GLAUCIA GABRIEL SASS

PROPOSTA DE UMA ARQUITETURA DE COMPONENTES PARA SISTEMAS DE INFORMAÇÃO BASEADOS NA WEB: NÍVEL LÓGICO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática pelo Curso de Pós-Graduação em Informática, do Setor de Ciências Exatas da Universidade Federal do Paraná, em convênio com o Departamento de Informática da Universidade Estadual de Maringá.

Orientadora: Prof.ª Dr.ª Itana Maria de Souza Gimenes

CURITIBA
2003

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, da aluna *Glaucia Gabriel Sass*, avaliamos o trabalho intitulado, *"Proposta de uma Arquitetura de Componentes para Sistemas de Informação Baseados na WEB: Nível Lógico"*, cuja defesa foi realizada no dia 11 de dezembro de 2003, às nove e trinta horas, no Auditório da Informática da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação da candidata.

Curitiba, 11 de dezembro de 2003.

Profª. Dra. Itana Maria de Souza Gimenes
UEM – Orientadora

Prof. Dr. Victor Francisco Araya Santander
UNIOESTE - Membro Externo

Prof. Dr. Marcos Sfair Sunye
DINF/UFPR - Membro Interno

Dedico ao meu esposo Sergio Ricardo,
meu filho Thales Augusto,
minha mãe Hulda
e a meus avós Augusto e Querubina

AGRADECIMENTOS

Primeiramente a Deus, que me guia pelos caminhos a serem seguidos, sempre presente em minha vida, ajudando-me em todos os momentos, auxiliando minhas decisões.

A toda minha família, principalmente a meu esposo, pelo apoio e por entender minhas ausências, e a meu filho que mesmo sem entender a minha ausência, sempre me recebia de braços abertos. A todos que sempre me diziam "não desanime".

Aos meus alunos André Luis Borges de Paula e Eduardo Hiroshi Nakamura pelo auxílio nos trabalhos. E a todas as turmas de graduação, que compreenderam a minha ausências em sala de aula.

A todos aqueles que de forma direta ou indireta, influenciaram e incentivaram a realização deste trabalho.

Agradecimento especial à Profª. Itana, pessoa que Deus colocou em meu caminho para me ensinar que os obstáculos são superados um de cada vez. Agradeço também a ajuda e compreensão das minhas limitações e dificuldades.

Instruir-te-ei, e ensinar-te-ei o caminho que deves seguir,
guiar-te-ei com os meus olhos.

Salmo 32:8.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

B2B	- *Business-to-Business*
B2C	- *Business-to-Consumer*
C2C	- *Consumer-to-Consumer*
CIS	- *Community Information Systems*
CMM	- *Capability Maturity Model*
COTS	- *Commercial Off-The-Self*
DBC	- Desenvolvimento Baseado em Componentes
EJB	- *Enterprise JavaBeans™*
G2B	- *Government-to-Business*
G2C	- *Government-to-Citizen*
G2G	- *Government-to-Government*
GUI	- *Graphical User Interface*
HTML	- *HyperText Markup Language*
J2EE	- *Java™2 Plataform, Enterprise Edition*
JSP	- *Java Server Page*
OCL	- *Object Constraint Language*
OO	- Orientação a Objetos
SGBD	- Sistema Gerenciador de Banco de Dados
SI	- Sistema de Informação
WbIS	- *Web-based Information System*

RESUMO

A *Web* vem apresentado um grande crescimento nos últimos anos. Os Sistemas de Informação baseados na *Web ou Web based Information System (WbIS)* oferecem recursos para automatizar e gerenciar os negócios. Esses sistemas demandam técnicas de engenharia de software que facilitem a sua construção para obtenção de produtos com alta qualidade e que sejam economicamente viáveis. A reutilização está entre essas técnicas, partindo do princípio que, reutilizando partes bem especificadas, desenvolvidas e testadas, é possível construir software em menor tempo e com maior confiabilidade. Entre as técnicas de reutilização estão: *frameworks*, padrões, arquitetura de software e desenvolvimento baseado em componentes.

Os *WbIS* podem ser categorizados dentro do domínio de aplicação para *Web*. Dentro das diversas categorias de *WbIS* é possível encontrar semelhanças entre seus artefatos, podendo definir assim restrições de uso e disponibilização de produtos e serviços. A categoria que apresenta uma maior demanda por sistemas é a de *e-commerce*. Para os *WbIS* falta uma proposta de arquitetura que reúna um conjunto de componentes que possam ser aplicados ao desenvolvimento de sistemas semelhantes, como aqueles na categoria de *e-commerce*.

