JOSÉ ANDRÉ GOMES JÚNIOR

MODELAGEM DE UM COMPONENTE DE SOFTWARE DISTRIBUÍDO:
COMPONENTE BASE

Trabalho de conclusão de curso apresentado ao Curso de Ciência da Computação.

UNIVERSIDADE PRESIDENTE ANTÔNIO CARLOS

Orientador: Prof. Ms. Frederico de Miranda Coelho

BARBACENA
2004

JOSÉ ANDRÉ GOMES JÚNIOR

MODELAGEM DE UM COMPONENTE DE SOFTWARE DISTRIBUÍDO: COMPONENTE BASE

Este trabalho de conclusão de curso foi julgado adequado à obtenção do grau de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação da Universidade Presidente Antônio Carlos.

Barbacena – MG, 22 de Junho de 2004.

_____

Prof. Ms. Frederico de Miranda Coelho - Orientador do Trabalho

_____

Prof. Ms. Elio Lovisi - Membro da Banca Examinadora

_____

Prof. Eduardo de Macedo Bhering - Membro da Banca Examinadora

AGRADECIMENTOS

A Deus pela força de vencer.

À minha mãe por todo incentivo e “puxões” de orelha que me fizeram ser um homem de caráter.

À minha irmã pelo carinho.

Ao meu pai, que mesmo não estando presente, sempre pude sentir seu afeto.

A todos os professores pela boa vontade em compartilhar os seus conhecimentos.

Ao orientador Frederico Coelho de Miranda pela disponibilidade, atenção e boa vontade desprendidas ao longo deste trabalho.

Aos verdadeiros amigos que me apoiaram em momentos difíceis, quando eu quase desisti.

À galera do “Buzu” pelas divertidas viagens ao longo desses 4 anos de faculdade.

A todos aqueles que me desejaram boa sorte nesta longa caminhada.

RESUMO

De certa forma pode-se considerar que a tão falada globalização também atingiu a informática, uma vez que a cada dia torna-se mais necessário integrar os sistemas computacionais existentes. Principalmente no âmbito empresarial, onde as empresas necessitam de estar mais próximas de seus clientes, parceiros, fornecedores, podendo assim ganhar mais tempo em suas negociações e conseqüentemente economizar dinheiro. Dessa forma, a Computação Distribuída aliada ao desenvolvimento baseado em Componentes de Software passam a ocupar um lugar de destaque no cenário da Informática. Uma vez que através da utilização desses dois paradigmas a comunicação entre Sistemas de Softwares pode acontecer de forma simplificada e mais transparente possível, mesmos estes sendo desenvolvidos em linguagens de programação distintas.

Sendo assim busca-se neste trabalho o estudo das tecnologias citadas anteriormente, através da modelagem de um Componente de Software com a aplicação do Padrão de Objetos Distribuídos CORBA. De forma que se possa verificar onde o esse padrão deve ser aplicado no Componente para que ele se torne distribuído.

Palavras-chave: Computação Distribuída, Componente de Software, Objetos Distribuídos, Corba

SUMÁRIO

FIGURAS

1 INTRODUÇÃO

A grande tendência da Tecnologia da Informação nos tempos atuais é a integração dos sistemas computacionais, independente de plataforma e linguagem de programação, onde é possível que Componentes localizados em computadores ligados através de uma rede de computadores se comuniquem e coordenem suas operações através de troca de mensagens, ou seja, a Computação Distribuída [Mari02]. Paralelo a essa tendência, a utilização de Componentes de softwares no desenvolvimento de sistemas também está em evidência, visto que através desses Componentes é possível reutilizar código já pronto de forma consistente, economizando assim tempo e dinheiro no desenvolvimento de softwares de qualidade. Com isso essas duas tecnologias ganham cada vez mais espaço dentro dos sistemas de informação, portanto busca-se através deste projeto um maior conhecimento dessas duas áreas.

1.1 CONTEXTO

De certa forma é possível enxergar os sistemas distribuídos como uma evolução do modelo cliente/servidor, uma vez que dentro de uma arquitetura distribuída, os componentes terão inteligência para operar através de sistemas operacionais heterogêneos. A partir daí, o conceito de objetos distribuídos está revolucionando a arquitetura, desenvolvimento, empacotamento, distribuição e manutenção de softwares. Isso é possível porque um Componente de Software Distribuído possui interfaces bem definidas, permitindo que o objeto seja acessado por outros objetos de forma bem transparente, ou seja, o invocador desconhece o local que está localizado o objeto invocado e o sistema operacional ao qual está sendo executado, podendo assim ambos estar na mesma máquina ou em máquinas diferentes [Mari02].

Para poder alcançar o objetivo dos sistemas distribuídos existe vários modelos de padrões de desenvolvimento de componentes distribuídos, dentre os quais se destacam o DCOM (*Distributed Component Object Model*) e CORBA (*Common Object Request Broker Architecture*), idealizados pela Microsoft e OMG (*Object Management Group*), respectivamente [Mari02].

Já o desenvolvimento baseado em Componentes de Software foi impulsionado pela Orientação a Objeto, um paradigma que trouxe grande avanço para a Engenharia de Softwares, que dentre outras vantagens, permite que os desenvolvedores reutilizem código de sistema já implementados, para compor os novos softwares. Porém, a Orientação a Objetos pura não permite que partes significativas do sistema sejam reutilizadas [Tava03].

A idéia de Componentes de softwares parte do princípio que unidades de softwares, baseadas em um determinado domínio, possam se integrar de forma a constituir um sistema de software, sem que essas unidades se preocupem com os detalhes de implementação de ambas as partes [Nirv01].

Para facilitar o desenvolvimento dos Componentes de Softwares é utilizado, também, com grande freqüência, os Padrões de Projeto, que são procedimentos que descrevem

soluções para determinados problemas que de forma repetitiva. Dessa forma consegue-se uma melhor estruturação dos Componentes de Softwares, tornando-os mais reutilizáveis [Roch03].

Devido a essas tecnologias vários trabalhos forma desenvolvidos dentro da comunidade da Engenharia de Software, sempre objetivando alcançar o desenvolvimento de Sistemas de Softwares de forma mais rápida e eficiente.

Tendo como mesmo objetivo, será apresentada neste trabalho a modelagem de um Componente de Software com o intuito de verificar como os Padrões de Projeto podem ser aplicados a ele, tornando-o mais reutilizável possível e também fazendo uso da tecnologia da computação distribuída, para que esse Componente ao ser implementado possa se interoperar com outras aplicações, mesmo que estas estejam desenvolvidas em linguagens de programação diferentes.

1.2 PROPOSTA

O Trabalho consiste em modelar um Componente de Software de uma Arquitetura Estruturada em Componentes, e submetê-lo a um processo de reengenharia onde o mesmo será remodelado após ser aplicado os Padrões de Projeto, condizentes ao seu domínio, em sua estrutura. Também será mostrado onde deverá ser aplicada, neste Componente, a tecnologia de objetos distribuídos baseada no Padrão CORBA, de maneira que ao ser implementado ele se torne um Componente de Software Distribuído, ou seja, possa se comunicar com outras aplicações independentemente de linguagens de programação, plataformas e hardware, desde que esta aplicação também seja desenvolvida baseada no Padrão CORBA.

O Componente a ser desenvolvido será o Componente Base, que compõe Arquitetura Estruturada em Componentes de um Framework voltado para a gestão de Recurso de Negócios [Coel02].

1.3 OBJETIVOS

Os principais objetivos almejados com este trabalho são:

Estudo das principais etapas propostas pela Engenharia de Software para o desenvolvimento de sistemas;

Estudo da Tecnologia de Componentes de Software;

Estudo da Tecnologia de Padrões de Projeto;

Aplicabilidade dos Padrões de Projeto em um Componente de Software;

Estudo da Interoperabilidade entre Sistemas de Software, através do Padrão de Objetos Distribuídos CORBA;

1.4 ORGANIZAÇÃO DA MONOGRAFIA

O Capítulo 1 apresenta uma pequena introdução das Tecnologias que impulsionaram o desenvolvimento deste trabalho;

O Capítulo 2 apresenta as definições das Tecnologias utilizadas no decorrer deste trabalho.

O Capítulo 3 apresenta todas as etapas desenvolvidas para a modelagem do Componente, e também a sua reestruturação através da aplicação dos Padrões de Projeto.

O Capítulo 4 apresenta a tecnologia necessária para que o Componente Base ao ser implementado possa se tornar um Componente Distribuído, bem como onde essa tecnologia deverá ser aplicada, no Componente Base, para que isso se torne possível.

Por fim, no Capítulo 5, são descritos as conclusões desse trabalho de conclusão de curso, suas contribuições e os planos de trabalhos futuros.

2 CONCEITOS INICIAIS

Antes de iniciar a modelagem do Componente Distribuídos, de acordo com o padrão CORBA, é necessário definir alguns conceitos que serão utilizados durante o discernimento deste trabalho e que são fundamentais para o seu entendimento.

2.1 COMPONENTES DE SOFTWARE

Pode-se adotar a seguinte definição para componentes de softwares: “uma parte não trivial de um sistema, praticamente independente e substituível, que preenche uma função clara no contexto de uma arquitetura bem definida” [Pres02].

É uma unidade de software independente em nível de aplicativo, desenvolvida para um propósito específico e não para um aplicativo, pois essa unidade, que é o componente, pode ser usada em outros sistemas, bastando para isso apenas adapta-lo ao sistema em questão [Nirv, 00].

A estratégia de desenvolvimento de software baseada em componentes de softwares é fundamentada nos paradigmas da Programação Orientada a Objetos, sendo que um componente é composto por várias classes e interfaces. Sendo assim pode-se dizer que o ideal é que se use os componentes de softwares conhecendo apenas suas interfaces, uma vez que dessa forma o sistema desenvolvido estaria totalmente independente de qualquer implementação particular de algum Componente de Software [Tava03].

É possível considerar que durante sua execução um componente se torna "vivo", visto que ele é parte do sistema como um todo que possui uma função bem definida dentro do contexto geral da aplicação. Em um componente de software os detalhes de implementação ficam ocultos nas interfaces, onde estas encapsulam de forma a isolar as funcionalidades do componente, ou seja, diferencia o que o sistema faz de como o sistema faz, com isso a forma pela qual o componente foi implementado fica oculta do usuário. É através desta interface que os componentes se conectam ao "mundo exterior", pois na interface que o conjunto de operações referente ao componente será ativado, quando um cliente fizer uma requisição. Portanto é de suma importância que a interface dos componentes seja bem definida, uma vez que somente através dela que o componente torna-se acessível às demais partes do sistema [Nirv00].

Segundo a literatura pode-se considerar que os componentes podem ser divididos em 2 tipos:

Componentes Lógicos: são unidades de composição que possuem um conjunto de interfaces bem definidas e dependências de contexto bem definidas.

Componentes Físicos: são aqueles que realmente possuem um código binário, ou seja, são eles que são capazes de executar as tarefas que fazem com que outros módulos do sistema possam utilizar os serviços por ele oferecidos. Esses componentes podem ser constituídos por um ou mais componentes lógicos.

Os componentes permitem que sistemas possam ser montados a partir de componentes pré-frabricados, simplesmente interligando-os, visto que um componente é auto contido, ou seja, independente de ambiente e aplicativo.

A seguir são apresentadas algumas vantagens dos componentes de softwares [Nirv00]:

Independente: o componente de software é uma unidade generalizada, dentro de um domínio, podendo ser utilizada em outros sistemas que abrangem o mesmo domínio.

Reutilizável: o componente de software possui um poder de reutilização bem maior que soluções específicas a um determinado problema, uma vez que podem ser utilizados em diversas aplicações do mesmo domínio.

Montagem: através da montagem e adaptação de vários componentes pode-se obter um sistema completo.

Fácil utilização: o componente pode ser atualizado individualmente sem comprometer o restante do sistema.

Localização Transparente: o componente pode estar em qualquer lugar da rede, ou seja, podem ser instalados em computadores mais adequados à necessidade do componente.

Distribuídos: o componente quando projetado de acordo com padrões distribuídos podem ser alocados em uma rede empresarial de forma a interagirem com diversas tecnologias diferentes.

2.1.1 INTERFACES

As interfaces podem ser consideradas como um dos principais pontos a serem considerados ao desenvolver um Componente de Software. Pois elas separam “o que o Componente faz” de “como o Componente faz”, ou seja, as interfaces disponibilizam os serviços oferecidos pelo Componente, de forma que a implementação desses serviços fique oculta do usuário. Sendo assim, os clientes ao requisitarem um serviço oferecido pelo Componente de Software não precisa se preocupar com a maneira que esse serviço será executado, simples-

mente é necessário conhecer as interfaces disponibilizadas pelo Componente de Software [Tava03].

2.2 REDES DE COMPUTADORES

As redes de computadores fazem com que trabalhos sejam realizados por um grande número de computadores autônomos separados, porém interconectados através de algum meio físico, sendo estes meios: fio de cobre, fibra ótica ou ar.

É possível classificar as redes de computadores conforme 2 critérios [Tane03]:

- Tecnologia de Transmissão
- Área de Abrangência

Dentro da Tecnologia de Transmissão podemos dividir as redes de computadores em:

Ponto-a-Ponto: são as redes onde entre cada par de máquinas existem linhas de comunicação, interconectando-as.

Multiponto: são as redes onde a comunicação entre as máquinas é feita através de um meio compartilhado.

Dentro da Área de Abrangência podem-se dividir as redes de computadores em:

Lan's: são redes consideradas como redes locais, ou seja, possuem tamanho restrito.

Man's: essas redes nada mais são do que uma versão maior das Lan's, ou seja, abrangem uma área de comunicação mais ampla.

Wan´s: são as redes consideradas de longa distância,ou seja, cobrem áreas geograficamente grandes, como continentes.

Um fato importante a ser observado é a confusão existente na literatura entre uma rede de computadores e um sistema distribuído. Sendo que a principal diferença entre eles é que em um sistema distribuídos os computadores que o compõe, aparecem para o usuário como um único computador, já nas redes de computadores isso não existe, ou seja, se as máquinas que constituem as redes tiverem hardwares diferentes e sistemas operacionais distintos isso será totalmente visível para o usuário [Tane03].

Na prática, um sistema distribuído é um sistema de software instalado em uma rede de computadores, o qual dará alto grau de coesão e transparência ao sistema, com isso o que diferencia um sistema distribuído de uma rede é o software e não o hardware. Deve-se levar em consideração também que em uma rede e em um sistema distribuído existe a manipulação de arquivos, e aí também existirá uma diferença, pois nos sistema distribuído quem manipula esses arquivos é o próprio sistema, já em uma rede isso será feito pelo usuário [Tane03].

2.3 COMPUTAÇÃO DISTRIBUÍDA

Uma das grandes características das redes de computadores corporativas, onde várias empresas consolidam suas aplicações, é a heterogeneidade, ou seja, essas grandes redes heterogêneas compartilham de vários sistemas operacionais, que rodam em múltiplos componentes de hardware e software [Nirv00].

Através da computação distribuída é possível que uma aplicação seja dividida em diferentes partes, na qual cada uma se comunicará com outra através de linhas de comunicação, sendo que cada parte poderá ser processada em um sistema independente. Dessa forma a computação distribuída faz transparecer para os usuários que toda a aplicação nada mais é do

que um único sistema, ao invés de um conjunto de componentes distintos e localizados em diferentes máquinas [Pime99].

Com isso a computação distribuída surge com o intuito de proporcionar uma infra-estrutura capaz de fazer com que os diversos componentes de software que constituem uma rede de computadores heterogênea possam se comunicar[Sand99].

Paralelo ao paradigma da computação distribuída surge o conceito da comunicação de componentes, desenvolvido a partir de tecnologias diferentes. Dentro deste conceito entra em cena a computação de objetos distribuídos (componentes), que permite que objetos possam ser distribuídos em uma rede de computadores heterogênea. Mesmo que esses objetos estejam distribuídos em computadores diferentes eles irão aparecer como parte de um todo [Nirv00].

2.3.1 OBJETOS DISTRIBUÍDOS

São componentes de softwares capazes de operar em sistemas operacionais heterogêneos, desenvolvidos em várias linguagens de programação. O que diferencia objetos distribuídos dos objetos clássicos é que os compiladores e linguagens de programação utilizadas para construí-los são totalmente transparentes à implementação desses, ou seja, eles possuem uma interface bem definida onde os compiladores geram um código a mais para serem acessados por outros objetos de forma totalmente transparente. Com isso o objeto que invoca o serviço desconhece o local do objeto invocado, bem como o sistema operacional que este objeto está sendo executado e a linguagem de programação na qual ele foi desenvolvido [Sand 99].

2.3.2 PADRÕES DE OBJETOS DISTRIBUÍDOS

São procedimentos para o desenvolvimento de componentes que permitem que esses componentes possam ser acessados remotamente através de chamadas a procedimentos, isto significa que um objeto distribuído pode ser usado como um objeto estático, porém estando ele em qualquer lugar da rede [Sand99].

Os padrões distribuídos permitem que aplicações desenvolvidas em linguagens de programação diferentes, sendo executadas em sistemas operacionais distintos, possam comunicar entre si, uma vez que a aplicação cliente contactará a aplicação servidora através das chamadas remotas a procedimentos, providas pelas interfaces dos componentes em questão[Roch03].

2.3.2.1 PADRÃO CORBA (*COMMOM OBJECT REQUEST BROKER ARCHITETURE*)

É um padrão proposto pela OMG (*Object Management Group*) para promover a intercomunicação entre objetos distribuídos. Com isso é possível que um programa baseado em CORBA desenvolvido em uma linguagem de programação possa interoperar com outro programa desenvolvido em uma linguagem de programação diferente [Andr01].

A OMG é uma organização internacional formada por mais de 800 membros, nesta organização estão incluídos empresas de sistema de informação, empresas de desenvolvimento de softwares, universidades e utilizadores do padrão CORBA. Até mesmo a Microsft que desenvolve um padrão concorrente, o DCOM (*Distributed Component Object Model*), participa desta organização [Grou03].

O CORBA é utilizado para desenvolver objetos que farão parte de uma rede de computadores distribuída e heterogênea. Uma rede é considerada distribuída porque os objetos podem estar localizados em diferentes computadores, e é considerada heterogênea porque esses computadores podem estar executando em diferentes sistemas operacionais e os objetos podem ser implementados em diferentes linguagens de programação [Nirv00].

O CORBA faz com que isso seja possível porque os objetos desenvolvidos neste padrão terão suas interfaces especificadas em uma linguagem de especificação de programação neutra, a IDL (*Interface Definition Linguage*). É a IDL que permite que um objeto desenvolvido em uma linguagem de programação possa ativar os métodos de outro objeto escrito em uma linguagem de programação. A IDL permite essa flexibilidade porque nela não serão implementados os métodos do objeto, estes serão apenas especificados, ou seja, para cada objeto servidor, uma interface IDL será especificada, na qual todas as constantes, tipos, exceções, atributos e métodos associados ao objeto serão listados [Nirv00]. Este padrão será estudado mais detalhadamente no capítulo 4, no qual serão mostrados suas principais funcionalidades e onde ele deverá ser aplicado no Componente, a ser modelado, para que ele se torne distribuído.

