um grupo de entidades possuindo os mesmos atributos forma um conjunto ou uma classe de entidades. Uma classe de entidades é apresentada por um grupo de entidades que possuem características comuns. Exemplo de classes de entidades: o conjunto de todos os empregados, fornecedores, departamentos, alunos e etc.

2.1.1.2 RELACIONAMENTO

Pompilho (1994) define relacionamento como um mapeamento (regra de associação entre dois conjuntos) entre duas classes de entidades. Por exemplo, o relacionamento que expressa todas as encomendas feitas a um fornecedor pode ser assim definido: fornecedor fornece encomenda ou encomenda é fornecida por fornecedor, onde fornecedor e encomenda são classes de entidades.

2.1.1.3 ATRIBUTO

O interesse em identificar entidades está na necessidade de armazenar dados a seu respeito. Tais dados representam características destas entidades. As características ou propriedades das entidades recebem a denominação de atributos. Segundo Pompilho (1994), atributo pode ser entendido como sendo aquilo que se percebe a respeito da entidade como constituindo a essência dela. Todas as entidades de uma mesma classe possuem os mesmos atributos. A entidade pode conter um ou mais atributos como sendo único que é denominado chave primária, assim nenhuma outra entidade poderá ter o mesmo valor para tal atributo que já contenha nesta entidade. Um atributo é obrigatório, ou não nulo, quando não for permitida a omissão desta informação.

2.1.1.4 DOMÍNIO

Pompilho (1994) descreve domínio como sendo um conjunto de valores válidos para um determinado atributo. Cada atributo de uma entidade é associado a um domínio de valores, que podem ser um conjunto de números inteiros, reais, cadeia de caracteres ou qualquer outro tipo de valores.

2.1.1.5 GENERALIZAÇÃO OU HERANÇA

Generalização ou herança é uma abstração poderosa para compartilhar similaridades entre entidades e ao mesmo tempo preservar suas diferenças. De acordo com Cougo (1997) a estrutura de generalização ou herança, procura representar graficamente o fato de uma entidade aproveitar a estrutura de outra entidade que tenha em comum algumas características.

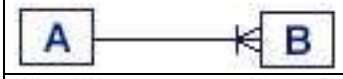
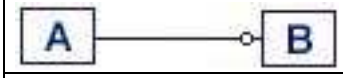
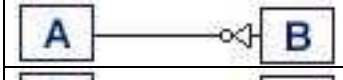
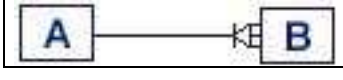
2.1.1.6 REPRESENTAÇÃO GRÁFICA

Para a representação gráfica do DER existem várias notações. Neste trabalho foi utilizada a notação de Martin (1991) também chamada de notação “pé-de-galinha”. As entidades são representadas por retângulos, que na parte superior apresentam o nome da entidade e opcionalmente em seguida a lista de atributos, onde o atributo identificador aparece sublinhado. A Figura 2 mostra duas entidades, aluno e veículo.

FIGURA 2 - REPRESENTAÇÃO DE ENTIDADES

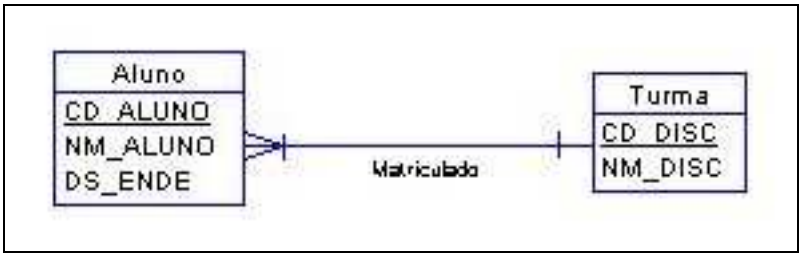
As linhas que ligam uma entidade a outra representam os relacionamentos entre as entidades. Os símbolos nas extremidades das linhas mostram o tipo de mapeamento válido entre as entidades. Os símbolos que representam o tipo de mapeamento e seu respectivo significado são apresentados no Quadro 1:

QUADRO 1 - RELACIONAMENTOS DE ENTIDADES

Relacionamento	Mínimo	Máximo	Descrição
	1	1	Cada entidade da classe “A” está associada a uma única entidade da classe “B”
	1	Várias	Cada entidade da classe “A” está associada a uma ou várias entidades da classe “B”
	0	1	Cada entidade da classe “A” está associada a zero ou a uma única entidade da classe “B”
	0	Várias	Cada entidade da classe “A” está associada a zero, uma ou várias entidades da classe “B”
	0	1	Entidade “B” depende de entidade “A”
	1	Várias	Entidade “B” depende de entidade “A”

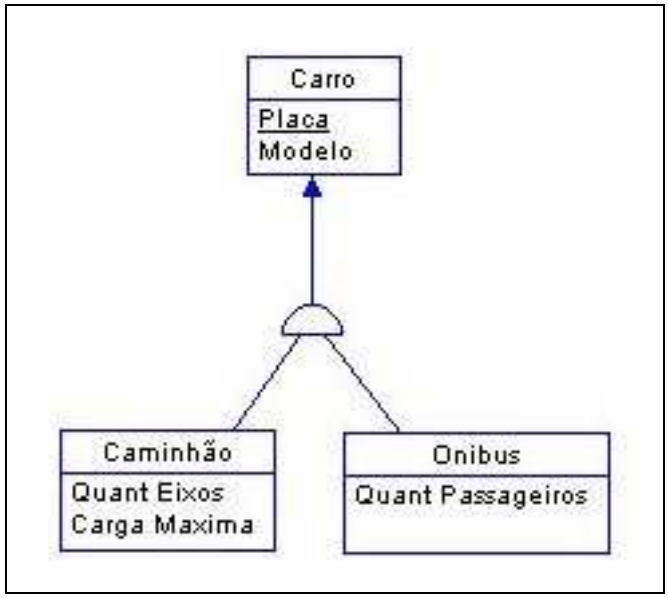
A Figura 3 apresenta um exemplo de relacionamento, onde o aluno está matriculado a uma turma e a turma possui um ou mais alunos.

FIGURA 3 - RELACIONAMENTO ENTRE DUAS ENTIDADES



A representação gráfica de generalização se dá através de uma semicircunferência ligada por uma seta a entidade de origem da herança. Na parte inferior desta semicircunferência estão ligadas as entidades que herdam características da entidade conforme Figura 4.

FIGURA 4 - REPRESENTAÇÃO DE GENERALIZAÇÃO



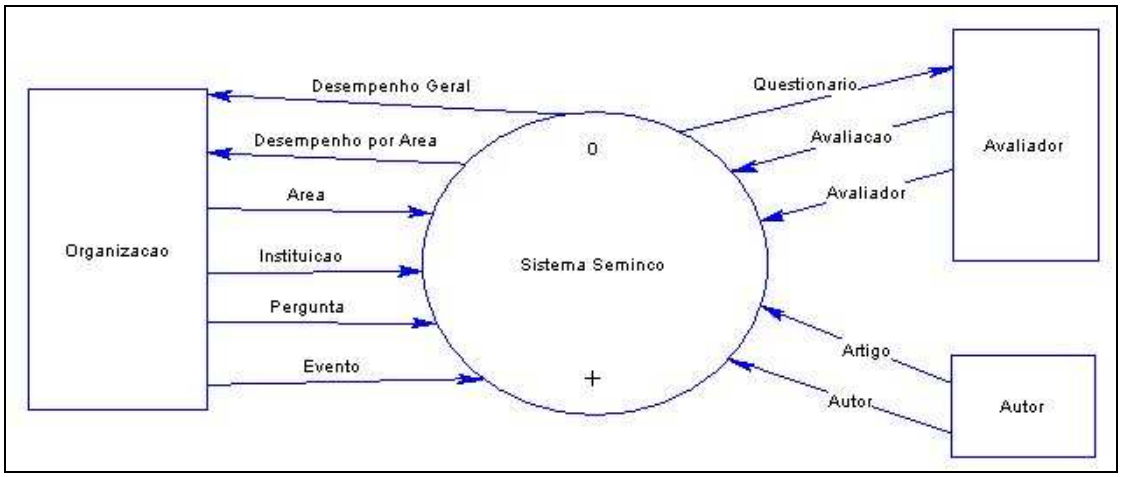
2.1.2 DIAGRAMA DE CONTEXTO

Outro componente da especificação estruturada é o diagrama de contexto. Pompilho (1994) descreve que o diagrama de contexto representa o sistema por um único processo e suas interligações com as entidades externas, mostrando apenas as interfaces do sistema com o ambiente em que ele está inserido. Não são apresentados detalhes do processamento interno do sistema. As entidades externas devem ser denominadas pelo papel que elas desempenham em relação ao sistema, e não pela pessoa ou entidade específica que o faz. No diagrama de contexto não pode haver fluxo de dados partindo diretamente de uma entidade externa em

direção a outra. Segundo Pompiho (1994), tal fluxo estaria fora do contexto do sistema, pois não entra nem sai da bolha que representa o sistema como um todo. No diagrama de contexto não deve aparecer o depósito de dados.

Todo sistema pode, a partir do diagrama de contexto, ser decomposto em diversas funções que se interligam. As entidades externas no diagrama de fluxo de dados expandido (DFD que apresenta as funções do sistema) são as mesmas do diagrama de contexto no qual, entretanto, são mostrados apenas os fluxos de dados que representam a interface do sistema com as entidades externas. Para cada função do sistema pode-se aplicar esse mesmo princípio e decompô-la em funções mais simples. A Figura 5 apresenta um exemplo de diagrama de contexto.

FIGURA 5 - DIAGRAMA DE CONTEXTO



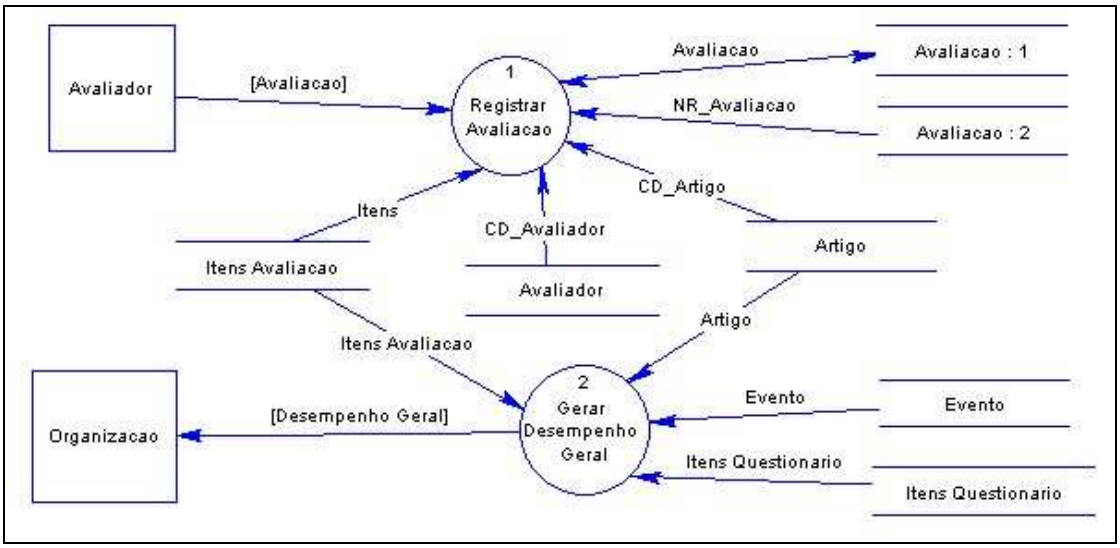
2.1.3 DIAGRAMA DE FLUXO DE DADOS (DFD)

Pompilho (1994) define DFD como uma forma gráfica de mostrar a interdependência das funções que compõem um sistema, apresentando fluxos de dados entre elas. Mostra ainda os arquivos lógicos de dados, que são denominados depósitos de dados, bem como as entidades externas, denominação dada tanto à origem dos fluxos de dados que chegam ao sistema, como ao destino dos fluxos que dele partem.

Na definição de Martin (1991), o diagrama de fluxo de dados apresenta os processos e o fluxo de dados entre eles. Em alto nível, é usado para mostrar eventos de negócios e as transações resultantes desses eventos, sejam elas feitas através de papéis ou por computador.

Em nível mais baixo, é usado para mostrar programas ou módulos de programas e o fluxo de dados entre as rotinas (Figura 6).

FIGURA 6 - DIAGRAMA DE FLUXO DE DADOS



O DFD é composto pelos seguintes componentes (Gane, 1983): entidade externa, processo, fluxo de dados e depósito.

2.1.3.1 ENTIDADE EXTERNA

Segundo Gane (1983), as entidades externas são, com maior freqüência, categorias lógicas de coisas ou pessoas que representas uma fonte ou destino para transações; por exemplo, clientes, empregados e aeronaves. Elas também podem ser uma fonte ou um destino específico como, Departamento de Contas, Receita Federal e Presidente. Quando o sistema enfocado recebe dados de um outro sistema ou fornece dados a ele, aquele outro sistema é considerado entidade externa.

Yourdon (1990) também cita entidade externa pelo nome de terminador, e define que entidade externa é a representação de uma pessoa ou grupo de pessoas, uma organização externa ou uma empresa do governo ou setor que esteja dentro da mesma companhia, mas fora do controle do sistema que está sendo modelado. A entidade externa tem a função de fornecer ou receber informações do sistema.

2.1.3.2 PROCESSO

Yourdon (1990) descreve processo como sendo o primeiro componente do DFD, que também pode ser chamado de função ou transformação. O processo mostra uma parte do sistema, a que transforma entradas em saídas, isto é, mostra como uma ou mais entradas são convertidas em saídas.

Para Pompilho (1994), que identifica processo pelo nome de função, o processo é um componente de um sistema onde somente os dados de entrada e os dados de saída são conhecidos. Não se conhece explicitamente nada a respeito do processo interno de transformação dos dados de entrada em dados de saída. As funções representam as ações que o sistema executa.

2.1.3.3 FLUXO DE DADOS

O fluxo de dados conduz o fluxo de informações através dos processos de um sistema, o fluxo de dados mostra como os processos são interligados com as entidades externas, que fornecem e recebem informação, e depósitos de dados, que armazenam as informações necessárias (Martin, 1991).

2.1.3.4 DEPÓSITO

Na definição de Yourdon (1990), depósito de dados é utilizado para se modelar uma coleção de pacotes de dados em repouso. O depósito de dados representa um arquivo lógico, arquivo ou banco de dados onde as informações serão armazenadas.

2.1.3.5 REPRESENTAÇÃO GRÁFICA

Para a representação gráfica do DFD existem diversas notações, as mais utilizadas são de Gane (1983) e Yourdon (1990), porém neste trabalho será utilizada a notação de Yourdon (1990).

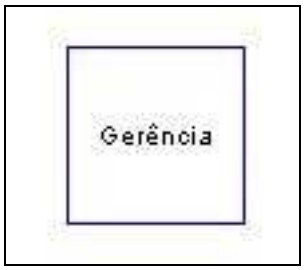
Yourdon (1990) representa o processo com um círculo e ao centro da circunferência o nome do processo. Martin (1991) representa o processo com um círculo ou um retângulo com os vértices arredondados. O nome do processo descrito dentro do círculo. Deve-se utilizar nomes significativos para definir a operação executada pelo processo, geralmente utilizando

verbos. Na parte superior co círculo opcionalmente pode-se adicionar o número que identifica o processo. Processo de número zero identifica que está em nível de contexto. Nenhuma outra informação é apresentada no processo (Figura 7).

FIGURA 7 - REPRESENTAÇÃO DE PROCESSO

A entidade externa é representada por um retângulo e ao centro do retângulo é apresentado o nome da entidade externa (Figura 8).

FIGURA 8 - REPRESENTAÇÃO DE ENTIDADE EXTERNA

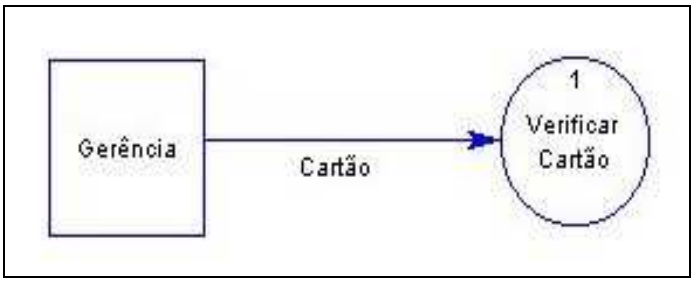


O depósito de dados é representado por duas linha paralelas em formato de retângulo e ao centro o nome do depósito (Figura 9).

FIGURA 9 - REPRESENTAÇÃO DE DEPÓSITO DE DADOS

O fluxo de dados é representado por uma linha que liga entidades externas a processos ou processos a depósitos de dados. Nas extremidades desta linha pode aparecer uma flecha indicando se o fluxo está realizando uma operação de leitura ou gravação. A flecha sinaliza a direção em que os dados estão fluindo. Em paralelo a esta linha aparece o nome que identifica o fluxo (Figura 10).

FIGURA 10 - REPRESENTAÇÃO DE FLUXO DE DADOS



2.1.4 DICIONÁRIO DE DADOS

Segundo DeMarco (1989) o dicionário de dados é parte integrante da especificação estruturada, sem ele, os diagramas de entidade relacionamento e fluxo de dados, são apenas imagens que transmitem alguma idéia do que está acontecendo em um sistema. O papel mais importante de qualquer dicionário é fornecer-lhe um único lugar para que possa ser procurada nele definição de termos não conhecida.

O dicionário de dados é um conjunto de definições formais de todos os dados que aparecem como fluxos, depósitos, entidades ou atributos nos diagramas estruturados.

O Quadro 2 apresenta a notação tipicamente utilizada.

QUADRO 2 - NOTAÇÃO DE DICIONÁRIO DE DADOS

Simbolo	Descrição	Alternativa
=	Composição	
+	Concatenação	
{ }	Iteração: 0 (zero) ou mais ocorrências	
[ ]	Seleção de 1 (um) entre os presentes	
( )	Opção	
“”	Valor discreto	
@	Chave de identificação	Sublinhado
|	Separador de alternativas	,
* *	Comentários	

O Quadro 3 apresenta um exemplo de dicionário de dados.

QUADRO 3 - EXEMPLO DE DICIONÁRIO DE DADOS

```
nome = titulo_de_cortesia + (promeir_nome) + ultimo_nome
titulo_de_costesia = [Sr. | Sra. | Prof. | Dr. | Arq. | Eng.]
ultimo_nome = {caracter}
caracter = [ A-Z | a-z |  | -]
```

2.2 ESPECIFICAÇÃO ORIENTADA A OBJETOS

A orientação a objetos introduz diversos conceitos que são imprescindíveis para o entendimento do processo de especificação orientada a objetos. Ambler (1998), Coad (1992), Coleman (1996), Martin (1995), Page-Jones (2001) entre outros autores, descrevem em suas obras todos os conceitos sobre orientação a objetos.

O modelo orientado a objetos deve ser especificado em uma linguagem de modelagem de objetos. A UML tem por objetivo criar uma modelagem padrão de desenvolvimento de sistemas orientados a objetos, ela incorpora as noções de desenvolvimento de software de forma totalmente gráfica.

2.2.1 *UNIFIED MODELING LANGUAGE* (UML)

Segundo Furlan (1998), a *Unified Modeling Language* (UML) é a linguagem padrão para especificar, visualizar, documentar e construir artefatos de um sistema que pode ser utilizado com todos os processos ao longo do ciclo de desenvolvimento e através de diferentes tecnologias de implementação.

A UML é uma tentativa de padronizar a modelagem orientada a objetos de uma forma que qualquer sistema, seja qual for o tipo, possa ser modelado corretamente, com consistência, fácil de se comunicar com outras aplicações, simples de ser atualizado e compreensível. A UML é uma linguagem de modelagem e não uma metodologia (Furlan 1998).

Para Fowler (2001), a UML é sucessora da onda de métodos de análise e projeto orientado a objetos que surgiu no final dos anos oitenta e no início dos noventa. Mais especificamente, ela unifica os métodos de Booch, Rumbaugh (OMT) e Jacobson, mas o seu alcance é bem maior.

A UML é composta pelas seguintes partes (Knop, 1999):

a) visões: As visões mostram diferentes aspectos do sistema que está sendo modelado. A visão não é um gráfico, mas uma abstração consistindo em uma série de diagramas;
b) modelos de elementos: Os conceitos usados nos diagramas são modelos de elementos que representam definições comuns da orientação a objetos como as classes, objetos, mensagem, relacionamento, dependências e heranças;
c) mecanismos gerais: Os mecanismos gerais provêm comentários suplementares, informações, ou semântica sobre os elementos que compõem os modelos;
d) diagramas: Os diagramas são gráficos que descrevem o conteúdo em uma visão. A UML possui nove tipos de diagramas que são usados em combinação para prover todas as visões do sistema.

2.2.2 DIAGRAMAS DA UML

O modo de descrever os vários aspectos de modelagem pela UML é através da notação definida pelos seus vários tipos de diagramas. Furlan (1998) define diagrama como uma apresentação gráfica de uma coleção de elementos de modelo, freqüentemente mostrado como um gráfico conectado de arcos (relacionamentos) e vértices (os outros elementos).

Para Booch (2000) os diagramas são utilizados para a visualização de blocos de construções básicos como classes, interfaces, colaborações, componentes, nós, dependências, generalização e associações. Os diagramas são utilizados para visualizar o sistema sob diferentes perspectivas. Uma vez que nenhum sistema complexo pode ser compreendido em sua totalidade a partir de uma única perspectiva, a UML define um número de diagramas que permite dirigir o foco para aspectos diferentes de seu sistema de maneira independente.

A UML fornece os seguintes diagramas (Furlan, 1998):

a) diagrama de classe;
b) diagrama de caso de uso;
c) diagrama de seqüência;
d) diagrama de colaboração;
e) diagrama de estado;
f) diagrama de atividade;

g) diagrama de componente;
h) diagrama de implantação.

Para a realização do mapeamento são utilizados neste trabalho os diagramas de caso de uso, diagrama de classes e diagrama de seqüência por serem os diagramas mais utilizados no meio acadêmico e serão descritos a seguir.

