

Desenvolvimento Baseado em Componentes de um Framework do domínio de Cardiologia

João Luís Cardoso de Moraes

Antonio Francisco do Prado

moraes.uol@uol.com.br

prado@dc.ufscar.br

Universidade Federal de São Carlos - DC

C.P. 676 – 13565-905 – São Carlos (SP)

RESUMO

Este artigo apresenta um Processo de Desenvolvimento Baseado em Componentes de um Framework, do Domínio de Cardiologia. O Framework, denominado “FrameCardio”, foi desenvolvido em 4 passos: *Definição do Domínio do Problema*, *Especificação dos Componentes*, *Projeto Interno dos Componentes* e *Implementação dos Componentes*. Na *Definição do Domínio do Problema* foram identificados os requisitos do framework, com base nas experiências no desenvolvimento de um sistema de cardiologia com 320 classes. No passo *Especificação dos Componentes* foram definidos os comportamentos externos dos componentes, com suas responsabilidades e escopos, operações e interfaces. Em seguida, no passo *Projeto Interno dos Componentes*, os Componentes Especificados são refinados, considerando as tecnologias de implementação. Toda modelagem foi suportada por uma ferramenta CASE. Finalmente, no passo *Implementação dos Componentes* foi gerado o código dos componentes na linguagem *ObjectPascal*, usando um Sistema Transformacional.

O FrameCardio foi estruturado em camadas e organizado em pacotes de componentes, disponíveis na ferramenta CASE, para serem reutilizados pelas aplicações. Uma aplicação do Domínio de Cardiologia é apresentada para mostrar a reutilização de componentes do Framework.

Palavras chaves: Sistema Transformacional, *Framework*, Catalysis, Desenvolvimento Baseado em Componentes.

1 INTRODUÇÃO

A reutilização é um princípio essencial na área de Engenharia de Software para garantir a redução de esforços e custos no Desenvolvimento de Software e a redundância de código. Na tecnologia orientada a objetos, a reutilização de software pode ser assegurada com a adoção de *patterns*, *frameworks* de domínios específicos e componentes de software já existentes e testados.

Diferentes Processos de Desenvolvimento de Software (PDS) têm sido pesquisados para melhorar a produção de software. Pesquisas têm explorado diferentes tecnologias, incluindo o uso de ferramentas CASE (*Computer-Aided Software Engineering*), *frameworks*, Sistemas de Transformação de Software e linguagens de programação, orientadas a objetos, para obter software de melhor qualidade e com menor custo.

Com o objetivo de melhorar o PDS, este artigo apresenta o Desenvolvimento de um Framework, Baseado em Componentes, realizado em quatro passos. Os três primeiros passos, responsáveis pela modelagem do framework, são realizados com apoio de uma ferramenta CASE, e correspondem aos três níveis do método Catalysis [1] de Desenvolvimento Baseado em Componentes. O quarto passo responsável pela implementação dos componentes na linguagem *ObjectPascal* [2], é realizado com apoio de um Sistema de Transformação de Software, denominado Draco-PUC [3, 4]. Os componentes do framework podem ser reutilizados na construção de aplicações do domínio de Cardiologia, facilitando a modelagem, diminuindo a redundância de códigos e os custos da manutenção.

Para facilitar o desenvolvimento das aplicações, reutilizando componentes do framework, utiliza-se também a ferramenta CASE Rational Rose 2001 [5], para modelagem, e o Sistema de Transformação Draco-PUC para gerar código *ObjectPascal*. O código na linguagem *ObjectPascal* é gerado a partir das descrições das especificações do projeto das aplicações. São gerados códigos das estruturas das classes e interfaces dos componentes com seus atributos e protótipos de métodos.

Para suportar o desenvolvimento do framework FrameCardio e das Aplicações são utilizadas diferentes tecnologias:

- a) O método *Catalysis* de Desenvolvimento de Software Baseado em Componentes;
- b) O Sistema de Transformação Draco-PUC, para geração de código *ObjectPascal*; e
- c) A linguagem Orientada a Objetos, *ObjectPascal*, para implementação dos Componentes do *framework* FrameCardio, e das Aplicações desenvolvidas, segundo o PDS proposto.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta as principais tecnologias integradas no processo de desenvolvimento; a Seção 3 apresenta o Processo de Desenvolvimento Baseado em Componentes do framework do domínio de Cardiologia; a Seção 4 apresenta um Estudo de Caso, de uma aplicação reutilizando o FrameCardio; e, finalmente, a Seção 5 apresenta uma conclusão deste projeto de pesquisa.

2 PRINCIPAIS TECNOLOGIAS DO PDS BASEADO EM COMPONENTES

Segue a apresentação do método *Catalysis*, usado na modelagem do Framework e das aplicações, na ferramenta CASE Rational Rose 2001.

2.1 Método Catalysis

Catalysis é um método de desenvolvimento de software baseado em componentes que integra técnicas, Padrões e *Frameworks*. *Catalysis* começou em 1991 como uma formalização do OMT e teve influências dos métodos *Fusion* [6] e UML [7, 8]. Suporta as características das tecnologias Orientadas a Objetos recentes como Java, CORBA e DCOM e sua notação é baseada na *Unified Modeling Language* (UML).

Catalysis tem como princípios a **Abstração**, a **Precisão** e os **Componentes “Plug-In”**. O princípio *Abstração* orienta o Engenheiro de Software na busca dos aspectos essenciais do sistema, dispensando detalhes que não são relevantes para o contexto do sistema. O princípio *Precisão* objetiva descobrir erros e inconsistências na modelagem e o princípio *Componentes “Plug-In”* visa a reutilização de componentes para construir outros componentes [9].

O PDS em *Catalysis* segue as características do modelo Espiral da Engenharia de Software [10], e está dividido em três níveis lógicos: **Domínio do Problema**, **Especificação dos Componentes** e **Projeto Interno dos Componentes**, correspondendo às atividades tradicionais do ciclo de vida do software: **Planejamento**, **Especificação**, **Projeto** e **Implementação**, que são executadas de forma incremental e evolutiva, resultando na geração de uma nova versão de um protótipo a cada ciclo realizado.

No nível *Domínio do Problema* é dada ênfase na identificação dos requisitos do sistema, especificando “o que” o sistema deve fazer para solucionar o problema. Identificam-se os tipos de objetos e ações, agrupando-os em diferentes visões por áreas de negócio.