2.4 PADRÕES DE PROJETO

Pode-se considerar a seguinte definição para Padrões de Projeto: “Um padrão fornece uma solução para um problema básico em um determinado contexto” [Roch03].

Os Padrões de Projeto funcionam como uma ferramenta integrada à Engenharia de Software que permite desenvolver softwares com grande poder de reutilização [Tava03].

A função dos padrões de projeto é documentar os resultados e soluções encontradas para problemas que repetem com freqüência dentro de um mesmo domínio, visto que eles têm como base experimentos obtidos durante anos de prática profissional. Na verdade esses padrões resolvem problemas específicos de projetos orientados a objeto mais flexíveis, ou seja, reutilizáveis. Através deles é mais fácil reutilizar projetos e arquiteturas bem sucedidas, uma vez que ajudam a escolher alternativas de projeto que tornam o sistema mais reutilizável, podendo até mesmo melhorar a documentação e manutenção dos sistemas, pois fornecem uma especificação explícita das interações existentes nas classes e objetos que constituem o sistema [Gamm00].

Como os Padrões de Projeto são utilizados em diversos domínios, eles podem ser classificados em diversas categorias, com o intuito de facilitar o reuso. Porém essa classificação não é muito rigorosa, visto que os padrões podem estar em mais de uma categoria. Estas categorias são listadas a seguir [Coel02].

1. **Padrões de Processo:** esses padrões mostram soluções para a resolução de problemas encontrados nos processos executados durante a engenharia de software, como desenvolvimento, modelagem, etc.

2. **Padrões Arquiteturais:** esses padrões "expressam o esquema ou organização estrutural fundamental de sistemas de softwares e hardware".

3. **Padrões de Padrões:** esses padrões definem as regras para o formato no quais os padrões devem ser escritos.

4. **Padrões de Análise:** esses padrões especificam soluções para a resolução de problemas encontrados na análise de sistemas.

5. **Padrões de Projeto:** esses padrões descrevem soluções para a resolução de problemas encontrados durante o projeto de softwares.

6. **Padrões de Programação:** esses padrões especificam soluções para problemas relacionados à programação de linguagens de programação específicas.

Existe também a Linguagem de Padrões, que nada mais é do que o conjunto de Padrões de projetos utilizados em um contexto comum, ou seja, a Linguagem de Padrões engloba todos os possíveis aspectos importantes dentro domínio que a aplicação foi desenvolvida [Tava03].

2.5 FRAMEWORK

O Framework é uma tecnologia que engloba o desenvolvimento de um sistema de software reutilizável, com o intuito de solucionar problemas semelhantes existentes em aplicações que abrangem um mesmo domínio.

Como o Framework pode ter tanto seu código fonte como o projeto de sua arquitetura reutilizado, o desenvolvedor irá personalizá-lo de acordo com suas necessidades para que ele possa atender às particularidades da aplicação a ser desenvolvida. Com isso os Frameworks trazem alguns benefícios como: a modularidade, a reusabilidade, a extensibilidade, fornecendo assim alto índice de reusabilidade para aplicações de um domínio específico [Coel02].

Os Frameworks podem ser classificados de acordo com a forma pela qual eles podem ser estendidos para gerarem uma nova aplicação [Tava03]:

Caixa Branca: são os Frameworks que fazem uso de herança e ligação dinâmica para conseguir sua extensabilidade.

Caixa Preta: são os Frameworks estruturados usando a composição e a delegação dos objetos, ao invés da herança.

Caixa Cinza: são os Framewoks que unem as duas classificações acima.

Os Frameworks baseados em componentes, cujo esses componentes são caixa preta, será considerado um Framework caixa preta, uma vez que a reutilização independerá da implementação dos componentes de software [Tava03].

2.6 MODELAGEM UML (*UNIFIED MODELING LANGUAGE*)

A modelagem é uma das etapas mais importantes durante o desenvolvimento de um sistema de boa qualidade, pois ela permite que a estrutura completa do software e seu comportamento possam ser visualizados, com isso é possível obter uma melhor compreensão do sistema que está sendo construído [Booc99].

A UML é uma linguagem de modelagem a qual possibilita que os artefatos do sistema de software em construção possam ser especificados, visualizados, construídos e documentados [Melo02].

A UML será usada neste trabalho para criar diagramas de Classes e diagramas de Casos de Uso gerados durante a modelagem do componente a ser desenvolvido.

2.7 JAVA

É uma linguagem de programação totalmente orientada a objetos que possui uma grande portabilidade entre as variadas plataformas e sistemas operacionais existentes. Essa portabilidade é alcançada pois esta linguagem não é compilada e sim interpretada, ou seja, o compilador gera o *byte-code*, que é um código independente de plataforma. Este código ao ser executado será interpretado pela máquina virtual instalada na máquina que se deseja executar o programa desenvolvido. Para portar o código gerado por outra plataforma basta instalar a máquina virtual correspondente a essa plataforma. O Java além de ser integrada à internet, é também uma excelente linguagem para desenvolvimento de aplicações em geral [Marc98].

O Java será usado neste trabalho para, caso seja necessário, exemplificar algum código de programação referente ao Componente Modelado.

2.8 RATIONAL ROSE

Ferramenta que permite que os diagramas, como: Diagramas de Classes, Diagramas de Casos de Uso, Diagramas de Estado e Diagramas de Seqüência, que serão gerados na Análise e Projeto dos componentes a serem desenvolvidos possam ser criados. Estes diagramas serão gerados de acordo com a Linguagem de Modelagem UML (seção 2.6), com o intuito de possibilitar um melhor entendimento da estrutura dos componentes [Carl99].

Neste trabalho o Rational Rose será utilizado para criar os diagramas de Casos de Uso, referentes ao Componente Modelado.

2.9 VISIO 2003

O Visio 2003 é um programa de diagramação utilizado para desenhar diagramas comerciais e técnicos que documentam e organizam as etapas de desenvolvimento de um projeto. Os diagramas criados no Visio 2003 permitem que as informações referentes ao projeto em desenvolvimento sejam visualizadas com clareza, consistência e eficiência de forma que texto e números não conseguem fazer sozinhos [Micr03].

O Visio 2003 será utilizado neste trabalho para desenhar os diagramas de Classes e o DER (Diagrama de Entidade e Relacionamento), gerados durante as etapas de modelagem do Componente.

2.10 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentadas as tecnologias utilizadas durante o desenvolvimento deste trabalho, bem como as ferramentas necessárias para que as etapas de modelagem do Componente pudessem ser realizadas.

3 DESENVOLVIMENTO DO COMPONENTE

Antes de começar a implementação de um software é preciso definir o que realmente se pretende desenvolver, analisar as possíveis soluções, projetar esse software, modelá-lo em diferentes níveis de abstração para que a estrutura e os riscos do projeto possam ser observados e discutidos detalhadamente, para então começar implementá-lo. Por isso se faz uso da Engenharia de Software, para que o processo de desenvolvimento de Software possa transcorrer de maneira consciente e segura, uma vez que através da Engenharia de Software é possível seguir etapas de forma a se evitar complicações futuras.

3.1 MÉTODO DE DESENVOLVIMENTO DO COMPONENTE

Para desenvolver um sistema de software para uma empresa qualquer, a maior parte das informações e requisitos necessários à sua construção são obtidos através de entrevistas realizadas com os funcionários da empresa em questão. Porém, quando se pretende de-

senvolver um Componente de Software que pertence a uma determinada Arquitetura já existente, o levantamento das informações, requisitos e funcionalidades do Componente a ser desenvolvido não é tão simples, uma vez que o Analista nem sempre terá acesso a quem idealizou o componente. Outra questão que deve ser levada em conta é que se o Componente for desenvolvido exatamente da forma como o mesmo se encontra na Arquitetura, será quase impossível que esse componente possa ser reutilizado isoladamente como é pretendido neste trabalho, visto que algumas de suas funcionalidades dependem de outros Componentes da Arquitetura para que determinadas tarefas possam ser executadas. Com o intuito de resolver esse problema algumas etapas, definidas conforme a Figura 3.1, serão seguidas para desenvolver o Componente, sendo que algumas destas etapas estão descritas em [Tava03].

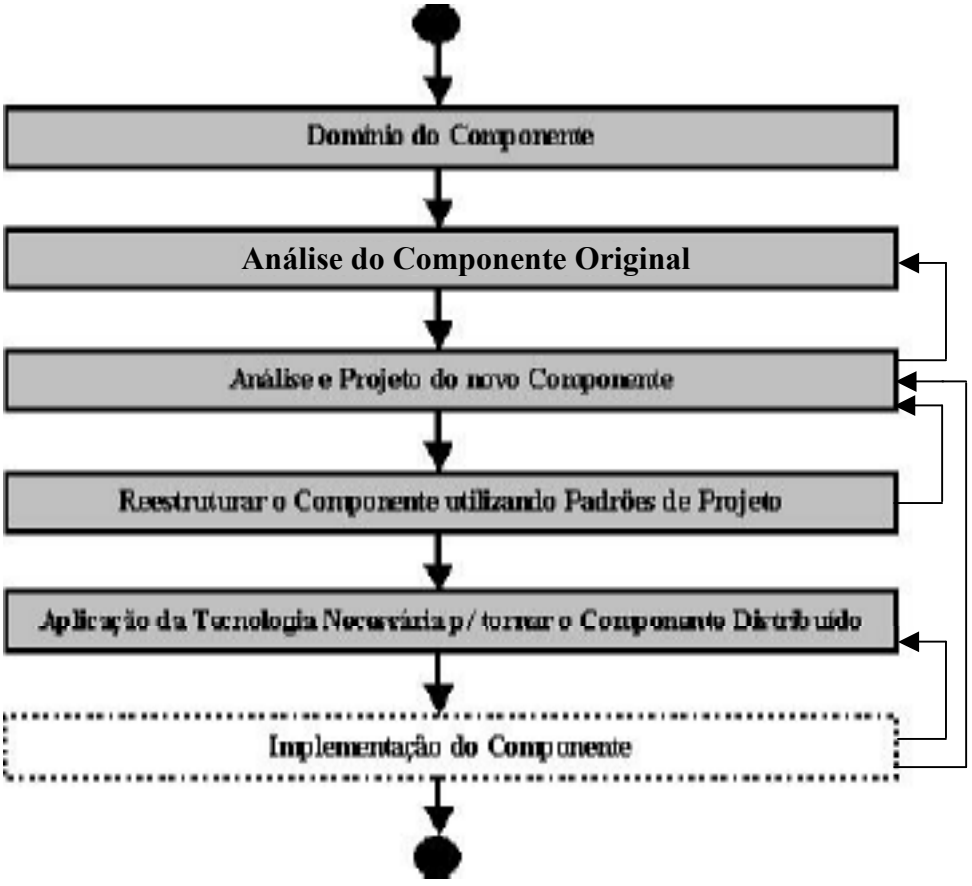


Figura 3.1 - Etapas de Desenvolvimento do Componente

A seguir serão descritas sucintamente as etapas descritas acima:

1ª Etapa: Domínio do Componente – Nesta etapa o Analista de Sistemas deve se preocupar em compreender bem o Domínio a qual pertence o Componente a ser desenvolvido.

2ª Etapa: Análise do Componente Original – Esta etapa é uma continuidade da descoberta do domínio do Componente, porém, em um nível de abstração inferior, preocupando-se em obter as características dos Componentes que pertencem ao Domínio em questão.

3ª Etapa: Análise e Projeto do Novo Componente – Esta etapa preocupa-se em, após compreensão do Domínio e da Arquitetura do Componente, iniciar a Análise e Projeto do novo Componente, ou seja, personalisando-o de acordo com as funcionalidades específicas do problema a ser resolvido.

4ª Etapa: Reestruturar o Componente Utilizando Padrões de Projeto – Com o Componente totalmente Projetado é possível identificar alguns Padrões de Projeto que se aplicam à sua estrutura e, se for do interesse do desenvolvedor, aplicar estes Padrões de Projeto e refazer o Projeto do Componente.

5ª Etapa: Aplicação da Tecnologia necessária para tornar o Componente Distribuído – Nesta etapa será mostrada qual a Tecnologia usada, no projeto, para tornar o Componente Base distribuído, ou seja, Tecnologia esta que permitirá ao componente base comunicar-se com outras aplicações, mesmo que elas tenham sido desenvolvidas em linguagens de programação diferentes.

6ª Etapa: Implementação do Componente - Esta é a etapa tão desejada, onde se inicia a construção física do Componente. Portanto, antes de sair programando é aconselhável uma revisão em todo o processo de Análise e Projeto do Componente, pois, alguma falha nessas etapas pode acarretar em sérios problemas durante a Implementação.

Esta etapa está representada com linha pontilhada pois, mesmo sendo de total interesse executá-la, ela ficará como sugestão para trabalhos futuros, pois não houve tempo hábil para que esta etapa fosse executada por completo.

3.2 DOMÍNIO DO COMPONENTE

O componente a ser desenvolvido nesse trabalho, Componente Base, foi escolhido de uma Arquitetura Estruturada em Componentes definida em [Coel02] construída a partir de uma reengenharia da Arquitetura do Framework GREN (Gestão de Recurso de Negócio) descrito em [Brag01], cuja documentação foi essencial para a criação de tal Arquitetura. Portanto é de suma importância que se tenha conhecimento do domínio do Framework GREN, descrito detalhadamente em [Tava03] e [Coel02], para que se possa compreender as características do componente escolhido e onde ele poderá ser aplicado.

Nesta etapa foi possível entender detalhadamente o domínio o qual pertence o Componente Base, ou seja, em que área de atuação se encontra os possíveis problemas que ele pode solucionar

3.3 ANÁLISE DO COMPONENTE ORIGINAL

Nesta seção dar-se-á início à análise do Componente original Base e projeto do novo Componente Base, na qual será baseado em alguns padrões de projetos, tendo as mesmas funcionalidades do Componente original. Porém será desenvolvido de acordo com o padrão de Objetos Distribuídos CORBA, visto que a intenção desse trabalho é remodelar o Componente Base de forma que ele deixe de ser um componente estático e torne-se um componente distribuído, onde o mesmo poderá ser acessado através de uma rede de computadores por outros componentes ou aplicações, que dependam dele para executar alguma tarefa, mesmo que estas tenham sido desenvolvidas em linguagens de programações diferentes.

3.3.1 ANÁLISE DA ARQUITETURA DO DOMÍNIO

A Arquitetura Estruturada em Componentes do Framework GREN (Gestão de Recursos de Negócio) descrita em [Coel02] é a fonte do Componente a ser modelado neste trabalho, sendo necessário o entendimento dessa arquitetura para que se possa compreender o domínio no qual o componente pertence.

A Arquitetura Estruturada em Componentes foi definida segundo o processo de reengenharia definido por [Coel02], tendo em mente que o projeto da Arquitetura Estruturada em Componentes seria projetada a partir da análise de uma Linguagem de Padrões (Linguagem de Padrões GRN). Os passos propostos por coelho [Coel02] para se obter essa Arquitetura são:

- Especificação dos Componentes da Nova Arquitetura e suas Interfaces;
- Refinamento dos Componentes;
- Acoplamento dos Componentes;
- Refinamento da Arquitetura Estruturada em Componentes;

Sendo que a Linguagem de Padrões GRN foi a principal fonte de recursos usada para conclusão desse processo de reengenharia [Tava03]. A Relação entre os componentes constituintes da Arquitetura Estruturada em Componentes e os padrões da Linguagem de Padrões GRN podem ser verificados em Coelho [Coel02].

A Arquitetura Estruturada em Componentes no domínio de gestão de recursos de negócios proposta por Coelho[Coel02], bem como a definição de cada uma de suas camadas pode ser visualizada em Tavares[Tava03].

A distribuição dos componentes que constituem a Arquitetura Estruturada em Componentes pode ser visualizada em Tavares [Tava03].

3.3.2 ANÁLISE DA ESTRUTURA DO COMPONENTE

A estrutura do Componente original será obtida a partir de um estudo da documentação da Arquitetura Estruturada em Componentes e dos Casos de Uso definidos em [Coel02] e da Linguagem de Padrões GRN [Brag01].

A especificação do Componente original Base é fornecer a capacidade de gravar as informações sobre os recursos, clientes, empresas e fornecedores que podem eventualmente participar de uma transação, bem como cadastrar, deletar, localizar ou alterar algum recurso, cliente, empresa ou fornecedor. Através deste componente é possível também verificar os detalhes de cada um dos itens cadastrados, sendo que esses detalhes correspondem a todos os campos existentes na tabela em questão com seus respectivos valores. O componente base permite que se faça o levantamento de todos os recursos cadastrados e também gerencie uma transação a ser efetuada, através da sua inclusão, exclusão, localização do participante da transação ou visualização dos detalhes dos participantes da transação, sendo que o gerenciamento de transação estende à transação envolvendo fornecedores [Coel02].

Para que seja possível gravar as informações relativas às Pessoas Físicas e Empresas, é preciso que se faça o cadastro do endereço das mesmas e que este esteja disponível para o Componente Base. Além do gerenciamento de Pessoas Físicas e Empresas, é necessário que o Recurso a ser cadastrado seja classificado e quantificado. Essas funcionalidades não são provenientes do Componente Base e sim da combinação de outros componentes,

O Componente Base é responsável exclusivamente por gravar as informações que futuramente farão parte de uma transação, sendo que alguns pré-requisitos para essas tarefas são de responsabilidades de outros componentes, portanto, se o Componente Base for distribuído isoladamente ele não terá condições de realizar suas funções, pois necessitaria de informações e operações que são de responsabilidades de outros componentes.

A figura 3.2 mostra as interfaces providas do Componente original Base obtida em [Coel02]:

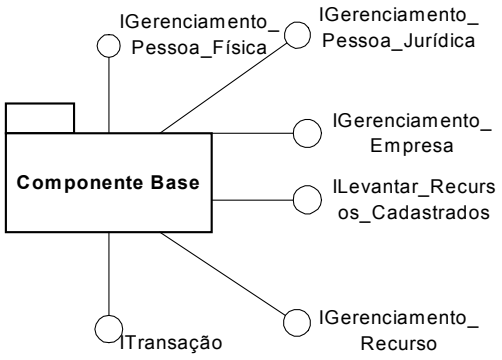


Figura 3.2 - Componente Base Original

Analisando a sua estrutura é possível observar que algumas de suas funcionalidades são obtidas implicitamente do seu relacionamento com outros componentes, conforme descrito em [Coel02]. Por exemplo, o endereço das Pessoas Físicas e Empresas cadastradas no sistema são controlados pelo Componente Endereço. A classificação de um recurso, ou seja, a sua categorização de acordo com o interesse da organização é realizada pelo Componente Classificação. A quantificação de um recurso com relação ao mesmo ser um recurso simples, mensurável, instanciável ou em lotes é provida por interfaces obtidas através do componente Quantificação de Recurso.