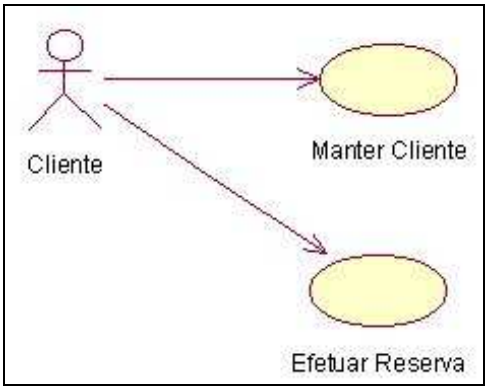
2.2.3 DIAGRAMA DE CASO DE USO

Os diagramas de casos de uso são um dos cinco diagramas disponíveis na UML para a modelagem de aspectos dinâmicos de sistemas (Figura 11). Para Booch (2000) os diagramas de casos de uso têm um papel central para a modelagem do comportamento de um sistema, de um subsistema ou uma classe.

Furlan (1998) apresenta o propósito primário dos casos de uso:

a) descrever os requisitos funcionais do sistema de maneira consensual entre usuários e desenvolvedores de sistemas;
b) fornecer uma descrição consistente e clara sobre as responsabilidades que devem ser cumpridas pelo sistema, além de formar a base para a fase de desenho;
c) oferecer as possíveis situações do mundo real para o teste do sistema.

FIGURA 11 - DIAGRAMA DE CASO DE USO



2.2.3.1 ATOR

Segundo Furlan (1998) o mundo externo é representado por atores que desempenham papéis. Um ator é um agente que interage com o sistema, um tipo de usuário ou categoria com papel definido, podendo incluir seres humanos, máquinas, dispositivos ou outros sistemas.

2.2.3.2 CASO DE USO

O propósito de um caso de uso é definir o comportamento de uma classe passiva sem revelar sua estrutura interna. Para Furlan (1998) casos de uso representam uma primeira ordem de divisão do domínio de problema em seus comportamentos fundamentais e cada um representa um conjunto de cenários de controle que é encapsulado dentro de um único objeto.

2.2.3.3 INTERAÇÃO

Um ator comunica-se com o caso de uso. Assim cada participação sua é mostrada conectando-se o símbolo de caso de uso por um caminho sólido. Este caminho sólido é chamado de interação (Furlan, 1998).

Para Furlan (1998), existem três tipos de interação:

a) comunicação ou associação: um ator comunica-se com um caso de uso, assim cada participação sua é mostrada conectando-se o símbolo de ator ao caso de uso por um caminho sólido;
b) extensão: mostra o comportamento de exceção e casos especiais que aumentariam a quantidade de casos de uso no modelo;
c) uso ou inclusão: quando um número de casos de uso tem comportamento comum, esse comportamento pode ser modelado em um simples caso de uso que é utilizado por outros casos.

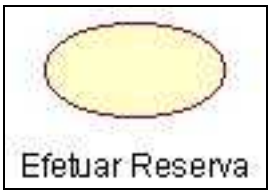
2.2.3.4 REPRESENTAÇÃO GRÁFICA

O ator é representado por uma imagem de um boneco em traços finos e abaixo do boneco aparece o nome do ator, como mostra a Figura 12.

FIGURA 12 - REPRESENTAÇÃO DO ATOR

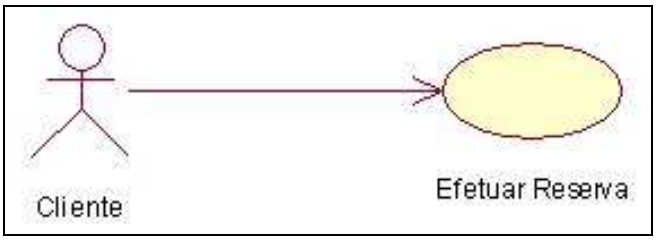
O caso de uso é representado por uma circunferência e ao centro ou abaixo aparece o nome do caso de uso (Figura 13).

FIGURA 13 - REPRESENTAÇÃO DE CASO DE USO



A interação é representada por uma linha reta interligando o ator e o caso de uso, possibilita também a interligação entre casos. A comunicação é simbolizada por uma linha sólida com uma seta aberta (Figura 14), a extensão é simbolizada por uma linha tracejada e uma seta aberta e a inclusão é simbolizada por uma linha tracejada com uma seta fechada.

FIGURA 14 - REPRESENTAÇÃO DE UMA INTERAÇÃO

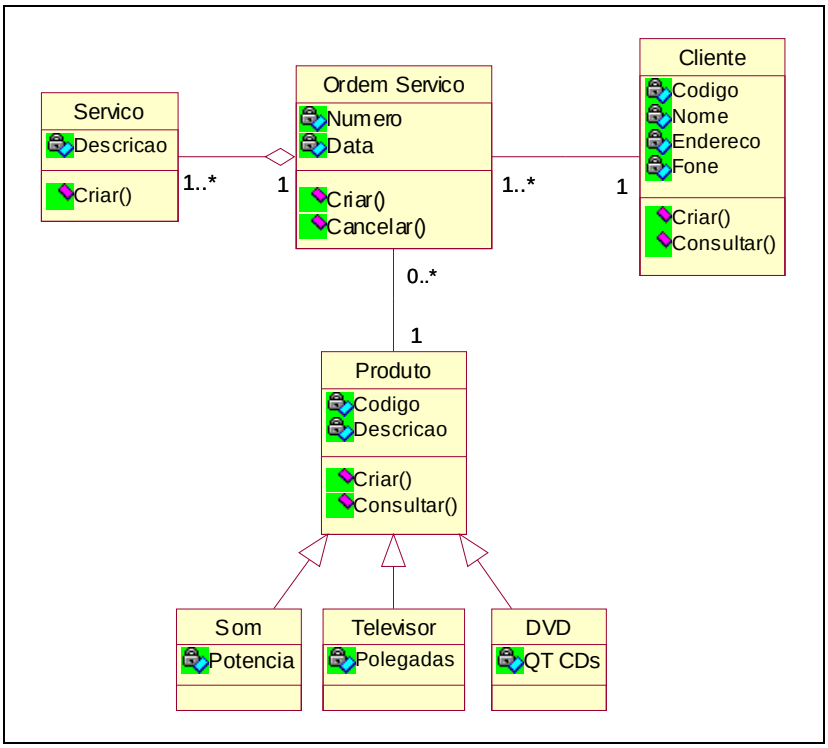


2.2.4 DIAGRAMA DE CLASSES

Os diagramas de classes são os diagramas encontrados com maior freqüência na modelagem de sistemas orientados a objetos. Um diagrama de classes mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos. Segundo Booch (2000) o diagrama de classes é utilizado para fazer a modelagem da visão estática do projeto de um sistema. Eles são importantes não só para a visualização, a especificação e a documentação de modelos estruturais, mas também para a construção de sistemas executáveis por intermédio de engenharia de produção e reversa.

Furlan (1998) descreve o diagrama de classes como uma estrutura lógica estática em uma superfície de duas dimensões mostrando uma coleção de elementos declarativos de modelo, como classes, tipos e seus respectivos conteúdos e relações (Figura 15).

FIGURA 15 - DIAGRAMA DE CLASSES



2.2.4.1 CLASSE

Assim como na definição de orientação a objetos, a classe no diagrama de classes representa um conjunto de coisas reais ou abstratas que são reconhecidas como sendo do mesmo tipo por compartilhar as mesmas características de atributos, métodos, relações e semânticas (Furlan, 1998).

A classe permite uma definição de visibilidade a seus atributos e métodos e é definido a seguir (Furlan, 1998):

a) visibilidade pública: significa que todos têm acesso podendo o atributo ou serviço ser utilizado pela própria classe e também pelas outras classes. É representada pelo sinal de adição (+). Este símbolo é colocado ao lado esquerdo do atributo ou método da classe. Em algumas ferramentas pode ser encontrado em formato gráfico como um retângulo em diagonal;

b) visibilidade protegida: significa que o atributo ou método é acessado através da própria classe ou por classes ao longo do pacote no qual a classe foi definida. É representada pelo sinal de sustenido (#) e em algumas ferramentas pode ser encontrado em formato de uma chave com um retângulo em diagonal;

c) visibilidade privada: significa que o atributo ou o método podem apenas ser acessados pela própria classe. É representada pelo sinal de menos (-) e em algumas ferramentas pode ser encontrado em formato de um cadeado com um retângulo em diagonal.

2.2.4.2 ASSOCIAÇÕES

Há quatro tipos principais de relacionamentos conforme Furlan (1998):

a) generalização ou especialização: indica o relacionamento entre um elemento mais geral e uma elemento mais específico (respectivamente, superclasse e subclasse) também conhecido como herança;

b) agregação: usada para denotar relacionamentos todo/parte, geralmente usado para representar que uma classe é parte integrante de outra classe;

c) associação: utilizada para denotar relacionamentos entre classes não correlatas. Na UML, uma associação é definida como um relacionamento que descreve um conjunto de vínculos, onde vinculo é definido como uma conexão semântica entre tuplas de objetos;

d) dependência: é um relacionamento entre elementos, um independente e outro dependente, onde uma mudança no elemento independente afetará o elemento dependente.

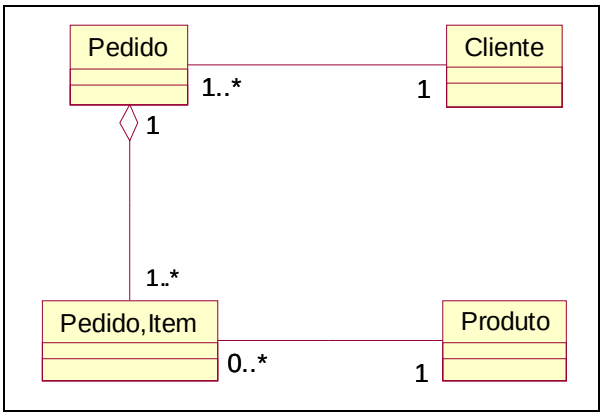
2.2.4.3 REPRESENTAÇÃO GRÁFICA

A classe é representada por um retângulo onde na parte superior encontra-se o nome da classe e a seguir seus atributos e métodos (Figura 16).

FIGURA 16 - REPRESENTAÇÃO DE CLASSE

As associações são representadas por linhas que interconectam as diversas classes do sistema. A generalização ou herança é representada por uma seta triangular no lado da classe que herda suas características. A agregação é representada por um losango no lado da classe que representa o todo. A associação é representada apenas por uma linha e em cada lado apresenta a cardinalidade possível para as classes (Figura 17). A dependência é representada por uma linha tracejada com uma seta no lado da classe independente.

FIGURA 17 - REPRESENTAÇÃO DE ASSOCIAÇÕES

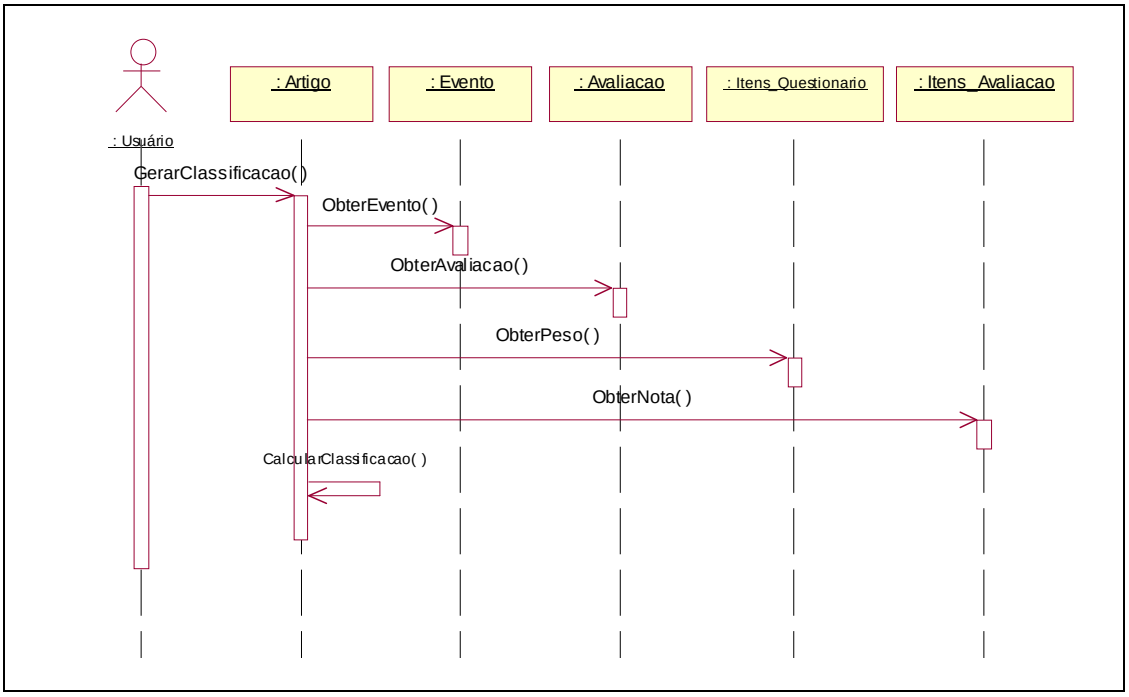


2.2.5 DIAGRAMA DE SEQÜÊNCIA

O diagrama de seqüência descreve de maneira genérica as interações entre os objetos organizados numa seqüência de tempo e de troca de mensagens. Neste diagrama percebe-se a seqüência de mensagens trocadas entre os objetos e também se observa alguns comportamentos em um dado instante da execução do programa (Furlan 1998).

Segundo Booch (2000) diagrama de seqüência é um diagrama de interação que dá ênfase à ordenação temporal de mensagens. Graficamente, um diagrama de seqüência é uma tabela que mostra objetos distribuídos no eixo X e mensagens em ordem crescente no tempo no eixo Y (Figura 18).

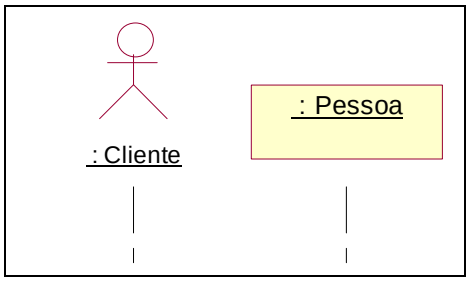
FIGURA 18 - DIAGRAMA DE SEQÜÊNCIA



2.2.5.1 REPRESENTAÇÃO GRÁFICA

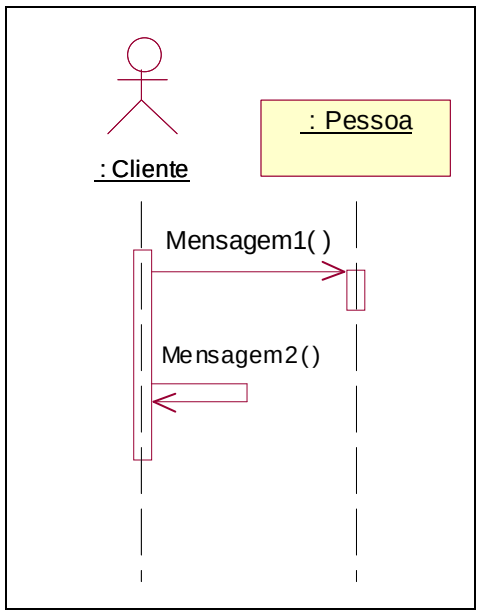
Em um diagrama de seqüência um objeto é mostrado como uma caixa na parte superior de uma linha tracejada vertical. Esta linha vertical é chamada de linha de vida do objeto (Figura 19).

FIGURA 19 - REPRESENTAÇÃO DE OBJETO



Cada mensagem é representada por uma flecha entre as linhas de vida de dois objetos. A ordem na qual estas mensagens ocorrem é mostrada da parte superior para a parte inferior da página. Cada mensagem é rotulada, no mínimo, com o nome da mensagem. Pode existir uma autochamada no qual um objeto envia uma mensagem para si mesmo enviando a flecha de mensagem de volta para a mesma linha de vida (Figura 20).

FIGURA 20 - REPRESENTAÇÃO DE MENSAGEM



3 ESTRATÉGIA DE MAPEAMENTO

O professor Joseph George do Departamento de Ciências da Computação da Universidade de Mississippi, elaborou uma estratégia de mapeamento de especificações estruturadas para especificações orientadas a objetos. A estratégia consiste em ler os diagramas de entidade relacionamentos e fluxos de dados da especificação estruturada e mapear para relacionamentos de objetos, baseados nos conceitos de orientação a objetos.

George (1996) estabeleceu seis fases para o mapeamento:

a) fase 1: identificação dos objetos;
b) fase 2: identificação dos atributos;
c) fase 3: identificação e associação entre operações;
d) fase 4: identificação de associação entre objetos;
e) fase 5: identificação de mensagens entre os objetos;
f) fase 6: criação e refinamento da estrutura de objetos e diagrama de mensagens (OSMD).

A descrição de todas as fases da estratégia de George (1996) pode ser encontrada no trabalho de Zibell (1996) e Saldanha (1999). Zibell (1996) utilizou a estratégia de George (1996) no desenvolvimento do seu trabalho de conclusão de curso, que consistia na obtenção de especificação estruturada e o resultado do mapeamento é um diagrama de relacionamento de objetos da abordagem OMT de Rumbaugh. O protótipo desenvolvido no trabalho de Zibell utilizou as informações do DER, DFD e dicionários de dados obtidas através da leitura dos arquivos exportados pela ferramenta CASE ET-SADS. Segundo Zibell (1996) a técnica de George (1996) foi escolhida por apresentar uma maior consistência na definição das etapas a serem realizadas. Os resultados obtidos com a utilização da estratégia foram satisfatórios e demonstram a viabilidade do mapeamento das especificações. Zibell (1996) sugere em seu trabalho a utilização de outras abordagens além da OMT, como por exemplo a UML que será utilizada neste trabalho. Saldanha (1999) também utilizou a técnica proposta por George (1996) em seu trabalho de conclusão de curso. As informações da especificação estruturada são obtidas através da leitura do repositório da ferramenta CASE System Architect. O resultado do mapeamento é uma lista de relacionamento de objetos. Saldanha (1999) sugeriu estudar outras ferramentas CASE para obter de seus repositórios as informações da especificação estruturada e gerar os diagramas de forma gráfica.

George (1996) sugere apenas o mapeamento para um diagrama de objetos. Esta proposta foi adaptada para o diagrama de classes da UML. A estratégia de mapeamento para o diagrama de caso de uso e para o diagrama de seqüência foi proposta a partir dos estudos realizados neste trabalho.

A estratégia de mapeamento descrita para diagrama de classes está baseada na idéia de se obter as classes, atributos, métodos, relacionamentos e heranças através da interpretação do DER e DFD. Zibell (1996) concluiu em seu trabalho que a eliminação de alguns dos passos descritos por George (1996) não afetariam de forma acentuada o resultado final do mapeamento. Estes passos constituem em sua maioria de técnicas de refinamento do resultado final. Como tentativa de maximizar a eficiência do mapeamento, alguns passos foram acrescentados à estratégia e consistem na criação de novos métodos às classes, baseados nas operações realizadas pelos fluxos sobre os depósitos.

A estratégia de mapeamento proposta neste trabalho para diagrama de caso de uso, diagrama de classes e diagrama de seqüência será descrita a seguir.

3.1 MAPEAMENTO PARA DIAGRAMA DE CASO DE USO

Os passo descritos a seguir tem por objetivo interpretar os diagramas de fluxo de dados e gerar o diagrama de caso de uso da UML, composto por atores, casos de uso e associações:

a) identificar atores: obter todas as entidades externas e para cada uma delas criar um ator;
b) identificar casos de uso: obter todos os processo de nível 1 e para cada processo cria-se um caso de uso;
c) identificar associações: obter todos os fluxos que interligam entidades externas e processo de nível 1, para cada fluxo cria-se uma associação entre os respectivos atores e casos criados pelos passos anteriores, que identifiquem os terminais do fluxo.

Os passos descritos a seguir geram a descrição do caso de uso. Este passo pode ser opcional caso não seja necessário descrever os casos. Os seguintes passos devem ser realizados para cada caso de uso identificado na fase anterior:

a) identificar no processo que gerou o caso de uso a existência de um fluxo de dados partindo de uma entidade externa para um processo e adicionar na descrição que o caso está recebendo informação do ator identificado na fase anterior;
b) identificar todas os depósitos de dados que contenham fluxo de dados ligado ao processo que gerou o caso de uso. Para cada fluxo deve-se criar uma descrição considerando a operação do fluxo de dados e a direção do fluxo entre o processo e entidade. Se o fluxo entre o processo e a entidade for de entrada a operação de leitura representa uma verificação no depósito, se o fluxo for de saída então a operação de leitura será de obtenção de valor no depósito;
c) identificar se o fluxo entre processo e entidade tem direção de saída, então deve descrever que o ator está recebendo informações.

3.2 MAPEAMENTO PARA DIAGRAMA DE CLASSES

Os passo descritos a seguir foram baseados na estratégia desenvolvida por George (1996), que consistem na interpretação dos diagramas estruturados resultando em diagramas de classes da UML, compostos por classes, atributos, métodos e relacionamentos.

3.2.1 FASE 1: IDENTIFICAÇÃO DAS CLASSES

A primeira fase identifica as principais classes do sistema. Os passos que constituem a primeira fase da estratégia de mapeamento são apresentados a seguir:

a) mapear cada entidade no DER em uma única classe;
b) mapear cada entidade externa no DFD em uma única classe;
c) mapear cada depósito no DFD em uma única classe;
d) aperfeiçoar a lista de classes criadas pela execução dos passos de “a” até “c” excluindo todos as classes repetidas e com diferentes nomes possíveis, derivados com resultado de diferentes critérios de aplicação.

3.2.2 FASE 2: IDENTIFICAÇÃO DOS ATRIBUTOS

Esta fase é responsável pela identificação dos atributos associados as classes identificadas na fase 1. Os atributos de uma classe fornecem muitas informações referentes ao

estado atual da classe e também ajudam a definir com maiores detalhes o que na verdade representa a classe.

Os atributos são obtidos quase que completamente do dicionário de dados. Os passos específicos envolvidos nesta fase são:

a) para cada classe na lista de classes, examinar o dicionário de dados em busca de qualquer definição abstrata de tipos de dados que contenham o mesmo nome de uma classe ou correspondam a uma classe. Se deste modo existir uma definição e constituir uma seqüência de componentes, então mapeie os componentes desta definição como atributos deste objeto;
b) listar todos os fluxos de dados no DFD. Para cada item de dados, determinar se é possível que um atributo possa ser de várias classes. Se deste modo, adicione este item de dados para a lista de atributos da classe identificada.