No nível *Especificação dos Componentes* dá-se ênfase na identificação, comportamento e responsabilidades dos componentes. Neste nível faz-se o refinamento dos modelos de negócios, do domínio do problema.

No nível do *Projeto Interno dos Componentes* dá-se ênfase na implementação dos requisitos especificados para os componentes do sistema, preocupando-se com suas distribuições físicas.

Para facilitar a modelagem segundo *Catalysis* foi utilizada a ferramenta CASE Rational Rose 2001, que disponibiliza técnicas para construção dos principais modelos de *Catalysis*.

2.2 Sistema de Transformação Draco-PUC

O Sistema de Transformação Draco-PUC é orientado a domínio e fundamenta-se na reutilização de componentes de software comuns nas diversas implementações, refletindo objetos e operações.

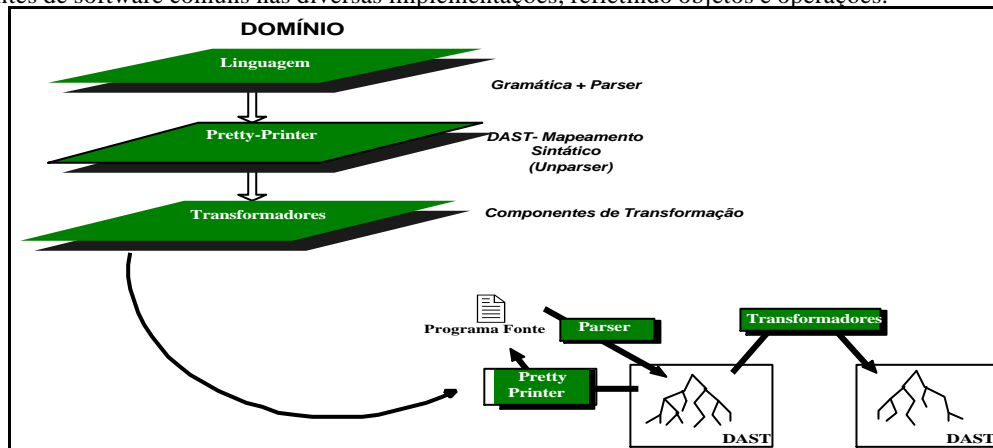


Figura 1: Partes de um domínio no Sistema de Transformação Draco-PUC.

No Sistema de Transformação Draco-PUC, um domínio é composto de três partes, **Linguagem**, **Prettyprinter** e **Transformadores**, conforme mostra a Figura 1. A linguagem é definida através de sua gramática e seu *parser*, que analisa um programa qualquer do domínio, e gera sua representação interna no Draco-PUC. Esta representação interna, denominada *Draco Abstract Syntax Tree (DAST)*, é usada para aplicar os componentes de transformação, que geram uma nova *DAST* no mesmo ou em outro domínio. A função do *prettyprinter* é exibir a *DAST*, na forma textual, orientado pela sintaxe da linguagem do domínio. Os transformadores são pacotes de componentes de transformação que atuam numa *DAST*, para gerar uma nova *DAST*, conforme mostra a Figura 1.

O Sistema de Transformação (ST) Draco-PUC foi usado como principal mecanismo para a geração de código do FrameCardio e de suas aplicações. Para suportar a geração do código foram construídos dois domínios

denominados MDL (Modeling Domain Language) e *ObjectPascal*. As especificações UML, modeladas na ferramenta CASE, segundo Catalysis, são armazenadas num arquivo de descrições textuais. As transformações que geram código baseiam-se nas gramáticas dos dois domínios, MDL e *ObjectPascal*, apresentadas parcialmente na Figura 2.

MDL	ObjectPascal
<pre> class_Object : 'CLASS' STR1 classAttributes; classAttributes : quidu .nl stereotype quid classAttributes_Attr ; classAttributes_Attr : (documentation)? (stereotype)? (superClasses)? (used_nodes)? superClasses : 'superclasses' .nl inheritance_relationship_list*; inheritance_relationship_list : '(list' .sp 'inheritance_relationship_list' .sp inheritance_Relationship*)'; </pre>	<pre> compilation_file : program_file unit_file package_file ; package_file : 'PACKAGE' .sp IDENTIFIER ';' requires_clause contains_clause ; unit_file : unit_heading interface_part? implementation_part? initialization_part? ';' unit_heading : 'UNIT' IDENTIFIER ';'; IDENTIFIER : [A-Za-z_][A-Za-z_0-9]*; </pre>

Figura 2: Gramáticas MDL e ObjectPascal

2.3 Linguagem ObjectPascal

ObjectPascal é a linguagem de desenvolvimento do ambiente Delphi [11] Baseado em Componentes. Dentre suas principais características destacam-se o suporte para configuração de projetos e arquivos fonte, a programação visual e a manutenção de informação de dependência entre unidades.

A Figura 3 mostra as telas principais do Delphi, onde se destacam as janelas com barra de ferramentas, página de componentes, Inspetor de Objetos e *Form designer* e seu código fonte.

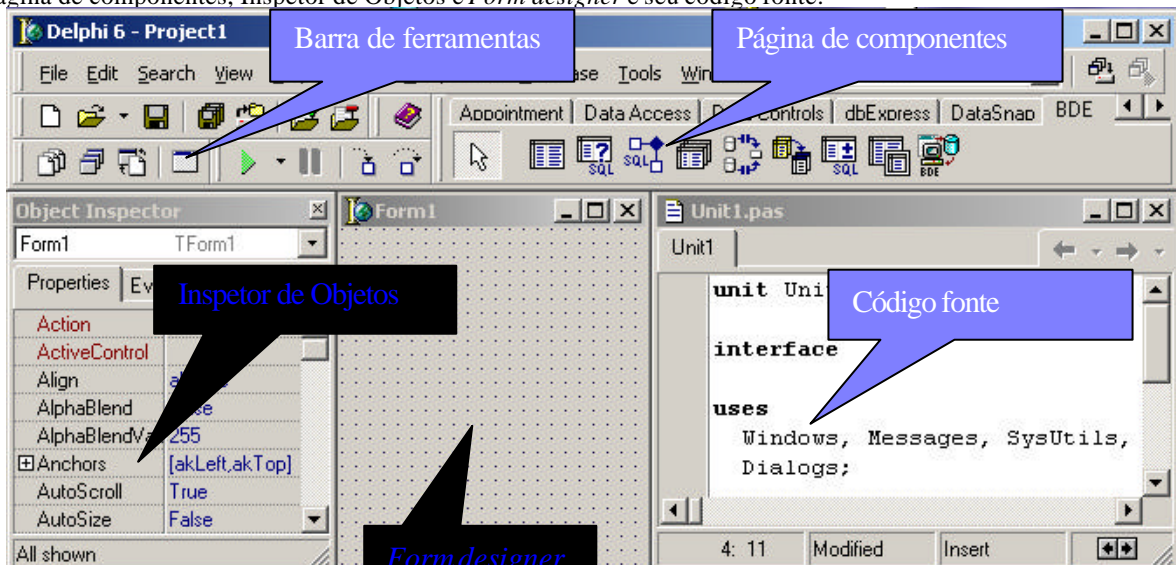


Figura 3: Ambiente Delphi.