3.4 ANÁLISE E PROJETO DO NOVO COMPONENTE

Como é visível na figura 3.3 o Componente Base possui alguma interfaces que dependem de outras interfaces providas pelos componentes Endereço, Classificação e Quantificação do Recurso, pois para gravar as informações a respeito de Clientes_Físicos e Clientes_Jurídicos é necessário que o endereço das mesmas esteja cadastrado, e para gravar as informações a respeito de um recurso é necessário que o mesmo seja obrigatoriamente quantificado e opcionalmente classificado.

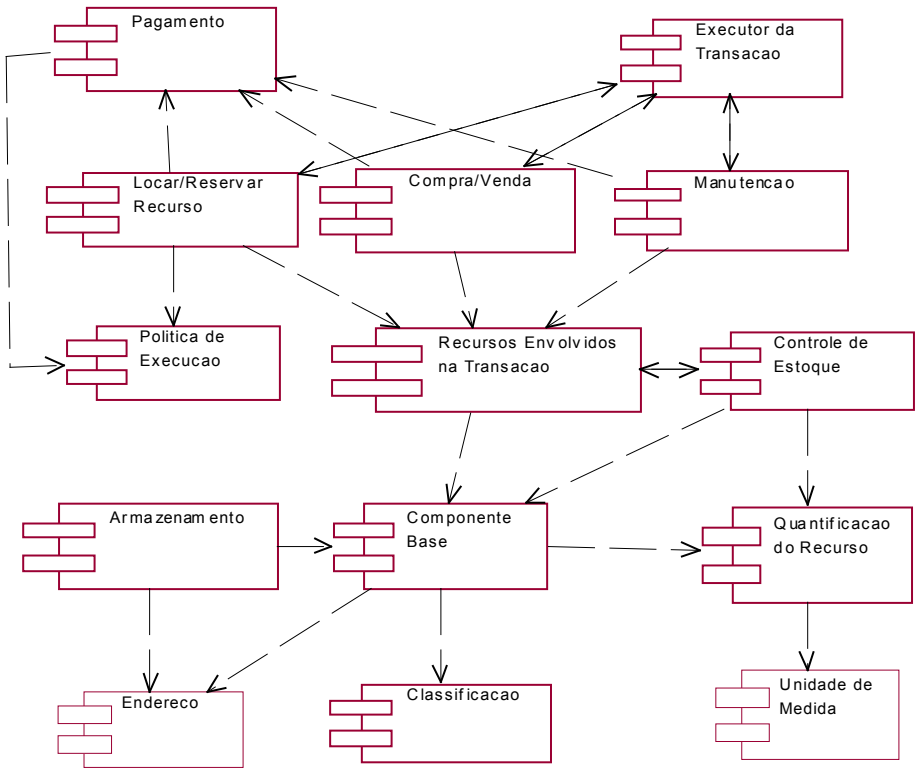


Figura 3.3 - Arquitetura Estruturada em Componentes

Como o objetivo deste trabalho é a modelagem de apenas um componente, tornando-o distribuído para fazer uso da aplicação de sistemas distribuídos em um framework, e devido a este componente depender de outros componentes (figura 3.2), algumas interfaces serão alteradas com o intuito de facilitar o seu entendimento, seu desenvolvimento e para que seja possível desenvolver um sistema que faça uso desse componente, eliminando-se a necessidade da criação de outros componentes, da Arquitetura GREN.

É importante deixar bem claro, que a alteração das interfaces se faz necessário apenas para tornar o componente independente dos outros, mas se o objetivo fosse o desenvolvimento de todos os componentes que compõe a Arquitetura Estruturada em Componentes [Coel02], as alterações que serão feitas não seriam necessárias. A figura 3.4 mostra o novo Componente Base com suas interfaces alteradas. As operações de cada uma das interfaces do Componente encontram-se no Anexo A.

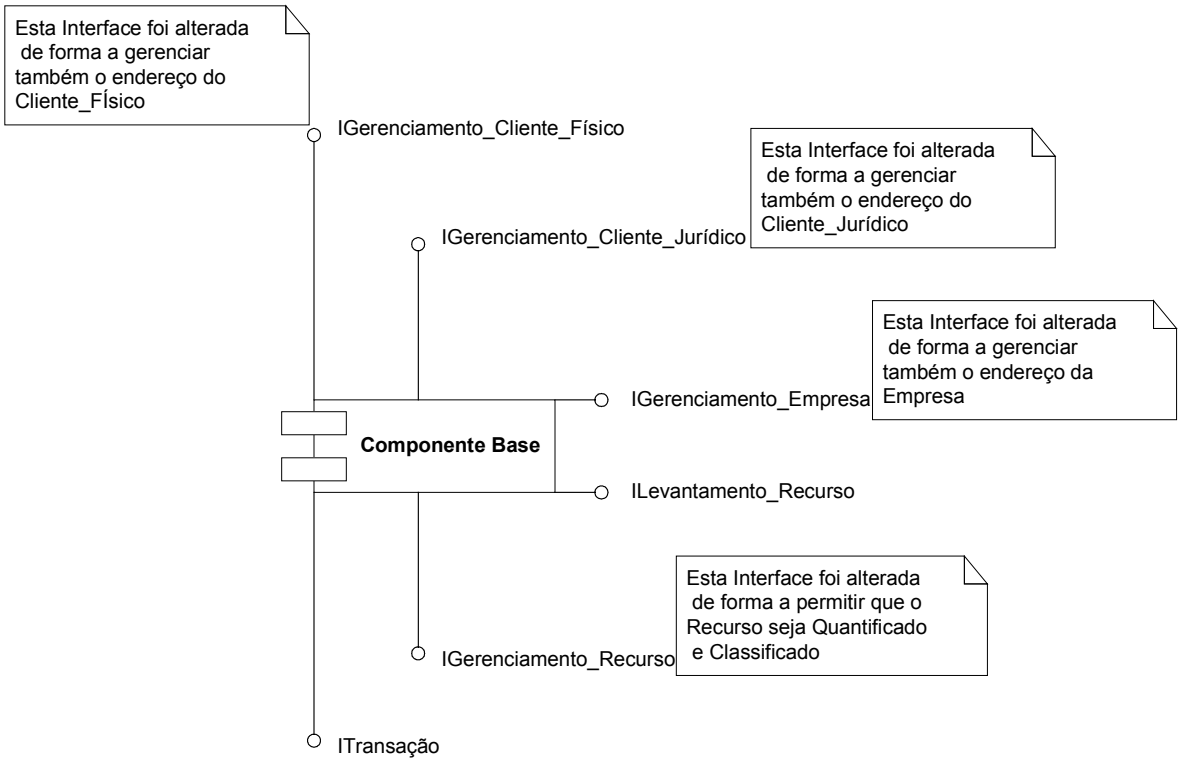


Figura 3.4 - Componente Base Remodelado

As interfaces e os motivos de suas presenças no componente serão descritos a seguir:

Interface IGerenciamento_Cliente_Físico

A interface IGerenciamento_Cliente_Físico trata das questões relativas ao gerenciamento de um novo cliente, representado por uma pessoa física. Ela fornece métodos para que se possam controlar todas as informações referentes ao cadastramento, exclusão, edição e busca de dados dos clientes físicos, envolvidos em uma transação. Com o novo propósito do componente, torna-se necessário a alteração dessa interface para que ela gerencie também o endereço dos Clientes_Físicos cadastrados.

Interface IGerenciamento_Cliente_Jurídico

Idem ao anterior, só que agora tratando de Cliente_Jurídico.

Interface IGerenciamento_Empresa

A interface IGerenciamento_Empresa trata das questões relativas ao gerenciamento de empresas que possam vir a participar de alguma transação. Ela fornece métodos para que se possam controlar todas as informações referentes ao cadastramento, exclusão, edição e busca de dados de empresas. Com o novo propósito do componente, torna-se necessário a alteração dessa interface para que ela gerencie também o endereço das Empresas cadastradas.

Interface IGerenciamento_Recurso

A interface IGerenciamento_Recurso trata das questões relativas ao gerenciamento do recurso que poderá ser envolvido em uma transação. Ela fornece métodos para que se possa controlar todas as informações referentes ao cadastramento, exclusão, edição, detalhes e busca dos dados relativos ao recurso em questão. Com o novo propósito do componente, torna-se necessário a alteração dessa interface para que ela gerencie também a classificação e Quantificação do recurso a ser cadastrado.

Interface Ilevantamento_Recurso

A interface Ilevantamento_Recurso permite que seja feito um levantamento de recursos já cadastrados, permitindo também que seja feita a edição do recurso em questão.

Na figura 3.5 é apresentado o diagrama de classes do componente modelado para implementar as Interfaces descritas anteriormente. Esse diagrama fornece uma visão estática da estrutura do Componente e permite visualizar quais as classes que realizam determinadas Interfaces.

As classes Cliente_Físico, Cliente_Jurídica e Empresa, figura 3.5, possuem um conjunto de métodos para o cadastro, consulta, exclusão e alteração dos dados, gerenciados pela Aplicação do Cliente, referente aos clientes físicos, aos clientes jurídicos e às empresas que eventualmente estarão envolvidos em uma transação.

A classe Recurso, figura 3.5, é responsável pelo controle dos recursos da empresa e fornece além de métodos básicos (cadastrar, consultar, alterar, excluir) alguns métodos que permitem a consulta dos recursos classificados em uma categoria específica.

A classe Transação, figura 3.5, é responsável por controlar as informações referentes aos recursos, e participantes envolvidos em uma transação. Fornecendo métodos básicos para controle dessas informações.

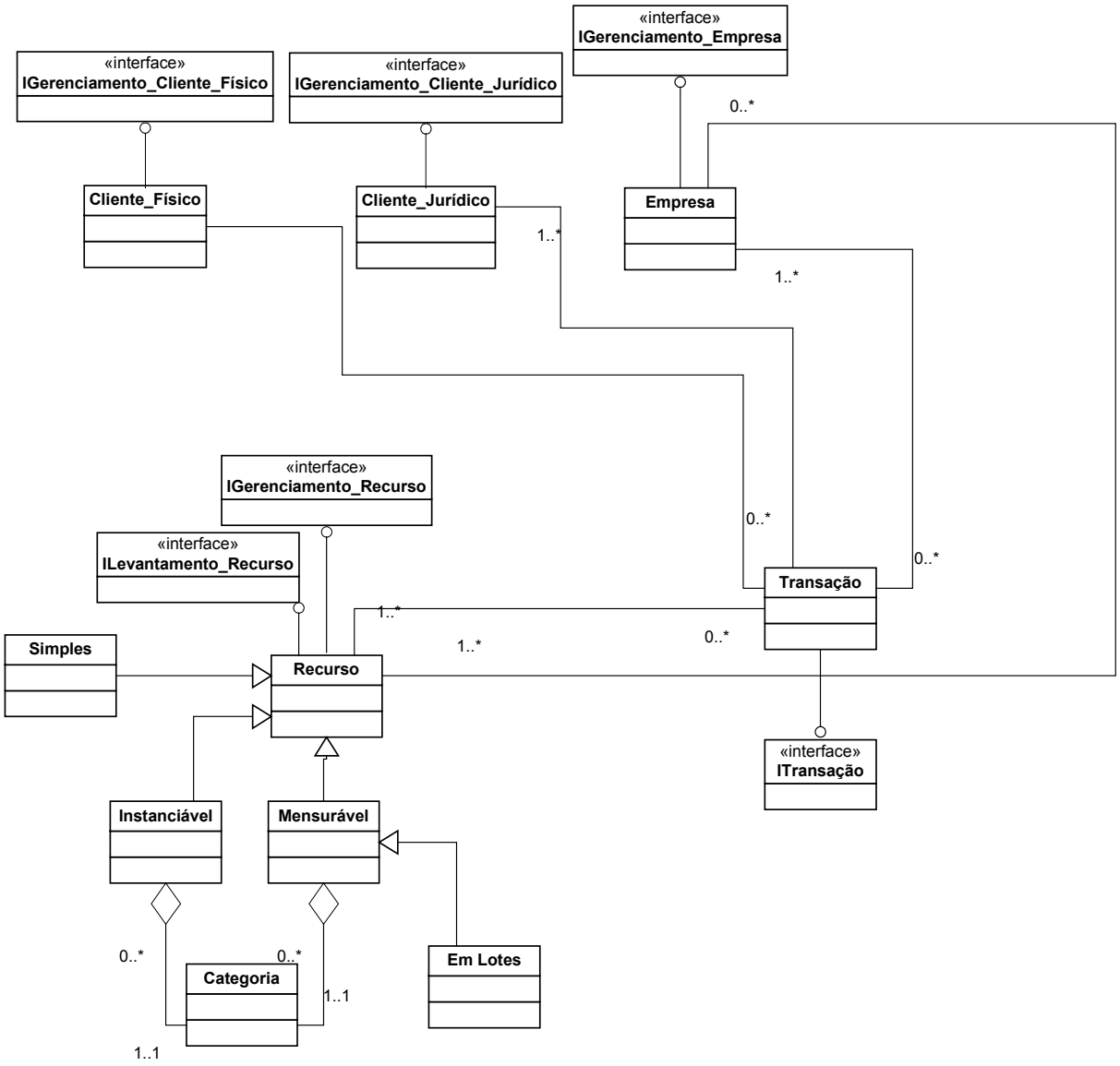


Figura 3.5 - Diagrama de Classes do Componente Base

3.4.1 DIAGRAMAS DE CASO DE USO DO COMPONENT BASE

Os diagramas de Caso de uso mostrados a seguir (figura 3.6 a figura 3.12), fornecem uma visão do comportamento do sistema, suas funcionalidades essenciais e a interação dos elementos envolvidos no sistema.

Os Casos de Uso Deletar, Alterar e Consultar não estão incluídos nos diagramas por serem considerados padrões em qualquer sistema. Os casos de uso Alterar são semelhantes aos casos de uso Cadastrar, a diferença entre eles ocorre porque no primeiro o sistema deverá buscar o registro a ser alterado e depois gravá-lo novamente com os novos valores enviados pela Aplicação Cliente. Nos casos de uso Deletar o sistema busca o registro a ser excluído e o exclui caso exista. Nos casos de uso Cadastrar e Deletar o sistema retorna uma mensagem, para a aplicação cliente, indicando se a operação foi bem sucedida ou não. Os casos de uso Consultar também são simples, a aplicação do cliente envia uma solicitação de consulta de um determinado registro e o sistema retorna todos os dados do registro que possui a chave correspondente.

A figura 3.6 mostra o Pacote que representa o cadastramento dos clientes, empresas e recursos que eventualmente poderão participar de uma transação.

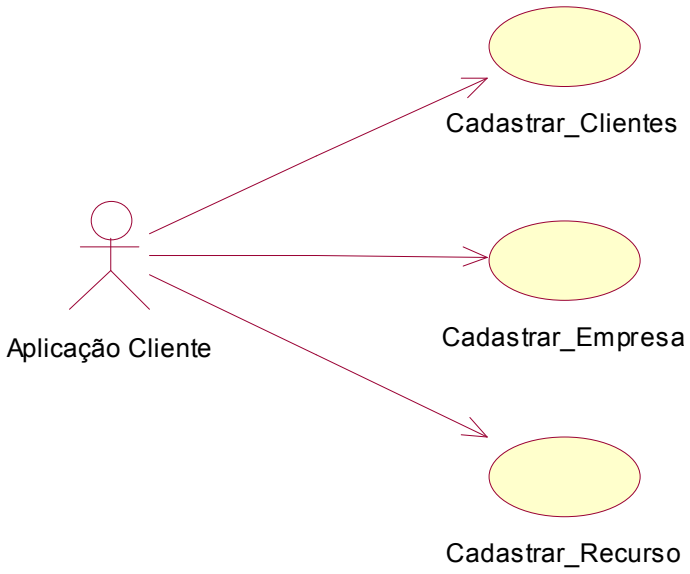


Figura 3.6 - Pacote Cadastro de Clientes, Empresas e Recursos

Na figura 3.7 está representado o caso de uso Cadastrar Clientes, que representa os serviços fornecidos pelo Componente para o cadastro de clientes.

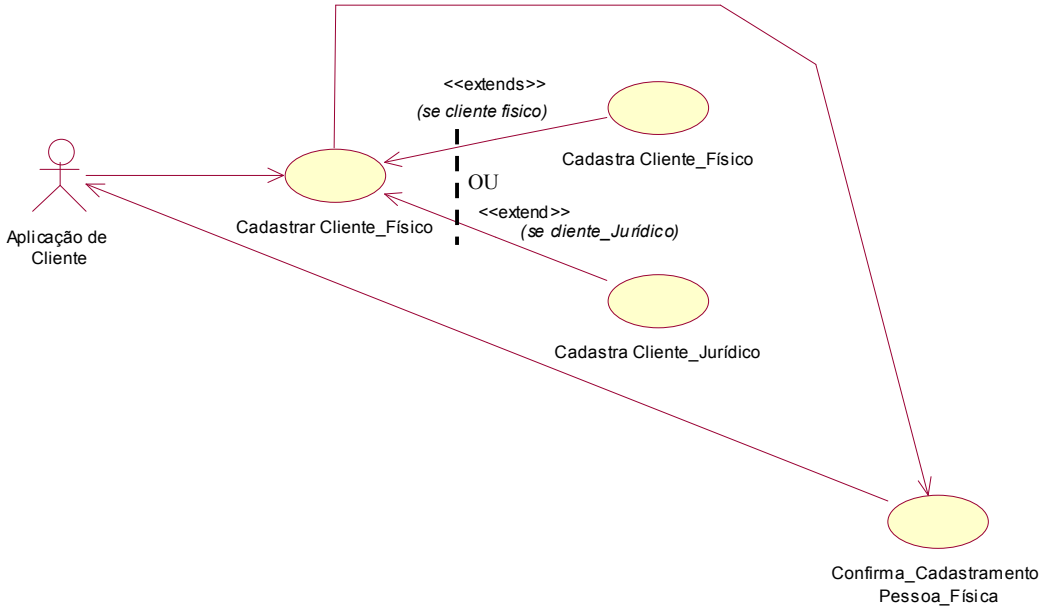


Figura 3.7 - Caso de Uso Cadastrar Clientes

Na figura 3.8 está representado o caso de uso Cadastrar Empresa, que representa os serviços fornecidos pelo Componente para cadastro de Empresas.