3.2.3 FASE 3: IDENTIFICAÇÃO E ATRIBUIÇÃO DOS MÉTODOS

Esta fase é provavelmente a mais complexa no processo do mapeamento. Esta fase é responsável pelo mapeamento de requisitos funcionais das classes do sistema em forma dos métodos que são definidas sobre as classes.

Os passos que constituem esta fase são: (repita o passo “b” para cada método)

a) mapear cada processo do DFD de nível 1 com um método correspondente do mesmo nome;
b) atribuir o método para uma classe da seguinte forma:
   - se um dos fluxos de entrada para o processo correspondente existir uma classe na lista de classes com o mesmo nome do fluxo correspondente, então o método é atribuído a classe;
   - se não foi identificada uma classe no passo anterior, então para cada fluxo de entrada para o processo correspondente ao método, identificar uma classe da lista de classes com um atributo que possua o mesmo nome do fluxo de dados;
   - se não foi identificada uma classe no passo anterior, então identificar qualquer classe que contenha um subconjunto de atributos semelhantes aos atributos do fluxo de entrada ao processo. Se mais de uma classe for identificada, então

atribui-se o método a uma classe que mais preenche as identificações do contexto do método;

c) mapear todos os fluxos de dados entre depósitos processos e atribuir um método aos respectivos depósitos identificados como classes na fase 1 considerando as operações (*Create, Read, Update* e *Delete*) realizadas pelo fluxo da seguinte forma:
   - se o fluxo for de entrada no depósito verificar as operações do fluxo. Se o fluxo tiver uma operação de "*Create*" então cria-se um método de construção da classe. Se o fluxo tiver uma operação de "*Update*" então cria-se um método de edição dos atributos da classe. Se o fluxo tiver uma operação de "*Delete*" então cria-se uma operação de destruição da classe;
   - se o fluxo for de entrada no processo verificar se existe um fluxo de entrada no processo proveniente de uma entidade externa. Se existir uma entrada então o método cria-se um método de consistência de dados do fluxo. Se o fluxo for de saída para uma entidade externa então cria-se um método de obtenção de dados na classe.

3.2.4 FASE 4: IDENTIFICAÇÃO DE ASSOCIAÇÕES ENTRE CLASSES

Esta fase identifica associações entre classes. Os passos que constituem esta fase são descritos abaixo:

a) para cada relacionamento no DER, se as entidades envolvidas no relacionamento tiverem os mesmos nomes das classes, então cria-se associações entre estas duas classes. A cardinalidade do DER deve ser mapeada para o relacionamento entre as classes. Casos de dependência nos relacionamentos do DER devem ser mapeados para uma agregação no diagrama de classes;

b) examinar pares de classes e determinar se elas correspondem a uma entidade externa e um depósito de dados no DFD que se comunicam com um mesmo processo, se ainda não existir um relacionamento entre as duas classes identificadas então cria-se uma associação entre as classes;

c) mapear cada generalização no DER e criar associações de herança conforme entidades relacionadas na generalização no DER.

3.3 MAPEAMENTO PARA DIAGRAMA DE SEQÜÊNCIA

O mapeamento do diagrama de seqüência consiste na obtenção das informações do diagrama de fluxo de dados para a criação dos objetos e identificação das mensagens entre os objetos. A estratégia descrita a seguir não apresenta uma ordem exata da ocorrência das mensagens, sendo necessário após a realização do mapeamento uma intervenção manual para a ordenação correta das mensagens. A seguir são descritos os passos para a realização do mapeamento:

a) identificar diagrama de seqüência: para cada processo de nível 1 do DFD, deve-se criar um diagrama de seqüência. Para cada diagrama criado neste passo deve-se executar os passos “b” e “c”;

b) identificar objetos:
   - obter todas as entidades externas que estejam ligadas ao processo (que originou o diagrama, conforme passo “a”) e criar para cada entidade um objeto no diagrama de seqüência;
   - obter a classe que contenha um método com o mesmo nome do processo, para esta classe deve criar um objeto no diagrama de seqüência;
   - obter as classes que representem os depósitos e para cada classe criar um objeto no diagrama de seqüência;

c) identificar mensagens:
   - criar uma mensagem entre o objeto da entidade externa e o objeto da classe que possui o método com o mesmo nome do processo. O nome da mensagem será o mesmo nome do processo;
   - criar as mensagens referentes às operações dos fluxos entre o processo e os depósitos entre os objetos criados a partir dos depósitos. Essas mensagens partem sempre do objeto que possui o método com o mesmo nome do processo com destino aos respectivos objetos referenciados pelo fluxo do DFD. Neste passo deve-se identificar apenas as operações de leitura que representam os métodos de verificação e obtenção de dados.

4 DESENVOLVIMENTO DO TRABALHO

Neste capítulo será apresentada uma ferramenta que foi desenvolvida para apoiar e automatizar parte da estratégia de mapeamento de especificação estruturada para especificação orientada a objetos.

4.1 REQUISITOS PRINCIPAIS DO PROBLEMA

O requisito para a realização do mapeamento é a existência de uma estratégia que possibilite interpretar o diagrama de entidade relacionamento e o diagrama de fluxo de dados resultando no diagrama de caso de uso, diagrama de classes e diagrama de seqüência da UML. Esta estratégia foi descrita no capítulo 3 e fundamenta a ferramenta implementada.

Os principais requisitos da ferramenta são:

a) permitir ao usuário editar modelos estruturados compostos pelos diagramas de entidade relacionamento, fluxo de dados e modelos orientados a objetos compostos pelos diagramas de caso de uso, classes e seqüência da UML;
b) permitir a importação de informações contidas em arquivos gerados pela ferramenta CASE *Power Designer* versão 6. Essas informações devem ser apresentadas em forma de diagrama gráfico;
c) permitir ao usuário editar os diagramas importados da ferramenta CASE *Power Designer*;
d) gerar a partir das informações dos diagramas de entidade relacionamento e fluxo de dados, os diagramas de caso de uso, classes e seqüência da UML utilizando a estratégia de mapeamento;
e) permitir ao usuário editar os diagramas gerados a partir do mapeamento.

4.2 ARQUIVO DATA ARCHITECT

O arquivo gerado pela ferramenta CASE *Power Designer – Data Architect* é um arquivo texto com extensão .CDM (*Conceptual Data Model).* Este arquivo armazena informações relacionadas ao diagrama de entidade relacionamento. Não foram encontradas referências sobre a especificação do arquivo. Sua estrutura foi identificada através da comparação de diversos diagramas editados através da ferramenta e as informações contidas

nos respectivos arquivos após a editoração. Desta forma foi identificado que o arquivo é dividido em seções específicas para cada objeto do projeto. O início de cada seção se dá através do identificador "/begin_table nome_seção", seguido da descrição dos campos e a identificação do tipo do campo. Para delimitar o fim de descrição dos campos aparece o identificador "/begin_data nome_seção". A seguir estão os valores de cada objeto pertencente a seção na mesma seqüência em que foram apresentadas as descrições. Para encerrar a seção aparece o identificador "/end_table nome_seção", conforme Quadro 4.

QUADRO 4 - EXEMPLO DE SEÇÃO DO *DATA ARCHITECT*

/BEGIN_TABLE AMCHEAD	Inicio de seção AMCHEAD
NAME A30	Nome do 1. campo + tipo do dado
VERS A8	Nome do 2. campo + tipo do dado
USER A30	Nome do 2. campo + tipo do dado
/BEGIN_DATA AMCHEAD	Inicio de dados da seção
PowerDesigner/CDM	Dados do 1. campo
6.1.0	Dados do 2. campo
	Dados do 2. campo
/END_TABLE AMCHEAD	Fim de dados da seção AMCHEAD

A descrição das seções do arquivo do *Data Architect* encontram-se no anexo 1.

4.3 ARQUIVO PROCESS ANALYST

O arquivo utilizado pela ferramenta CASE *Power Designer – Process Analyst* é um arquivo texto com extensão .PAM (*Process Analyst Model).* Este arquivo armazena informações sobre o diagrama de fluxo de dados. Sobre as especificações deste arquivo também não foram encontradas referências e o processo de identificação da estrutura foi igual ao processo realizado na identificação do arquivo do *Data Architect*. Este arquivo é divido em seções específicas que são apresentadas no anexo 2.

4.4 ESPECIFICAÇÃO DA FERRAMENTA

A especificação da ferramenta é apresentada na forma de alguns diagramas da UML. A ferramenta CASE utilizada para esta especificação foi o *Rational Rose* 2000 da *Rational* (Rational, 2002). A seguir são descritos alguns diagramas.

4.4.1 DIAGRAMA DE CASOS DE USO

O principal ator do sistema é o engenheiro de software interessado no processo de migração de especificações estruturadas para especificações da UML. Os seguintes casos de uso primários foram identificados (Figura 21):

a) importar DER: o engenheiro seleciona um arquivo .CDM gerado pelo *Power Designer*. A ferramenta lê o arquivo e gera o diagrama de entidade relacionamento referente às informações do arquivo;

b) importar DFD: o engenheiro seleciona uma arquivo .PAM gerado pelo *Power Designer*. A ferramenta lê o arquivo e gera o diagrama de contexto e o diagrama de fluxo de dados;

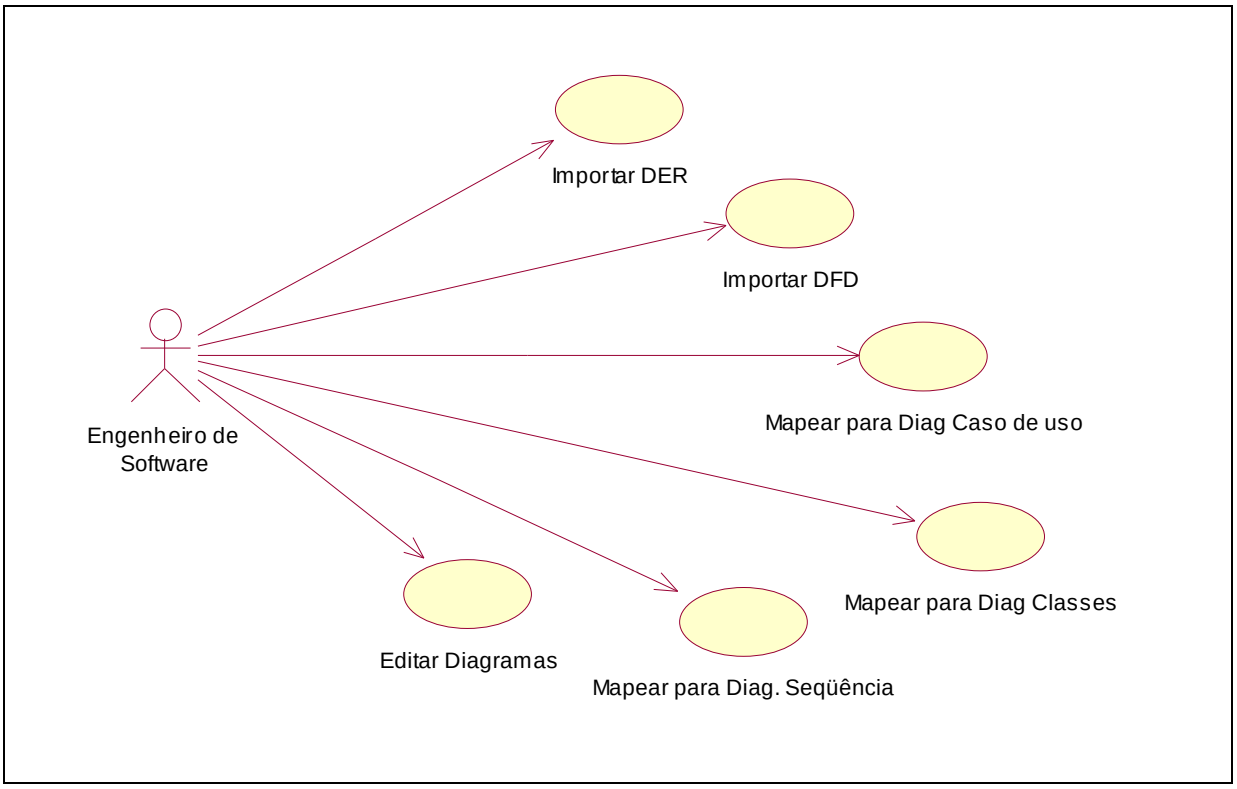
c) mapear para diagrama de caso de uso: a ferramenta gera o diagrama de caso de uso seguindo a estratégia de mapeamento;

d) mapear para diagrama de classes: a ferramenta gera o diagrama de classes seguindo a estratégia de mapeamento;

e) mapear para diagrama de seqüência: a ferramenta gera o diagrama de seqüência seguindo a estratégia de mapeamento;

f) editar diagramas: o engenheiro edita os diagramas estruturados e OO.

FIGURA 21 - DIAGRAMA DE CASOS DE USO



4.4.2 DIAGRAMA DE CLASSES

As classes identificadas no sistema estão separadas nos seguintes pacotes (Figura 22):

a) projeto: contém classes que agregam dicionários e é responsável pela operabilidade do sistema;

b) dicionário: contém classes de dicionário que agrega todos os seus componentes. Estas classes são responsáveis por toda manutenção de dicionário de dados;

c) objetos de modelo: contém as classes de componentes do dicionário;

d) objetos gráficos: contém as classes responsável pela representação gráfica dos componentes do modelo;

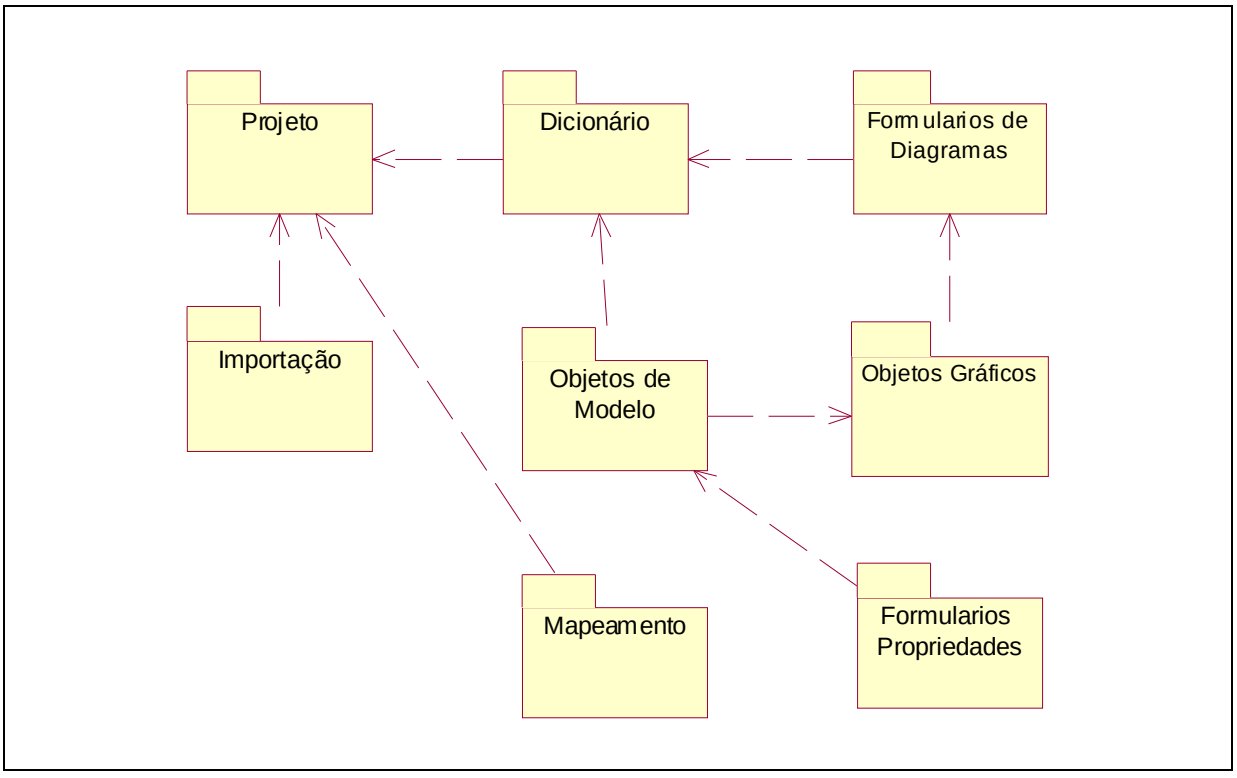
e) formulário de propriedades: contém as classes responsáveis por editar as propriedades dos objetos do modelo;

f) formulários de diagramas: contém as classes de formulários utilizados para apresentar os diagramas em forma gráfica;

g) importação: contém classes responsáveis pela leitura e importação de arquivo do *Power Designer*;

h) mapeamento: contém classes responsáveis pelo mapeamento.

FIGURA 22 - DIAGRAMA DE CLASSES - PACOTES



4.4.2.1 PACOTE DICIONÁRIO

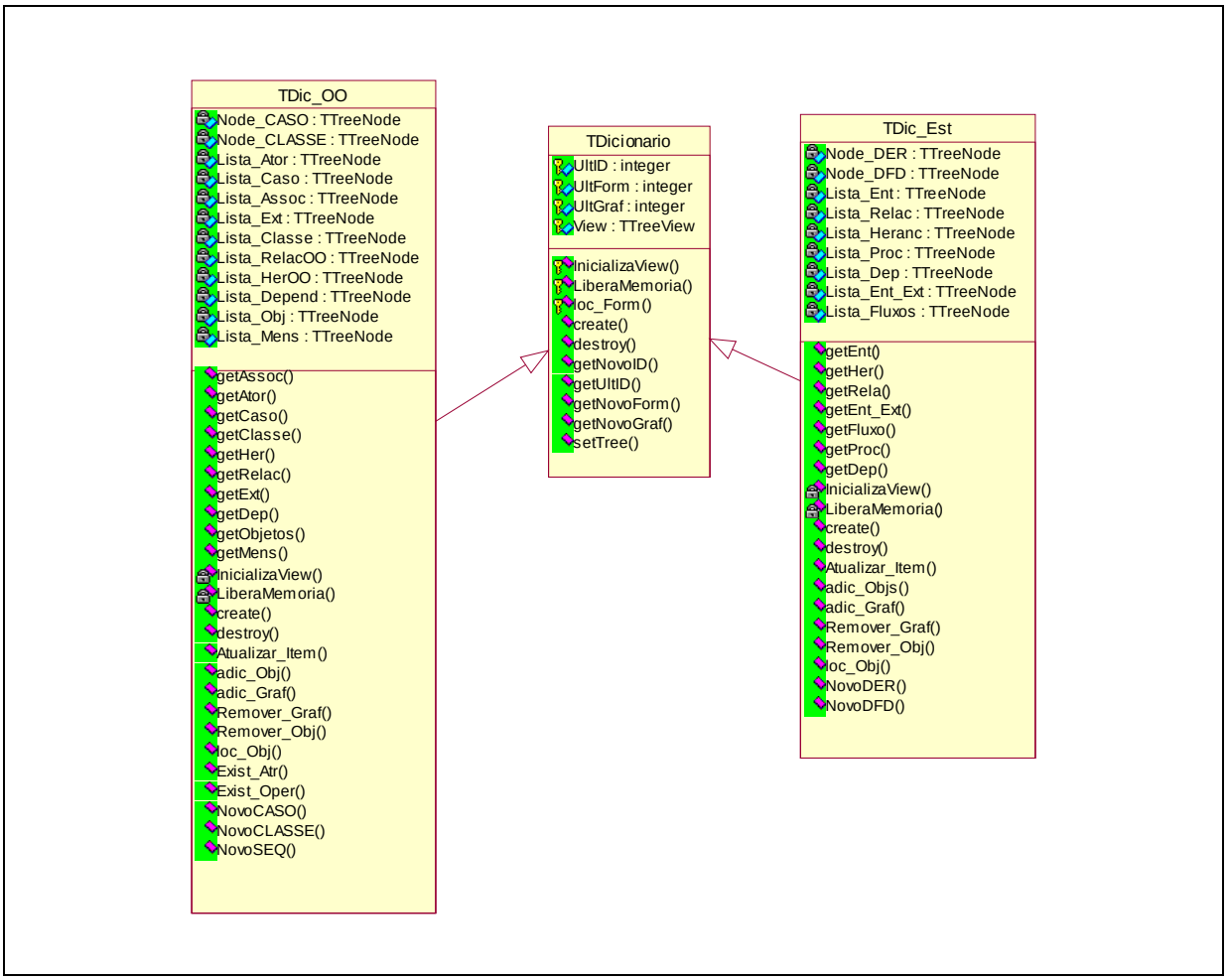
O pacote dicionário apresenta as classes de dicionário de dados descritas a seguir (Figura 23):

a) TDicionario: esta é a classe base para as classes de dicionário do modelo estruturado e dicionário do modelo orientado a objetos. Ela possui métodos para inicialização dos dicionários, gerenciamento de identificadores de objetos e formulários;

b) TDic_Est: esta classe é responsável pelo gerenciamento de todas as informações sobre os diagramas estruturados e possui funções de adicionar, remover objetos e localizar objetos;

c) TDic_OO: esta classe é responsável pelo gerenciamento de todas as informações sobre os diagramas orientados a objetos e possui funções de adicionar, remover e localizar objetos.