A linguagem *ObjectPascal* representa um conjunto de extensões, orientadas a objetos, do *Pascal* padrão. É uma linguagem de alto nível, compilada e fortemente tipificada que suporta a implementação orientada a componentes, usando *frameworks* e ambiente RAD (*Rapid Application Development*).

A *Visual Component Library* (VCL) [12] é uma hierarquia de classes, escritas em *ObjectPascal*, que suporta o desenvolvimento de aplicações através da reutilização de componentes e do *Object Inspector*, sendo, este último, usado para a inspeção e definição de objetos.

Todos os objetos da VCL descendem de *TObject*, uma classe abstrata cujos métodos encapsulam comportamentos fundamentais para construção e destruição de componentes e tratamento de mensagens. Componentes na VCL descendem da classe abstrata *TComponent* e podem ser utilizados em tempo de projeto. Componentes visuais, como *TForm* e *TButton*, que aparecem nas interfaces, são chamados de controles e descendem da *TControl*.

Reunindo estas tecnologias foi desenvolvidos um framework e suas aplicações, do domínio de cardiologia, conforme se segue.

3 DESENVOLVIMENTO DO FRAMECARDIO

O *FrameCardio* foi desenvolvido em quatro passos: *Definir Domínio do Problema, Especificar Componentes, Projetar Componentes e Implementar Componentes*.

Parte-se dos requisitos comuns do domínio de Cardiologia, para construir componentes que possam ser reutilizados pelas aplicações deste domínio. As linhas horizontais tracejadas separam os passos do processo de desenvolvimento, segundo os níveis de Abstração do método *Catalysis*: *Domínio do Problema, Especificação dos Componentes, Projeto Interno dos Componentes e Implementação Usando Transformações*, mostrados à direita da Figura 4.

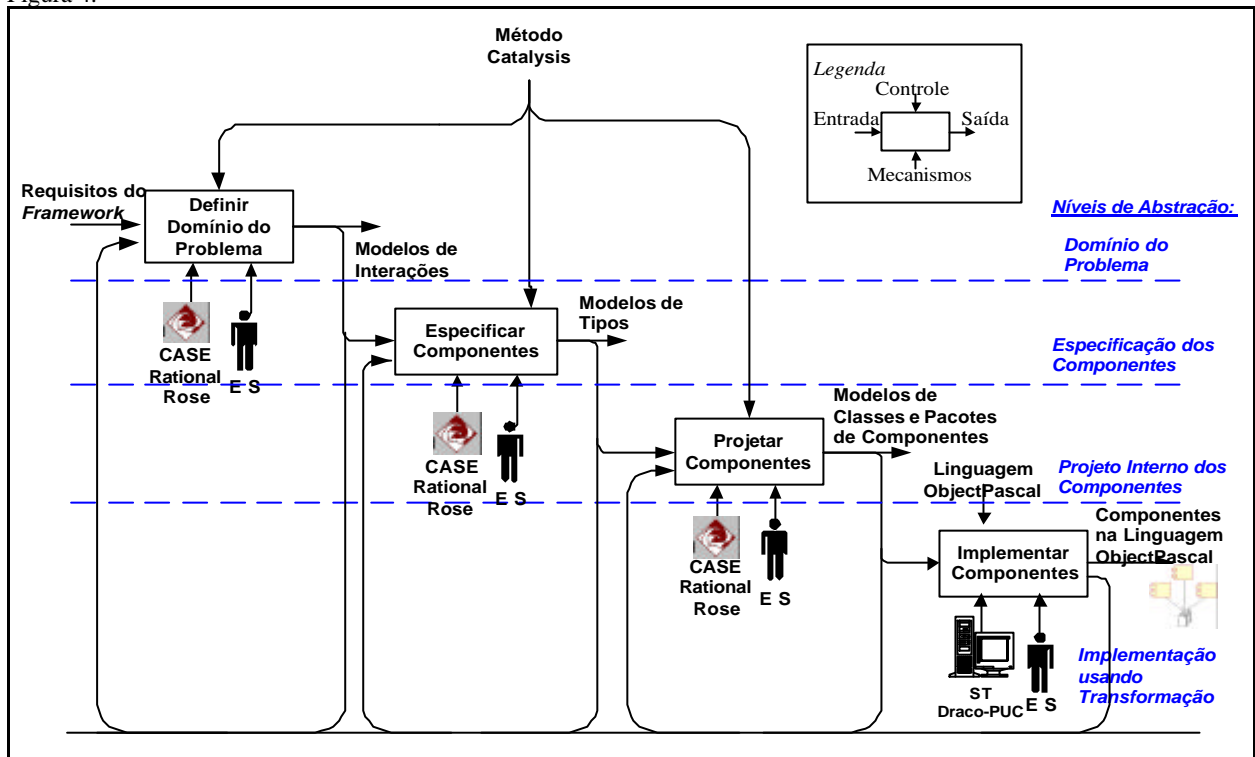


Figura 4: Processo de Desenvolvimento do Framework *FrameCardio*.

3.1 Definir Domínio do Problema

No passo Definir Domínio do Problema foram definidos: a terminologia do domínio do problema, o entendimento do processo de negócio, os papéis dos atores e as colaborações para especificar o comportamento de grupo de objetos do domínio de cardiologia. Os principais modelos de interações utilizados neste passo foram os de Casos de Uso, Ações e Colaborações. Os requisitos identificados são inicialmente especificados pelas Regras de Negócios que expressam o entendimento do domínio do problema, representado em um Modelo de Colaborações, incluindo associações e casos de uso, refletindo os processos existentes. A Figura 5 mostra um Modelo de Colaborações do *framework* com os atores Médico, Paciente e Funcionário e as ações: Realizar Consulta, Realizar Exame, Emitir Laudo e Realizar Cirurgia.