Esta dissertação apresenta uma proposta de arquitetura de componentes para *WbIS*. Esta arquitetura visa facilitar o desenvolvimento de sistemas *e-commerce* que possuam características comuns, mas que também possuam aspectos diferenciados de acordo com o negócio. O desenvolvimento da arquitetura proposta tomou como base o *CIS Framework*. O processo de desenvolvimento da arquitetura seguiu o método *UML Components*. A arquitetura foi obtida por meio da análise de domínio e desenvolvimento de estudos de casos.

Palavras-chaves: Sistemas de informação baseados na *Web;* Arquitetura de software; Desenvolvimento baseado em componentes; *UML Components;* Reutilização.

ABSTRACT

The Web has been significantly increasing in the past years. Web-based Information Systems (WbIS) provide means for business management and automation. These systems demand software engineering techniques that make their production easier so that high quality and low cost products can be obtained. Reuse is one of these techniques as by reusing well specified and tested artefacts it is possible to build more reliable software at lower costs. Reuse techniques include frameworks, patterns, software architecture and component based development. WbIS can be categorized within the domain of Web applications. This allows the establishment of similarities and differences amongst their artefacts that enable faster generation of products and services of these kind of applications from a well defined infrastructure. The Web application category that presents a greater demand for products is that of e-commerce. There is a demand for a generic architecture and components that can be applied to facilitate the production of these systems. This work aim to fill this gap. It proposes an architecture and a set of domain specific components for e-commerce applications. The architecture development was based on the CIS (Communitarian Information Systems) Framework The method followed to obtain the architecture was UML Components. This method was applied to case studies to form the basis for the proposed generalized architecture and their components.

Keywords: Web-based information systems; Software architecture; Component-based development; UML Components; Reuse.

1 INTRODUÇÃO

Uma arquitetura de software bem definida possibilita a reutilização de um conjunto de estruturas em sistemas similares. Em uma organização que produz sistemas semelhantes, a utilização da mesma arquitetura e de seus elementos para o desenvolvimento dos sistemas, pode trazer benefícios como a redução dos custos de produção e do tempo para disponibilizar este sistema no mercado.

Uma arquitetura de software é uma estrutura que inclui componentes de software, suas propriedades externamente visíveis e os relacionamentos entre esses componentes (BOSCH, 2000). A idéia de reutilização de componentes por meio de uma arquitetura tem motivado o estudo dos possíveis componentes que podem ser agregados a um Sistema de Informação baseado na *Web* (*Web based Information System - WbIS*).

Os métodos de desenvolvimento baseado em componentes (DBC) podem basear-se no desenvolvimento de componentes específicos do Sistema de Informação (SI) e/ou adquirir componentes de terceiros e através da arquitetura fazer a conexão entre estes diversos componentes.

Os *WbIS* são sistemas de informação que permitem combinar as características dos sistemas de informação tradicionais com as vantagens da *Web*. Os *WbIS* têm como foco as grandes comunidades de usuários e a computação baseada em redes de computadores, quer local, quer a longo alcance (SILVA; DELGADO, 2002).

A quantidade de recursos de informações e de aplicações da *Web* nas diversas áreas do conhecimento tem crescido bastante, apresentando novos desafios de pesquisa para o desenvolvimento dos sistemas de informação. A área que mais cresce é a do comércio eletrônico (*e-commerce*). As transações comerciais realizadas na *Web* têm representado um diferencial para o crescimento das empresas.

A relevância em projetar uma arquitetura para *WbIS* está no aumento da qualidade, na redução do tempo e custo de construção desses sistemas. A

velocidade em que a *Web* cresce e a necessidade que as empresas têm em manter-se no mercado gera uma busca por componentes que possam ser cada vez mais genéricos e reutilizáveis. Baseado nos conceitos de DBC e na necessidade atual justifica-se a criação desta arquitetura implantando o que a comunidade de software vem buscando a tempo, a efetiva reutilização de artefatos de software. Cada implementação de um *e-commerce* pode adaptar componentes ou interfaces de acordo com as necessidades específicas da aplicação. *WbIS* com características semelhantes, porém com especificidades diferentes são necessários nas mais diversas empresas que pretendem conquistar uma parte do mercado *Web*. O domínio dos *WbIS* é altamente favorável à aplicação de uma arquitetura genérica que possa ser ajustada à maioria dos *e-commerce*.