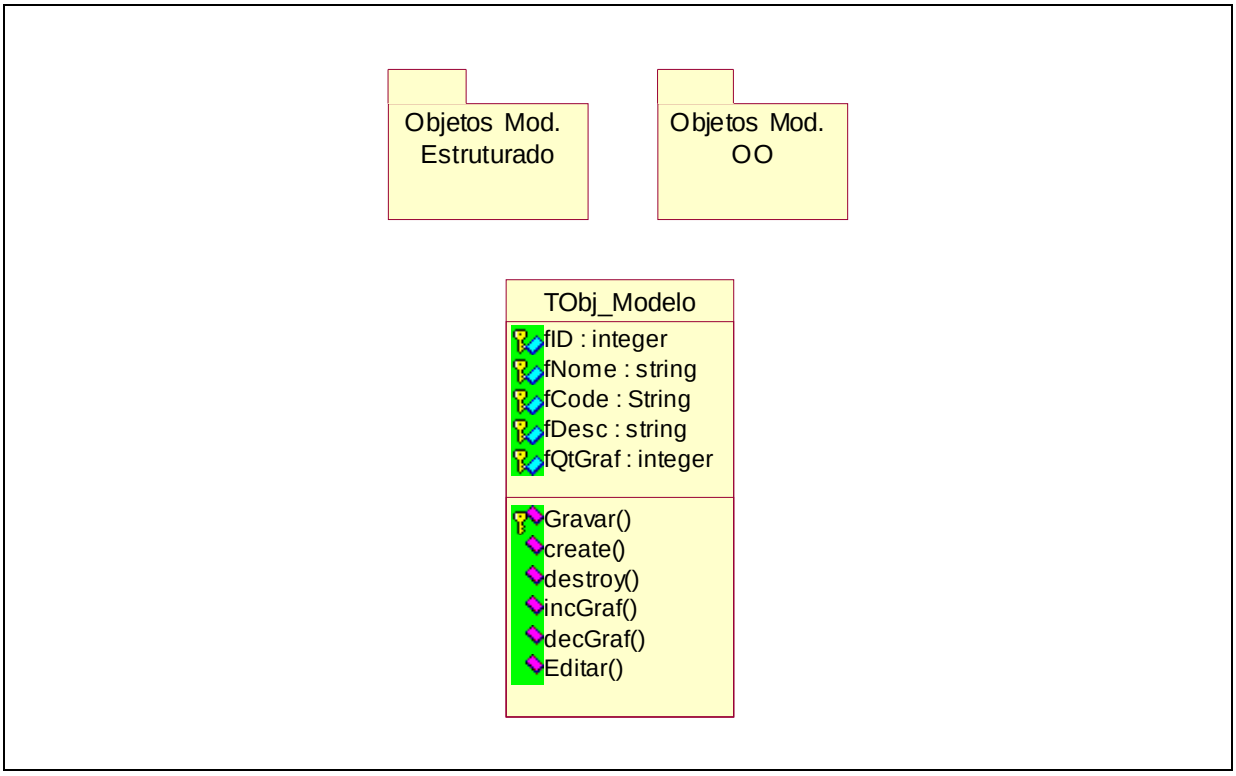
FIGURA 23 - DIAGRAMA DE CLASSES – PACOTE DICIONÁRIO



4.4.2.2 PACOTE OBJETOS DE MODELO

Este pacote se divide em outros dois pacotes: os objetos de modelo estruturado e modelo orientado a objetos. O pacote de objetos de modelo possui uma classe que é base para os demais objetos de modelo. Esta classe implementa atributos e métodos comuns ao demais objetos, como identificador, nome, descrição e edição de propriedades (Figura 24).

FIGURA 24 - DIAGRAMA DE CLASSES – PACOTE OBJETOS DE MODELO



As classes descritas nos pacotes objetos do modelo estruturado e OO são responsáveis pela manipulação das informações de cada componente do modelo. Essas classes são subclasses de TObj_Modelo. As classes do pacote de objeto de modelo estruturado serão descritas a seguir e podem ser visualizadas na Figura 25:

a) TObj_Entidade: é responsável pelas informações da entidade do DER e possui uma lista de atributos;
b) TObj_Atributo: esta classe possui as informações sobre o atributo que esta associado as entidades;
c) TObj_Relac: é responsável pelo relacionamento entre as entidades do DER;
d) TObj_Herança: é responsável pelas heranças entre entidades do DER e possui um relacionamento com a entidade “pai” e uma lista de entidades “filhas”;

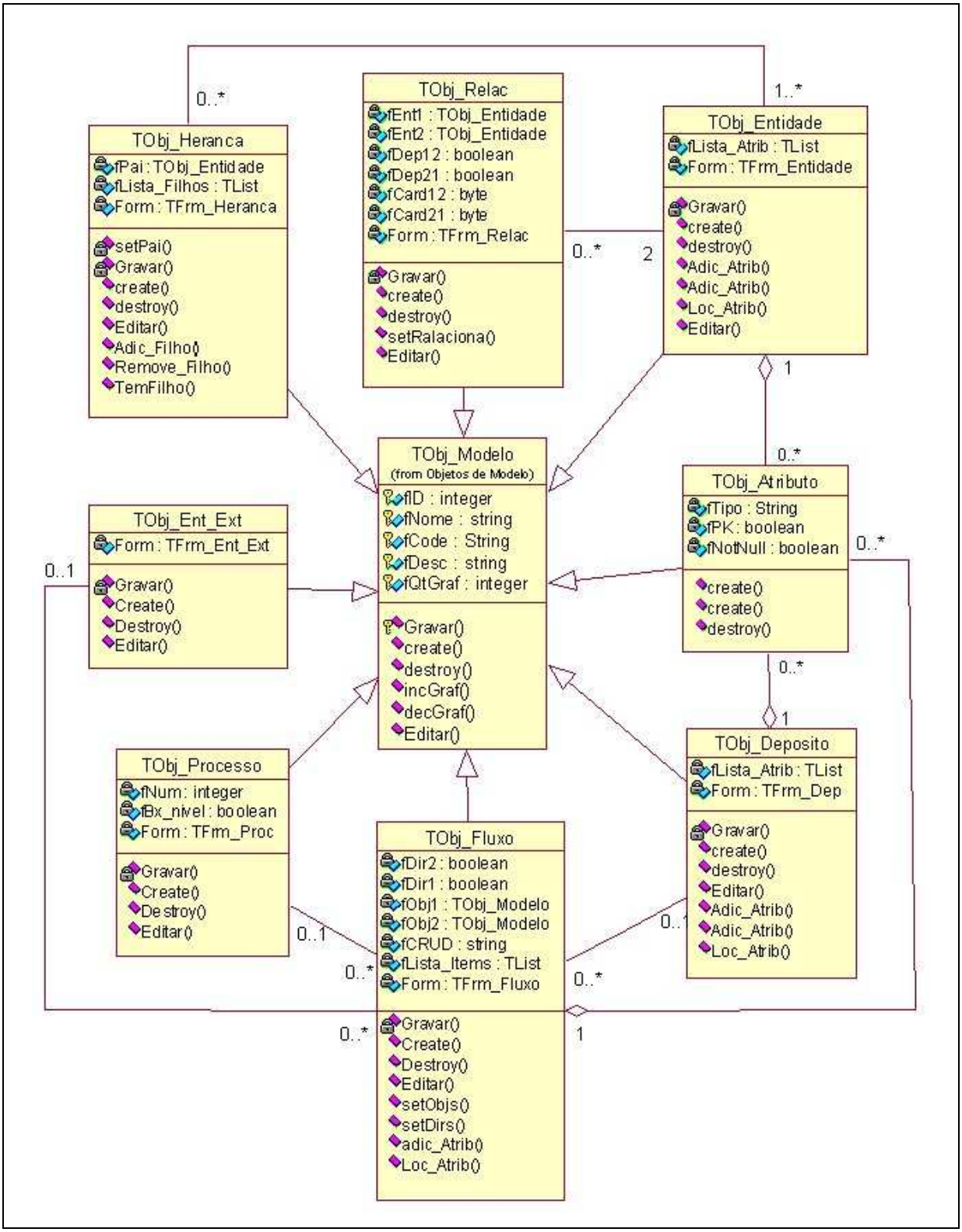
e) TObj_Ent_Ext: esta classe possui as informações sobre uma entidade externa do DFD;
f) TObj_Processo: é responsável pelas informações do processo do DFD;
g) TObj_Deposito: esta classe contém informações referentes ao depósito do DFD e possui uma lista de atributos;
h) TObj_Fluxo: esta classe é responsável pela associação entre um processo, depósito ou entidade externa e permite apenas associações entre dois objetos de tipo diferentes;

As classes do pacote de objeto de modelo orientado a objetos serão descritas a seguir e podem ser visualizadas na Figura 26:

a) TObj_Ator: esta classe é responsável pelas informações do ator utilizado no diagrama de caso de uso;
b) TObj_Caso: possui as informações sobre o caso e sua descrição detalhada;
c) TObj_Assoc: esta classe é responsável pela associação entre atores e casos de uso;
d) TObj_Extend: possui informações sobre as extensões/inclusões de casos de uso;
e) TObj_Classe: é responsável pelas informações da classe e possui uma lista de atributos e operações;
f) TObj_AtribOO: possui informações sobre o atributo que está associado a classe;
g) TObj_OperOO: possui informações sobre a operação que está associada a classe;
h) TObj_RelacOO: esta classe é responsável pela associação entre as classes e possui informações sobre o relacionamento, como cardinalidade e agregação;
i) TObj_HerOO: esta classe é responsável pela representação de herança entre classes;
j) TObj_Depend: possui informações sobre o relacionamento de classes dependentes;
k) TObj_Objeto: é responsável pelas informações de objeto no diagrama de seqüência;
l) TObj_Mens: é responsável pelas informações sobre mensagens entre objetos no diagrama de seqüência.

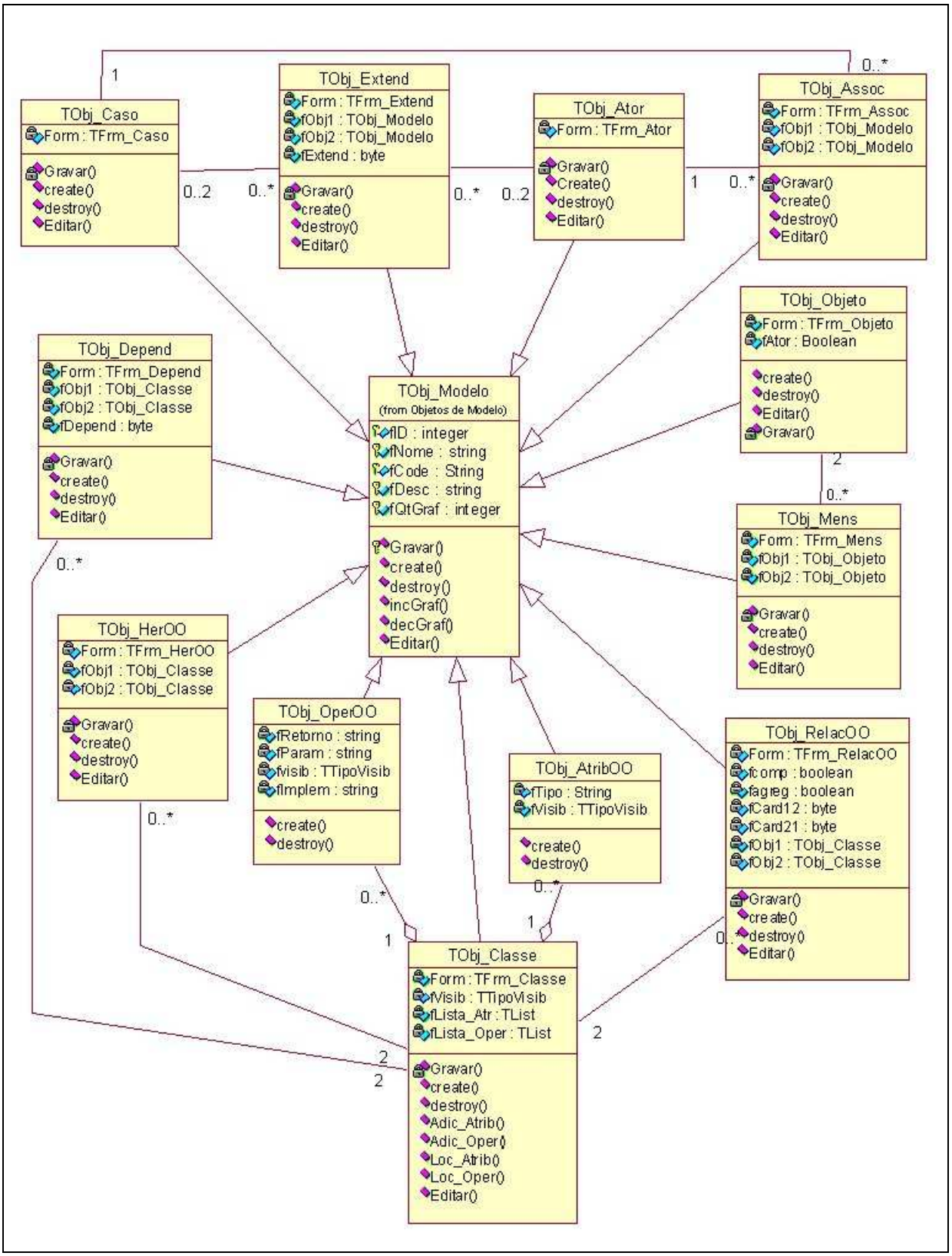
O pacote de objetos de modelo estruturado apresenta as classes utilizadas nos diagramas de entidade relacionamento e diagrama de fluxo de dados que são apresentadas na Figura 25.

FIGURA 25 - DIAGRAMA DE CLASSES – OBJ. DE MODELO ESTRUTURADO



O pacote de objetos de modelo orientado a objetos apresenta as classes utilizadas nos diagramas de caso de uso, classes e seqüência que são apresentadas na Figura 26.

FIGURA 26 - DIAGRAMA DE CLASSES – OBJ. DE MODELO OO



4.4.2.3 PACOTE IMPORTAÇÃO

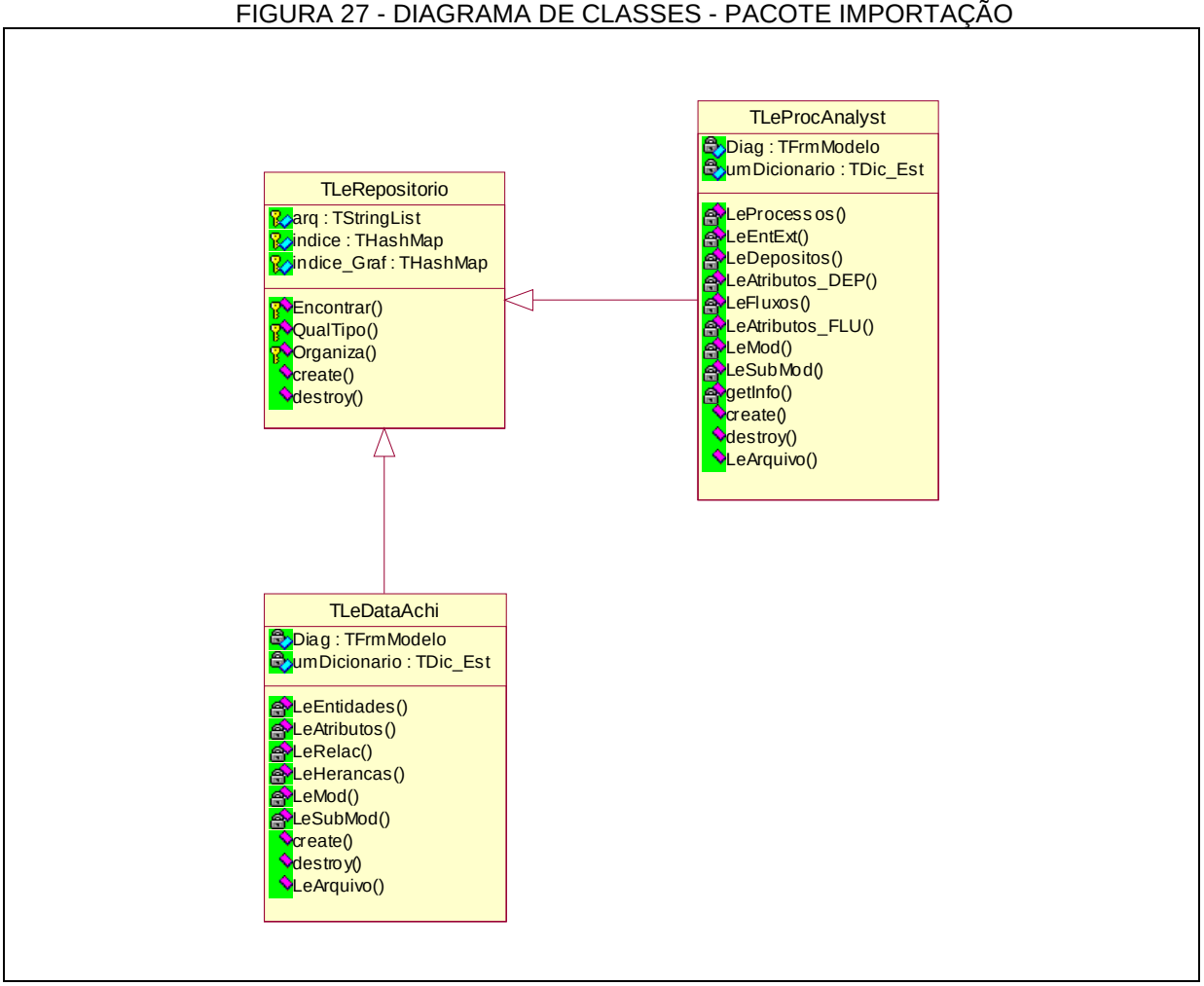
As classes que compõem o pacote importação são descritas a seguir (Figura 27):

a) TLeRepositorio: esta é a classe base para as demais classes de importação. Ela possui métodos para encontrar seção, organizar arquivo de leitura e detectar qual o tipo do atributo;

b) TLeDataArchi: esta classe é responsável pela leitura do arquivo do *Power Designer Data Architect* e possui métodos para ler entidades, atributos, relacionamentos e demais componentes dos diagramas;

c) TLeProcAnalyst: esta classe é responsável pela leitura do arquivo do *Power Designer Process Analyst* e possui métodos para ler processos, depósitos, atributos, fluxos e demais componentes dos diagramas.

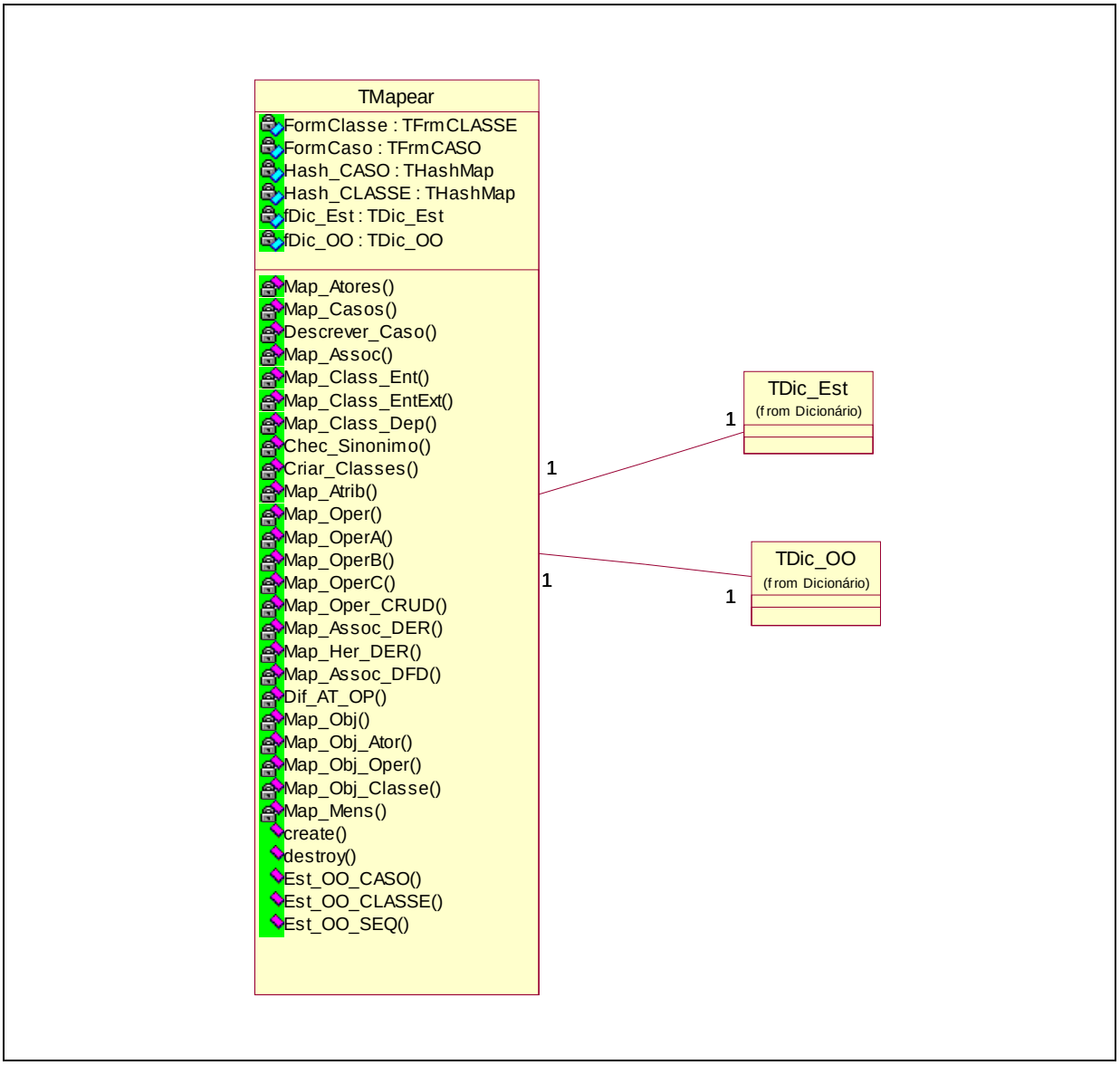
FIGURA 27 - DIAGRAMA DE CLASSES - PACOTE IMPORTAÇÃO



4.4.2.4 PACOTE MAPEAMENTO

O pacote mapeamento apresenta apenas uma classe chamada TMapear. Esta classe é responsável pelo mapeamento da especificação estruturada para especificação orientada a objetos (Figura 28). Através do método EST_OO_CASO é iniciada a leitura do dicionário de dados para a obtenção das informações do modelo estruturado e posteriormente gerar o diagrama de caso de uso. Outro método é o EST_OO_CLASSE que inicia a leitura do dicionário de dados para a obtenção das informações do modelo estruturado e posteriormente gerar o diagrama de classes. O método EST_OO_SEQ inicia a leitura do dicionário para a geração do diagrama de seqüência. As classes TDic_EST e TDic_OO foram descritas no pacote dicionário (Figura 23).