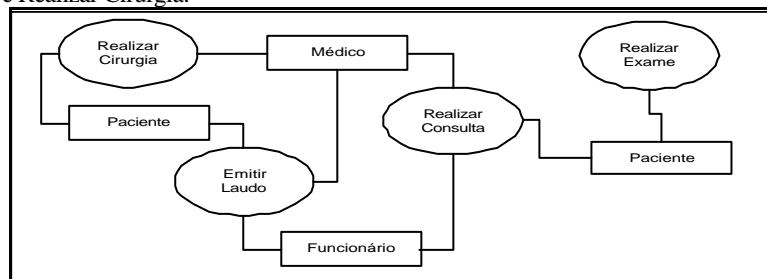


Figura 5– Modelo de Colaborações

Profissionais e médicos da área de cardiologia e um sistema de software do Instituto do Coração de Marília (ICM) foram as principais fontes para identificação dos requisitos. O sistema do ICM foi desenvolvido e implantado por um dos autores que vem realizando sua manutenção desde 1997. Reuniões, entrevistas, estudo de sistemas legados e observações de cenários de uso do sistema foram empregados no levantamento e identificação dos requisitos. Com a utilização do sistema surgiram novos requisitos e, ao longo do tempo, foram necessárias atualizações para acompanhar mudanças de tecnologias. Todas estas experiências foram importantes para conhecer o domínio de Cardiologia, facilitando o desenvolvimento do Framework.

Através do refinamento do *Modelo de Colaborações*, buscando um melhor entendimento das funcionalidades do *framework*, definem-se as principais ações do sistema especificando-se os casos de uso. Os casos de uso representam cenários de uso dos atores do sistema. A Figura 6 mostra o Modelo de Casos de Uso, obtido do refinamento do Modelo de Colaborações da Figura 5. Ações podem ser modificadas ou adicionadas, como por exemplo, os casos de uso *cadastrarPaciente*, *cadastrarMédico*, *registrarConsulta* e *solicitarConsultasRealizadas*, identificados a partir da ação **Realizar Consulta**.

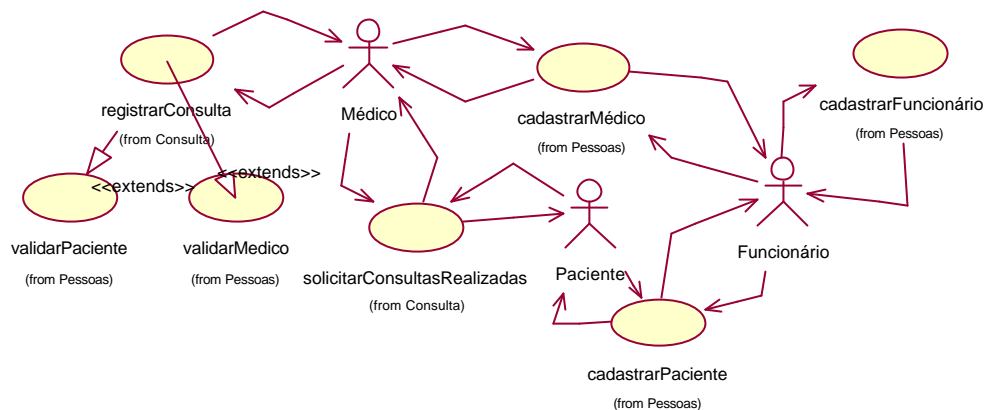


Figura 6 – Modelo de Casos de Uso

3.2 Especificação dos Componentes

No segundo passo, *Especificação de Componentes*, foram definidos os comportamentos externos dos componentes, com suas responsabilidades e escopos, operações e interfaces. O principal modelo é o de Tipos, que descreve o comportamento externo de um objeto, independente das decisões de implementação. A Figura 7 mostra o Modelo de Tipos obtido do refinamento do Modelo de Casos de Uso da Figura 6. A notação <> indica os tipos que podem ser reutilizados pelas aplicações do FrameCardio. Os tipos estão relacionados através de associações, agregações ou herança, com as respectivas cardinalidades, que expressam as ocorrências mínimas e máximas de objetos participantes de um relacionamento.

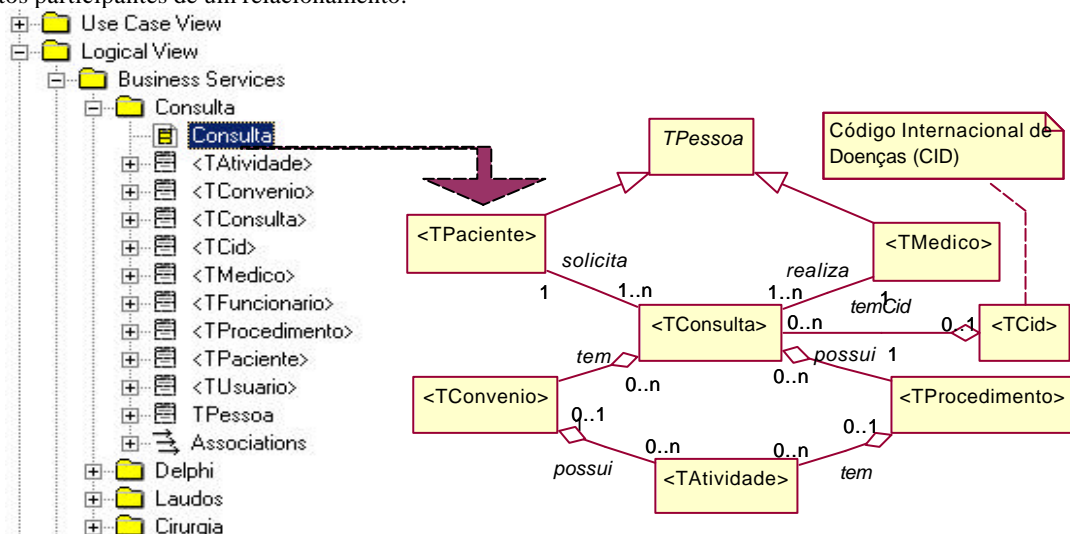


Figura 7 – Modelo de Tipos

3.3 Projetar Componentes

Em seguida, no passo *Projetar Componentes*, os Componentes Especificados, nos modelos de tipos e de sequência, são refinados, considerando as tecnologias de implementação. Seguindo as técnicas do método *Catalysis*, desta atividade obtém-se o Projeto Interno dos Componentes, com suas distribuições físicas. Dentro dos modelos desta etapa destacam-se os Modelos de: Classes de componentes, Componentes, e Pacotes de componentes.