Esta dissertação apresenta uma proposta de arquitetura de componentes para *WbIS*, nível lógico. Por meio dos estudos realizados como base para a proposta, identificou-se que havia estudos sobre os componentes que integram os níveis de infra-estrutura e de apresentação que compõem as arquiteturas *WbIS*, mas para o nível lógico havia somente algumas suposições sobre os possíveis componentes que integrariam, sendo assim escolhido como objeto de estudo para esta dissertação. Para a realização do estudo foi necessária à categorização dos *WbIS*, dentre as categorias foi escolhida a de *e-commerce B2C*. Esta arquitetura pode ser usada para facilitar o desenvolvimento de *WbIS*.

O desenvolvimento da arquitetura tomou como base o *framework* proposto por ALENCAR, COWAN e LUO (2002), a arquitetura *CIS Framework*. O processo de desenvolvimento da arquitetura seguiu o método *UML Components* (CHEESMAN; DANIELS, 2000), que tem como foco um processo de DBC. Para fins de validação da arquitetura foram desenvolvidos dois estudos de caso para *e-commerce*.

A principal contribuição deste trabalho é uma arquitetura de componentes para *WbIS*, em particular *e-commerce*. Pode-se com a identificação desses componentes desenvolver *WbIS* mais confiáveis e em menos tempo.

O texto está organizado da seguinte forma. O capítulo 2 apresenta os principais conceitos relacionados aos *WbIS*, sua categorização e suas aplicações nas mais diversas áreas do conhecimento.

O capítulo 3 apresenta conceitos relacionados à arquitetura de software, dando ênfase ao estilo de arquitetura usado para o desenvolvimento de *WbIS*. Também apresenta conceitos sobre DBC, e enfoca o método *UML Components* usado para o desenvolvimento deste trabalho.

O capítulo 4 apresenta os trabalhos relacionados a esta dissertação, abordando entre eles os conceitos do *CIS Architecture Framework with Patterns* (ALENCAR; COWAN; LUO, 2002).

O capítulo 5 apresenta a arquitetura de componentes para *WbIS*, bem como o processo seguido para o projeto desta arquitetura.

No capítulo 6 são apresentados às conclusões e os trabalhos futuros que podem ser realizados a partir deste estudo.

O apêndice 1 apresenta o estudo de caso *e-maganize*, desenvolvido para identificação dos componentes da arquitetura.

O apêndice 2 apresenta a aplicação da arquitetura no estudo de caso *e-rent-a-car* para validação da mesma.

2 SISTEMAS DE INFORMAÇÃO BASEADOS NA *WEB*

Na última década, os SIs evoluíram juntamente com a Internet. A *Web* trouxe novos horizontes para a distribuição e acesso às informações, mas com ela também surgiram novos problemas relacionados a segurança, portabilidade e manutenção, entre outros. Este capítulo conceitua o ambiente *Web* (seção 2.1), apresenta os sistemas de informação baseados na *Web*, seu funcionamento (seção 2.2) e a categorização dos *WbIS* (seção 2.2.1). Apresenta também como a engenharia para *Web* (seção 2.3) está tratando o desenvolvimento desses sistemas.

2.1 WORLD WIDE WEB

A *World Wide Web*, ou simplesmente *Web*, é um serviço oferecido pela Internet. A Internet foi originalmente estabelecida como um ambiente que permitia as agências governamentais, contratantes, universidades e instalações de pesquisa compartilharem o acesso a informações de maneira aberta. Foi fundada pelo Departamento de Defesa dos Estados Unidos da América em 1969, originalmente chamada de *Arpanet*. Para entender a *Web* é preciso entender a Internet, desde os detalhes técnicos até compreender que a *Web* é executada exclusivamente no topo da Internet (MATTISON; KILGER-MATTISON, 1999).

Para ter uma visão rápida da Internet, pode-se dizer que, basicamente ela tem uma arquitetura simples e direta, em que um grupo de computadores se comunica e compartilha funções de administração de rede de acordo com um conjunto de regras pré-estabelecidas.

A *Web* permite o acesso a documentos ligados e espalhados por milhares de máquinas na Internet. Em pouco tempo ela deixou de ser um meio de distribuir dados extremamente técnicos para se tornar a aplicação que milhões de pessoas consideram ser "A Internet". Sua enorme popularidade se deve à sua interface gráfica, que é de fácil utilização (TANEMBAUM, 1997).

A *Web* é atualmente utilizada para transmissão e disseminação de informações, acesso a dados e comunicação social (LIMA; LIFSCHITZ, 1998). Os documentos armazenados na *Web* podem ser de vários tipos, os mais comuns são documentos hipermídias[1].