FIGURA 28 - DIAGRAMA DE CLASSES – PACOTE MAPEAMENTO



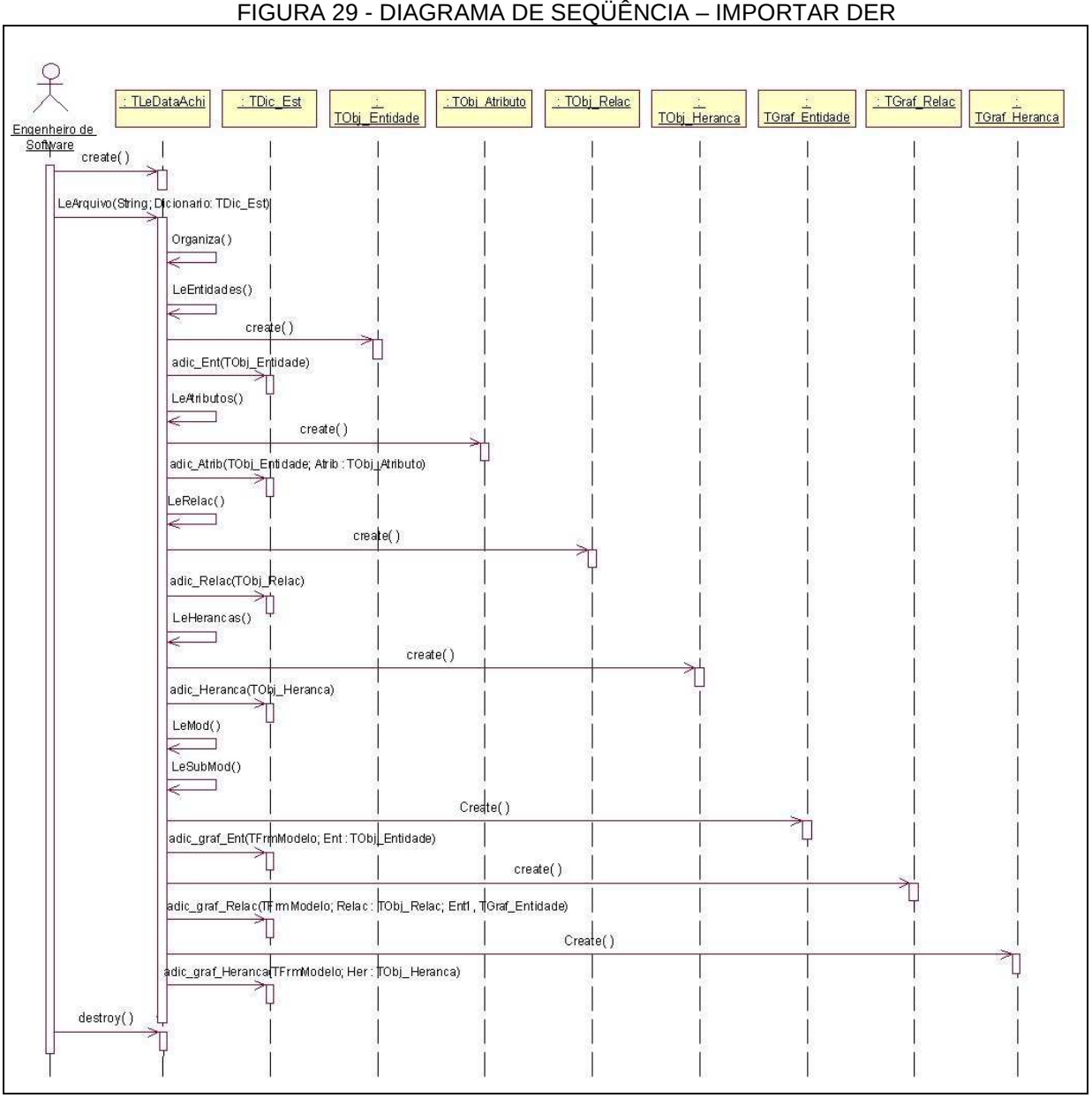
4.4.3 DIAGRAMA DE SEQÜÊNCIA

O diagrama de seqüência representa a seqüência em que as ações ocorrem dentro do sistema. Eles demonstram como é feita a troca de mensagens entre as classes (objetos). A seguir são apresentados os diagramas de seqüência dos casos de uso primários.

4.4.3.1 IMPORTAR DER

O diagrama apresentado na Figura 29 representa os passos executados para importar as informações do arquivo .CDM do *Power Designer*. Esse diagrama de seqüência é referente ao caso de uso "Importar DER" descrito na Figura 21.

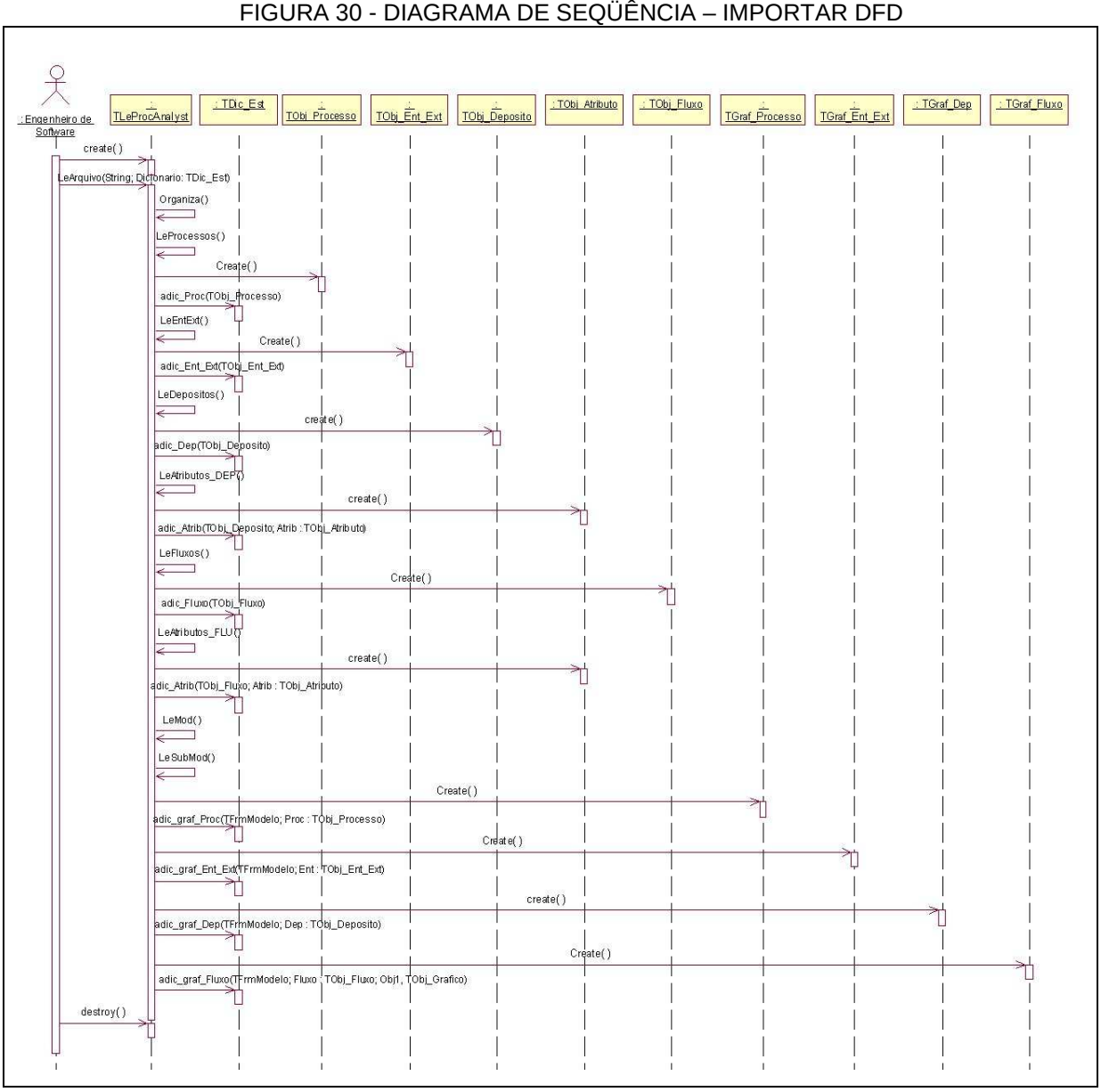
FIGURA 29 - DIAGRAMA DE SEQÜÊNCIA – IMPORTAR DER



4.4.3.2 IMPORTAR DFD

O diagrama apresentado na Figura 30 representa os passos executados para importar as informações do arquivo .PAM do *Power Designer*. Esse diagrama de seqüência é referente ao caso de uso “Importar Diagramas Estruturados” descrito na Figura 21.

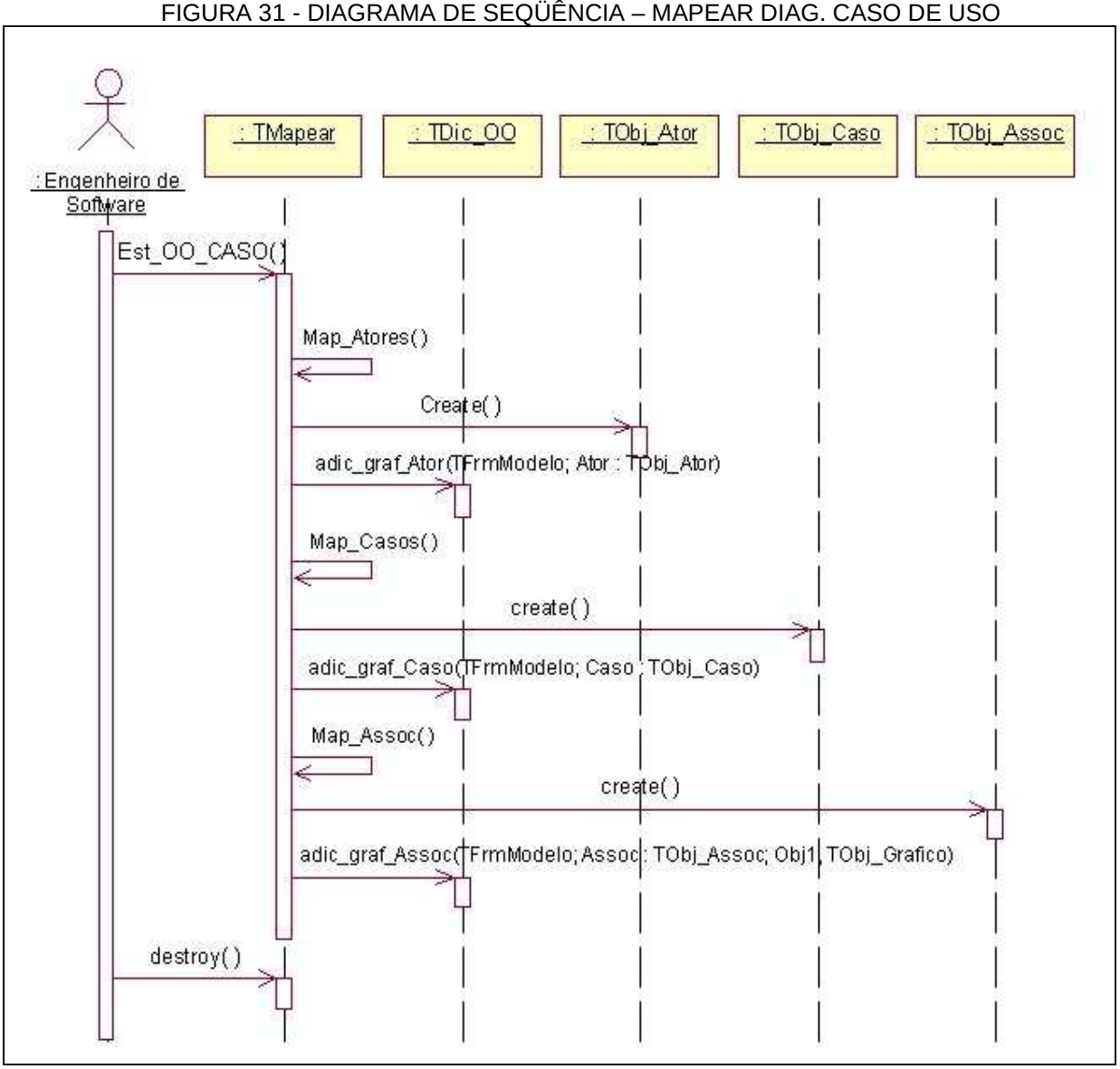
FIGURA 30 - DIAGRAMA DE SEQÜÊNCIA – IMPORTAR DFD



4.4.3.3 MAPEAR PARA DIAGRAMA DE CASO DE USO

O diagrama apresentado na Figura 31 representa os passos executados para gerar o diagrama de caso de uso baseado na proposta de mapeamento. Esse diagrama de seqüência é referente ao caso de uso “Mapear para UML” descrito na Figura 21.

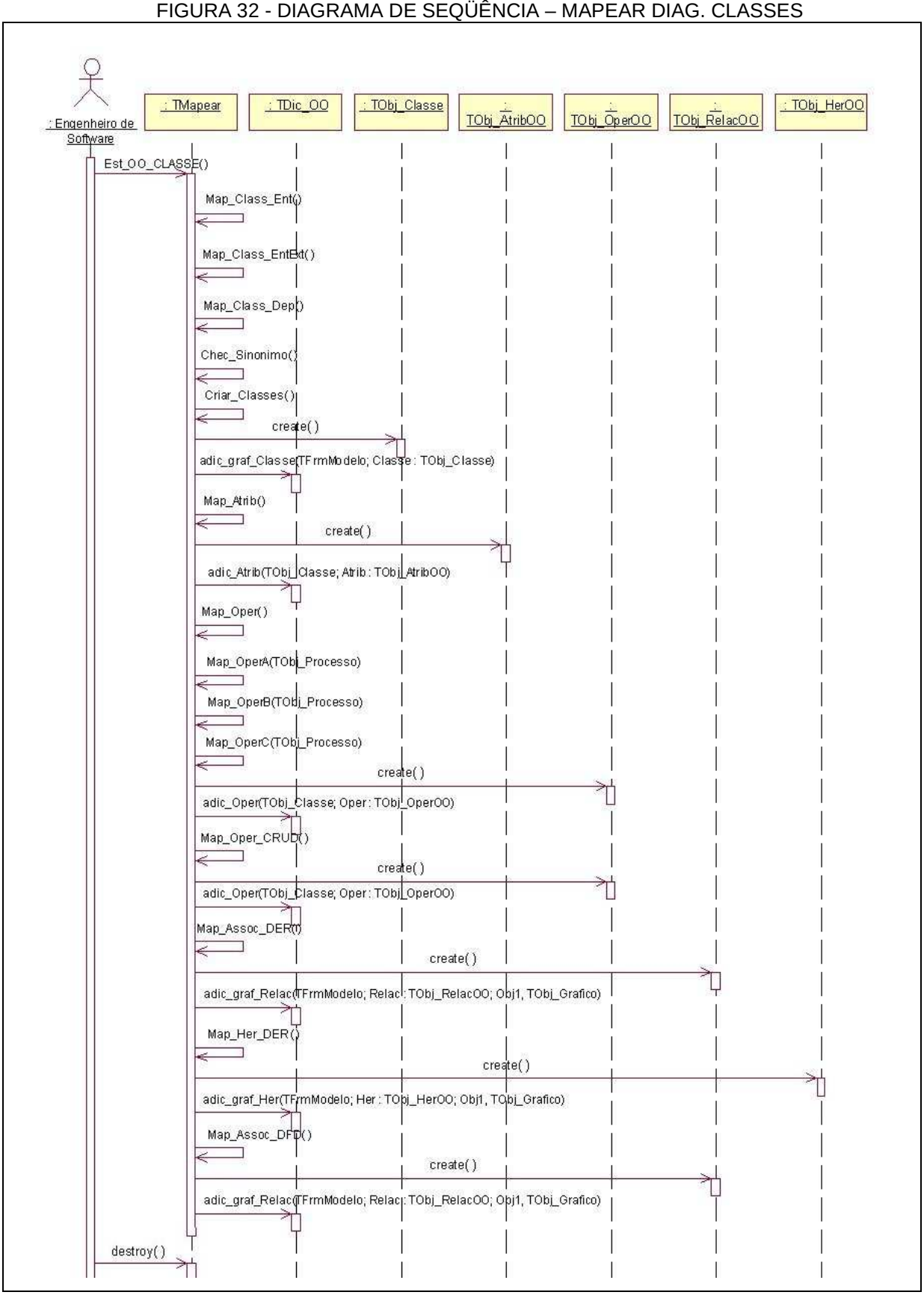
FIGURA 31 - DIAGRAMA DE SEQÜÊNCIA – MAPEAR DIAG. CASO DE USO



4.4.3.4 MAPEAR PARA DIAGRAMA DE CLASSES

O diagrama apresentado na Figura 32 representa os passos executados para gerar o diagrama de Classes baseado na proposta de mapeamento. Esse diagrama de seqüência é referente ao caso de uso “Mapear para UML” descrito na Figura 21.

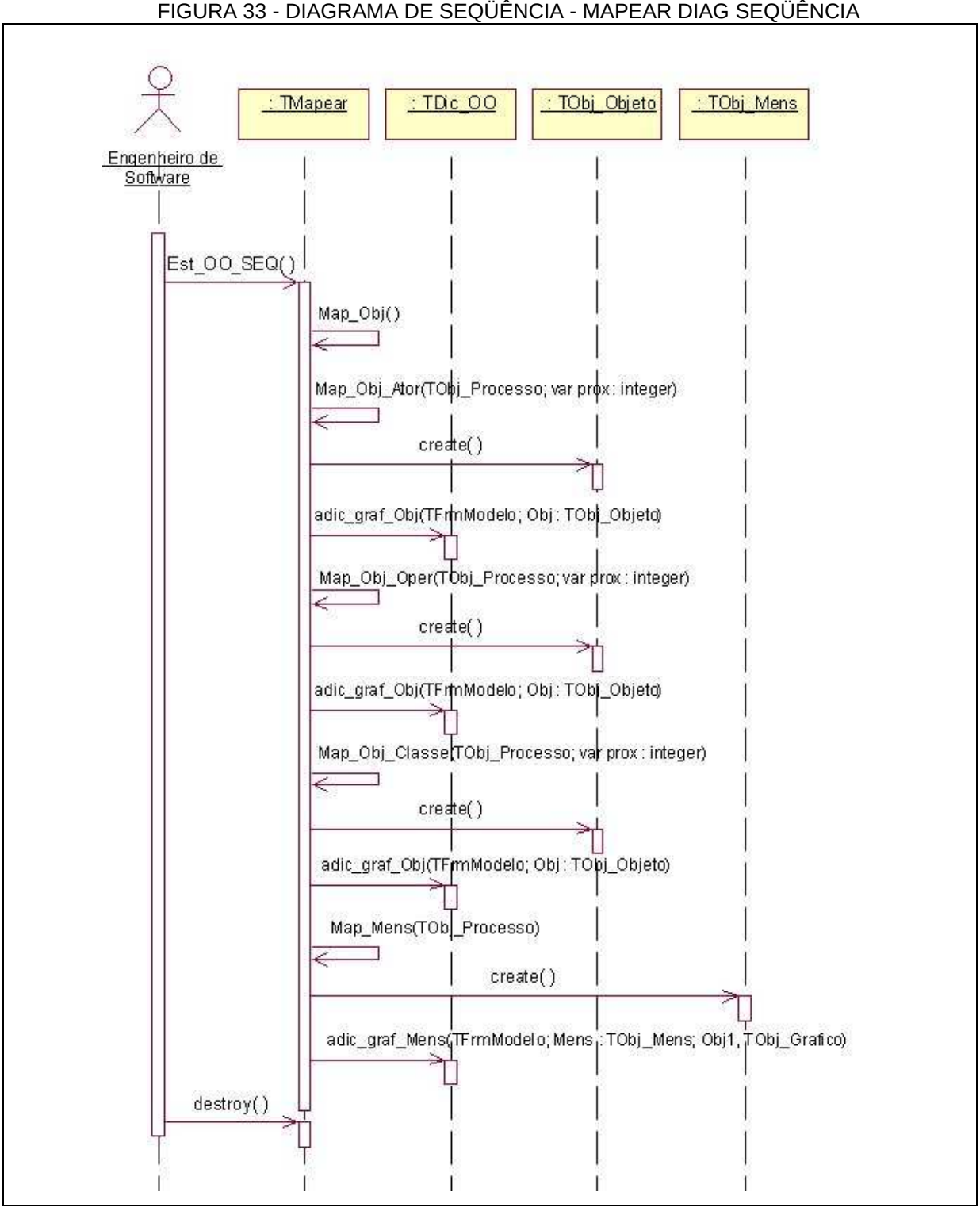
FIGURA 32 - DIAGRAMA DE SEQÜÊNCIA – MAPEAR DIAG. CLASSES



4.4.3.5 MAPEAR PARA DIAGRAMA DE SEQÜÊNCIA

O diagrama apresentado na Figura 33 representa os passos executados para gerar o diagrama de seqüência baseado na proposta de mapeamento. Esse diagrama de seqüência é referente ao caso de uso “Mapear para UML” descrito na Figura 21.

FIGURA 33 - DIAGRAMA DE SEQÜÊNCIA - MAPEAR DIAG SEQÜÊNCIA



4.5 IMPLEMENTAÇÃO

Considerações sobre as técnicas utilizadas para implementação do protótipo, bem como a forma de operação do mesmo, são apresentadas nesta seção.

4.5.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

A ferramenta foi implementada no ambiente de programação *Borland Delphi 6* (Cantu, 2000), para o sistema operacional *Windows 32 bits*. No desenvolvimento da ferramenta foram utilizados os conceitos de orientação a objetos.

Na interface gráfica para editoração dos diagramas foram utilizadas classes descendentes de *TShape* que é um componente gráfico do *Delphi* que permite o desenho de objetos padrão, como retângulo e círculos utilizados no desenho de entidades, processos e classes. Para a realização do desenho nos objetos se fez necessário a reescrita do método *Paint*. A interface gráfica teve como grande complexidade o desenho das retas que compõem as ligações entre objetos, como exemplo um relacionamento entre entidades do DER. Todos os objetos gráficos são desenhados sobre formulários que identificam cada diagrama.

O dicionário de dados foi implementado utilizando o componente visual do *Delphi* chamado *TTreeView* que permite a implementação de uma árvore de objetos. A maior dificuldade no dicionário foi manter o sincronismo entre a árvore de objetos e os símbolos gráficos inseridos nos diagramas.

A leitura do arquivo do *Power Designer* exigiu muito esforço, pois não se encontrou documentação sobre a especificação dos arquivos. Para a realização da leitura foram implementadas classes especializadas na leitura dos arquivos gerados pelo *Power Designer*. Essas classes fazem a leitura e interpretação do arquivo e adicionam os objetos no dicionário de dados.

A seguir são apresentadas algumas rotinas para demonstrar as implementações conforme as especificações apresentadas anteriormente.

4.5.1.1 LEITURA DO ARQUIVO DO *DATA ARCHITECT*

A leitura do arquivo do *Data Architect* está baseado no diagrama de seqüência apresentado na Figura 29. O Quadro 5 apresenta trecho do código fonte em *Delphi* do método LeArquivo da classe TLeDataArchitect.