O Modelo de Componentes representa a arquitetura física dos componentes, com suas interfaces para conexão e suas dependências, especificado a partir do Modelo de Classes refinado do Modelo de Tipos. Neste modelo têm-se as interfaces dos componentes, representadas pelos ícones em forma de círculos. Uma interface especifica os serviços realizados pelo componente. Um componente pode realizar várias interfaces, fornecendo um conjunto de métodos capazes de implementar adequadamente os serviços especificados na interface. A conexão entre componentes, ou entre um componente que tem acesso aos serviços de outro componente se dá por meio da interface e é representada por um relacionamento de dependência. A Figura 8 mostra um Diagrama de Componentes, destacando suas interfaces e associações. No caso o componente TConsulta conecta-se com o componente TPaciente através da interface IPaciente.

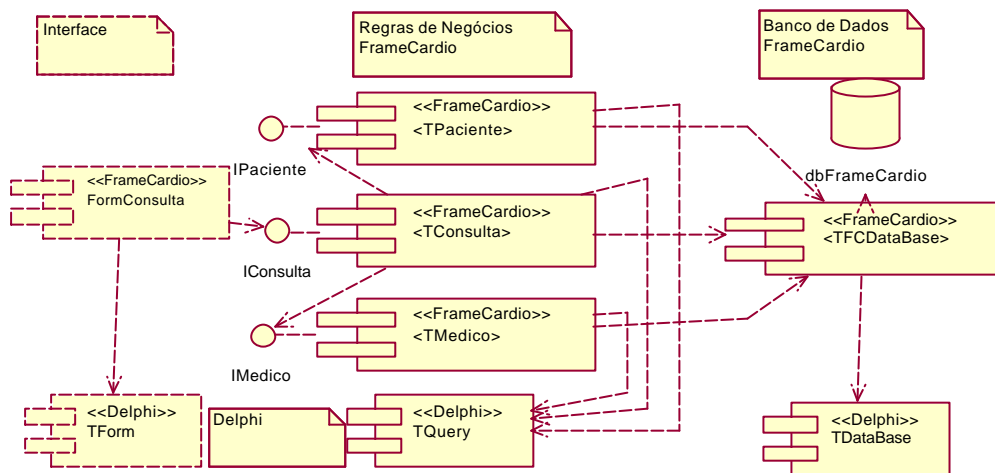


Figura 8: Modelo de Componentes do Pacote Consulta.

Para facilitar a reutilização dos 320 componentes do FrameCardio, estes foram organizados num **Modelo de Pacotes** conforme mostra a Figura 9. Os pacotes foram distribuídos nas seguintes camadas: *User Services*, *Business Services* e *Data Services*. A primeira camada *User Services* disponibiliza componentes para desenvolvimento de interfaces gráficas e visuais. A segunda camada, *Business Services*, provê componentes da área de cardiologia, em três pacotes: *Consultas*, *Cirurgias* e *Laudos*. Estes pacotes são reutilizados pelas diferentes aplicações do domínio de cardiologia. A terceira camada, *Data Services*, provê componentes de acesso a Banco de Dados relacional, que são utilizados pelas regras de negócio. Outros pacotes disponíveis, como o de componentes do Delphi, podem ser reutilizados, como mostra o Modelo da Figura 9.

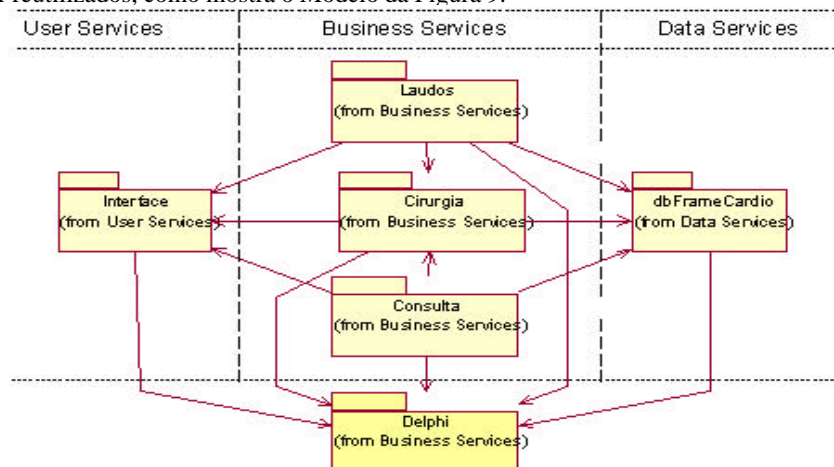


Figura 9: Arquitetura em Camadas do Framework FrameCardio.

A arquitetura em três camadas possibilita a reutilização de componentes, aumenta a independência e facilita a portabilidade das aplicações, que podem ser escritas em diferentes linguagens, acessar diferentes gerenciadores de Bancos de Dados e utilizar as mesmas regras de negócio. Esta organização em camadas torna o sistema mais flexível para suportar as mudanças tecnológicas sem comprometer sua estrutura, aumentando assim seu ciclo de vida.

3.4 Implementar Componentes

Os componentes são implementados com base nos seus *Projetos Internos*. Utiliza-se o Sistema Transformacional Draco-PUC para geração do Código *ObjectPascal*, a partir das descrições MDL dos projetos dos componentes. Para que se tenha uma idéia de como atuam as transformações construídas no Draco-PUC, a Figura 10 mostra a Transformação Classes_ModeloLogico, que reconhece a especificação de uma classe em MDL, no ponto de controle LHS e gera o correspondente código *ObjectPascal*, conforme o padrão de substituição especificado no ponto de controle RHS.

TRANSFORM Classes_ModeloLogico LHS: { {dast MDL (object Class [[STRI classe]] quid [[STRI nrquid]] class_attributes ([[classAttribute* atributosClasse]]) operations ([[operations* metodos]]) }}	RHS: { {dast ObjectPascal uses Windows, Messages, Classes, DB, Dialogs, DBTables; type [[IDENTIFIER classeType]] = class(TQuery) [[not_field_definition* ListaDeAtributos]] public [[not_method_definition* ListaDeMetodos]] end; } } RHS: { {dast delphi.unit_file unit [[IDENTIFIER nomeClasse]]; [[interface_part DeclaracaoUnit]] implementation [[impl_decl_sect* ListaCorpoMetodos]] end. } }
---	---

Figura 10: Transformador MdlParaDelphi

Para que se tenha uma idéia da geração de código, a Figura 11 mostra, à esquerda, a especificação MDL da Classe TConsulta, e à direita, o correspondente código *ObjectPascal* gerado. No caso, a especificação object Class de TConsulta deu origem ao Type TConsulta, que implementa uma classe derivada da classe TQuery. A especificação object Operation InserirConsulta deu origem à procedure FCInserirConsulta. As especificações object ClassAttribute CodigoConsulta e DtConsulta deram origem aos atributos FCodigoConsulta, do tipo Integer e FDtConsulta do tipo TDateTime. As bibliotecas de classes Windows, Messages e outras são adicionadas conforme as necessidades da implementação.