O usuário da *Web* enxerga textos formatados ilustrados por imagens. Esses textos possuem ligações associadas aos documentos que ao serem selecionadas apresentarão o conteúdo do documento referenciado. A interface de hipertexto[2] da *Web* é uma interface de navegação poderosa e visualmente atrativa.

As informações disponibilizadas pela *Web* muitas vezes não seguem nenhum padrão ou método específico para sua composição. Isso pode acarretar uma série de problemas, relacionados a manutenção, corretude da informação e segurança. O desenvolvimento desenfreado de *WbIS* tem causado uma série de transtornos. O crescimento e a demanda exagerada obrigam os profissionais a não usarem o "tempo" ou as "condições" necessárias para desenvolver ou seguir um método para organizar seus SIs.

2.2 CONCEITOS DE SISTEMAS DE INFORMAÇÃO BASEADOS NA *WEB*

Originalmente a *Web* era um sistema hipermídia distribuído para navegação e consulta de informação fracamente estruturada. Com o passar do tempo novas necessidades foram surgindo, então houve a introdução de novas capacidades e funcionalidades que já existiam nos sistemas de informação tradicionais, tais como formulários e interligação a bases de dados (SILVA; DELGADO, 2002).

O'BRIEN (2001) considera que um SI consiste em: pessoas, *hardware*, software, dados e redes. Os SIs utilizam esses recursos para transformar dados em produtos de informação. Com o surgimento da *Web*, o recurso rede foi estendido,

---

[1] Hipermídia: banco de dados de hipertexto consistindo de diferentes tipos de informação (texto, fotografias, som, voz e vídeo).

[2] Hipertexto: é a combinação de textos não lineares, de forma que se pode saltar de um texto para outro dinamicamente por meio de ligações (*links*).

agora com alcance mundial. A forma como os sistemas são disponibilizados mudou, assim como, a localização física do software e dos dados.

Atualmente a tecnologia para desenvolvimento e suporte a *WbIS* é alvo de interesse e investimento, porque permite combinar os sistemas de informação tradicionais com as vantagens da *Web*.

Existem alguns conceitos básicos (BEM-NATAN, 1997) importantes para o desenvolvimento de *WbIS*. Um *WbIS* pode ser dividido em duas partes principais - o *back-end* e o *front-end*. O *back-end* é a máquina computacional responsável pelas funções do *WbIS* e pela administração da informação, implementando a lógica do negócio e administrando os dados. O *front-end* é a interface que permite aos usuários do *WbIS* ver os dados e ativar seus processos. Todo *WbIS* em geral tem algum tipo de *front-end*.

Para construção do *front-end* são usadas interfaces gráficas (*Graphical User Interface – GUI*). O uso de *GUI* nos *front-end* é uma prática comum, utilizada com a tecnologia cliente/servidor na qual o cliente quase sempre é um *GUI*, junto com uma máquina de processo que envia dados do cliente para o servidor. Em sistemas de processamentos distribuídos é muito comum o uso de *front-ends* separados da lógica do SI.

O *front-end* usualmente reside em uma máquina separada do *back-end* Estas máquinas são conectadas por alguma infra-estrutura de comunicação, uma rede local, por exemplo. Os usuários acessam funcionalidades do *WbIS* usando o *front-end*. Dependendo da segmentação da parte lógica do *WbIS*, cada operação executada pelo usuário causará a comunicação entre o *front-end* e o *back-end* requerendo vários serviços.

O *front-end* sempre gerencia a apresentação do usuário e captura as entradas feitas por ele. Este pode implementar parte da lógica do *WbIS*, neste caso são freqüentemente chamados *fat front-ends* (*front-end* "gordo"). Neste caso, o *back-*