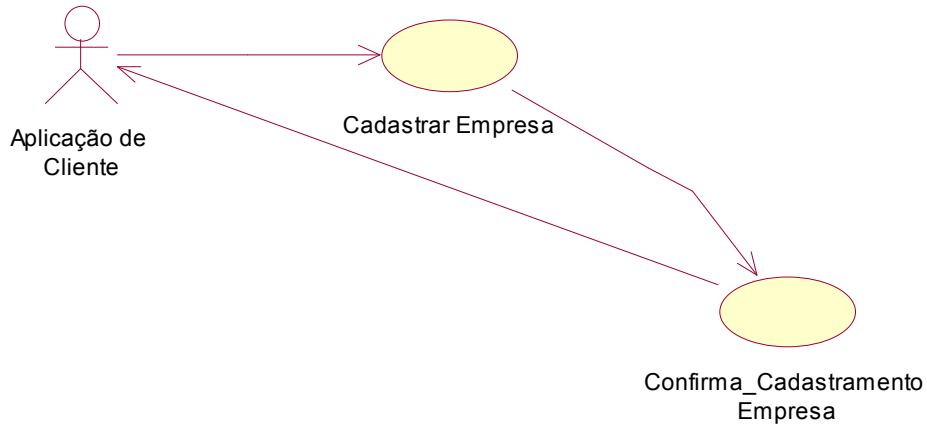


Figura 3.8 - Caso de Uso Cadastrar Empresa

Na figura 3.9 está representado o caso de uso Cadastrar Recurso, o qual representa os serviços fornecidos pelo componente para controle dos recursos que poderão ser envolvidos em uma transação. O caso de uso Cadastrar recurso permite a inclusão de novos recursos, bem como a quantificação e classificação do recurso cadastrado.

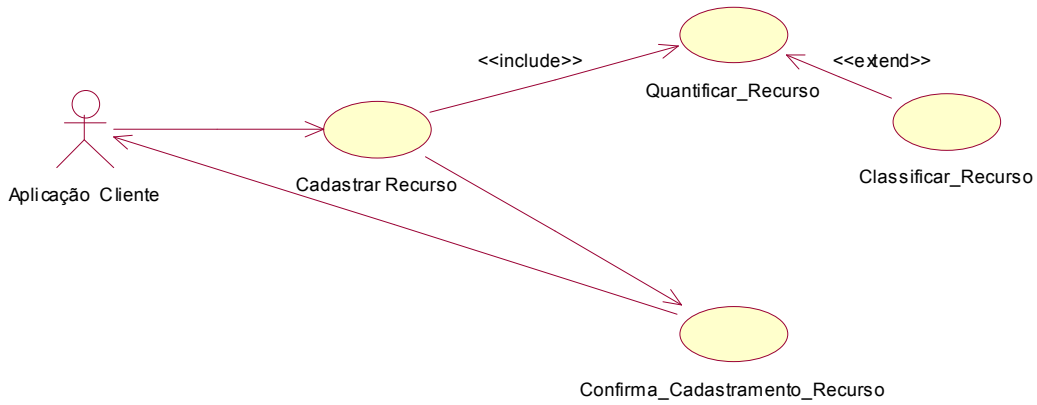


Figura 3.9 - Caso de Uso Cadastrar Recurso

Na figura 3.10 está representado o caso de uso Levantar Recurso, sendo que este demonstra como um recurso pode ser levantado, ou seja, pode ter todas as suas informações antes que seja envolvido em uma transação.

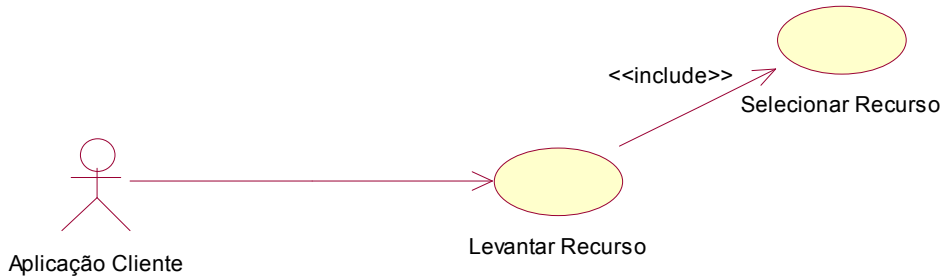


Figura 3.10 - Caso de Uso Levantar Recurso

Na figura 3.11 está representado o Pacote Transação, o qual representa os dados necessários para que uma transação possa ser efetuada.

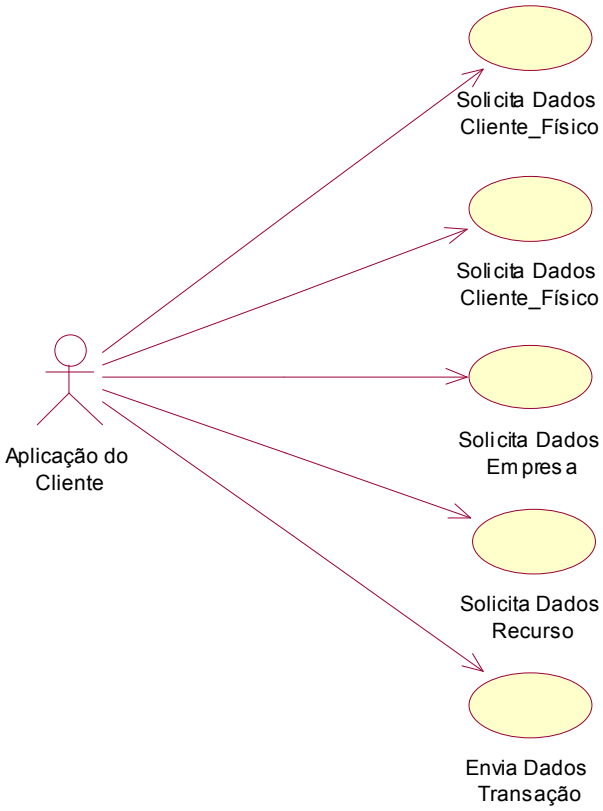


Figura 3.11 - Pacote Transação

A figura 3.12 está representando o caso de uso Efetuar Transação, que representa como decorre o processo de efetuar uma transação através do sistema proposto.

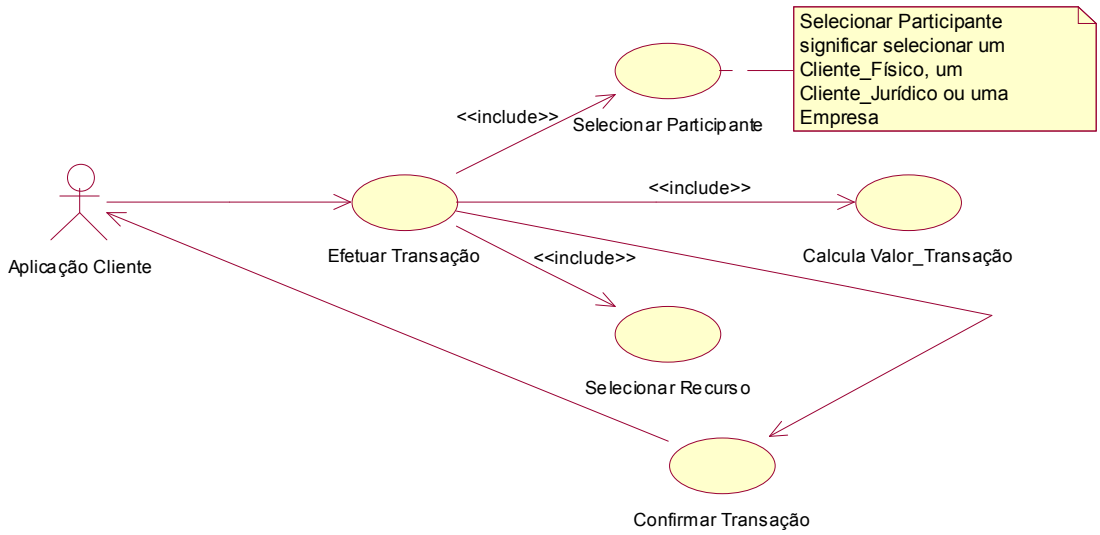


Figura 3.12 - Caso de Uso Efetuar Transação

A descrição completa dos Casos de Uso apresentados anteriormente poderá ser verificada no Anexo B.

3.4.2 ARMAZENAMENTO DE DADOS DO COMPONENTE BASE

Como a principal funcionalidade do Componente Base é prover o controle e armazenamento das informações dos clientes (físicos/jurídicos), empresas e recursos cadastrados no sistema e das transações efetuadas pela organização, é necessário que ele interaja com um banco de dados para que o armazenamento de dados possa ser feito.

No sistema proposto os dados serão armazenados em um Banco de Dados Relacional, o Diagrama de Entidade e Relacionamento (DER) do componente Base, que representa a forma com os dados serão armazenados, é apresentado na figura 3.13:

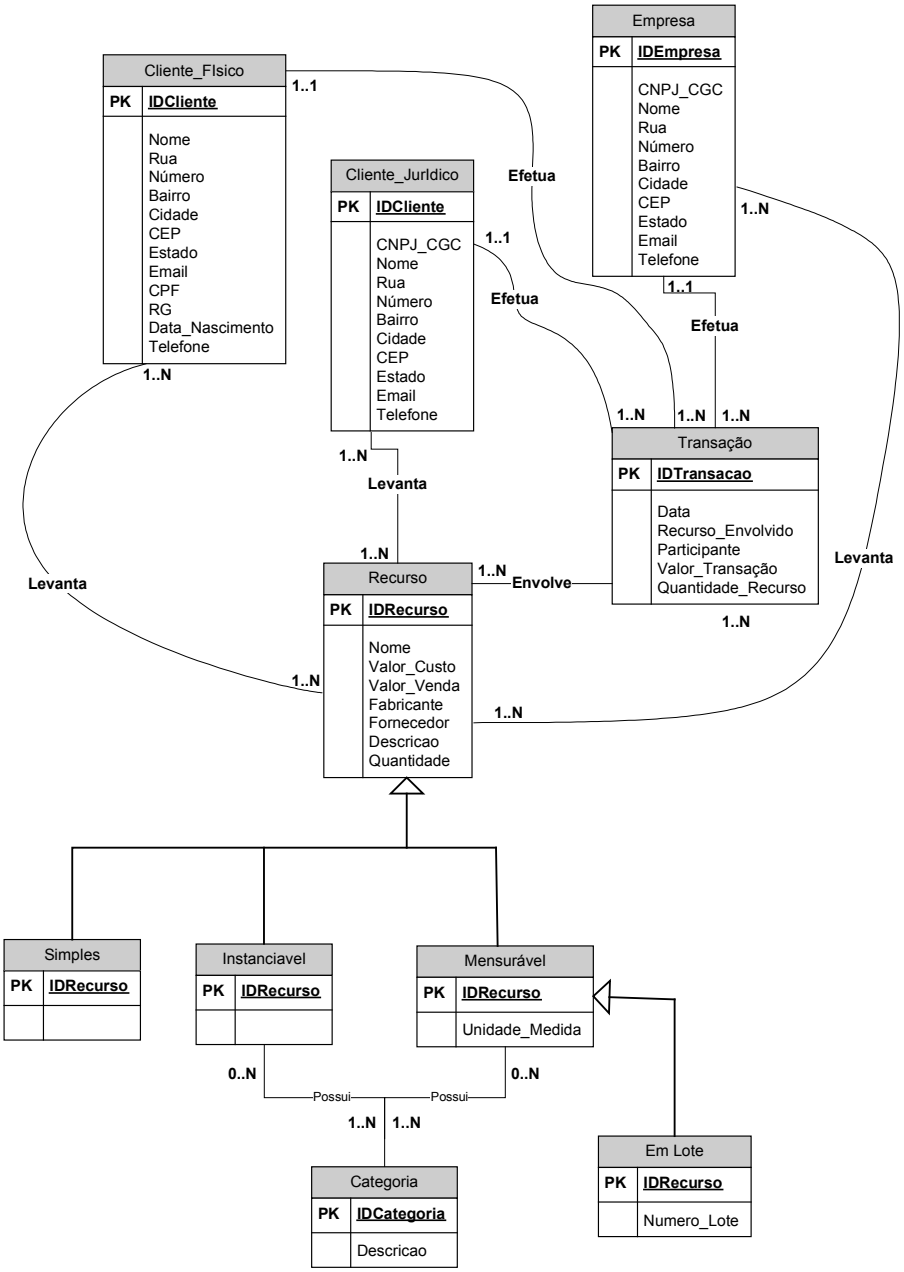


Figura 3.13 - Diagrama de Entidade e Relacionamento do Componente Base

3.5 REESTRUTURAR O COMPONENTE UTILIZANDO PADRÕES DE PROJETO

Após a Análise e Projeto do novo Componente foi realizada uma nova Análise, que tratou da verificação da estrutura do Componente em busca de partes de que tratam de problemas específicos, onde possa ser aplicado Padrões de Projeto. Com o intuito de trazer benefícios ao Componente, como torna-lo mais fácil de usar, manter e permitir também uma maior reusabilidade do mesmo.

Depois que tais problemas foram identificados, o passo a seguir foi realizar uma análise do Catálogo do Gamma [Gamm00], objetivando encontrar Padrões de Projeto que poderiam solucionar esses problemas.

O primeiro padrão identificado foi o FAÇADE (figura 3.14), padrão este que já fazia parte, implicitamente, da estrutura do Componente, ou seja, ao analisar o catálogo de Padrões de Projeto notou-se que este padrão estava sendo utilizado na estrutura do componente mesmo sem conhecê-lo. O que o padrão FAÇADE propõe é:

"Fornecer uma interface unificada para um conjunto de interfaces em um subsistema. FAÇADE define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado" [Gamm00].

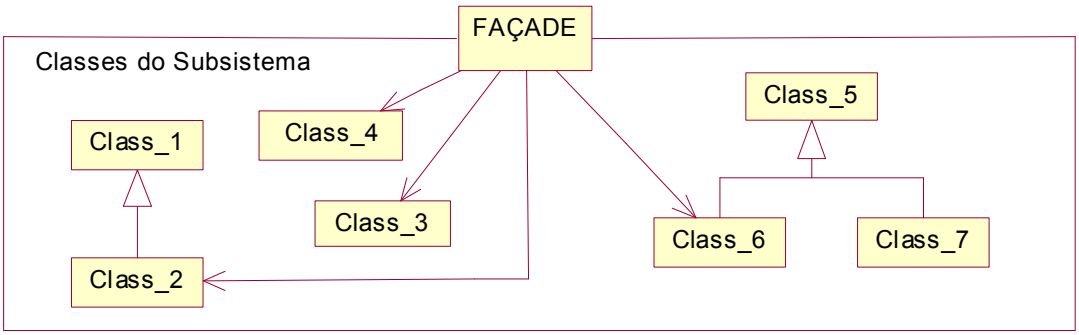


Figura 3.14 - Padrão Façade

De acordo com [Gamm00] os clientes se comunicam com um sub-sistema, que é o conjunto classes que constituem o sistema, através do envio de solicitações a FAÇADE que irá repassar para o(s) objeto(s) do sub-sistema que é responsável pela execução da tarefa.

O que a maioria dos projetos buscam é a diminuir ao máximo a comunicação e a dependência entre sub-sistemas, ou classes que compõe um sistema, e uma forma encontrada para que esse objetivo possa ser atingido é a introdução do objeto FAÇADE, que irá fornecer uma interface simplificada para os usuários desses sub-sistemas [Gamm00].

Dentre as diversas aplicações para o padrão Facade, segundo Gamma [Gamm00] é a estruturação do sistema em camadas, para isso estabelece, então, uma fachada (Façade) para cada nível do sistema.

Analisando o componente Base é possível observar que sua estrutura está organizada em níveis distintos, ou seja, suas interfaces podem ser consideradas “fachadas” (Façades), uma vez que cada interface trata de níveis exclusivos do componente, isolando os clientes dos elementos do sistema, reduzindo o número de objetos com que os clientes têm que interagir. Se a estrutura do componente Base não estivesse organizada dessa maneira a aplicação cliente seria obrigada a instanciar recursos que ela não estaria interessada em determinados momentos. Agora, uma vez estruturando o sistema em níveis distintos, a aplicação cliente tem a liberdade de utilizar apenas os níveis que lhe for conveniente, além de facilitar o seu entendimento e utilização. Além do mais o uso de Façades, segundo a literatura, provoca um

fraco acoplamento do sistema, permitindo que sejam feitas alterações no sub-sistema sem afetar os seus clientes.

A figura 3.15 ajuda a entender porque as interfaces do componente Base podem ser consideradas Façades. Cada uma das Interfaces fornece um conjunto de operações fáceis de entender e usar. Porém, para executar essas operações podem ser necessárias um emaranhado de métodos que são implementados pelas classes internas do Componente e que por sua vez fica oculto ao usuário.

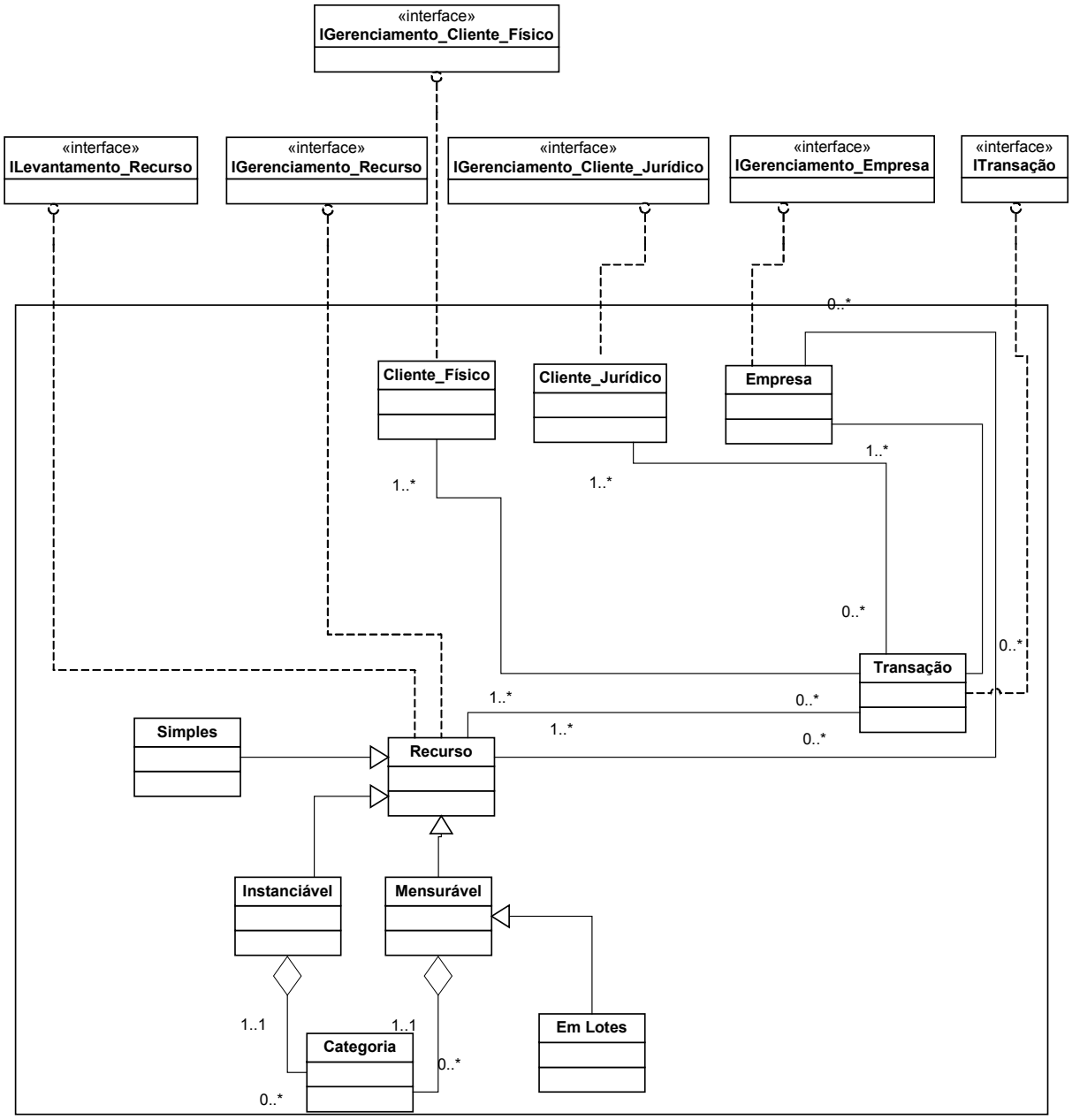


Figura 3.15 - Padrão Façade Presente no Componente Base

Outro padrão identificado, que poderia ser aplicado ao componente Base, foi o Padrão STATE que, segundo [Gamm00]:

"Permite a um objeto alterar seu comportamento quando o seu estado interno muda. O objeto parecerá ter mudado de classe."