Especificação MDL	Código ObjectPascal Gerado
(object Class "<TConsulta>" operations (list Operations (object Operation "InserirConsulta" class_attributes (list class_attribute_list (object ClassAttribute "CodigoConsulta" type "Integer") (object ClassAttribute "DtConsulta" type "Date")	unit UnitConsulta; interface uses Windows, Messages, SysUtils, Classes, Dialogs, DB, DBTables, UnitMedico, UnitPaciente; Type TConsulta = class(TQuery) FCodigoConsulta : Integer; FDtConsulta : TDateTime; procedure FCInserirConsulta;

Figura 11: Geração de Código ObjectPascal, a partir de especificações MDL

3.5 Ciclo de vida do framework

O PDS apresentado segue as características do modelo Espiral de desenvolvimento de software, no qual o Engenheiro de Software pode retornar aos passos anteriores para refinar os modelos especificados e obter nova implementação dos componentes. A cada ciclo, que compreende a execução dos quatro passos do PDS, têm-se novos protótipos dos componentes. Os protótipos permitem verificar se os requisitos especificados estão sendo atendidos, orientando as correções e melhorias dos componentes. Aplicações, reutilizando os componentes, são desenvolvidas para testar os componentes. Embora informais, os testes têm sido usados porque facilitam a depuração do software.

A Figura 12 mostra o código gerado de um componente com suas interfaces.

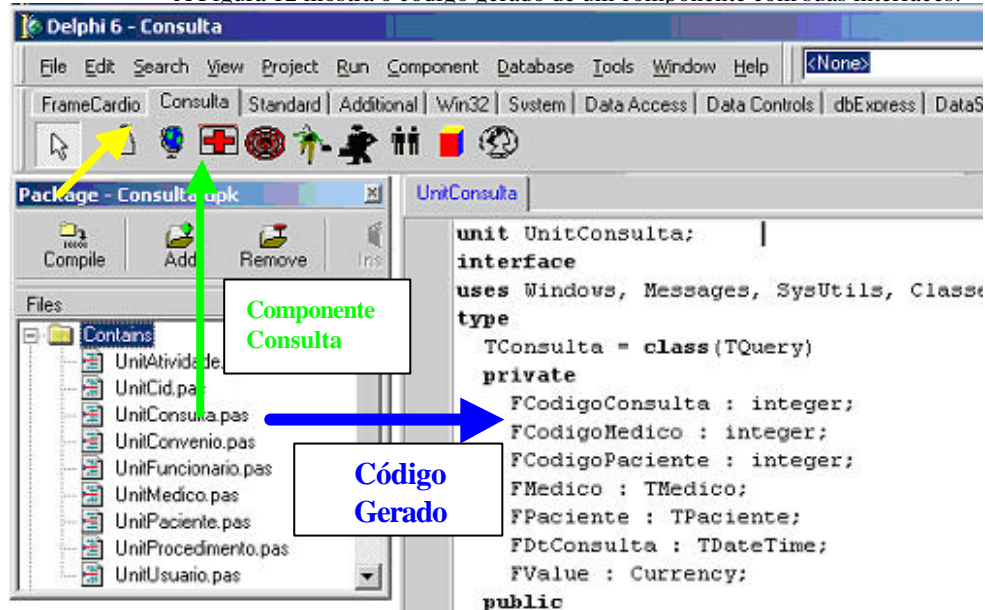


Figura 12 – Componente TConsulta implementado em ObjectPascal

A seguir será apresentado um Estudo de Caso do domínio de cardiologia, reutilizando componentes do FrameCardio.

4 ESTUDO DE CASO

Trata-se de um Sistema de Atendimento Médico aos Pacientes na clínica de Cardiologia. O paciente é atendido na recepção da Clínica por um funcionário e encaminhado até a sala do médico cardiologista. A consulta é realizada e os seus resultados são registrados no Banco de Dados. O Médico pode verificar se o Paciente já está cadastrado no sistema e atualizar suas informações.

Segue-se uma apresentação de cada passo do desenvolvimento desta aplicação, que compreendem: **Modelar, Implementar e Executar Aplicação.**

4.1 Modelar Aplicação

Inicialmente, o Engenheiro de Software, na ferramenta CASE, modela a aplicação conforme o primeiro nível de Catalysis. São especificados os requisitos da aplicação, preocupando-se com suas regras de negócio. A Figura 13 mostra os principais casos de uso da aplicação, com uma breve descrição, suas entradas e saídas.

Nr	Descrição	Caso de Uso	Entrada	Saída
01	Médico realiza Consulta	realizarConsulta	DadosConsulta	Msg01
02	Médico verifica Consultas	gerarConsultas	DadosGerarConsulta	Msg02
03	Funcionário cadastra Médico	cadastrarMédico	DadosMédico	Msg03
04	Funcionário cadastra Paciente	cadastrarPaciente	DadosPaciente	Msg04

Figura 13: Casos de Uso da Aplicação

Os casos de uso são modelados, em diagramas de Casos de Uso que mostram os atores interagindo com o sistema. A Figura 14 mostra, por exemplo, o ator médico interagindo para realizar uma consulta. Neste nível do problema, os relacionamentos dos atores com os casos de uso são indicados dispensando detalhes que não são relevantes para o contexto do sistema.