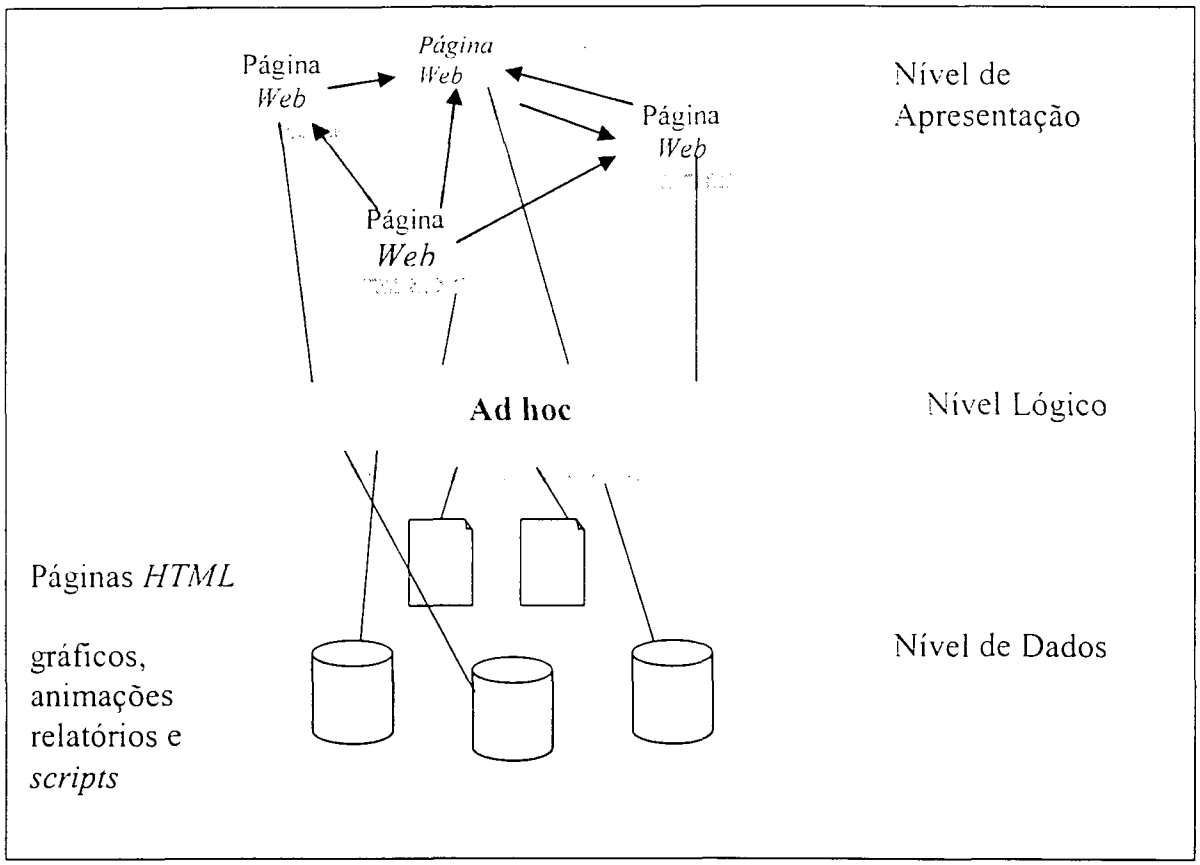
end funciona usualmente como um servidor de dados, servindo como elemento de estrutura de dados para o *front-end*.

Por outro lado existe o *front-end* que gerencia somente a apresentação, o teclado e o *mouse*, freqüentemente chamado de *thin front-end* (*front-end* "magro"). O *front-end* recebe a apresentação direto do *back-end* e envia os dados de entradas do usuário para o *back-end*. O *back-end* então executa algumas funções do *WbIS* e cria uma nova apresentação para ser usada pelo *front-end*.

Uma arquitetura básica para *WbIS* pode ser vista como uma arquitetura *thin front-end*. Neste caso o *front-end* é o *Web Browser*. Este recebe páginas *HTML (HyperText Markup Language)* que usa diretivas de apresentação, interpreta os elementos de marcação e organiza os dados na tela. Quando um evento é ativado na página *HTML*, como um clique no botão enviar de um formulário, as informações são empacotadas e enviadas ao servidor. Neste caso, o servidor ativa um *gateway*, dispositivo de tradução de protocolo, que retorna a informação ao *front-end*. O *Web Browser* representa o *front-end* do *WbIS*, enquanto o *back-end* neste ambiente vai ser representado pelo servidor e os programas *gateway*.

Segundo essas definições, pode-se entender a arquitetura geral de um *WbIS* como uma arquitetura três camadas, como apresentado na FIGURA 1.

FIGURA 1 - ARQUITETURA EM TRÊS CAMADAS PARA *WbIS*



FONTE: BEM-NATAN, 1997.