Suponha que uma empresa, para seu maior controle, queira generalizar seus clientes em diferentes tipos (A, B, C) podendo assim futuramente propiciar ou limitar benefícios para eles. O padrão State permite que esse objetivo seja alcançado.

O Componente em questão irá fornecer uma interface para que a Aplicação Cliente possa cadastrar os clientes em 2 classes, Cliente_Físicos e Clientes_Jurídicos, sendo assim através dessa classificação é possível que a empresa que utilizar esse sistema possa oferecer serviços diferenciados para seus clientes, sendo para isso necessário algoritmos diferenciados para cada estado da classe do cliente. O mesmo raciocínio pode ser aplicado ao se cadastrar um recurso, onde esse recurso poderá ser de 4 tipos (simples, instanciável, mensurável ou em lotes). Dessa forma o padrão STATE (figura 3.16) foi aplicado no componente, para permitir que futuras funcionalidades possam ser implementadas ao componente. Pois se algum dia for necessário incluir novas classificações para os clientes ou recursos, segundo outros critérios, com o padrão State ficará mais simples criar e incluir novos métodos para atender a essas novas classes, sem afetar as aplicações que já utilizavam o componente Base.

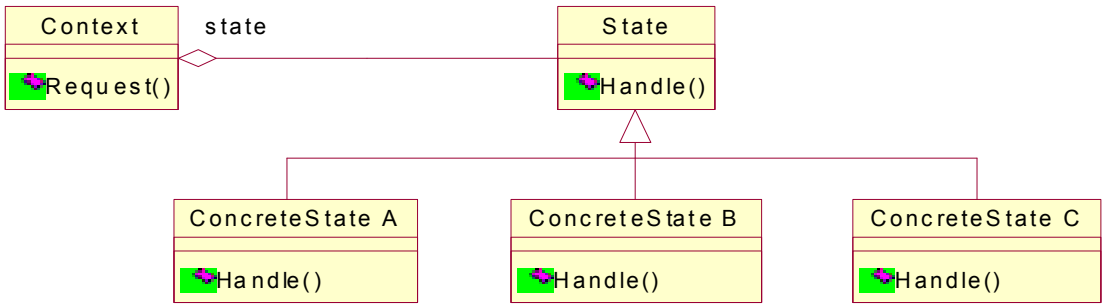


Figura 3.16 - Padrão State

Após a aplicação dos Padrões de Projeto à estrutura do componente Base, modificações significativas tiveram que ser efetuadas no mesmo. Alterações estas que foram de suma importância para a simplificação do sistema.

Na figura 3.17 está representado o novo componente Base, e suas interfaces, após aplicação dos Padrões de Projeto.

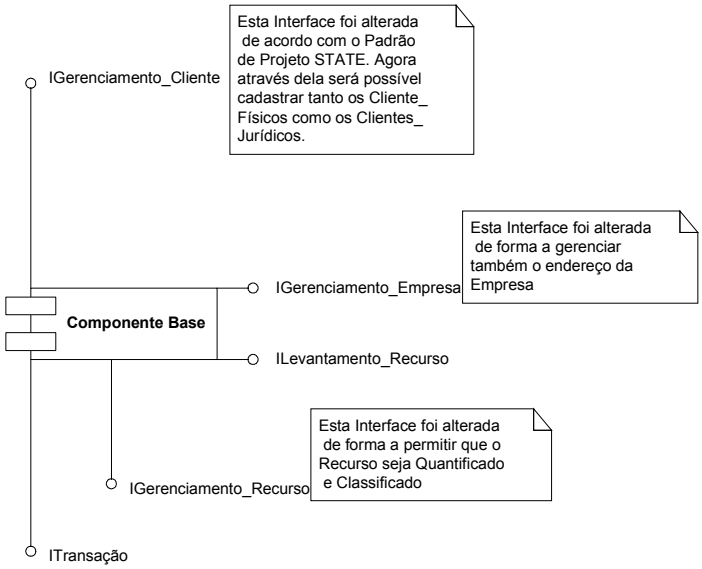


Figura 3.17 - Componente Base Após Aplicação dos Padrões de Projeto

O projeto original do componente Base propunha Interfaces distintas para o gerenciamento de clientes_físicos e clientes_jurídicos. Porém, após aplicação do Padrão de Projeto STATE, que permite a generalização de determinadas classes, foi possível unir as Interfaces antes existentes (IGerenciamento_Cliente_Físico e IGerenciamento_Cliente_Jurídico) em apenas uma Interface, IGerenciamento_Clientes) que permitirá que se escolha qual cliente será tratado.

O Diagrama de Classes do Componente Base depois de aplicados os Padrões de Projeto é mostrado na figura 3.18

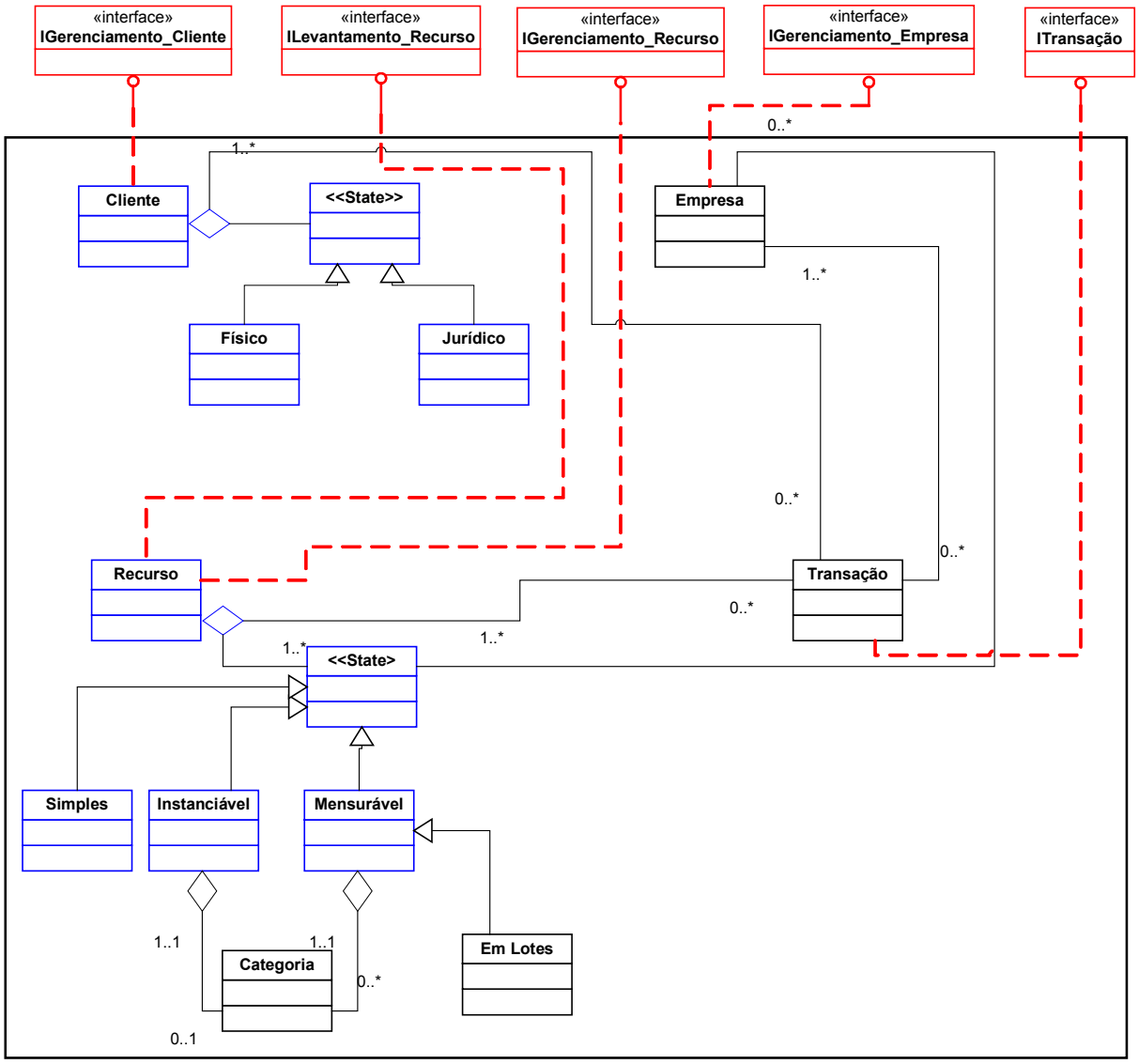


Figura 3.18 - Diagrama de Classes do Componente Base Após Aplicação dos Padrões de Projeto

O diagrama da figura 3.18 pode ser visto com mais detalhes no Anexo C, que contém o mesmo diagrama com seus atributos e métodos.

Devido às alterações promovidas pelo Padrão State, o Diagrama de Entidade e Relacionamento do Componente Base, que representa a forma como os dados serão armazenados, também teve que ser modificado, como é mostrado na Figura 3.19

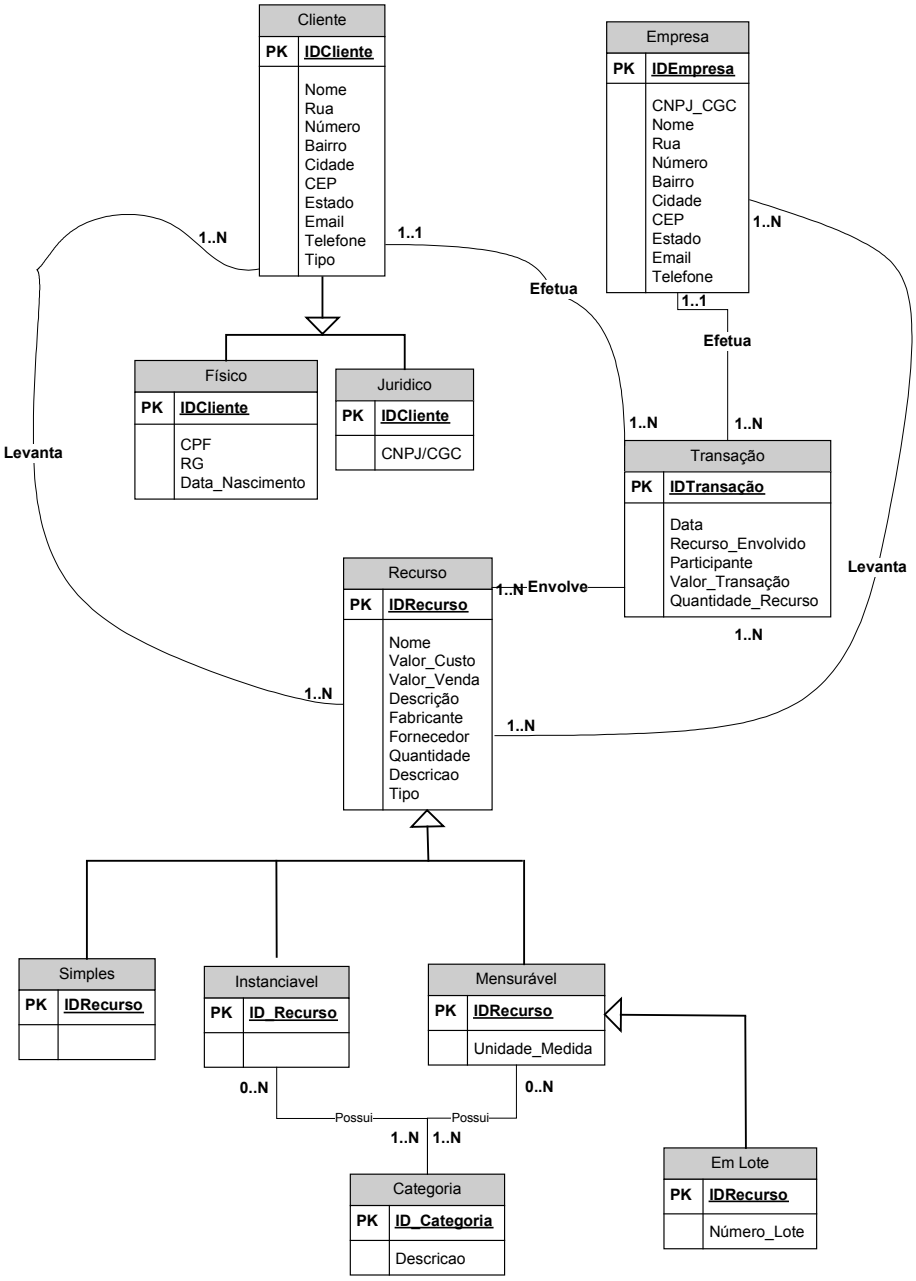


Figura 3.19 - Diagrama de Entidade e Relacionamento do Componente Base Após Aplicação dos Padrões de Projeto

3.6 CONSIDERÇÕES FINAIS

Neste capítulo foi feito todo um estudo sobre a Arquitetura de um Framework Baseado em Componentes, o que possibilitou entender como o uso de Componentes de Software pode tornar o desenvolvimento de um sistema bem mais simples, em relação às formas convencionais.

Além desse estudo, foi apresentada toda a modelagem do Componente Base, Arquitetura Estruturada em Componentes [Coel02], na qual foram seguidas etapas de desenvolvimento de software baseadas nas técnicas de Engenharia de Software Orientada a Objetos [Pres02].

Por fim, o Componente Base foi reestruturado através da aplicação de Padrões de Projeto, onde foi possível verificar a facilidade que estes trazem ao desenvolver um sistema. Visto que através deles é possível obter soluções, pré-definidas, para a resolução de problemas encontrados no processo de desenvolvimento do software, obtendo assim uma maior manutenbilidade e reusabilidade.

4 APLICAÇÃO DA TECNOLOGIA NECESSÁRIA PARA TORNAR O COMPONENTE BASE DISTRIBUÍDO

Neste capítulo será mostrado como o CORBA (*Commom Object Request Broker Arquiteture*), tecnologia que permite a distribuição de componentes de software, deverá ser aplicado no Componente Base para que ele se torne distribuído. Uma vez o Componente Base estando distribuído, ele poderá se comunicar com outros sistemas mesmo que estes estejam em computadores diferentes e sejam implementados em linguagens de programação distintas.

4.1 CORBA

No capítulo 2 foi apresentada uma introdução a respeito desse padrão de objetos distribuídos, CORBA. Neste capítulo o CORBA será descrito mais detalhadamente, de forma que se possa enfocar onde ele deverá ser aplicado no Componente Base para que esse, ao ser implementado, possa se interoperar com sistemas diferentes.

4.1.1 A LINGUAGEM DE DEFINIÇÃO DE INTERFACE (IDL)

A IDL (*Interface Definition Language*) é uma linguagem totalmente declarativa, sendo ela baseada no C++. O CORBA utiliza a *IDL* para que as interfaces que constituem o cliente e o servidor possam ser descritas. Através dela é possível garantir que os componentes implementados de acordo com o padrão CORBA, mesmo estando escritos em diferentes linguagens, possam se interagir através das redes e de sistemas operacionais [MONT97].

Na figura 4.1 pode-se verificar como a IDL permite a independência de linguagem de programação entre componentes [Bian03]:

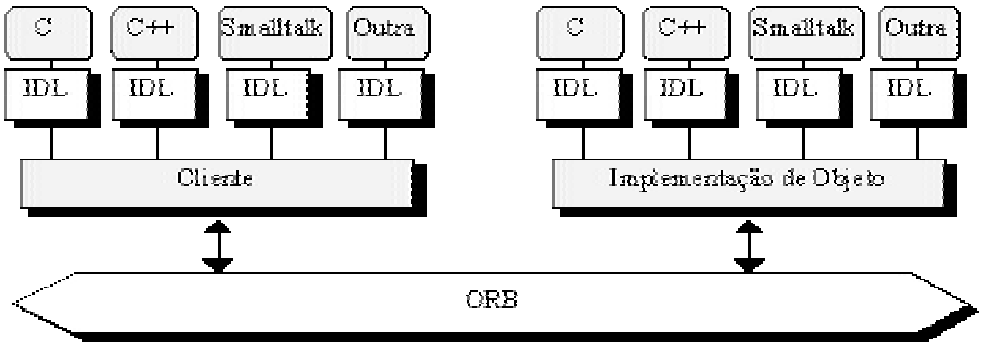


Figura 4.1 - Independência de Linguagem de Programação Através da IDL

Quando uma interface IDL é definida ela conterá a declaração de todos os métodos que um determinado objeto oferece. Com isso, o objeto que declarou uma interface, não divulga detalhes de sua implementação para os outros objetos, ou seja, na verdade ele apenas fornece, aos demais objetos, os serviços oferecidos por ele, como fazer para chamar esses serviços e o que devem esperar como respostas. Pode-se então dizer que a interface é um conjunto de informações dos objetos que são disponibilizadas aos demais objetos, utilizando para isso uma mesma linguagem (IDL), com a finalidade de permitir essa interação entre eles, sem que seja necessário se preocupar com detalhes individuais da implementação de cada objeto [Auto00].

É importante ressaltar que não é permitido nenhum tipo de programação em IDL. Os objetos CORBA que a IDL descreve deverão ser implementados em qualquer outra linguagem que tenha suporte a CORBA, por exemplo, Java, Delphi, C++, etc. As regras que convertem a IDL para a linguagem de programação na qual o objeto foi implementado são chamadas de Ligações de Linguagem, e estas são padronizadas pela OMG (*Object Management Group*). Dessa forma todos os fornecedores de serviços CORBA são obrigados a usar as mesmas regras para mapear construções IDL em uma linguagem de programação específica[Corn00].