QUADRO 5 - MÉTODO LEARQUIVO DA CLASSE TLEDATAARCHITECT

```
procedure TLeDataArchi.LeArquivo(Arquivo: String; Dicionario: TDic_Est);
begin
  // cria lista de strings para fazer leitura do arquivo
  arq := TStringList.Create;
  arq.LoadFromFile(arquivo);
  // recebe dicionario
  umDicionario := Dicionario;
  // chama método para organizar arquivo do Data Architect
  Organiza;
  // chama método para ler entidades
  LeEntidades;
  // chama metodo para ler atributos
  LeAtributos;
  // chama metodo para ler relacionamentos
  LeRelac;
  // chama metodo para ler herancas
  LeHerancas;
  // chama metodo para ler objetos graficos do modelo principal
  LeMod;
  // chama metodo para ler objetos graficos dos submodelos
  LeSubMod;
  umDicionario := nil;
  // destroi lista de strings
  arq.Free;
end;
```

O Quadro 6 apresenta trecho de código do método LeEntidades da classe TLeDataArchitect responsável pela leitura das entidades.

QUADRO 6 - MÉTODO LEENTIDADES DA CLASSE TLEDATAARCHITECT

```
procedure TLeDataArchi.LeEntidades;
var
  LIniTab,
  LIniDat,
  LFimTab,
  Cont : integer;
  Entidade : TObj_Entidade;
  PowerID : string; // Id do power designer
begin
  // encontrar linha de inicio da tabela de definicao
  // inicio de dados e fim da tabela/dados
  LIniTab := Encontrar(DEF_Ent_ini_Tab);
  LIniDat := Encontrar(DEF_Ent_ini_Dat);
  LFimTab := Encontrar(DEF_Ent_fim_Tab);
  // se nao for encontrado alguma das linha entao encerra rotina
  if (LIniTab < 0) or (LIniDat < 0) or (LFimTab < 0) then exit;
  Cont := LIniDat+1;
  // percorrer todos os dados das entidades
  while Cont < LFimTab do begin
    Entidade := TObj_Entidade.create;
    // recupera ID do power designer para uso futuro
    PowerID := arq.Strings[Cont+2];
    // mantem lista de ID / Objeto do power designer
    indice.put(PowerID,Entidade);
    // atualiza dados da entidade
    Entidade.Nome := arq.Strings[Cont+4];
    Entidade.Code := arq.Strings[Cont+5];
    Entidade.Desc := arq.strings[Cont+7];
    // Colocar Entidade no Dicionario
```

```
    umDicionario.Adic_Ent(Entidade);
    // passar para proxima entidade
    Cont := Cont + (LIniDat-LIniTab-1);
  end;
end;
```

4.5.1.2 MAPEAR PARA CASO DE USO

O processo de mapeamento para diagrama de caso de uso está baseado no diagrama de seqüência apresentado na Figura 31. O Quadro 7 apresenta trecho de código do método EST_OO_CASO da classe Tmapear.

QUADRO 7 - MÉTODO EST_OO_CASO DA CLASSE TMAPEAR

```
procedure TMapear.Est_OO_CASO;
begin
  // cria formulario de caso de uso
  FormCASO := TFrmCASO(Dic_OO.NovoCASO);
  // chama metodo para mapear atores
  Map_Atores;
  // chama metodo para mapear casos
  Map_Casos;
  // chama metodo para mapear associacoes
  Map_Assoc;
end;
```

O Quadro 8 apresenta trecho de código do método Map_Atores da classe Tmapear. Este método é responsável por mapear os atores do diagrama de caso de uso.

QUADRO 8 - MÉTODO MAP_ATORES DA CLASSE TMAPEAR

```
procedure TMapear.Map_Atores;
var
  x : integer;
  Ent : TObj_Ent_Ext;
  Ator : TObj_Ator;
  Graf : TObj_Grafico;
  prox : integer;
begin
  prox := 10;
  // percorrer entidades externas
  for x := 0 to Dic_Est.Ent_Ext.Count-1 do begin
    Ent := TObj_Ent_Ext(Dic_Est.Ent_Ext.item[x].Data);
    // criar ator
    Ator := TObj_Ator.Create;
    Ator.Nome := Ent.Nome;
    Ator.Code := Ent.Code;
    // adicionar no dicionario
    Dic_OO.adic_Ator(Ator);
    //  adicionar objeto grafico
    graf := Dic_OO.adic_graf_Ator(FormCASO,Ator);
    // posicionar objeto grafico
    graf.top := prox;
    prox := prox + 100;
    graf.left := 10;
    // adicionar na lista de controle
    Hash_CASO.put(inttostr(Ent.ID),Graf);
  end;
end;
```

A seguir será descrita a operacionalidade da ferramenta desenvolvida neste trabalho.

4.5.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

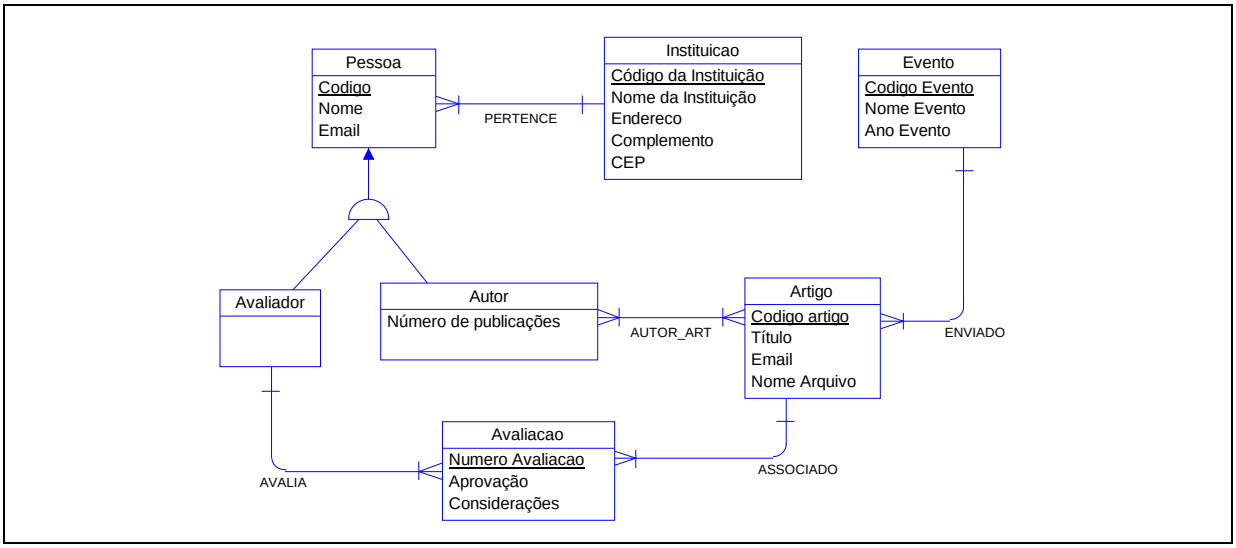
Para exemplificar a utilização da ferramenta foi elaborado um estudo de caso de um sistema para avaliação de artigos submetidos a um evento. O enunciado do estudo de caso mostrado no Quadro 9 é proveniente de trabalho acadêmico realizado na FURB.

QUADRO 9- ESTUDO DE CASO

A comissão organizadora do SEMINCO (Seminário de Computação), resolveu informatizar o controle das avaliações dos artigos que são submetidos para aceitação nas diversas edições do evento. A comissão é responsável pelas informações das instituições (código, nome, endereço, etc.) e eventos realizados. Quando algum autor submete um artigo, são guardados alguns dados para posterior avaliação (titulo do artigo, nome, e-mail e instituições dos autores). Após à chegada dos artigos, os mesmos são distribuídos para avaliação. Neste momento entram os avaliadores (nome, e-mail e instituição) do evento que preenchem uma ficha (nome do avaliador, titulo do artigo e parecer do avaliador) contendo o seu parecer sobre os artigos recebidos. Cada artigo pode ser avaliado por mais de um avaliador. Um avaliador sempre está ligado a uma instituição e pode estar apto a avaliar artigos em diversas áreas. Dias antes da edição do SEMINCO, a comissão receberá relatório da classificação dos artigos.

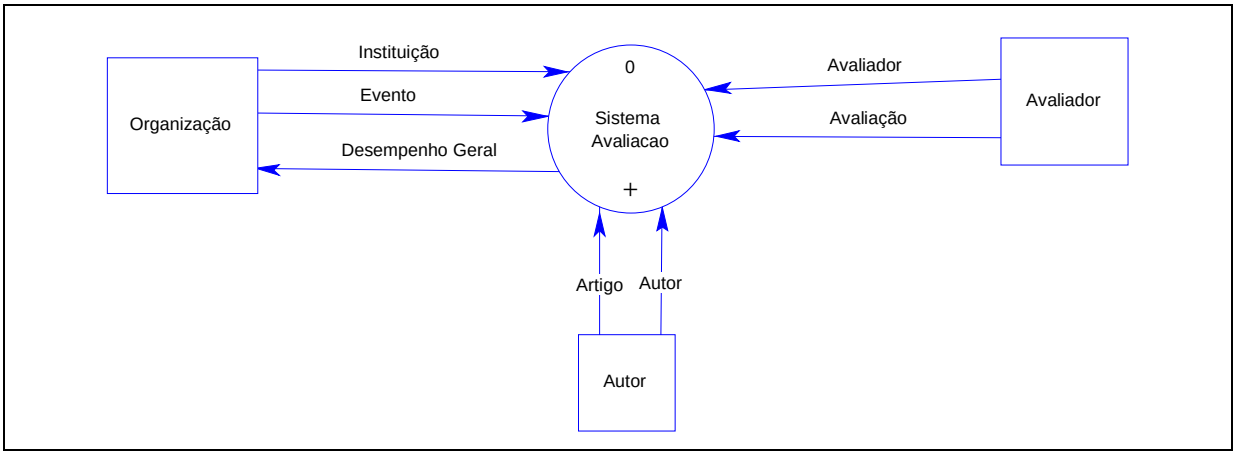
O problema foi especificado de forma estruturada utilizando a ferramenta *Power Designer 6*. O diagrama de entidade relacionamento pode ser visto na Figura 34 especificado na ferramenta *Data Architect*.

FIGURA 34 - ESTUDO DE CASO - DER



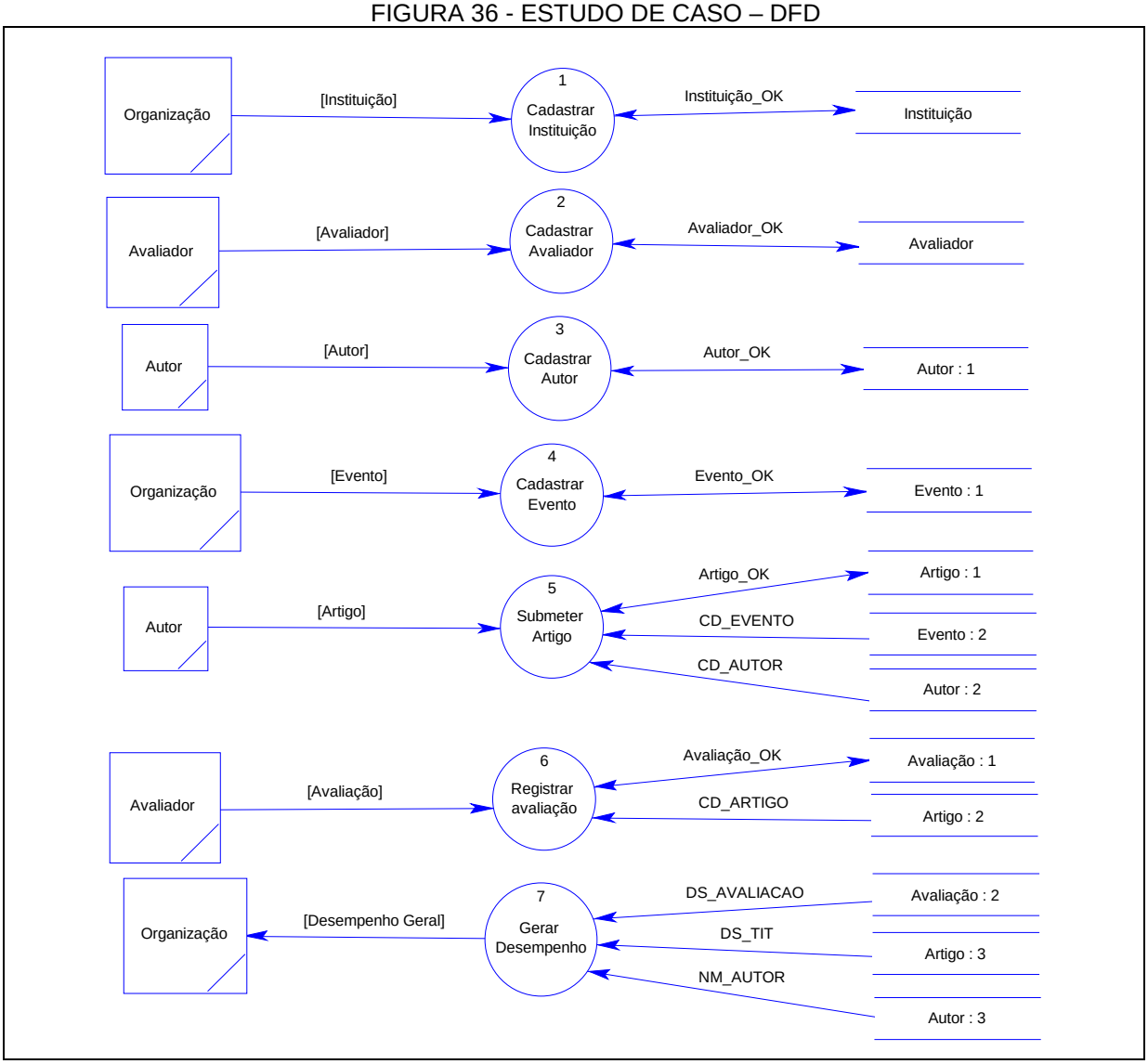
A Figura 35 apresenta o diagrama de contexto especificado na ferramenta *Process Analyst*.

FIGURA 35 - ESTUDO DE CASO - CONTEXTO



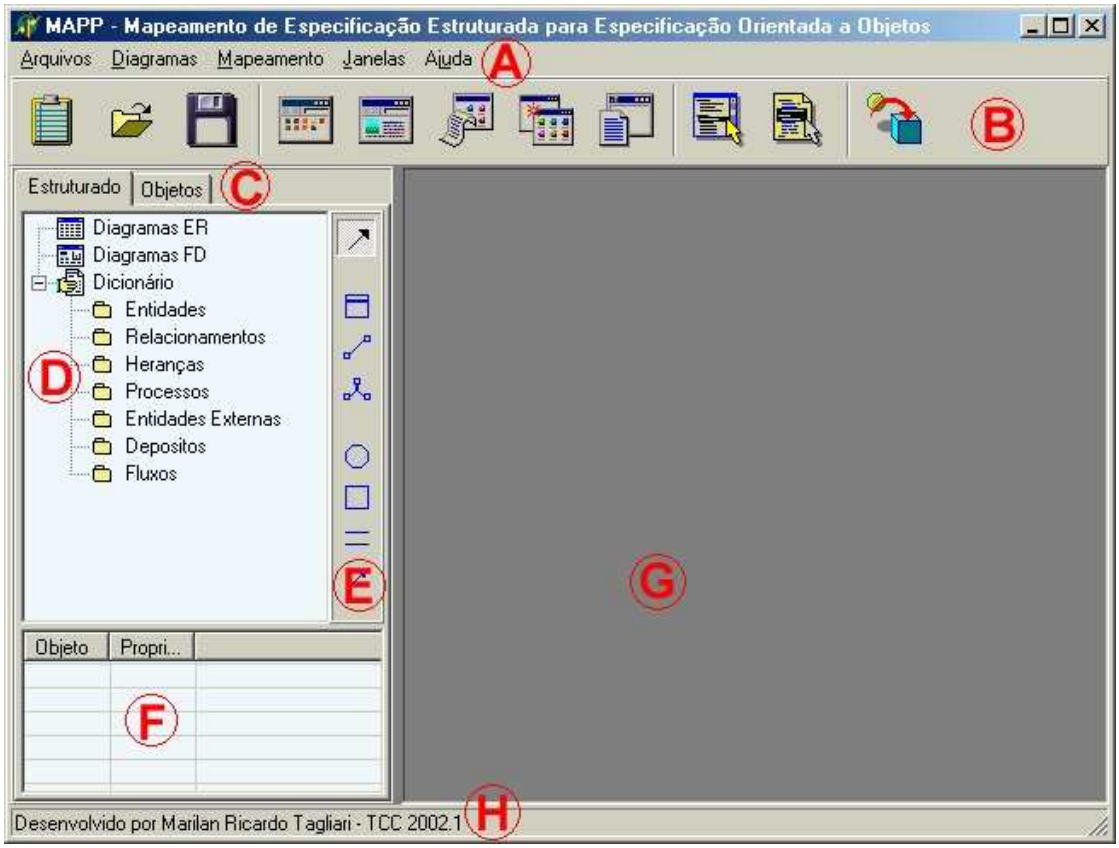
A Figura 36 apresenta o diagrama de fluxo de dados especificado na ferramenta *Process Analyst*.

FIGURA 36 - ESTUDO DE CASO – DFD



Para iniciar o mapeamento deve-se ter os diagramas de entidade relacionamento, diagrama de contexto e diagrama de fluxo de dados. Os diagrama apresentados anteriormente poderiam ter sido desenvolvidos na própria ferramenta, mas para demonstrar as importações de diagramas optou-se pelo uso do *Power Designer* para desenvolver os diagramas. A Figura 37 apresenta a tela principal da ferramenta de apoio ao mapeamento.

FIGURA 37 - TELA PRINCIPAL DA FERRAMENTA



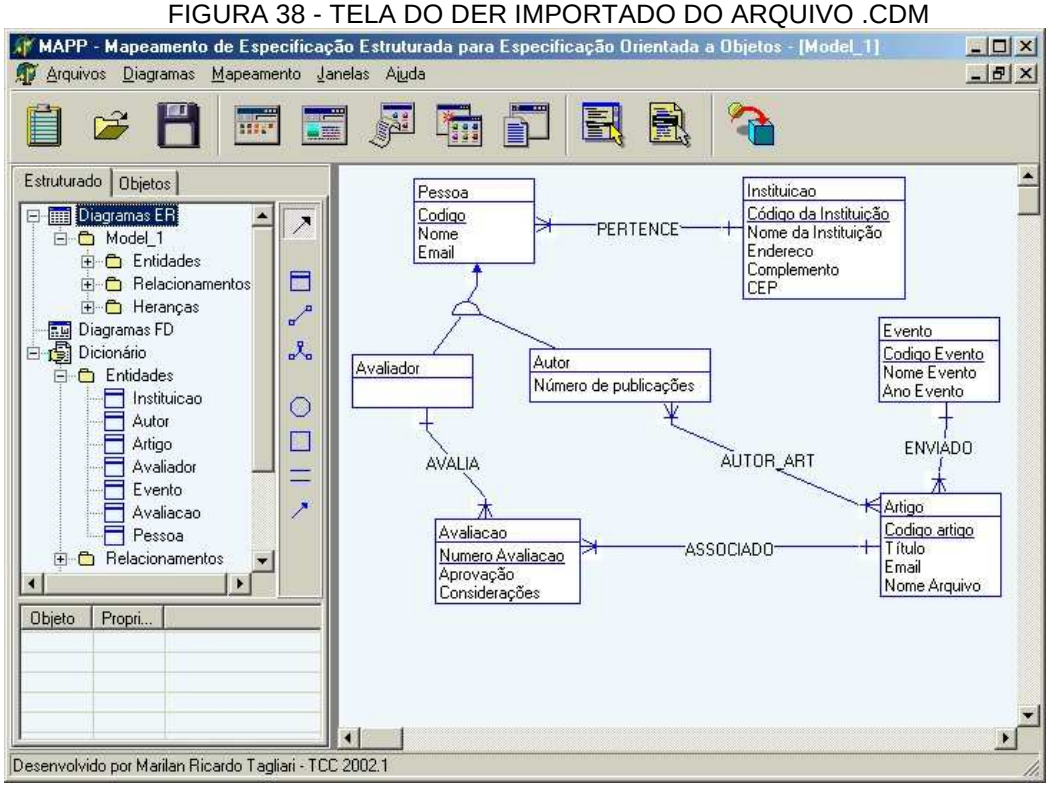
A seguir são descritos os pontos representados pelas letras na Figura 37:

a) <u>menus da ferramenta</u>: o menu “Arquivo” apresenta as funções de manipulação de arquivos como novo projeto, abrir, salvar, fechar, importar arquivos do *Power Designer*, gerar relatórios, gerar os arquivos de código fonte em Java e sair da ferramenta. O menu “Diagramas” apresenta as funções de criação de novos diagramas de entidade relacionamento, fluxo de dados, caso de uso, classes e seqüência. O menu “Mapeamento” é responsável pela ativação da tela de configuração do mapeamento e execução do mapeamento. O menu “Janelas” tem as funções de controle das janelas de diagramas. O menu “Ajuda” apresenta as janelas de informações sobre a ferramenta;

b) botões de acesso rápido: estes botões aceleram o trabalho com a ferramenta. Eles tem a função de chamar os procedimentos realizados pelas principais opções do menu. As funções dos botões são descritas na exata ordem em que eles aparecem: novo projeto, abrir, salvar, novo diagrama entidade relacionamento, novo diagrama de fluxo de dados, novo diagrama de caso de uso, novo diagrama de classes, novo diagrama de seqüência, importar diagrama de entidade relacionamento, importar diagrama de fluxo de dados e executar mapeamento;
c) dicionários: as duas abas representam os dicionários estruturado e orientado a objetos;
d) árvore de componentes: esta árvore apresenta todos os componentes do dicionário, como diagramas e elementos do dicionário de dados;
e) barra de ferramentas: esta barra apresenta as ferramentas para construção dos diagramas;
f) visualizador de propriedades: este visualizador tem a função de apresentar as características dos componentes do dicionário;
g) área de trabalho: a área de trabalho é utilizada para a abertura das telas dos diagramas;
h) barra de status: esta barra tema como finalidade informar ao usuário ocorrência da ferramenta.

Acessando o menu “Arquivo/Importar/Data Architect” abre-se uma janela para informar o caminho do arquivo .CDM do *Power Designer*. Esse procedimento importará o diagrama de entidade relacionamento, conforme Figura 38.