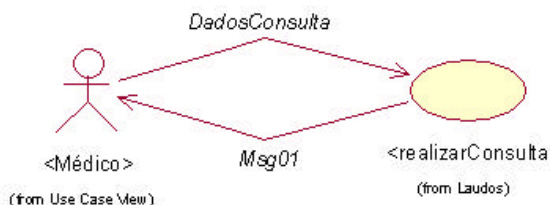


Figura 14: Modelo de Caso de Uso realizarConsulta

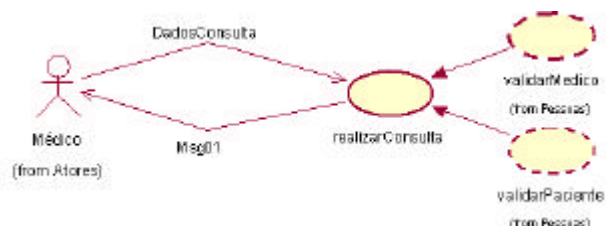


Figura 15: Modelo de Casos de Uso refinado

A reutilização acontece em diferentes níveis de abstração. A Figura 15 mostra o modelo de Casos de Uso, obtido do refinamento do Modelo da Figura 14. No caso, realizarConsulta reutiliza os métodos validarPaciente, do componente TPaciente, e validarMedico do componente TMedico, do FrameCardio.

Neste primeiro nível, especifica-se ainda o Modelo dos Tipos da aplicação, preocupando-se com “o que” a aplicação deve fazer para atender os seus requisitos. É um modelo de alto nível de abstração do domínio do problema, onde se buscam os tipos essenciais da aplicação. A Figura 16 mostra um Modelo de Tipos neste nível do desenvolvimento.

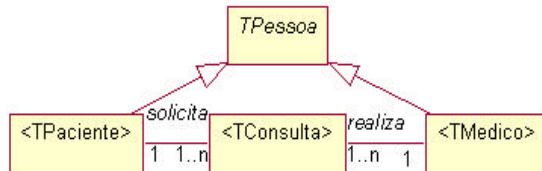


Figura 16: Modelo de Tipos

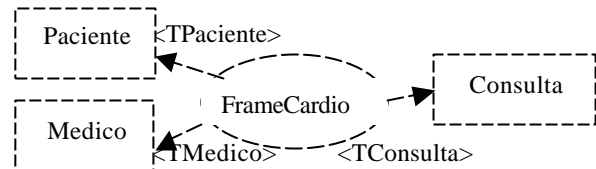


Figura 17: Modelo de Tipos da Aplicação

Em seguida, conforme o segundo nível de *Catalysis*, refinam-se os modelos do primeiro nível, especificando-se os componentes da aplicação. A Figura 17 mostra o Modelo da Aplicação, obtido do refinamento dos casos de uso, com os principais tipos importados do FrameCardio. No caso, Paciente, Medico e Consulta, importam do FrameCardio, os tipos <TPaciente>, <TMedico>, <TConsulta>, respectivamente.

Finalmente, no terceiro nível de *Catalysis*, o Engenheiro de Software especifica os Projetos Internos dos Componentes, dando ênfase às suas implementações e distribuições físicas. A Figura 18 mostra o Diagrama da Classe do componente Consulta, obtido do refinamento do modelo de tipos, com sua interface. Na interface IConsulta estão os protótipos dos métodos, implementados no componente TConsulta. No caso têm-se os métodos validarConsulta, deletarConsulta, inserirConsulta e selecionarConsulta.

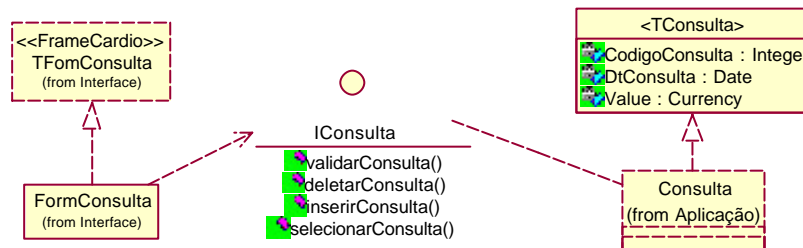


Figura 18 – Diagrama de classe do componente Consulta

A Figura 19 mostra um Modelo de Componentes da aplicação, onde os componentes reutilizados do FrameCardio, são indicados pelos delimitadores “<” e “>”. Pode-se observar que apenas os componentes FormConsulta, Consulta e DSConsulta, são específicos da aplicação. Os demais são do FrameCardio ou do próprio Delphi.

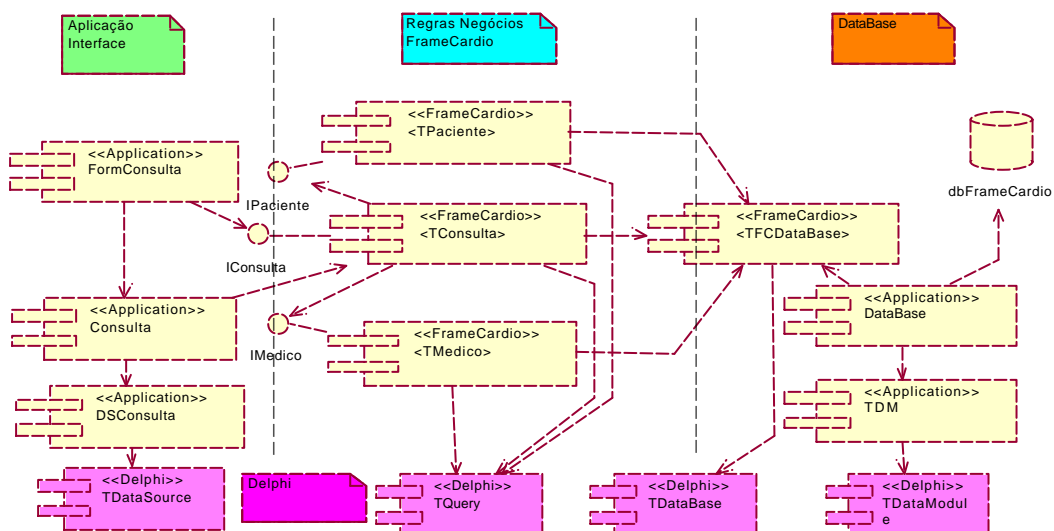


Figura 19: Modelo de Componentes da Aplicação.

4.2 Implementar Sistema

Neste passo, a aplicação é implementada em uma linguagem orientada a componentes. No caso, as especificações da aplicação, armazenadas em um arquivo MDL, são transformadas, pelo ST Draco-PUC, para a linguagem *ObjectPascal*. A Figura 20 mostra, à esquerda, parte de uma unitDM.pas, com o código *ObjectPascal*, do componente *Consulta* da aplicação e, à direita, o correspondente Diagrama que mostra a reutilização de componentes implementados do FrameCardio.