No nível de apresentação, o ponto principal é, "como" as informações serão apresentadas. Neste nível, deve-se escolher as informações que serão agrupadas, ou separadas em *hiperlinks*[3], dentro das unidades de apresentação, como páginas da *Web*. Os elementos que compõem este nível são processados no cliente *Web* onde está o *front-end*. O nível de dados descreve como as informações serão organizadas fisicamente, em termos de quais softwares de aplicação (ex. gerenciador de banco de dados e editor gráfico) serão necessários e quais arquivos serão usados. Neste nível o processamento é feito no servidor *Web* representando parte do *back-end*. Entre o nível de dados e o de apresentação, está o nível lógico que realiza o mapeamento das informações que serão armazenadas em bancos de dados e nos servidores *Web*. É mais comum processar os elementos do nível lógico no servidor *Web*, fazendo parte do *back-end*, mas isso nem sempre acontece. Os elementos que compõem o nível lógico podem ser processados no cliente *Web*, ou

[3] *Hiperlink*: ligações.

divididos em partes para ser processado tanto no cliente quanto no servidor *Web*, dividindo assim a localização física do *back-end*, que pode ser espalhada dentro de uma empresa (*Intranet*) ou em uma extensa área geográfica (Internet).

Os SIs que seguem esta estrutura são classificados conforme o domínio. Para o domínio *Web* existe uma classificação segundo os agentes envolvidos no processo, como apresentado na próxima seção.

2.2.1 Categorização dos *WbIS*

A Internet tornou-se o mais inovador ambiente para realização das mais diversas formas de transações comerciais. Os tradicionais SIs se adequaram para acompanhar a evolução. Como os SIs tradicionais, os *WbIS* também possuem suas classificações.

Existem várias classificações de *WbIS*, por ser uma tecnologia nova, não existe uma classificação única e padronizada. Dentre os materiais pesquisados, várias denominações e hierarquias desses SIs, foram encontradas, tentou-se extrair os conteúdos semelhantes de cada material para poder definir um padrão a ser seguido neste trabalho.

Em geral, as transações comerciais realizadas na *Web* são chamadas genericamente de *e-commerce*, mas essas transações comerciais são geralmente classificadas segundo os agentes envolvidos (TIGRE, 2003):

a) *Business-to-Business* (*B2B*) envolvendo a articulação das cadeias produtivas e a compra e venda de produtos e serviços pelas empresas;

b) *Business-to-Consumer* (*B2C*) focalizando o relacionamento entre empresas e seus clientes finais;

c) *Consumer-to-Consumer* (*C2C*) que liga pessoas físicas interessadas em realizar transações; e

d) o governo constitui outro agente de relacionamento eletrônico com empresas e indivíduos, através do *Government-to-Business* (*G2B*), *Government-to-Citizen* (*G2C*) e *Government-to-Government* (*G2G*).

O *e-commerce* é mais do que a mera compra e venda de produtos *on-line*. Ele engloba o processo de desenvolvimento do SIs *on-line* inteiro incluindo: *marketing*, venda, entrega, atendimento e pagamento por produtos e serviços comprados por comunidades mundiais de clientes virtuais, com o apoio de uma rede mundial de parceiros comerciais. O *e-commerce* pode incluir, por exemplo, processos de *marketing* interativo, pedidos e pagamentos na *Web*, acesso por *Extranet* a bancos de dados de estoque pelos clientes e fornecedores, acesso por *Intranet* a cadastro de clientes por representantes de vendas e atendimento ao consumidor e no desenvolvimento de produtos via grupo de notícias da *Web* e troca de *e-mail* (O'BRIEN, 2001).

A seguir é apresentada a definição de cada uma das classificações encontradas.

2.2.1.1 Comércio empresa-consumidor

No comércio empresa-consumidor (*B2C*), as empresas precisam desenvolver praças de mercado eletrônico atraentes para seduzir e vender produtos e serviços aos consumidores. As empresas podem oferecer, por exemplo, *sites* em multimídias que apresentam fachadas de lojas virtuais e centros comerciais virtuais, processamento interativo de pedidos e sistemas seguros de pagamentos eletrônicos.

Características desejáveis no comércio eletrônico B2C segundo (O'BRIEN, 2001) são:

a) desempenho e serviço: as pessoas não desejam ser mantidas na espera ao consultarem, escolherem ou pagarem por produtos em uma loja eletrônica. O *site* deve ser eficientemente projetado para facilidade de acesso, escolha e compra. Os processos de *marketing*, pedidos e atendimentos ao cliente também devem ser amigáveis e vantajosos, além de rápidos e fáceis;