FIGURA 38 - TELA DO DER IMPORTADO DO ARQUIVO .CDM



Acessando o menu “Arquivo/Importar/Process Analyst” abre-se uma janela para informar o caminho do arquivo .PAM *Power Designer*. Esse procedimento importará o diagrama de contexto e o diagrama de fluxo de dados, conforme Figura 39 e Figura 40.

FIGURA 39 - TELA DO CONTEXTO IMPORTADO DO ARQUIVO .PAM

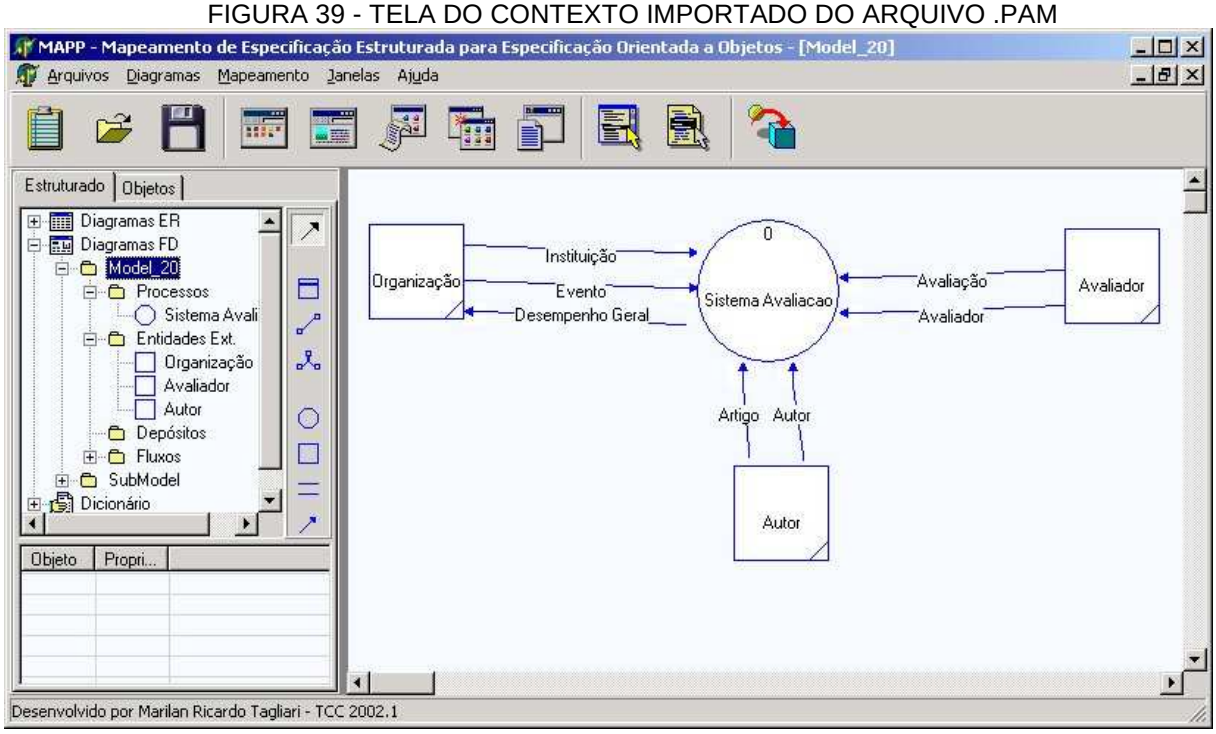
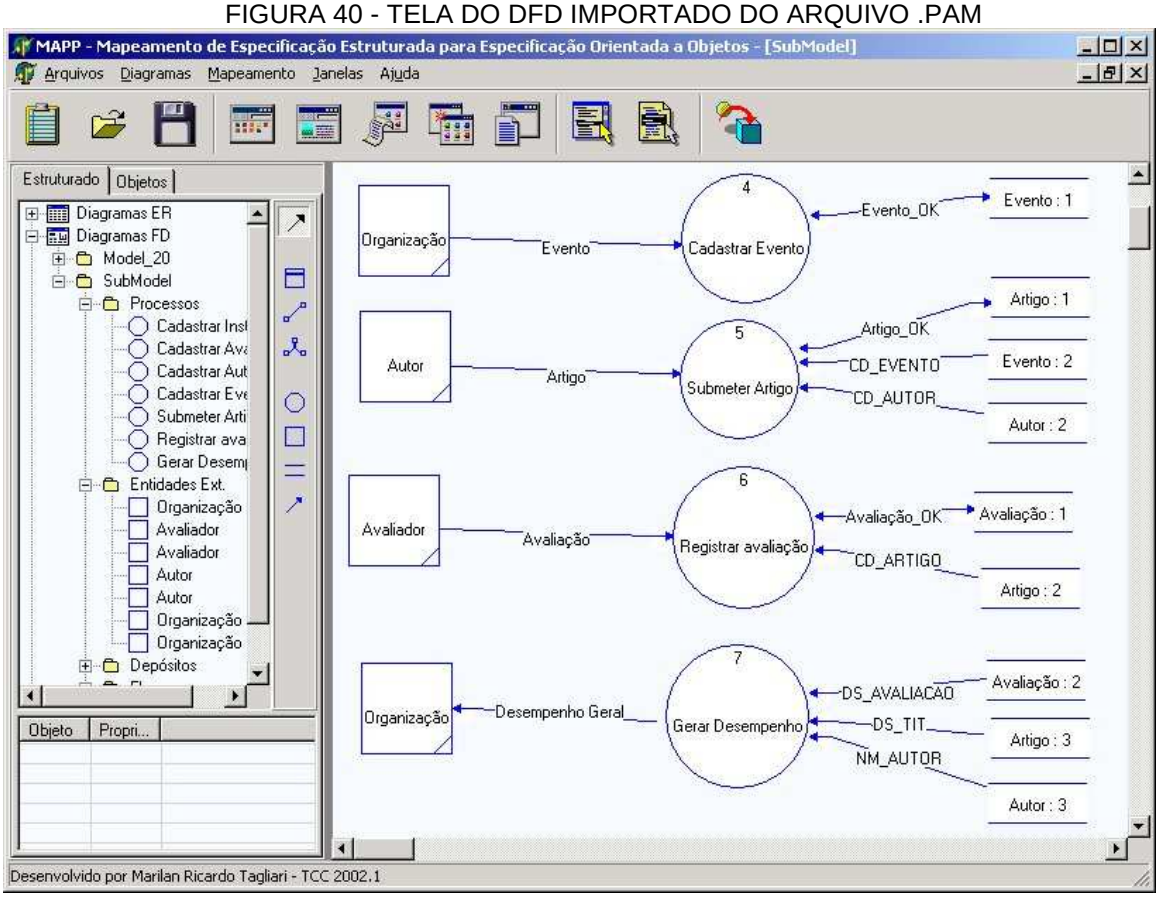


FIGURA 40 - TELA DO DFD IMPORTADO DO ARQUIVO .PAM



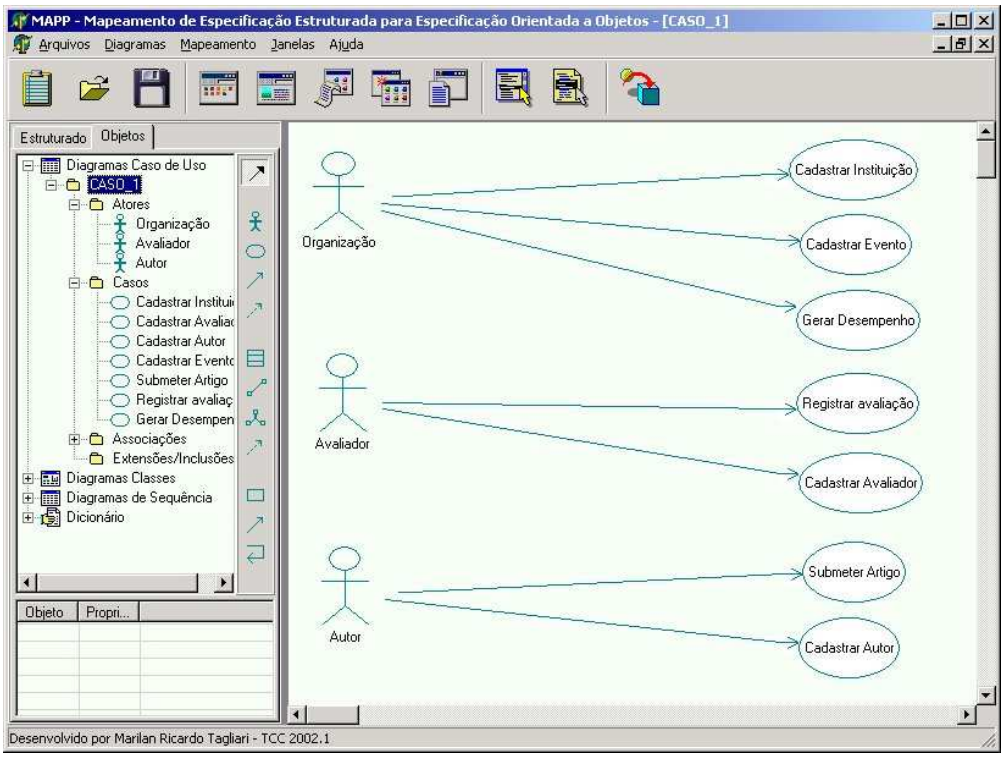
Neste momento a ferramenta permite a editoração dos diagramas importados. O próximo passo para a realização do mapeamento é a configuração das opções de mapeamento. A Figura 41 apresenta a tela de configuração de mapeamento.

FIGURA 41 - TELA DE CONFIGURAÇÃO DO MAPEAMENTO

A lista de sinônimos é utilizada para a realização do passo “d” na fase 1 da estratégia de mapeamento que consiste em excluir as classes repetidas ou com mesmos nomes possíveis. Sendo assim a ferramenta busca por classes que contenham o mesmo nome das classes de substituição. Caso encontre, associa os dois nomes à mesma classe de preferência, assim existindo apenas uma única classe, a preferencial. Nesta mesma tela é possível definir nome padrão para os métodos criados através do passo “c’ da fase 3, que consiste na criação de métodos baseados nas operações de fluxo de dados. No método de criação pode-se informar a palavra reservada “&Classe” indicando que o nome do método é o mesmo nome da classe, como é o caso do construtor na linguagem Java.

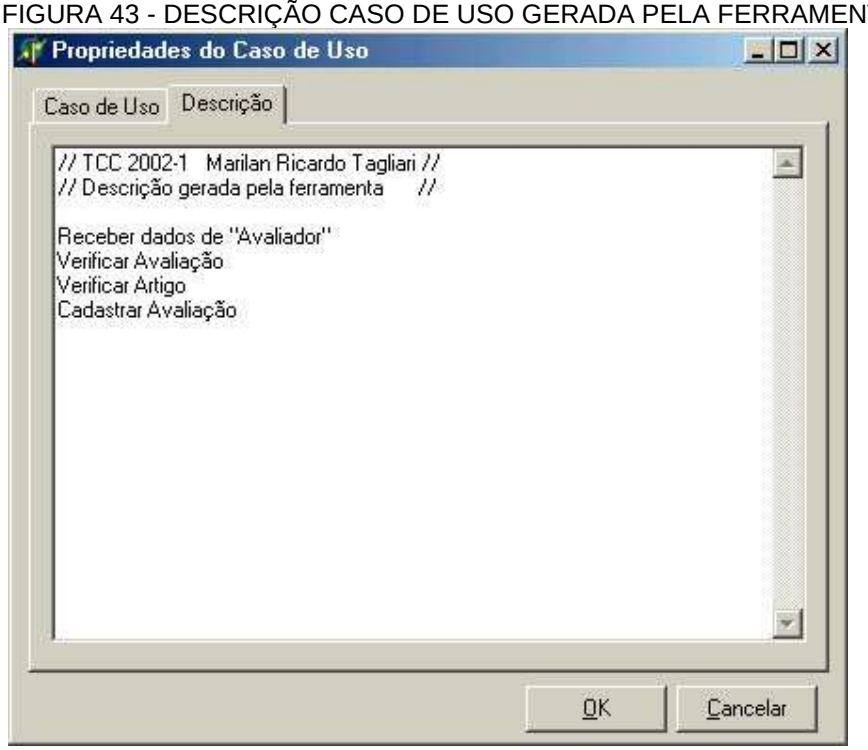
Após a configuração pode-se executar o mapeamento. A ferramenta disponibiliza a opção de mapear separadamente cada diagrama ou todos diagramas em uma única operação acessando o menu “Mapeamento/Mapear todos os Diagramas”. No momento da execução a ferramenta inicia o mapeamento para o diagrama de caso de uso, as descrições de cada caso, diagrama de classes e diagrama de seqüência para cada caso de uso. A estratégia de mapeamento gerou o diagrama de caso de uso (Figura 42) que apresenta três atores que conforme a estratégia são provenientes das entidades externas do DFD e sete casos de uso provenientes dos processo de nível 1 do DFD.

FIGURA 42 - DIAGRAMA DE CASO DE USO GERADO PELA FERRAMENTA



A ferramenta gera automaticamente a descrição dos casos de uso. A Figura 43 apresenta a descrição do caso de uso “Registrar Avaliação”. Percebe-se que a descrição gerada, apenas cita alguns passos básicos para que o engenheiro de software possa ampliar a descrição. Estes passos foram obtidos basicamente da interpretação do diagrama de fluxo de dados, conforme passos descritos na estratégia de mapeamento para diagrama de caso de uso.

FIGURA 43 - DESCRIÇÃO CASO DE USO GERADA PELA FERRAMENTA

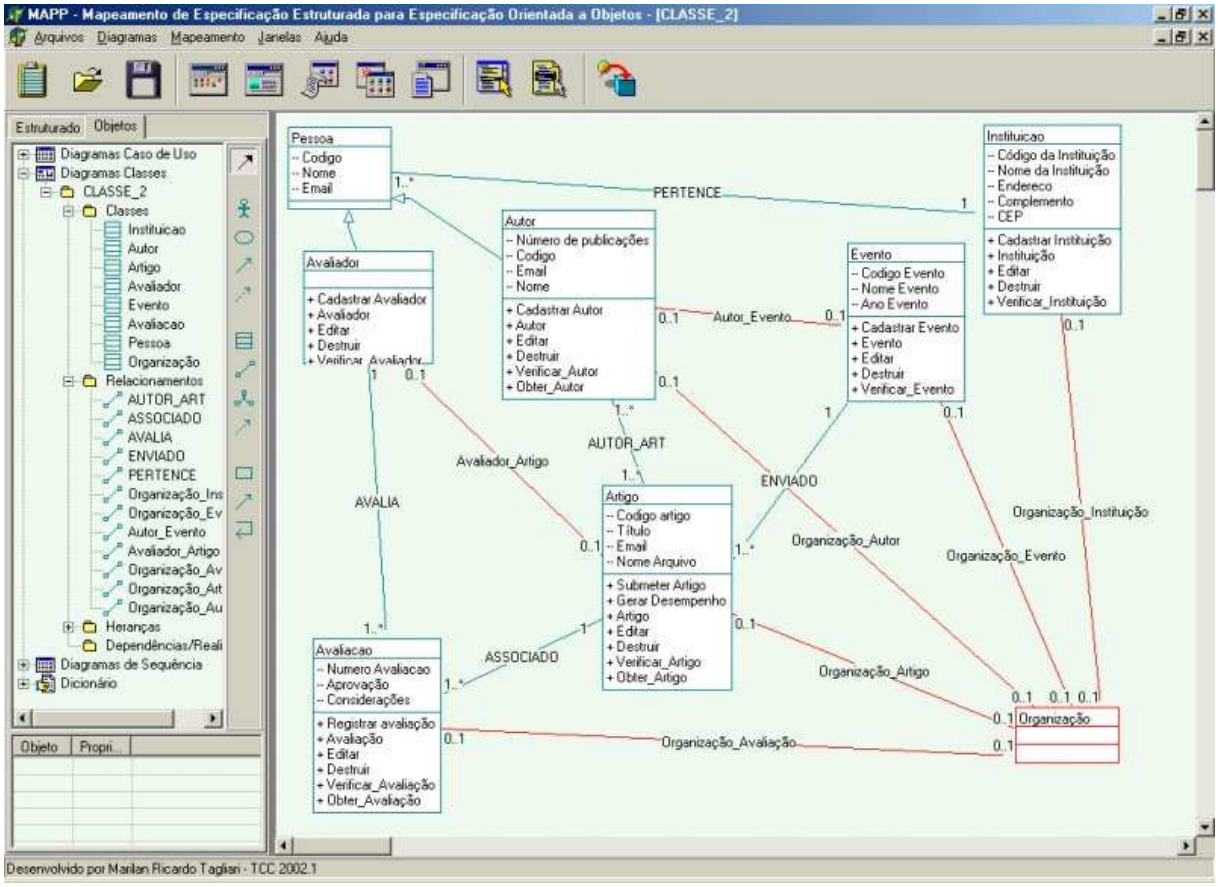


A ferramenta gerou o diagrama de classes (Figura 44) seguindo a estratégia de mapeamento que identificou oito classes (Pessoa, Avaliador, Autor, Instituição, Artigo, Evento, Avaliação e Organização). A classe “Organização” não apresentou atributos e métodos, por isso está na cor diferenciada sendo forte candidata a ser excluída do modelo. Após o mapeamento o usuário poderá realizar um refinamento manual possibilitando a exclusão ou alteração de propriedades. Os relacionamentos entre classes com cor diferenciada também são candidatos a exclusão, pois são provenientes da identificação de relacionamentos no DFD que na maioria dos casos testados são desnecessários. No caso do usuário não desejar excluir as classes ou relacionamentos poderá clicar com o botão esquerdo do mouse, e então aparecerá um menu para selecionar a opção “Diferenciar” alterando assim a cor do objeto.

O motivo pelo qual o objeto fica diferenciado ao outro se dá pelo fato deste objeto ser proveniente do mapeamento do diagrama de fluxo de dados, onde se observou em vários

exemplos que estes objetos não fariam parte do diagrama. Porém optou-se em manter esses objetos para que o usuário possa decidir se ele permanece ou não, apenas diferenciando ele dos demais.

FIGURA 44 - DIAGRAMA DE CLASSES GERADO PELA FERRAMENTA

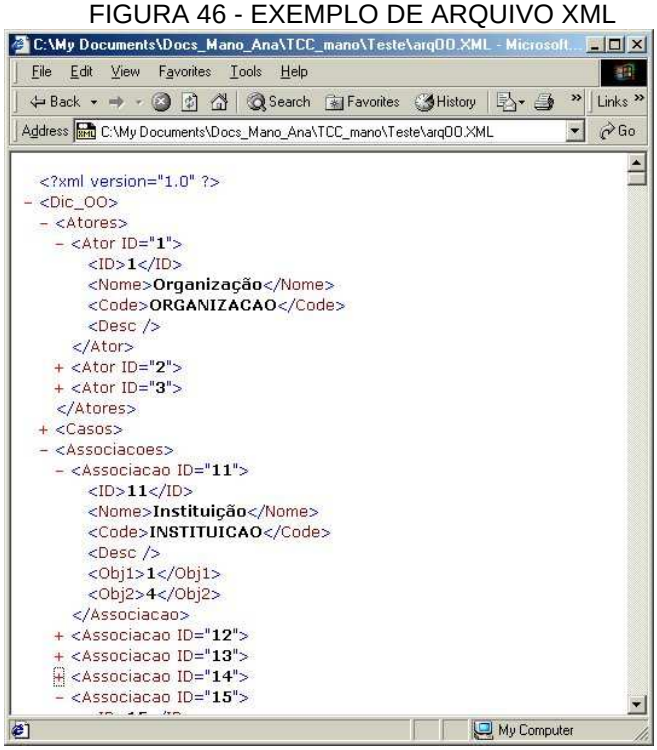


A ferramenta gerou para cada caso de uso um diagrama de seqüência. A Figura 45 apresenta o diagrama de seqüência do caso “Gerar Desempenho”. As mensagens identificadas poderão ser refinadas manualmente excluindo ou alterando a ordem das mensagens, pois mensagens que foram incluídas podem não fazer parte do diagrama. Esta versão da ferramenta não desenha a ativação dos métodos no diagrama de seqüência, para futuras versões pretende-se implementar esse recurso.

FIGURA 45 - DIAGRAMA DE SEQÜÊNCIA GERADO PELA FERRAMENTA

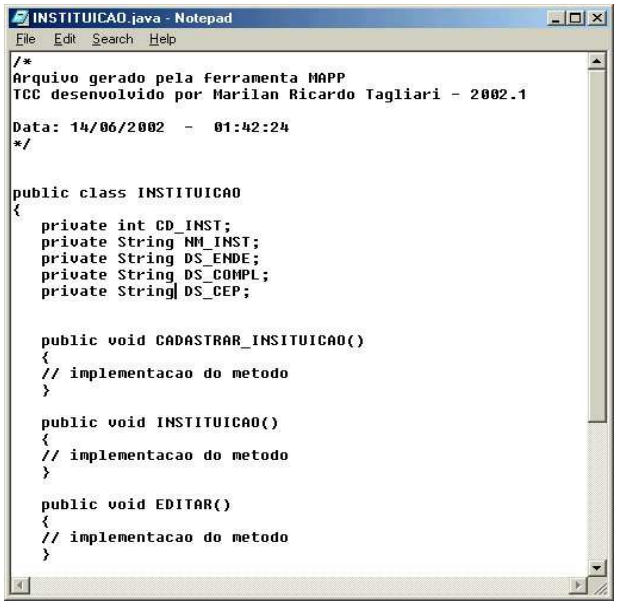
Como extensão do trabalho, a ferramenta possibilita ao usuário salvar as informações referentes ao diagramas em arquivo no formato XML. Esses arquivos poderão ser abertos na ferramenta restaurando os diagramas salvos nos arquivos XML. A Figura 46 apresenta um exemplo de arquivo salvo em XML sendo visualizado em um navegador de internet.

FIGURA 46 - EXEMPLO DE ARQUIVO XML



O diagrama de classes possibilitou a geração de código fonte para a linguagem Java. Através deste diagrama é possível gerar a estrutura das classes em Java. Para gerar código é necessário acessar o menu “Arquivo/Gerar Código Java” e então selecionar o diretório para criar os arquivos. A Figura 47 apresenta exemplo de código fonte em Java gerado pela ferramenta.