Para exemplificar como uma linguagem pode ser mapeada para a IDL, a seguir serão descritas algumas das várias diferenças existentes entre a linguagem IDL e a linguagem Java [Corn00]:

- Em IDL a definição de interface termina com um ponto-e-vírgula. Diferentemente do Java;
- Em IDL a palavra reservada string é escrita com letra minúscula, ao contrário do Java;
- Na linguagem Java as strings contêm caracteres Unicode de 16 bits, já no CORBA as strings contêm apenas caracteres de 8 bits. Se seu objeto receber uma string de 16 bits será lançada uma exceção, informando esta incompatibilidade. Os tipos de dados suportados pela IDL, bem como suas descrições podem ser verificados detalhadamente no Anexo E;

Cada linguagem de programação, que tem suporte a CORBA, possui um compilador específico que irá transformar as definições IDL em definições de interfaces da própria linguagem. Na linguagem Java, o compilador IDL correspondente (a partir da versão JDK – *Java Development Kit* 1.3) é o compilador **idlj (*Interface Definition Language Java*)** [Corn00]. Portanto, se for inserido a definição IDL IGerenciamento_Cliente em um arquivo IGerenciamento_Cliente.idl e executar o compilador **idlj,** o resultado será um arquivo IGerenciamento_Cliente.java.

4.2 ORB (OBJECT REQUEST BROKER)

O ORB é a estrutura mais importante da arquitetura CORBA. Ele é responsável por todos os mecanismos requeridos para encontrar o objeto, preparar a implementação deste objeto para receber a requisição e executar a requisição. É através dele que o cliente enxerga a requisição de forma independente de onde o objeto (servidor) está localizado, qual linguagem de programação ele foi implementado ou qualquer outro aspecto que não está refletido na interface do objeto. Também, é função do ORB, o retorno de parâmetros de saída da requisição para o cliente, se assim houver [RICC00].

O ORB trata-se de um conjunto de módulos de softwares que gerenciam a comunicação entre objetos, podendo ser denominado, segundo a literatura, como o "barramento de objetos". O grande potencial deste "barramento" está em permitir que objetos façam, de forma transparente, requisições a objetos que podem estar localizados localmente ou remotamente. Essa transparência assegura que o cliente (requisitante) não tenha conhecimento de quais os mecanismos utilizados para se comunicar com o objeto desejado [Corb02].

O ORB se estende desde o Stub do cliente (seção 4.2.4), onde é invocado o serviço, e vai até o Skeleton (seção 4.2.5) no servidor, ficando responsável por toda comunicação necessária entre esses dois componentes . Algumas das características mais importantes do ORB são listadas abaixo [Auto00]:

- **Transparência entre serviços Locais/Remotos:** o ORB pode executar uma operação sobre um objeto e é indiferente se ele (o objeto) se encontra localmente ou em algum lugar da rede que não seja sua máquina.

- **Utilização de Linguagens de Alto-Nível:** o ORB permite a utilização de linguagens consideradas de alto-nível, como por exemplo, o C, C++, Java, Ada e Smaltalk, na invocação de métodos no servidor, não importando em qual linguagem o servidor de objetos esteja implementado.

- **Estado de Execução de Objeto:** quando um cliente faz uma requisição a um determinado objeto, ele (cliente) não precisa saber se esse tal objeto es-

tá ativo ou não, pois o ORB se encarrega de ativá-lo, antes de entregar o método ao objeto.

Pode-se considerar que toda função que o ORB se resume a fazer, está vinculada às requisições de serviços que são feitas pelo cliente ao servidor de objetos, ou seja, chamadas de um objeto cliente a métodos que um objeto servidor oferece e estas são compostas por três componentes [Andr01]:

- **Referência ao Objeto**: identifica o objeto desejado;
- **Operação**: Identifica qual operação deve ser executada;
- **Parâmetros**: constitui-se os dados necessários para a execução da chamada;

A figura 4.2 representa a estrutura de um ORB [Manu99]:

Figura 4.2 - Estrutura do ORB

4.2.1 INTERFACE ORB

A interface ORB foi definida, dentro da especificação CORBA, como uma "interface abstrata" para o ORB, servindo como uma biblioteca de ajuda para os serviços locais do mesmo (ORB). Por se tratar de uma entidade lógica, a interface pode ser implementada de

diversas maneiras. Portanto, a interface ORB aparece justamente para desacoplar as aplicações dos detalhes de cada uma dessas implementações de ORB. Dentre as funcionalidades oferecidas por essa interface, a conversão de "referências a objetos" para strings e vice-versa é uma delas. Provê também a criação de listas de argumentos para as requisições que são realizadas através da DII (Interface de Invocação Dinâmica) [Corb02].

4.2.2 INTERFACE DE INVOCAÇÃO DINÂMICA (DII – *DYNAMIC INVOCATION INTERFACE*)

A Interface de Invocação Dinâmica permite ao cliente, por meio do descobrimento dos métodos em tempo de execução, um acesso direto aos mecanismos de requisição residentes em um ORB. Ou seja, através da DII é possível que um programa cliente faça chamadas a um método de um objeto CORBA (servidor) cujo tipo era desconhecido quando o cliente foi escrito [Mont97].

4.2.3 ADAPTADOR DE OBJETO

Os Adaptadores de Objetos, também chamados de AO, tem como principal função auxiliar o ORB no despacho de requisições para os objetos e na ativação de um determinado objeto. O AO é considerado o núcleo da comunicação do ORB, pois associa implementações de um objeto com o ORB. Existem vários tipos de adaptadores de objetos, pois é possível especializá-los para uma determinada função. Como exemplo de um AO especializado tem-se o OODB que são adaptadores para objetos, adaptadores persistentes e de bibliotecas para objetos não-remotos. Entretanto, independentemente de existir os AOs especializados, sempre será encontrado um "padrão" denominado **BOA** (*Basic Object Adapter*) que nada mais é do que um adaptador padrão presente em todos os ORB's [Bian03].

4.2.4 STUB

Os Stubs também são conhecidos como *proxies*, por possuírem a capacidade de localizar os objetos alvos. Estão localizados no lado do cliente e são gerados a partir da compilação da interface do cliente definida anteriormente em IDL. Quando um cliente deseja chamar um método de um objeto, basta apenas indicar qual o objeto que deseja. Tal chamada é repassada para o stub que terá como função receber essa chamada, localizar o objeto desejado, transformar a chamada para que possa ser enviada pela rede e transmiti-la para o ORB, dando assim continuidade no resto do processo. A comunicação entre o stub e o ORB se dá por meio de interfaces privadas [Corb02].

4.2.5 SKELETON

O Skeleton tem uma função parecida com o Stub do cliente, porém está situado do lado do servidor. A partir daí o Skeleton irá disponibilizar os métodos dos objetos servidores para o resto do sistema e também encontrar os serviços que são solicitados pelo Stub, ou seja, as requisições solicitadas pelo cliente [Bian03].

A descrição do funcionamento do Skeleton seria: receber um pacote (requisição) do ORB e enviar esse pacote para a implementação de objetos. A partir daí essa requisição será recebida pelo objeto que irá executá-la, quando esse objeto executar essa requisição, ele enviará o resultado para o skeleton, que irá passá-lo ao ORB, posteriormente pelo stub até chegar ao cliente [Auto00].

A figura 4.3 mostra como se dá a comunicação entre aplicações, desenvolvidas em CORBA, através do ORB:

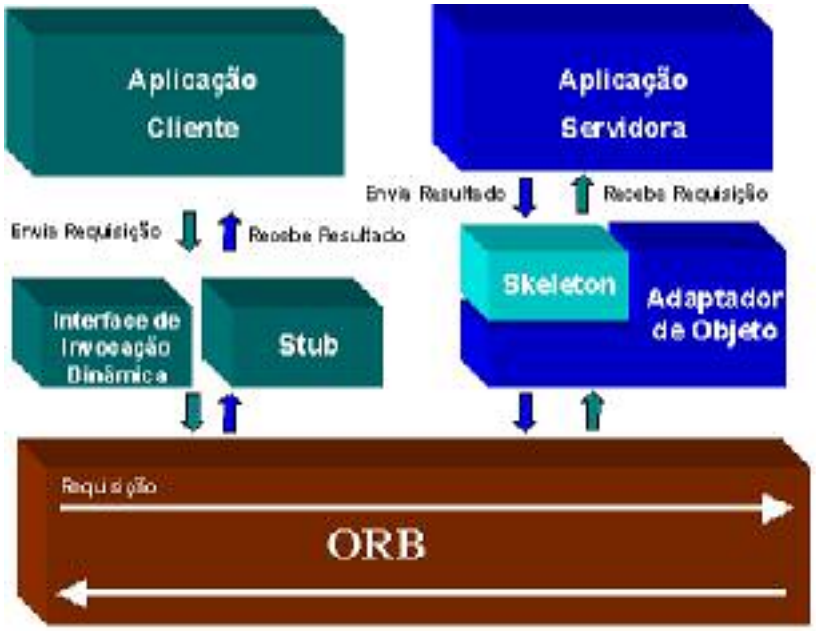


Figura 4.3 - Comunicação Entre Aplicações Através do ORB

Essa comunicação, ou seja, o cliente requisitando um serviço do servidor, acontece da seguinte forma: a Aplicação Cliente envia uma requisição de serviço, via Stub, até o ORB. Esta requisição "viaja" pelo ORB até o Skeleton no Servidor, onde esse enviará a requisição à classe responsável por executar a requisição. Após a requisição ter sido executada, essa classe envia o resultado, via Skeleton, para o ORB, sendo assim esse resultado percorre todo o ORB até chegar ao Stub, que por sua vez repassa ao cliente.

4.3 APLICAÇÃO DO PADRÃO CORBA NO COMPONENTE BASE

Após as principais funcionalidades do CORBA, terem sido apresentadas, será mostrado onde esse padrão deverá ser aplicado no Componente Base para que este possa ser considerado um componente de software (objeto) distribuído.

Como foi descrito na seção 4.1, a(s) interface(s) do objeto a ser distribuído deve ser especificada na linguagem IDL, para que assim outros objetos implementados em lingua-

gens de programação distintas, do objeto em questão, possam acessar os métodos disponibilizados por este objeto. Desde que este objeto também tenha sido implementado segundo o padrão Corba.

De acordo com o escopo desse trabalho o Componente Base é o objeto no qual deseja-se tornar distribuído. Como este componente possui várias interfaces, descritas na seção 3.5, ao implementá-lo, todas as interfaces terão que ser especificadas de acordo com a linguagem IDL. Dessa forma qualquer sistema que se comunique com este componente, sistema este que também deverá ser implementado baseando-se no padrão CORBA, poderá requisitar qualquer um dos métodos oferecidos pelas classes do Componente Base.

A figura 4.4 mostra a representação do diagrama de classes do Componente Base (seção 3.5) considerando que o mesmo, ao ser implementado, será baseado no padrão CORBA:

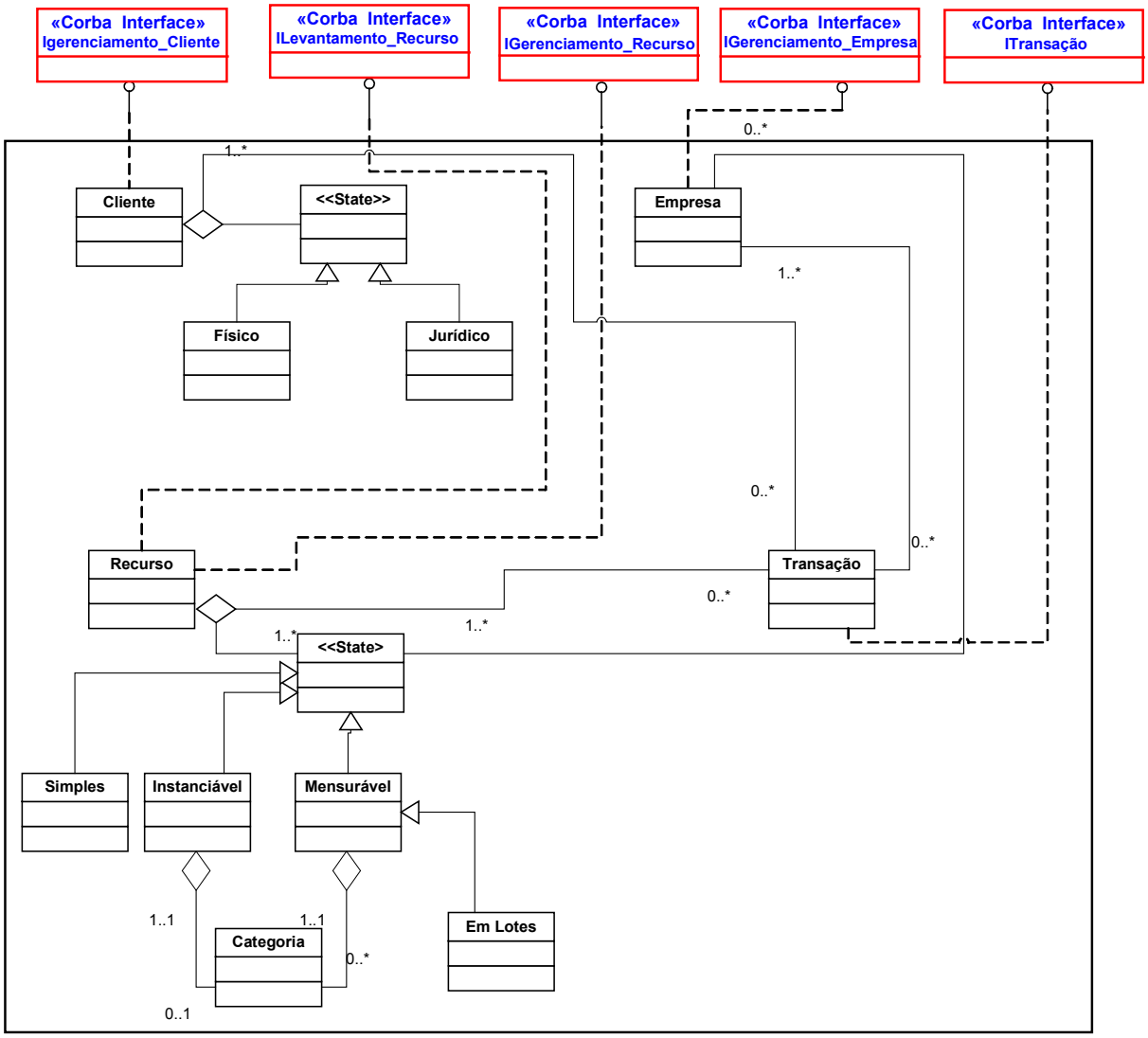


Figura 4.4 - Diagrama de Classes do Componente Base com a Aplicação do Padrão Corba

Este diagrama foi adaptado segundo a modelagem UML proposto em [Grou03], ou seja pode-se observa que o CORBA é aplicado diretamente nas interfaces do Componente.

Partindo do princípio que o Componente Base ao ser desenvolvido será implementado na linguagem de Programação Java, a seguir será apresentada a estrutura da interface IGerenciamento_Empresa provida pelo Componente Base, escrita de acordo com a linguagem IDL. A sintaxe para escrever as outras interfaces do Componente Base segue a mesma linha de raciocínio, tornando-se desnecessária a apresentação das mesmas ao longo do texto.

```
interface IGerenciamento_Empresa

{

    string Cadastrar_Empresa( in long idempresa, in string cnpj_cgc, in
string nome, in string rua, in long numero, in string bairro, in string cidade, in string
estado, in long cep, in long telefone , in string email, in string atividade);

    string Alterar_Empresa(in long idempresa, in string cnpj_cgc);

    string Deletar_Empresa( in string cnpj_cgc);

    string Consultar_Empresa( in string nome, in string cnpj_cgc);

    long Get_Numero_Empresas ( );

};
```

A interface escrita na linguagem Java, correspondente a essa interface IDL se encontra no Anexo F. O disquete que contém todas as interfaces providas pelo Componente Base está à disposição (Anexo G) para os interessados em verificar a compilação dessas interfaces.

Essa interface ao ser compilada através do compilador idlj (idlj IGerenciamento_Empresa.idl), será gerado os seguintes arquivos [Corn00]:

IGerenciamento_Empresa.java: a definição da interface;

IGerenciamento_EmpresaHolder.java: a classe recipientes para parâmetros **out**[1];

IGerenciamento_EmpresaHelder.java: classe auxiliar da interface IGerenciamento_Empresa[2].

---

[1] Um método armazena um valor em cada parâmetro out, antes de retorná-lo. A partir daí o objeto que chamar este método poderá recuperar os valores armazenados nos parâmetros out.

[2] Fornece o método *narrow* que faz a conversão da referência dos objetos de nomes (localiza outros objetos) em uma referência Igerenciamentio_Empresa, permitindo assim chamar os métodos dessa interface.

_ Igerenciamento_EmpresaOperations.java: a definição das operações oferecidas pela interface;

_ Igerenciamento_EmpresaStub.java: a classe de stub para comnuicação com ORB;

Uma vez que todas as interfaces do Componente Base estiverem sido escritas e compiladas de acordo com a sintaxe da linguagem IDL e os métodos descritos nessas interfaces tenham sido implementados, o Componente Base estará apto para fazer parte de um sistema distribuído. Visto que outras aplicações, que também tenham sido implementadas segundo o padrão CORBA, poderão ter acesso a seus métodos através das interfaces providas por ele.

Vale lembrar que, embora a Ligação de Linguagens seja padronizada, cada linguagem de programação possui um compilador IDL próprio. Então, ao ser compilada uma interface IDL, o conjunto de arquivos gerados se diferenciará de linguagem para linguagem [Core].

4.4 APLICAÇÃO DO COMPONENTE BASE EM UM SISTEMA DISTRIBUÍDO

Para exemplificar a utilização do Componente Base em um sistema distribuído pode-se pensar em uma Aplicação Cliente, desenvolvida na linguagem de programação Object Pascal (com interface gráfica Delphi), por exemplo, que irá acessar as funcionalidades do componente base. Essa Aplicação Cliente pode ser desenvolvida em qualquer outra linguagem que suporte a especificação CORBA. Pensando em termos de um sistema de grande porte é possível também que várias aplicações, localmente ou através de uma rede de computadores, façam uso das funcionalidades oferecidas pelo Componente Base, enviando, através do ORB, requisições aos métodos que estão disponibilizadas em suas interfaces.

A figura 4.5 demonstra com seria a comunicação entre uma Aplicação Cliente, desenvolvida em Delphi, C++, Java ou VB, ambas com suporte ao padrão CORBA, que faz requisições às operações implementadas pelas classes do componente.