b) personalização: personalizar a experiência de compra encoraja o comprador a fazer novas visitas. Muitos locais cadastram o cliente ou lhe pedem para traçar um perfil de seus interesses pessoais. Quando o cliente volta a acessar o *site*, é cordialmente saudado, e dirigido apenas às partes do *site* nas quais o cliente está interessado. Muitos *site* registram automaticamente detalhes de suas visitas e montam perfis de usuários com interesses similares;

c) socialização: dar aos clientes *on-line* com interesses semelhantes à sensação de pertencer a um grupo único de indivíduos de mentalidade parecida, ajuda a aumentar o valor para o cliente e sua lealdade. O *marketing* de relacionamento e o *marketing* de afinidades baseados na *Web* promovem essas comunidades virtuais de clientes, fornecedores, representantes da empresa e outros consumidores por meio de várias técnicas. Os exemplos incluem fóruns de discussão e salas de bate-papo entre os clientes, grupos de foco de produto e interligações para outros *sites* e grupos de notícias relacionados;

d) aparência e impressão: os *sites* na rede podem oferecer ao cliente uma fachada de loja virtual atraente. Isto pode variar desde o fornecimento de uma experiência de compra emocionante, com som, movimento e gráficos impressionantes, até uma aparência e impressão mais convencionais. O software da *Oracle* (ORACLE CORPORATION, 2002b), por exemplo, permite que os compradores consultem seções de produto, escolham produtos, coloquem-nos em um carrinho de compras virtual e se dirijam para um caixa virtual para pagarem pelo pedido;

e) incentivos: os *sites* da rede precisam oferecer aos compradores incentivos à compra e ao retorno. Normalmente, isto é feito por cupons, descontos, ofertas especiais e vales para outros serviços da rede, às vezes em outros *sites* interligados. Alguns *sites* lhe fornecem uma carteira eletrônica em que o cliente pode acumular cupons para uso futuro, além de recibos e informações sobre seu cartão de crédito;

f) segurança e confiabilidade: o cliente de uma loja da *Web* precisa sentir confiança de que seu cartão de crédito, informações pessoais e detalhes de suas transações estejam protegidos contra uso sem autorização. Medidas importantes para a confiabilidade de uma loja da *Web* é o preenchimento correto de seus pedidos e a entrega no prazo prometido.

2.2.1.2 Comércio empresa-empresa

SI empresa-a-empresa (*B2B*), ou *e-business*, envolvem mercados comerciais eletrônicos e ligações diretas de mercado entre as empresas. Muitas empresas, por exemplo, oferecem à comunidade dos negócios uma multiplicidade de informações de *marketing* e produtos da *Web*. Outras também recorrem ao intercâmbio eletrônico de dados (*EDI*)[4] pela Internet ou *Extranet* para a troca direta computador-a-computador de documentos de transações comerciais com seus clientes e fornecedores.

O *e-commerce B2B* é o lado atacadista do processo comercial. Por exemplo, uma empresa deseja montar e vender um produto para outras empresas. Ela precisa comprar matérias-primas e uma série de serviços contratados de outras empresas. As inter-relações com outras empresas, necessárias para montar e vender um produto, constituem uma rede de relações comerciais que é chamada de cadeia de suprimentos. Sistemas de *e-commerce*, como o *EDI*, e processos de administração das empresas, como o gerenciamento da cadeia de suprimento, procuram reformular e agilizar os processo tradicionais.

Entre as tecnologias empregadas nesta categoria está incluído o *e-mail*, formulários comerciais eletrônicos, *EDI*, transferências eletrônicas de fundos, *sites* com informações de *marketing* e catálogo de produtos multimídias, e sistemas de processamento interativo de pedidos.

---

[4] *EDI – Eletronic Data Interchange* – troca eletrônica de dados ou troca eletrônica de documentos. Utiliza sistemas de comunicação entre empresas para a troca de dados e documentos tais como uma montadora e os seus fornecedores, um distribuidor de medicamentos e as farmácias, um banco e os seus melhores clientes., etc.

ALLEN (2001) relaciona os tipos de serviços que poderiam fazer parte de uma solução *e-business*. Aqui estão alguns exemplos:

a) relacionamento com cliente: um grupo de serviços para descobrir, armazenar, manter e usar perfis de clientes. Isto inclui todas as interações do cliente com a organização, localizando tendências de comportamentos e fazendo isto em tempo real;

b) catálogo de produto: um grupo de serviços apropriado e acesso consistente as informações dos produtos do catálogo incluindo estimativas, ofertas especiais, níveis de estoque e "estratégias";

c) pagamento: um grupo de serviços fornecendo verificação de crédito e administração das transações de pagamentos, é altamente provável que isto seja fornecido por um provedor externo de serviços financeiros;

d) pedidos: um grupo de serviços que lida com entradas de pedidos através da entrega, mantendo o cliente atualizado, possivelmente por *e-mail*, do estado do pedido;

e) lista: um grupo de serviços que fornece uma versão pública e digital de telefones que permite aos clientes negociar com as pessoas da organização.