FIGURA 47 - EXEMPLO DE CÓDIGO FONTE EM JAVA



4.6 RESULTADOS E DISCUSSÃO

Para validar os resultados, três estudos de caso foram submetidos ao mapeamento utilizando a ferramenta desenvolvida. Os estudos de caso são de diferentes graus de complexidade e domínio do problema. O primeiro caso foi modelado no *Power Designer* e depois importado para a ferramenta demonstrando assim a utilização da importação. Os outros dois casos foram desenvolvidos na própria ferramenta demonstrando a capacidade de editoração de diagramas.

O estudo de caso 1 enfatiza o problema de uma empresa de eletrotécnica que deseja informatizar seu atendimento. O Quadro 10 apresenta o estudo de caso 1.

QUADRO 10 - ESTUDO DE CASO 1

Uma eletrotécnica tem enfrentado problemas para atender a demanda de serviço de consertos que tem recebido. Cada vez que um cliente traz um aparelho para consertar, o atendente cadastra o aparelho que é caracterizado pelo número de série, descrição do aparelho, marca, numero da nota fiscal e a data da aquisição que após abre uma nova ordem de serviço (uma por aparelho). Esta ordem de serviço diz respeito a um cliente, que deixa seu nome, endereço e telefone para contato. Um mesmo cliente pode ter outras ordens de serviço suas em andamento, e a empresa mantém um cadastro de todos os seus clientes. Na ordem de serviço constam ainda a data de recebimento do aparelho, número de série do aparelho, a descrição do defeito, e a data de previsão da entrega do aparelho ao cliente, e a data na qual foi efetivamente retirado. Para resolver o problema dos atrasos, a firma optou por designar um técnico experiente como gerente. Este gerente, inicia o dia examinando todas as ordens de serviço novas, designando então um conjunto de técnicos. A ordem pode ter vários técnicos. Cada técnico pode estar associado a várias ordens de serviço. Os técnicos são caracterizados por seu nome, endereço, numero de telefone para contato, e contrato de trabalho. Semanalmente são emitidos três relatórios para o gerente: Ordens de Serviço por Tipo de Defeito e Ordens de serviço por Técnico, Ordens de serviço por Ferramenta.

O estudo de caso 2 enfatiza o problema de uma empresa de turismo que deseja informatizar seu sistema de reservas. O Quadro 11 apresenta o estudo de caso 2.

QUADRO 11 - ESTUDO DE CASO 2

Uma agência de turismo deseja informatizar as rotinas de vendas de pacotes de viagens. O cliente que deseja viajar, dirige-se até a agência. Após uma fase inicial de cadastramento dos dados do cliente, onde um funcionário preenche uma ficha de com os dados do mesmo (Nome, Endereço, Telefone, E-Mail, CPF, RG), para poder efetuar uma reserva. Cada reserva está associada somente a um cliente e cada reserva de viagem pode ter um ou mais destinos (pacote). O funcionário também é responsável pelo cadastramento das opções de pacotes de viagem (Destino, Permanência, Período Promocional), que envolve as redes de hotéis (Nome, Endereço, Telefone, E-mail, CPNJ, Preço Diária) e do cadastramento das empresas de transporte (Nome, Endereço, Telefone, E-mail, CNPJ, Valor Passagem, Tipo de Transporte). Cada pacote de viagens possui no mínimo 1 hotel e 1 meio de transporte. Periodicamente, os clientes cadastrados são informados via e-mail, dos pacotes que estão em promoção no período (Nome do Cliente, e-mail, período promocional, destino). Mensalmente, o gerente da agência recebe um relatório dos pacotes de viagens mais vendidos (Número de reserva da viagem, destino, quantidade). Trimestralmente, é gerada uma relação dos clientes com maior número de reservas de viagens na agência (Número da reserva, nome do cliente).

O estudo de caso 3 enfatiza o problema de uma escola de informática que deseja informatizar seu sistema acadêmico. O Quadro 12 apresenta o estudo de caso 3.

QUADRO 12 - ESTUDO DE CASO 3

Uma escola de informática quer informatizar o controle acadêmico. A escola oferece aos seus alunos várias opções de cursos em diversos horários. Cada turma pode ter no máximo 20 alunos matriculados para fazer o curso. Apenas serão oferecidos os cursos que tiverem pelo menos 10 alunos matriculados. Cada curso pode possuir uma ou mais turmas cadastradas. Cada turma tem o seu horário especifico, que será estabelecido pela direção da escola de Informática. Cada turma terá apenas uma aula por semana. A quantidade de horas semanais

varia de acordo com a carga horário do curso desejado. Cada curso possui um preço específico e, alguns deles, possuem uma tacha de inscrição. O curso é pago mensalmente pelo aluno. Cada Curso tem apenas um instrutor. São guardados os seguintes dados dos instrutores (Nome, Endereço, Cidade, Estado, Data de Nascimento, CPF, Rg, e-mail, Telefone, Formação Escolar, Informações de cursos já realizados, Salário Fixo e % Comissão por Cursos). Cada curso, pode ainda, ter ou não algum material de apoio (livro, apostila, ...). Além dos instrutores, a escola ainda possui um cadastro de funcionários, que atuam na secretaria e em serviços de limpeza e manutenção da escola. Estes, por sua vez, informam (Nome, Endereço, Cidade, Estado, Data de Nascimento, CPF, Rg, e-mail (caso possuam) e Telefone para Contato. Todos os funcionários recebem apenas um salário fixo. Ao efetuar a matricula, o aluno deve se cadastrar informando o seus dados pessoais (Nome Completo, Endereço, Cidade, Estado, Data de Nascimento, CPF, Rg, e-mail, Formação Escolar e Telefone para contato) e o curso de seu interesse . O aluno poderá fazer apenas a matricula em um curso por vez, portanto se quiser fazer dois cursos ao mesmo tempo. ele vai fazer a matricula duas vezes. Ao término da matrícula ele recebe um comprovante de matricula (Resumo de Matricula). Cada turma terá uma sala especifica. São enviados para a direção da escola os seguintes relatórios: Trimestralmente a relação de cursos com o maior número de alunos matriculados; Mensalmente a relação dos alunos aniversariantes

A eficiência da estratégia de mapeamento é caracterizada pelos resultados obtidos no processo de identificação dos componentes dos diagrama de caso de uso, classes e seqüência. O Quadro 13 demonstra os resultados obtidos nos três casos submetidos ao mapeamento:

QUADRO 13 - RESULTADOS OBTIDOS

Componentes	**Caso 1** Mapeados/Desejáveis	**Caso 2** Mapeados/Desejáveis	**Caso 3** Mapeados/Desejáveis
Diagrama de Caso de Uso			
Atores	3 / 3	3 / 3	4 / 4
Casos de uso	5 / 5	8 / 8	12 / 12
Associações	5 / 5	8 / 8	12 / 12
Diagrama de Classes			
Classes	6 / 4	8 / 6	12 / 11
Atributos	11 / 11	28 / 28	52 / 52
Métodos	18 / 12	30 / 30	48 / 44
Relacionamentos	4 / 2	12 / 5	21 / 10
Heranças	2 / 2	0 / 0	3 / 3
Diagrama de Seqüência			
Objetos	12 / 12	23 / 23	32 / 31
Mensagens	20 / 20	37 / 37	54 / 50

O mapeamento do diagrama de caso de uso apresentou bons resultados, obtendo os atores e caso de uso conforme o desejável. As descrições dos casos de uso geradas pela ferramenta são básicas e servem apenas como idéia inicial da descrição. É claro que a ferramenta mapeia todos os casos de uso, não destacando os casos de uso primários ou principais.

No mapeamento do diagrama de classes, a estratégia mapeia algumas classes e relacionamentos provenientes do DFD que não são compatíveis com o problema e ficam diferenciados aguardando um refinamento manual e podem ser excluídos do diagrama. As heranças tiveram bons resultados sendo mapeadas de forma correta, porém alguns métodos das classes filhas poderiam estar na classe pai, assim compartilhando estes métodos. Um exemplo é o estudo de caso 3 que mapeou 48 métodos quando o desejável seria 44 métodos.

No mapeamento dos diagramas de seqüência os objetos apresentados foram compatíveis com o problema. A ferramenta gerou algumas mensagens que poderiam ser removidas. A estratégia não consegue identificar a ordem correta de envio das mensagens, sendo assim o engenheiro necessita realizar uma intervenção manual no processo de mapeamento.

5 CONCLUSÕES

Através deste estudo foi possível automatizar parte da estratégia de mapeamento de especificações estruturadas para especificações orientadas a objetos. Este trabalho possibilitou a criação de uma estratégia para o mapeamento do diagrama de caso de uso e seqüência, além dos melhoramentos realizados na estratégia de mapeamento do diagrama de classes inicialmente elaborada por George (1996). As extensões e melhorias na estratégia de mapeamento atenderam aos objetivos propostos para a realização deste trabalho.

A importação de diagramas gerados pela ferramenta CASE *Power Designer* foi realizado com sucesso. Esse processo de importação é importante, pois além de facilitar o mapeamento possibilita a reutilização de diagramas.

A ferramenta apresenta bons recursos gráficos para a editoração de diagramas estruturados e orientados a objetos utilizando a UML. Essa ferramenta apresenta um potencial para uso acadêmico nas disciplinas de engenharia de software e orientação a objetos.

Nos diagramas de caso de uso, a estratégia demonstrou eficiência, como mostra os resultados apresentados no Quadro 13. A estratégia identificou com sucesso todos os atores e casos de uso esperados com o mapeamento. As descrições dos casos de uso apresentam apenas uma idéia inicial sendo necessário que o engenheiro de software complete essa descrição.

No diagrama de classe, que representa a etapa mais complexa do mapeamento, a estratégia foi bem sucedida, identificando a maiorias das classes desejáveis. Os atributos foram identificados sem maiores problemas e os métodos dependem de um bom diagrama de fluxo de dados, pois os métodos em sua maioria dependem dos fluxos e suas operações. Neste diagrama é imprescindível que o usuário faça um refinamento manual no diagrama. As alterações efetuadas na estratégia de George não prejudicaram o mapeamento e as etapas adicionadas à estratégia cumpriram o objetivo proposto.

Nos diagramas de seqüência, a estratégia identificou a maioria dos objetos e mensagens, porém é necessário que o usuário remova e adicione algumas mensagens e altere a ordem das mesmas.

A ferramenta criada não automatiza todo o processo de mapeamento sendo necessário uma intervenção manual por parte do usuário para o refinamento dos diagramas. Apesar disso, de modo geral, os primeiros resultados alcançados são satisfatórios e demonstram a viabilidade do mapeamento das especificações estruturadas para especificações orientadas a objetos. É importante salientar que a qualidade do resultado final da estratégia depende da qualidade dos modelos de origem, devido à estratégia se basear plenamente nos elementos dos modelos.

Ao longo do trabalho foi possível desenvolver também a opção de geração de código fonte para a linguagem de programação Java. Esse código apresenta apenas a estrutura inicial da classe com seus atributos e métodos. A partir desse código o programador pode inserir o código das rotinas de cada método.

Foi possível implementar a geração de arquivos XML com o objetivo de que os modelos não sejam de uso exclusivo desta ferramenta, o que facilita a interligação com outras aplicações.

Sobre as limitações verificadas na ferramenta pode-se destacar a importação exclusiva de arquivos da ferramenta CASE *Power Designer* versão 6 e a geração de código fonte apenas para linguagem Java.

5.1 EXTENSÕES

A seguir serão apresentadas algumas sugestões para aperfeiçoar e dar continuidade ao trabalho desenvolvido:

a) permitir a leitura de outros repositórios disponibilizados por ferramentas CASE como *ERWin, System Architect, Rational Rose, Oracle Designer* entre outras;
b) permitir a leitura de outras versões do *Power Designer;*
c) aperfeiçoar a estratégia de mapeamento proposta;
d) permitir a editoração de outros diagrama da UML, como diagrama de estados;
e) permitir o mapeamento das especificações estruturadas e orientadas a objetos para especificações de aplicações WEB;
f) geração de código fonte para diversas linguagens como por exemplo *Delphi, C++ e SmallTalk.*

ANEXO 1 – SEÇÕES DO ARQUIVO DO *DATA ARCHITECT*

Neste anexo são apresentadas às seções e os campos utilizados na leitura do arquivo do *Data Architect*.

DESCRIÇÃO DA SEÇÃO MODELO (AMCMODL) – *DATA ARCHITECT*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome do modelo
CODE	A80	Code do modelo
DESC	TXT	Descrição sobre o modelo

DESCRIÇÃO DA SEÇÃO DOMÍNIO (AMCDOMN) – *DATA ARCHITECT*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome do domínio
CODE	A80	Code do domínio
DTTP	A30	Tipo do domínio

DESCRIÇÃO DA SEÇÃO INFORMAÇÃO DE ATRIBUTO (AMCINFO) – *DATA ARCHITECT*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome da informação
CODE	A80	Code da informação
DOMN	N8	Domínio da informação
DTTP	A30	Tipo da informação

DESCRIÇÃO DA SEÇÃO ENTIDADE (AMCENTT) – *DATA ARCHITECT*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome da entidade
CODE	A80	Code da entidade
DESC	TXT	Descrição da entidade

DESCRIÇÃO DA SEÇÃO RELACIONAMENTO (AMCRLSH) – *DATA ARCHITECT*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome da entidade
CODE	A80	Code da entidade
DESC	TXT	Descrição da entidade
ENTT1	N8	Entidade relacionada 1
ENTT2	N8	Entidade relacionada 2
CMIN1	A10	Cardinalidade mínima em entidade 1
CMAX1	A10	Cardinalidade máxima em entidade 1

CMIN2	A10	Cardinalidade mínima em entidade 2
CMAX2	A10	Cardinalidade máxima em entidade 2
RELT	N1	Dependência de entidade 2 em relação a entidade 1

DESCRIÇÃO DA SEÇÃO ATRIBUTO (AMCPENT) – *DATA ARCHITECT*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
ENTT	N8	Entidade pertencente
INFO	N8	Informação do atributo lido na seção AMCINFO
IDTF	A1	Define chave primária
MAND	A1	Define não nulo

DESCRIÇÃO DA SEÇÃO HERANÇA (AMCINHR) – *DATA ARCHITECT*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
ENTT	N8	Entidade pai
NAME	A80	Nome do relacionamento
CODE	A80	Code do relacionamento

DESCRIÇÃO DA SEÇÃO ENTIDADES DA HERANÇA (AMCLINH) – *DATA ARCHITECT*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
INHR	N8	Objeto de herança
ENTT	N8	Entidade filha

DESCRIÇÃO DA SEÇÃO DE SUBMODELOS (AMCSUBM) – *DATA ARCHITECT*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome do sub-modelo
CODE	A80	Code do sub-modelo
DESC	TXT	Descrição sobre o sub-modelo

DESCRIÇÃO DA SEÇÃO SÍMBOLOS (AMCSYMB) – *DATA ARCHITECT*

Nome Campo		**Descrição**
SID	N8	Identificador do símbolo
SID1	N8	Símbolo relacionado 1
SID2	N8	Símbolo relacionado 2
OID	N8	Objeto representado pelo símbolo

DESCRIÇÃO DA SEÇÃO GRUPO (AMCSUBG) – *DATA ARCHITECT*

Nome Campo		**Descrição**
SUBG	N8	Identificador do grupo
SUBM	N8	Identifica o sub-modelo

DESCRIÇÃO DA SEÇÃO SÍMBOLO DO GRUPO (AMCSUBS) – *DATA ARCHITECT*

Nome Campo		**Descrição**
SID	N8	Identificador do símbolo
SID1	N8	Símbolo relacionado 1
SID2	N8	Símbolo relacionado 2
GRPH	N8	Sub-grupo identificando o sub-modelo
OID	N8	Objeto representado pelo símbolo

ANEXO 2 – SEÇÕES DO ARQUIVO DO *PROCESS ANALYST*

Neste anexo são apresentadas às seções e os campos utilizados na leitura do arquivo do *Process Analyst*:

DESCRIÇÃO DA SEÇÃO MODELO (AMCMODL) – *PROCESS ANALYST*

Nome Campo		Descrição
OID	N8	Identificador do objeto
NAME	A80	Nome do modelo
CODE	A80	Code do modelo
DESC	TXT	Descrição sobre o modelo

DESCRIÇÃO DA SEÇÃO PROCESSO (AMCPRCS) – *PROCESS ANALYST*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome do processo
CODE	A80	Code do processo
DESC	TXT	Descrição sobre o processo
NUMR	N8	Número do processo
LLVL	N1	Define mais baixo nível

DESCRIÇÃO DA SEÇÃO ENTIDADE EXTERNA (AMCENTT) – *PROCESS ANALYST*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome da entidade
CODE	A80	Code da entidade
DESC	TXT	Descrição sobre a entidade

DESCRIÇÃO DA SEÇÃO DEPÓSITO (AMCSTOR) – *PROCESS ANALYST*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome do depósito
CODE	A80	Code do depósito
DESC	TXT	Descrição sobre o depósito

DESCRIÇÃO DA SEÇÃO DOMÍNIO (AMCDOMN) – *PROCESS ANALYST*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome do domínio
CODE	A80	Code do domínio
DTTP	A30	Tipo do dom;inio

DESCRIÇÃO DA SEÇÃO INFORMAÇÃO DE ATRIBUTO (AMCINFO) – *PROCESS ANALYST*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome da informação
CODE	A80	Code da informação
DOMN	N8	Domínio da informação
DTTP	A30	Tipo da informação

DESCRIÇÃO DA SEÇÃO FLUXO (AMCFLOW) – *PROCESS ANALYST*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
OBJ1	N8	Objeto 1
OBJ2	N8	Objeto 2
NAME	A80	Nome do fluxo
CODE	A80	Code do fluxo
DESC	TXT	Descrição sobre o fluxo
ENB1	N1	Direção da flecha em objeto 1
ENB2	N1	Direção da flecha em objeto 2
TYPE	N5	Informação CRUD

DESCRIÇÃO DA SEÇÃO ATRIBUTOS DO FLUXO (AMCDFDI) – *PROCESS ANALYST*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
FLOW	N8	Objeto de fluxo de dados
INFO	N8	Objeto de atributo

DESCRIÇÃO DA SEÇÃO ATRIBUTOS DO DEPÓSITO (AMCDSDI) – *PROCESS ANALYST*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
STOR	N8	Objeto de depósito
INFO	N8	Objeto de atributo

DESCRIÇÃO DA SEÇÃO SUBMODELO (AMCSUBM) – *PROCESS ANALYST*

Nome Campo		**Descrição**
OID	N8	Identificador do objeto
NAME	A80	Nome do sub-modelo
CODE	A80	Code do sub-modelo
DESC	TXT	Descrição sobre o sub-modelo

DESCRIÇÃO DA SEÇÃO SÍMBOLO (AMCSYMB) – *PROCESS ANALYST*

Nome Campo		**Descrição**
SID	N8	Identificador do símbolo
SID1	N8	Símbolo relacionado 1
SID2	N8	Símbolo relacionado 2
OID	N8	Objeto representado pelo símbolo

DESCRIÇÃO DA SEÇÃO GRUPO (AMCSUBG) – *PROCESS ANALYST*

Nome Campo		Descrição
SUBG	N8	Identificador do grupo
SUBM	N8	Identifica o sub-modelo

DESCRIÇÃO DA SEÇÃO SÍMBOLOS DO GRUPO (AMCSUBS) – *PROCESS ANALYST*

Nome Campo		**Descrição**
SID	N8	Identificador do símbolo
SID1	N8	Símbolo relacionado 1
SID2	N8	Símbolo relacionado 2
GRPH	N8	Sub-grupo identificando o sub-modelo
OID	N8	Objeto representado pelo símbolo

REFERÊNCIAS BIBLIOGRÁFICAS

AMBLER, Scott W. **Análise e projeto orientados a objetos**. Rio de Janeiro: Infobook, 1998.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: guia do usuário. Rio de Janeiro: Campus, 2000.

CANTÚ, Marco. **Dominando o Delphi 5** – a bíblia. Tradução João E.N. Tortello. São Paulo: Makron Books, 2000.

COAD, Peter; YOURDON, E. **Análise baseada em objetos**. Rio de Janeiro: Campus, 1992.

COLEMAN, Derek. **Desenvolvimento orientado a objetos**: o método fusion. Rio de Janeiro: Campus, 1996.

COUGO, Paulo. **Modelagem conceitual e projeto de banco de dados**. Rio de Janeiro: Campus, 1997.

DEMARCO, Tom. **Análise estruturada e especificação de sistemas**. Rio de Janeiro: Campus, 1989.

FOWLER, Martin; SCOTT, Kendall. **UML essencial** - um breve guia para linguagem-padrão de modelagem de objetos. Porto Alegre: Bookman, 2ed, 2001.

FURLAN, José David. **Modelagem de objetos através da UML**. São Paulo: Makron Books, 1998.

GANE, Chris. **Análise estruturada de sistemas**. Rio de Janeiro: Livros Técnicos e Científicos, 1983.

GEORGE, Joseph. A strategy for mapping from function-oriented software models to object-oriented models. **Software Engineering Notes (ACM SIGSOFT)**, v. 21, n. 2, p. 56-63, Março de 1996.

KNOP, Jeferson. **Comparação de metodologias orientadas a objetos através da especificação de um software**. 1999. 114 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MARTIN, James. **Análise e projeto orientados a objeto**. São Paulo: Makron Books, 1995.

MARTIN, James; MCCLURE, Carma. **Técnicas estruturadas e case**. São Paulo: Makron Books & McGraw-HILL, 1991.

PAGE-JONES, Meilir. **Fundamentos do desenho orientado a objeto com UML**. São Paulo: Makron Books, 2001.

POMPILHO, S. **Análise essencial**. Rio de Janeiro: Infobook S.A, 1994.

PRESSMAN, Roger S. **Engenharia de software**. São Paulo: Makron Books, 1995.

RATIONAL, SOFTWARE CORPORATION. **Rational support.** EUA, [2002]. Disponível em: <http://www.rational.com/support/documentation/manuals/rose.jsp>. Acesso em: 20 maio 2002.

SALDANHA, Emerson B. **Protótipo de uma ferramenta de apoio à migração de especificação estruturada para especificação orientada por objetos**. 1999. 46 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SYBASE, Inc. **Products manuals.** EUA, [2002]. Disponível em: <http://sybooks.sybase.com/pdd0800e.html>. Acesso em: 20 maio 2002.

YOURDON, Edward. **Análise estruturada moderna**. Rio de Janeiro: Campus, 1990.

ZIBELL, Dílson. **Protótipo de auxílio ao mapeamento de especificações estruturadas para especificações orientadas a objetos**. 1996. 67 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.