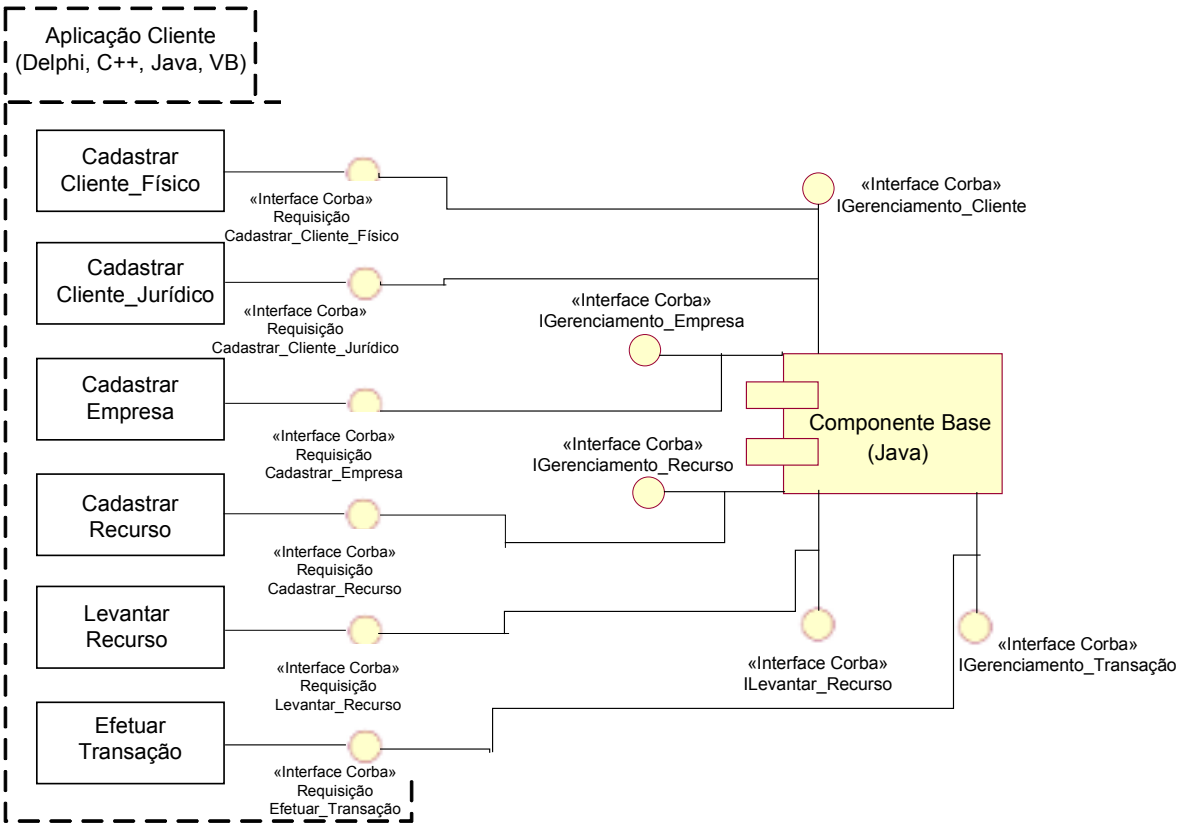


Figura 4.5 - Representação da Comunicação Entre uma Aplicação Cliente e o Componente Base

4.5 CONSIDERAÇÕES FINAIS

Após ter sido feito um estudo de forma mais aprofundada sobre a tecnologia para a interoperabilidade entre objetos implementados em linguagens de programação diferentes, o padrão CORBA, foi possível verificar o quanto um sistema torna-se mais flexível a partir do momento que essa tecnologia é aplicada a ele. Pois, a partir daí consegue obter um alto grau de reusabilidade, não só de código, mas de uma aplicação por completo, que irá influenciar

diretamente na economia do tempo de montagem do sistema. Porque se já existe uma aplicação desenvolvida, que é solução de um determinado problema, dentro do domínio que o sistema irá atuar, basta incorpora-la ao sistema apenas adaptando ambas as partes de forma que eles possam conversar entre si, sem maiores problemas. E o CORBA permite que isso seja feito de forma simplificada e totalmente transparente.

5 CONCLUSÕES

Foi apresentada neste trabalho a modelagem de um componente de software, proveniente de um Framework baseado em Componentes de Software. Bem como a estruturação desse componente através de Padrões de Projeto e a aplicação de uma tecnologia, para distribuição de objetos, que permite que esse componente ao ser implementado possa se Distribuído.

As principais conclusões que esse trabalho proporcionou foram:

- Através do uso correto da Engenharia de Software é possível modelar de forma adequada e transparente um sistema de software. Visando assim o mínimo de erro possível ao implementar o software, principalmente com relação ao entendimento das funcionalidades que esse software deverá fornecer.

- A aplicação de Padrões de Projeto durante a modelagem de um sistema de software, permite encontrar soluções para determinadas situações que irão reduzir de forma significativa à complexidade dos esforços necessários ao desenvolvimento de um software de qualidade.

- Através da estruturação de um Sistema de Software a ser desenvolvido, utilizando Componentes de Software, busca-se alcançar, com isso, um alto grau de reusabilidade do software. Uma vez que se pode incluir neste sistema, Componentes de Software já prontos, pertencentes ao domínio da aplicação, fazendo apenas pequenas alterações de forma que esse Componente possa atender eficientemente às necessidades do sistema em questão.

- A manutenção de Sistemas baseados em Componentes de Software é feita com muito mais facilidade, do que a manutenção realizada em Sistemas tradicionais. Pois, as alterações serão efetuadas internamente no Componente e já que ele dispõe suas funcionalidades através de interfaces, os clientes não serão afetados, visto que estes conversam com o Componente somente através dessas interfaces.

- Além de toda a flexibilidade que os Componentes de software trazem ao sistema. Quando esses Componentes são projetados de forma distribuída eles atribuem ainda mais rapidez e economia no desenvolvimento do sistema que eles irão compor. Já que uma vez os Componentes de Software estando distribuídos eles podem se comunicar com outras aplicações desenvolvidas em linguagens de programação diferentes e consequentemente plataformas e hardwares distintos. Com isso o investimento em novas tecnologias é reduzido significadamente.

As conclusões anteriormente apresentadas podem fazer a diferença para empresas desenvolvedoras de sistemas, pois seus softwares serão desenvolvidos com mais rapidez, menor custo e terão maior flexibilidade. Porém essas vantagens serão percebidas a médio e longo prazo, já que a complexidade e o tempo necessário para desenvolvimento de Componentes de Softwares não é trivial, mais à medida que aplicações pertencentes ao domínio do componente são desenvolvidas, essas vantagens são incontestáveis.

5.1 CONTRIBUIÇÕES

Dentre as contribuições oferecidas por este trabalho, pode-se destacar as seguintes:

- A modelagem do Componente Base tende a oferecer uma base para estudantes que desejam desenvolver habilidade em Engenharia de Software, uma vez que este projeto documentou algumas das principais etapas de modelagem desse componente.

- Um estudo analítico sobre como os Padrões de Projeto ajudam no desenvolvimento de um Componente de Software e quais as vantagens de se utilizar esses padrões.

- O estudo sobre o Padrão de Objetos Distribuídos CORBA *(Common Object Request Broker Architecture)* oferece uma boa base para programadores que desejam desenvolver softwares baseados nesse padrão, ou seja Softwares Distribuídos.

- A conclusão deste projeto permitiu um maior conhecimento dos sistemas distribuídos, onde a interoperabilidade desses sistemas é um dos principais

focos da computação dos dias de hoje, visto que cada dia torna-se mais necessário que as organizações integrem seus sistemas.

5.2 TRABALHOS FUTUROS

Alguns trabalhos poderão ser feitos para dar continuidade a essa monografia:

1. Implementar o Componente Base, seguindo a modelagem proposta neste trabalho.

2. Desenvolver uma Aplicação, implementada em linguagem diferente da linguagem aplicada ao Componente Base, para verificar a interoperabilidade entre sistemas proveniente do padrão CORBA.

3. Desenvolver os demais Componentes da Arquitetura Estruturada em Componentes [Coel02] e criar um Framework que facilite a criação de aplicações utilizando esses componentes.

REFERÊNCIAS BIBLIOGRÁFICAS

[Andr01] ANDRÉ, D. S., Corba: Reutilização de Componentes de Softwares. Url:http://www.dcc.ufrj.br/~schneide/es/2001/1/g01/corba.htm. Visitado em 15/09/2003. Ufrj. 2001.

[Auto00] AUTOMAÇÃO, G. T. Corba. Url:http://www.gta.ufrj.br/grad/00_2/corba. Visitada em 25/11/2003. Ufrj. 2000.

[Bian03] BIANCHETTI, L. L. Sistema Distribuído com o Padrão de Arquitetura Corba. Trabalho Final de Curso. Unipac. 2003.

[Booc99] BOOC, G., UML Guia do Usuário: O Mais Avançado Tutorial Sobre Unified Modeling Language (UML). Campus. 1999.

[Brag01] BRAGA, R. T. V. Um Framework Baseado em Uma Linguagem de Padrões para Sistemas de Gestão de Recursos de Negócios. Tese de Doutorado em Ciência da Computação Url:http://www.icmc.sc.usp.br/~rtvb/GRENFramework.html. Usp. 2001.

[Carl99] CARLA, W. K. L. Modelagem do Estudo de Caso Utilizando Rational Rose. Url:http://rochel.inf.ufrgs.br/amadeus/carlawkl/cmp159/trabalhofinal.html. Visitada em 14/10/2003. Ufrgs. 1999.

[Coel02] COELHO, F. M. Uso de Componentes de Software no Desenvolvimento de Frameworks Orientados a Objetos. Tese de Mestrado em Ciência da Computação.Unicamp. 2002.

[Corb02] CORBA, G. U. Corba. Url:http://www.sucesusp.com.br/html/grupos/corba/corba.html. 2002.

[Corn00] CORNELL, G. Core Java 2: Recursos Avançados. Makron Books. 2000.

[Gamm00] GAMMA, E., Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos. Bookman. 2000.

[Group03] GROUP. O. M. Uml Profile for Corba Specification. Url:http://www.omg.org/docs/formal/02-04-01.pdf. Visitada em 05/03/2004.

[Manu99] MANUEL, V. G. R., Arquiteturas Distribuídas Cliente Servidor: Corba, DCOM, JavaRMI. Tese de Mestrado em Engenharia Eletronica e Telecomunicações. Universidade Aveiro.1999.

[Marc98] MARCOS, A. S. K., Uma Introdução à Java. Url:http://java.icmc.sc.usp.br/library/tutorial_java.ppt. Visitada em 13/10/2003. Usp. 1998.

[Mari02] MARINHO, P. B., Objetos Distribuídos – Sistemas Distribuídos Modelo Cliente Servidor. Url:http://www.ing.unisinos.br/~marinho/Teaching/Especializacao/SistDist/11-objdist.pdf. Visitado em 03/10/2003. Unisinos.2002.

[Melo02] MELO, A. C., Desenvolvendo Aplicações com UML – Do Conceitual à Implementação. Brasporte. 2002.

[Mont97] MONTEZ, C. Um Modelo de Programação para Aplicações de Tempo Real em Sistemas Abertos. Monografia do Exame de Qualificação de Doutorado.Ufsc. 1997.

[Nirv00] NIRVA, M. L., Oracle 8i – Programação de Componentes Java em EJB, Corba e JSP: Desenvolva Aplicativos em Nível Empresarial com Componentes Java. Campus. 2000.

[Pime01] PIMENTEL, G. Objetos Distribuídos. Url:http://www.linuxdb.hg.ig.com.br/artigos/art2.htm. Visitada em 23/10/2003.

[Pres02] PRESSMAN, R. S. Engenharia de Software. MacGrawHill. 2002.

[Ricc00] RICCIONI, P. UML: Guia do Usuário. Campus. 2000.

[Roch03] ROCHA, H., Padrões de Projeto. Url:http://www.argonavis.com.br/palestras/java/j930/index.html. Visitado em 03/11/2003.

[Sand99] SANDRO., Faq. Url:http://www.gta.ufrj.br/grad/99_2/sandro/faq.htm. Visitada em 23/09/2003. Ufrj. 1999.

[Tane03] TANENBAUM, A. S. Redes de Computadores. Campus. 2003.

[Tava03] TAVARES, A. M. Desenvolvimento de um Componente de Software para o Domínio de Gestão de Recursos de Negócios: Componente Locar_Reservar_Recurso. Trabalho Final de Curso.Unipac. 2003.

[Visi03] VISIO, M. Microsoft OfficeVisio 2003. Url:http://www.microsoft.com/brasil/office/visio/default.asp. Microsoft. 2003.

ANEXO A – INTERFACES DO COMPONENTE BASE

A seguir serão descritos todos os métodos, e suas funcionalidades, que compõem cada Interface que constitui o Componente Base.

Interface IGerenciamento_Cliente

Cadastrar_Cliente_Físico (IDCliente, Nome, Data_Nascimento, RG, CPF, Rua, Número, Bairro, Cidade, CEP, Estado, Telefone, Email): Integer;

Permite o cadastro de um cliente físico. Sendo que o parâmetro IDCliente deverá ser fornecido pelo próprio sistema. Este método retorna 0 se o cliente for cadastrado com sucesso e 1 se ocorrer algum erro durante a gravação do registro no banco de dados.

Cadastrar_Cliente_Jurídico (IDCliente, Nome, CNPJ/CGC, Rua, Número, Bairro, Cidade, CEP, Estado, Telefone, Email): Integer;

Permite o cadastro de um cliente jurídico. Sendo que o parâmetro IDCliente deverá ser fornecido pelo próprio sistema. Este método retorna 0 se o cliente for cadastrado com sucesso e 1 se ocorrer algum erro durante a gravação do registro no banco de dados.

Alterar_Cliente_Físico (IDCliente, Nome, Data_Nascimento, RG, CPF, Rua, Número, Bairro, Cidade, CEP, Estado, Telefone, Email): Integer;

Permite alterar o registro de um cliente físico passando-se o IDCliente ou CPF como parâmetros. Este método Retorna 0 se o cliente for encontrado e tiver seus dados alterados com sucesso e 1 se o registro não for encontrado.

Alterar_Cliente_Jurídico (IDCliente, Nome, CNPJ_CGC, Rua, Número, Bairro, Cidade, CEP, Estado, Telefone, Email): Integer;

Permite alterar o registro de um cliente físico passando-se o IDCliente ou CNPJ_CGC como parâmetros. Este método Retorna 0 se o cliente for encontrado e tiver seus dados alterados com sucesso e 1 se o registro não for encontrado.

Deletar_Cliente (IDCliente): Integer;

Permite a remoção do cliente, tanto físico como jurídico, passando-se o IDCliente como parâmetro. Retorna a 0 se o cliente for excluído e 1 caso contrário.

Consultar_Cliente_Físico (IDCliente, Nome, Data_Nascimento, RG, CPF, Rua, Número, Bairro, Cidade, CEP, Estado, Telefone, Email): Integer;

Permite consultar todos os atributos de um cliente físico passando-se o nome, IDCliente ou CPF como parâmetros. Se o cliente não for encontrado será retornado 1.

Consultar_Cliente_Jurídico (IDCliente, Nome, CNPJ_CGC, Rua, Número, Bairro, Cidade, CEP, Estado, Telefone, Email): Integer;

Permite consultar todos os atributos de um cliente físico passando-se o nome, IDCliente ou CNPJ_CGC como parâmetros. Se o cliente não for encontrado será retornada 1.

getNumero_Clientes_Físicos ( ) :integer;

Retorna o número de registros de clientes fisicos existentes na base de dados de clientes.

getNumero_Clientes_Jurídicos ( ) :integer;

Retorna o número de registros de clientes jurídicos existentes na base de dados de clientes.

Interface IGerenciamento_Empresa

Cadastrar_Empresa (IDEmpresa, CNPJ/CGC, Nome, Rua, Número, Bairro, Cidade, Estado, CEP, Telefone, Email, Atividade): Integer;

Permite o cadastro de uma empresa. O parâmetro IDEmpresa deverá ser fornecido pelo próprio sistema . Este método retornará 0 se a empresa for cadastrada com sucesso e 1 se ocorrer algum erro durante a gravação dos dados no banco de dados.

Alterar_Empresa (IDEmpresa, CNPJ_CGC, Nome, Rua, Número, Bairro, Cidade, Estado, CEP, Telefone, Email, Atividade): Integer;

Permite Alterar o registro de uma empresa com o CNPJ_CGC ou IDEmpresa passados como parâmetro. Retorna 0 se a empresa for encontrada e seus dados alterados e a 1 se o registro não for encontrado.

Deletar_Empresa (IDEmpresa): Integer;

Permite remover uma empresa do arquivo de empresas. Retorna 0 se o registro for encontrado e removido e 1 caso contrário.

Consultar_Empresa (Nome, IDEmpresa, CNPJ_CGC): Integer;

Permite consultar uma empresa através de seu nome, IDEmpresa ou CNPJ_CGC passados como parâmetro. Retorna 1 se o registro não for encontrado.

getNumero_Empresas ( ) :integer;

Retorna o número de empresas existentes no arquivo de empresas.

Interface IGerenciamento_Recurso

Cadastrar_Recurso_Simples (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Simples, Status): Integer;

Permite o cadastro de recurso simples. O parâmetro IDRecurso deverá ser fornecido pelo próprio sistema, o parâmetro Cod_Simples deverá ser único para todos os recursos simples . Este método 0 se o recurso for cadastrado com sucesso e 1 se ocorrer algum erro durante a gravação dos dados no banco de dados.

Cadastrar_Recurso_Instanciável (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Instanciável, Status, Cód_Categ_Instanciável, Descrição): Integer;

Permite o cadastro de recurso instanciável. O parâmetro IDRecurso deverá ser fornecido pelo próprio sistema, os parâmetros Cod_Instanciável e Cód_Categ_Instanciável deverão ser únicos para todos os recursos instanciáveis . Este método retornará 0 se o recurso for cadastrado com sucesso e 1 se ocorrer algum erro durante a gravação dos dados no banco de dados.

Cadastrar_Recurso_Mensurável (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Mensurável, Unidade_Medida, Status, Cód_Categ_Mensurável, Descrição): Integer;

Permite o cadastro de recurso mensurável. O parâmetro IDRecurso deverá ser fornecido pelo próprio sistema, os parâmetros Cod_Mensurável e Cód_Categ_Mensurável deverão ser únicos para todos os recursos mensuráveis. Este método retornará 0 se o recurso for cadastrado com sucesso e 1 se ocorrer algum erro durante a gravação dos dados no banco de dados.

Cadastrar_Recurso_Lotes (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Lotes, Num_Lote, Status): Integer;

Permite o cadastro de recurso em lotes. O parâmetro IDRecurso deverá ser fornecido pelo próprio sistema, o parâmetro Cod_Lotes deverá ser único para todos os recursos em lotes . Este método retornará 0 se o recurso for cadastrado com sucesso e 1 se ocorrer algum erro durante a gravação dos dados no banco de dados.

Alterar_Recurso_Simples (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Simples, Status): Integer;

Permite alterar o registro de um recurso simples passando-se o IDRecurso como parâmetro. Este método Retorna 0 se o recurso for encontrado e tiver seus dados alterados com sucesso e 1 se o registro não for encontrado.