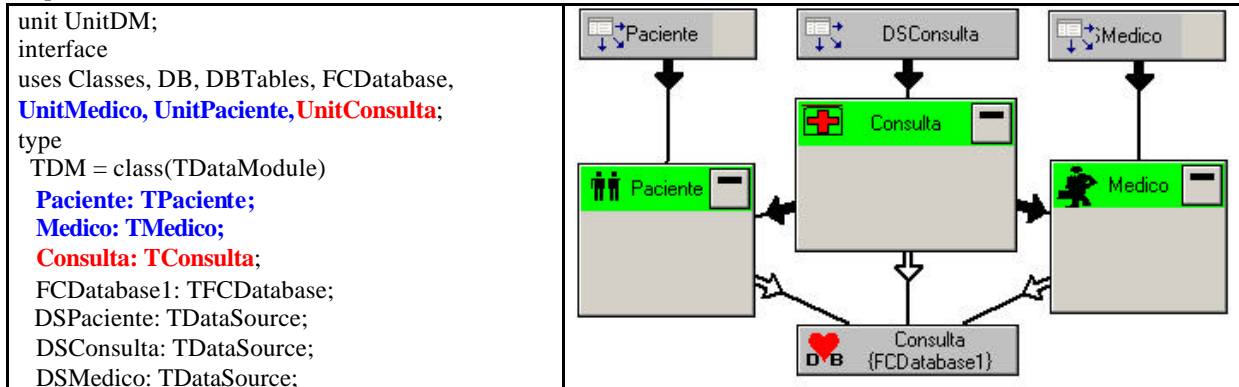


Figura 20: Código *ObjectPascal* gerado

4.3 Executar Sistema

Neste passo, o Engenheiro de software importa o código gerado, no Ambiente Delphi, para executá-lo. No Delphi, o código é reunido em um projeto da aplicação para facilitar seu gerenciamento. A implementação em *ObjectPascal*, gerada no passo Implementar Sistema pode não ser suficiente para atender todos os requisitos, principalmente os não funcionais, relacionados com a interface, segurança, validação de dados e acesso a banco de dados. Assim, o Engenheiro de software, utilizando os recursos visuais do Delphi, pode complementar o projeto com outros componentes que implementam estes requisitos. O código gerado pelo Sistema de Transformação Draco-PUC, integrado ao código desenvolvido no Delphi, resulta na implementação de todo o sistema.

Estando todo o sistema implementado, pode-se finalmente executá-lo para verificar se o mesmo atende aos requisitos especificados. Caso ocorram problemas, ou surjam novos requisitos, pode-se retornar aos passos anteriores para correções ou adições de novos requisitos e, novamente, reimplementar o sistema.

A Figura 21 mostra, à esquerda, um componente de interface gerado e, à direita, o método do FrameCardio *FCValidarConsulta*, sendo acessado pela Interface *IConsulta* do Componente *Consulta*.

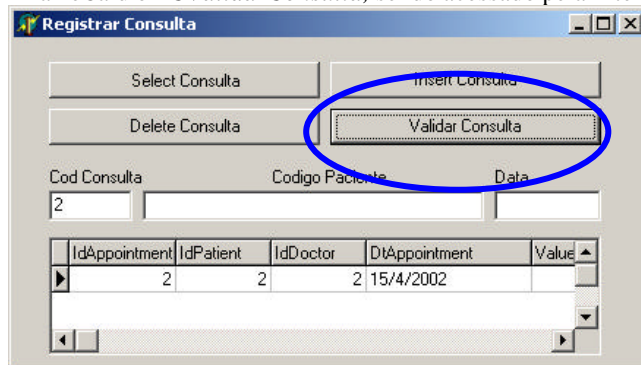


Figura 21: Execução do Sistema

5 CONCLUSÃO

Este artigo apresentou um PDS baseado em componentes que integra diferentes tecnologias, destacando-se as do método Catalysis e de um Sistema de Transformação de software, para desenvolver um *framework* de componentes e suas aplicações.

Na ferramenta CASE, reutilizando componentes do FrameCardio, obtém-se o Projeto da Aplicação Orientado a Componentes. As descrições das especificações em MDL, que representam o projeto, são utilizadas para gerar código do sistema em uma linguagem orientada a componentes. O Sistema de Transformação Draco-PUC automatizou grande parte da geração do código *ObjectPascal*, do *framework* e suas aplicações.

Dada a capacidade do Sistema de Transformação Draco-PUC, de suportar diferentes domínios de modelagem e de aplicação, outras linguagens, diferentes de *Catalysis* e *ObjectPascal*, podem ser utilizadas no PDS proposto.

O PDS proposto dá mais um passo na automatização de grande parte das tarefas do Engenheiro de Software, o que pode contribuir na redução do tempo e custos do desenvolvimento de uma aplicação do Domínio de Cardiologia.

6 REFERÊNCIAS

- [1] D'SOUZA, D.; WILLS, A. **Objects, Components and Frameworks with UML – The Catalysis Approach**. USA: Addison Wesley, 1998.
- [2] BORLAND/INPRISE. *Object Pascal Reference*.
- [3] LEITE, JCS., FREITAS, F.G., SANTANNA M. Draco-PUC Machine: A Technology Assembly for Domain Oriented Software Development. *3rd International Conference of Software Reuse*. IEEE Computer Society Press. In proceedings, pp. 94-100. Rio de Janeiro. 1994.
- [4] NEIGHBORS, J.M. The Draco approach to Constructing Software from Reusable Components. *IEEE Transactions on Software Engineering*. v.se-10, n.5, pp.564-574, September, 1984.
- [5] RATIONAL SOFTWARE CORPORATION.,
<http://www.rational.com/products/rose/prodinfo/index.jtmpl>.
- [6] COLEMAN, D. et al. **Object-Oriented Development – The Fusion Method** Prentice Hall, 1994.
- [7] FOWLER, M. **UML Distilled. Applying the Standard Object Modeling Language**. England: Addison Wesley, 1997.
- [8] BOOCH, G. et al. **The Unified Modeling Language – User Guide**. USA: Addison Wesley, 1999.
- [9] FUKUDA, Ana Paula. **Refinamento Automático de Sistemas Orientados a Objetos Distribuídos**. Dissertação de Mestrado. UFSCar, 2000
- [10] PRESSMAN, R. S. **Engenharia de Software**. Makron Books: São Paulo, 1995.
- [11] BORLAND/INPRISE. *Programming with Delphi 2001*.
- [12] BORLAND/INPRISE. *Visual Component Library Reference*.