Alterar_Recurso_Instanciável (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Instanciável, Status, Cód_Categ_Instanciável, Descrição): Integer;

Permite alterar o registro de um recurso instanciável passando-se o IDRecurso como parâmetro. Este método Retorna 0 se o recurso for encontrado e tiver seus dados alterados com sucesso e 1 se o registro não for encontrado.

Alterar_Recurso_Mensurável (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Mensurável, Unidade_Medida, Status, Cód_Categ_Mensurável, Descrição): Integer;

Permite alterar o registro de um recurso mensurável passando-se o IDRecurso como parâmetro. Este método Retorna 0 se o recurso for encontrado e tiver seus dados alterados com sucesso e 1 se o registro não for encontrado.

Alterar_Recurso_Lotes (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Lotes, Num_Lote, Status): Integer;

Permite alterar o registro de um recurso em lotes passando-se o IDRecurso como parâmetro. Este método Retorna 0 se o recurso for encontrado e tiver seus dados alterados com sucesso e 1 se o registro não for encontrado.

Deletar_Recurso (IDRecurso): Integer;

Permite a remoção de um determinado recurso, tanto simples, instanciável, mensurável ou lotes, passando-se o IDRecurso como parâmetro. Retorna 0 se o cliente for excluído e 1 caso contrário.

Consultar_Recurso_Simples (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Simples, Status): Integer;

Permite consultar todos os atributos de um recurso simples passando-se o IDRecurso como parâmetro. Se o recurso não for encontrado será retornado 1.

Consultar_Recurso_Instanciável (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Instanciável, Status, Cód_Categ_Instanciável, Descrição): Integer;

Permite consultar todos os atributos de um recurso instanciável passando-se o IDRecurso como parâmetro. Se o recurso não for encontrado será retornado 1.

Consultar_Recurso_Mensurável (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Mensurável, Unidade_Medida, Status, Cód_Categ_Mensurável, Descrição): Integer;

Permite consultar todos os atributos de um recurso mensurável passando-se o IDRecurso como parâmetro. Se o recurso não for encontrado será retornado 1.

Consultar_Recurso_Lotes (IDRecurso, Nome, Descrição, Valor_Custo, Valor_Venda, Quantidade, Fabricante, Fornecedor, Cod_Lotes, Num_Lote, Status): Integer;

Permite consultar todos os atributos de um recurso lotes passando-se o IDRecurso como parâmetro. Se o recurso não for encontrado será retornado 1.

setAltera_Valor_Venda (IDRecurso): currency,

Permite alterar o valor de venda de um determinado recurso.

getNumero_Recurso_Simples ( ) :integer;

Retorna o número de recursos simples existentes na base de dados.

getNumero_Recurso_Intanciável ( ) :integer;

Retorna o número de recursos instanciáveis existentes na base de dados.

getNumero_Recurso_Mensurável ( ) :integer;

Retorna o número de recursos mensuráveis existentes na base de dados.

getNumero_Recurso_Lotes ( ) :integer;

Retorna o número de recursos lotes existentes na base de dados.

Interface Itransação

Iniciar_Transação (IDTransação, Data, Recurso_Envolvido, Quantidade_Recurso, Valor_Venda, Participante, Valor_Transação): Integer;

Permite startar e armazenar uma transação. O parâmetro IDTransação deverá ser fornecido pelo próprio sistema. Este método retornará 0 se a transação for armazenada sem ser concluída.

Consultar_Transação (IDTransação, Data): Integer;

Permite consultar uma transação através do IDTransação ou Data passados como parâmetros. Retorna 1 se o registro não for encontrado.

Alterar_Transação (IDTransação, Data, Recurso_Envolvido, Quantidade_Recurso, Participante, Valor_Transação): Integer;

Permite alterar o registro de um transação passando-se o IDTransação como parâmetro. Este método Retorna 0 se a transação for encontrado e tiver seus dados alterados com sucesso e 1 se o registro não for encontrado.

Deletar_Transação (IDTransação): Integer;

Permite a remoção de uma transação passando-se o IDTransação como parâmetro. Retorna 0 se a transação for excluída e 1 caso contrário.

Incluir_Participante_Físico (IDCliente, CPF): Integer;

Permite a inclusão de um participante físico passando-se o IDCliente ou CPF como parâmetros. Retorna 0 se o participante for incluído e 1 caso o registro não for encontrado.

Incluir_Participante_Jurídico (IDCliente, CNPJ_CGC): Integer;

Permite a inclusão de um participante jurídico passando-se o IDCliente ou CNPJ_CGC como parâmetros. Retorna 0 se o participante for incluído e 0 caso o registro não for encontrado.

Incluir_Participante_Empresa (IDEmpresa, CNPJ_CGC): Integer;

Permite a inclusão de um participante empresa passando-se o IDEmpresa ou CNPJ_CGC como parâmetros. Retorna 0 se o participante for incluído e 1caso o registro não for encontrado.

Incluir_Recurso_Simples (IDRecurso): Integer;

Permite a inclusão um recurso simples passando-se o IDRecurso como parâmetro. Retorna 0 se o recurso for incluído e 1 caso o registro não for encontrado.

Incluir_Recurso_Instanciável (IDRecurso): Integer;

Permite a inclusão um recurso instanciável passando-se o IDRecurso como parâmetro. Retorna 0 se o recurso for incluído e 1 caso o registro não for encontrado.

Incluir_Recurso_Mensurável (IDRecurso): Integer;

Permite a inclusão um recurso mensurável passando-se o IDRecurso como parâmetro. Retorna 0 se o recurso for incluído e 1 caso o registro não for encontrado.

Incluir_Recurso_Lotes(IDRecurso): Integer;

Permite a inclusão um recurso lotes passando-se o IDRecurso como parâmetro. Retorna 0 se o recurso for incluído e 1 caso o registro não for encontrado.

Set_Concluir_Transação (IDTransação): Integer;

Permite concluir uma transação, passando-se o IDTransação como parâmetro. Este método retornará 0 se a transação for concluída (status = concluída) com sucesso.

setValor_Transação (IDTRansação): currency;

Calcula o valor da transação em questão.

getNumero_Transações ( ); :integer;

Retorna o número de transações existentes na base de dados.

ILevantamento_Recurso

getTodos_Recurso ( ): String;

Retorna o nome e o código de todos os recursos

getRecursos_Simples ( ): String;

Retorna o nome, o código do recurso, e o código da classificação simples de todos os recursos simples cadastrados.

getRecursos_Intanciaveis ( ): String;

Retorna o nome, o código do recurso, o código da classificação instanciável e o código da categoria instanciável de todos os recursos instanciáveis cadastrados.

getRecursos_Mensuráveis ( ): String;

Retorna o nome, o código do recurso, o código da classificação mensurável, o código da categoria mensurável e a unidade de medida de todos os recursos simples cadastrados.

getRecursos_Lotes ( ): String;

Retorna o nome, o código do recurso, o código da classificação em lote e o número do lote de todos os recursos em lote cadastrados.

GetClassificação_Recurso ( ): String;

Retorna o nome e o código de todos os tipos de classificações cadastradas.

GetQuantificaçãoRecurso ( ):

Retorna o código e a descrição de todos os tipos de quantificações cadastradas

ANEXO B – DESCRIÇÃO DOS CASOS DE USO DO COMPONENTE BASE

Ator

Aplicação Cliente: É a aplicação (Software) do cliente que utiliza o Componente Base. Solicita a execução de tarefas específicas através de chamadas às operações disponibilizadas pelas Interfaces do Componente Base, passando os dados necessários à execução dessas tarefas. É, em geral, a fonte primária das informações a serem gerenciadas e armazenadas pelo Componente.

Pacote: Cadastro de Clientes, Empresas e Recursos

Caso de Uso: Cadastrar Clientes

Ator: Aplicação Cliente

Pré-Condição: O cliente não estar cadastrado.

Pós-Condição: O cliente cadastrado

Descrição:

1. A Aplicação Cliente faz uma chamada à uma das operações referentes ao cadastro de clientes, disponibilizadas pela Interface Igerenciamento_Clientes:

    - Cadastrar Cliente_Físico
    - Cadastrar Cliente_Jurídico

Passando como parâmetro seu nome, ID, CPF e os demais dados referentes ao tipo de cliente que está sendo cadastrado.

2. A Aplicação Cliente envia os dados referentes ao cliente a ser cadastrado para o Componente Base.

3. O Componente Base retorna uma mensagem confirmando se o cliente foi cadastrado ou não.

Caso de Uso: Cadastrar Empresas

Ator: Aplicação Cliente

Pré-Condição: A Empresa não estar cadastrada.

Pós-Condição: A Empresa cadastrada

Descrição:

1. A Aplicação Cliente faz uma chamada à operação Cadastrar_Empresa, disponibilizada pela Interface IGerenciamento_Empresa, passando como parâmetro seu nome, ID, CNPJ/CGC e os demais dados referentes à empresa que está sendo cadastrada.

2. A Aplicação Cliente envia os dados, referentes à empresa a ser cadastrada, para o Componente Base.

3. O Componente Base retorna uma mensagem confirmando se a empresa foi cadastrada ou não.

Caso de Uso: Cadastrar Recurso

Ator: Aplicação Cliente

Pré-Condição: O recurso não estar cadastrado.

Pós-Condição: O recurso cadastrado

Descrição:

1. A Aplicação Cliente faz uma chamada à operação referente ao cadastro de recursos, passando como parâmetro os dados referentes ao recurso a ser cadastrado:

    - Cadastrar_Recurso_Simples
    - Cadastrar_Recurso_Instanciável
    - Cadastrar_Recurso_Mensurável
    - Cadastrar_Recurso_Lotes

2. O sistema exibe as possibilidades de Quantificação do Recurso

3. A Aplicação Cliente seleciona a opção mais adequada para a quantificação do recurso.

4. a: Caso escolha a opção “Quantificar Recurso Simples”:

   - O sistema exibe o campo relacionado à opção desejada;
   - O sistema disponibiliza os campos referentes ao recurso que está sendo quantificado;
   - A Aplicação Cliente envia os dados para o Componente Base;

4. b: Caso escolha a opção “Quantificar Recurso Instanciável”:

   - O sistema exibe o campo relacionado à opção desejada;
   - O sistema disponibiliza os campos referentes ao recurso que está sendo quantificado.
   - O sistema permite a Classificação do Recurso, disponibilizando os campos referentes à descrição da classificação;
   - A Aplicação Cliente envia os dados para o Componente Base

4. c: Caso escolha a opção “Quantificar Recurso Mensurável”:

   - O sistema exibe os campos relacionados à opção desejada;
   - O sistema disponibiliza os campos referentes ao recurso que está sendo quantificado.
   - O sistema disponibiliza o campo para que a Unidade_Medida seja selecionada
   - O sistema permite a Classificação do Recurso, disponibilizando os campos referentes à descrição da classificação;

- A Aplicação Cliente envia os dados para o Componente Base;

4. d: Caso escolha a opção “Quantificar Recurso Lotes”:

- O sistema exibe o campo relacionado à opção desejada;
- O sistema disponibiliza o campo Número do Lote;
- A Aplicação Cliente envia os dados para o Componente Base;
- O Componente Base retorna uma mensagem confirmando se o recurso foi cadastrado ou não.

Pacote: Gravar Informações Cadastrais

Caso de Uso: Gravar_Informações_Clientes, Gravar_Informações_Empresa, Gravar_Informações_Recursos

Ator: Componente Base

Pré-Condição: Receber as informações da Aplicação Cliente.

Pós-Condição: Informações gravadas no banco de dados.

Descrição:

O Componente Base recebe as informações, referente ao que foi cadastrado, da Aplicação Cliente e envia essas informações para serem gravadas no banco de dados.

Caso de Uso: Levantar Recurso

Ator: Aplicação Cliente

Pré-Condição: O recurso estar cadastrado.

Pós-Condição: Todas as informações do cliente visualizadas.

Descrição:

1. A Aplicação Cliente seleciona o produto a ser cadastrado, através de seu Identificador ou Nome, e em seguida é mostrada todas as informações referente ao produto.

2. Caso o registro não for encontrado o sistema deverá retornar uma mensagem informando que o registro não foi encontrado.

Pacote: Transação

Caso de Uso: Efetuar Transação

Ator: Aplicação Cliente

Pré-Condição: Clientes, Empresas e Recursos estarem cadastrados.

Pós-Condição: Informações da Transação cadastradas.

Descrição:

1. A Aplicação Cliente seleciona o participante da Transação.

2. A Aplicação Cliente seleciona o recurso envolvido na Transação, de acordo com sua quantificação:

    - Selecionar Recurso_Simples
    - Selecionar Recurso_Instanciável
    - Selecionar Recurso_Mensurável
    - Selecionar Recurso_Lotes

3. A Aplicação Cliente faz uma chamada à operação que calcula o valor da Transação.

4. A Aplicação Cliente envia os dados referentes à Transação, em questão, para o Componente Base.

5. O Componente Base retorna uma mensagem confirmando o armazenamento das informações referentes à transação.

Pacote: Gravar Informações Transações

Caso de Uso: Gravar_Informações_Transações

Ator: Componente Base

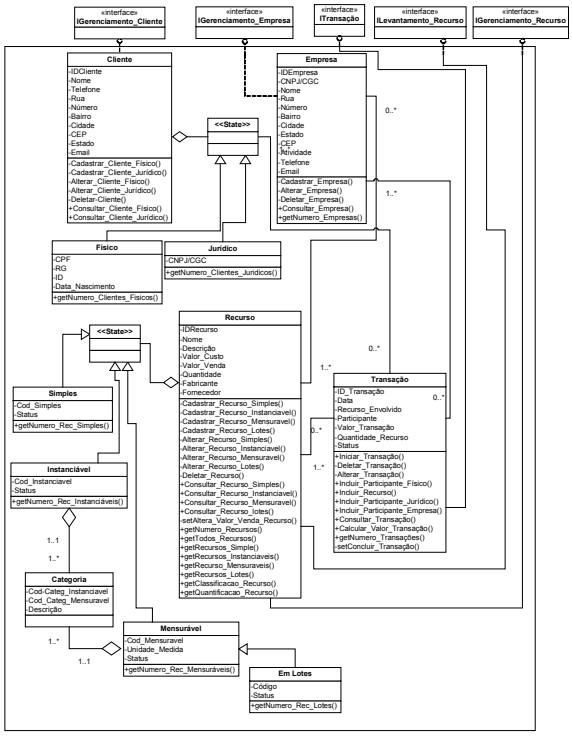
Pré-Condição: Receber as informações da Aplicação Cliente.

Pós-Condição: Informações da Transação gravada no banco de dados.

Descrição:

O Componente Base recebe as informações, referente ao que foi cadastrado, da Aplicação Cliente e envia essas informações para serem gravadas no banco de dados.

ANEXO C – DIAGRAMA DE CLASSES COM ATRIBUTOS E MÉTODOS



ANEXO D – DICIONÁRIO DE DADOS DO COMPONENTE BASE

CLIENTE			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
IDCliente	Integer		*
Nome	String [30]		
Rua	String [30]		
Numero	Integer		
Bairro	String [15]		
Cidade	String [30]		
CEP	Integer		
Estado	String [2]		
Telefone	Integer		
E-mail	String [30]		
Tipo	String [10]		

CLIENTE FÍSICO			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
CPF	Integer		
IDCliente	Integer		*
RG	String [10]		
Data_Nascimento	TDateTime		

CLIENTE JURÍDICO			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
CNPJ_CGC	String [18]		
IDCliente	Integer		*

EMPRESA			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
IDEmpresa	Integer		*
CNPJ_CGC	String [18]		
Nome	String [30]		
Rua	Rua [30]		
Número	String [8]		
Bairro	String [15]		
Cidade	String [30]		
CEP	Integer		
Estado	String[2]		
Telefone	Integer		
Email	String [30]		

TRANSAÇÃO			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
IDTransacao	Integer		*
Data	TDateTme		
Recurso_Envolvido	String [20]		
Participante	String [30]		
Valor_Transação	Currency		
Quantidade_Recurso	Integer		

RECURSO			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
IDRecurso	Integer		*
Nome	String [20]		
Valor_Custo	Currency		
Valor_Venda	Currency		
Fabricante	String [20]		
Fornecedor	String [20]		
Quantidade	Integer		
Descrição	String [50]		
Tipo	String [10]		

RECURSO SIMPLES			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
IDRecurso	Integer		*

RECURSO INSTANCIÁVEL			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
IDRecurso	Integer		*

RECURSO MENSURÁVEL			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
IDRecurso	Integer		*
Unidade Medida	String [15]		

RECURSO EM LOTES			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
IDRecurso	Integer		*
Número_Lote	String [10]		

CATEGORIA			
ATRIBUTO	TIPO	OBSERVAÇÃO	CHAVE
IDCategoria	Integer		*
Descrição	String [50]		

ANEXO E – TIPOS DE DADOS SUPORTADOS PELA IDL

	TIPO	DESCRIÇÃO
BÁSICOS	**(unsigned) short**	inteiro 16 bits - signed [unsigned]
	(unsigned) long	inteiro 32 bits - signed [unsigned]
	float	número vírgula flutuante - 36 bits.
	double	número vírgula flutuante - 64 bits.
	char	caracter ISO Latin-1 (8859.1).
	boolean	tipo lógico/boleano. Toma os valores verdade (true) ou falso (false).
	string	conjunto de caracteres.
	octet	tipo genérico de 8 bits.
	enum	identificadores com numeração implícita.
	any	pode representar um valor de qualquer um dos tipos IDL, básico ou estruturado.

ESTRUTURADOS	**structure**	registo:
		struct struct_type_name {
		type1 member_name1;
		type2 member_name2;
		};

	union	é o cruzamento das declarações union e switch do C:
		union union_type_name switch(discriminator_type) {
		case value1 : type1 member_name1;
		case value2 : type2 member_name2;
		default : type3 member_name3;
		}

	array	lista indexada de comprimento fixo.
		typedef array_type_name1 member_type1(10);
		typedef array_type_name2 member_type2(10)(60);

	sequence	lista indexada de comprimento variável que pode ter um limite superior. typedef bounded_seq_type_name sequence <member_type1, 30>; typedef unbounded_seq_type_name sequence <member_type2>;

ANEXO F – INTERFACE IGERENCIAMENTO_EMPRESA, ESCRITA NA LINGUAGEM JAVA

```
public interface IGerenciamento_Empresa {

        public String Cadastrar_Empresa (int idempresa, String cnpj_cgc, String nome,
                                         String rua, int numero, String bairro,
                                         String Cidade, String estado, int cep,
                                         String atividade, int telefone, String email);

        public String Alterar_Empresa (int idempresa, String cnpj_cgc);

        public String Deletar_Empresa (int idempresa);

        public String Consultar_Empresa(String nome, String cnpj_cgc);

        public int getNum_Empresa ();
}
```

ANEXO G – DISQUETE CONTENDO TODAS AS INTERFACES DO COMPONENTE BASE – ESCRITAS EM JAVA E IDL