2.2.1.3 Comércio consumidor-consumidor

O comércio consumidor-consumidor (*C2C*) possui as mesmas características do *B2C*, mas realiza transações entre consumidores finais. Exemplos deste tipo de comércio são os *sites* de leilões e classificados *on-line*. As características indicadas por O'BRIEN (2001) também são desejáveis neste tipo de SI. Os SIs que oferecem este tipo de serviço têm que garantir desempenho, oferecer personalização, socialização, incentivos, segurança e confiabilidade além de possuírem características agradáveis na aparência e impressão.

2.2.1.4 Transações com o governo

Os governos mundiais têm, sem dúvida, grande papel nas revoluções e, como em todo tipo de evolução, acabarão sendo superados se não acompanharem o ritmo do progresso. Isso ocorrerá porque a Internet modificou a maneira como organizações, comunidades e indivíduos estudam, trabalham e interagem.

Criar um *e-government* (governo eletrônico) não é somente "usar alguns computadores ou construir um *site* para acesso a informação; e sim transformar a relação fundamental entre o governo e o público" (Pardo, 2000).

E-government significa utilizar as novas tecnologias digitais para habilitar os moradores, contribuintes, visitantes, enfim, cidadãos, o acesso as informações e aos serviços outrora indisponíveis eletronicamente, bem como uma interação com seu governo.

Em outras palavras, *e-government* é uma maneira dos governos usarem as novas tecnologias de informação para fornecer às pessoas um acesso mais conveniente às informações e serviços governamentais, para, assim, melhorar a qualidade destes serviços e possibilitar maiores oportunidades para participação popular em processos e instituições mais democráticas.

O *e-government* trás junto aos seus conceitos tecnológicos, a proposta de criação e o desenvolvimento de uma economia inovadora e socialmente inclusiva. Mesmo porque a idéia principal do *e-government* é a inclusão digital. Essa inclusão também se ajusta perfeitamente às necessidades do *e-commerce*. Juntos, o *e-government* e o *e-commerce* estão criando um novo nicho econômico de alta representatividade na economia geral dos países.

Dentro dos SIs classificados como *e-government* é possível ainda fazer três classificações internas:

a) os SIs de transação Governo-Empresa (*G2B*): corresponde a ações do governo que envolve interação com entidades externas. O exemplo mais

concreto deste tipo é a condução de compras, contratações e licitações, via meios eletrônicos;

b) os SIs de transação Governo-Cidadão (*G2C*): corresponde às ações do governo de prestação (ou recebimento) de informações e serviços ao cidadão via meios eletrônicos. O exemplo mais comum deste tipo é a veiculação de informações em um *site* de um órgão do governo, aberto a quaisquer interessados;

c) os SIs de transação Governo-Governo (*G2G*): corresponde às funções que integram ações do governo horizontalmente (exemplo: em nível Federal, ou dentro do Executivo) ou verticalmente (exemplo: entre o Governo Federal e um Governo Estadual).

Segundo COOK (2002) *e-government* é o uso da informática para apoiar operações do governo, "'contratos com" cidadãos, e fornecer serviços do governo. Dentro desta definição estão quatro dimensões que refletem as funções do governo nisto:

a) *e-services*: disponibilização eletrônica de informações do governo, programas, e serviços freqüentemente pela *Web*;

b) *e-management*: uso da informática para melhorar a administração do governo, agilizar processos empresariais e melhorar o fluxo de informação dentro das agências do governo;

c) *e-democracy:* uso dos veículos de comunicação eletrônicos, como *e-mail* e a *Web*, para aumentar a participação do cidadão no processo de formação da decisão pública;

d) *e-commerce*: a troca de dinheiro para bens e serviços pela *Web,* como cidadãos que pagam impostos e contas de serviços públicos, renovam inscrições de veículo, pagam por programas de recreação, ou o governo que compra materiais de escritório e leiloa equipamento em excesso.

2.3 ENGENHARIA PARA *WEB*

A engenharia de software vê a *Web* como uma nova classe de SI. Como acontece no surgimento de novas classes de SIs, no início existe a preocupação em apoiar o desenvolvimento dos SIs propondo novas tecnologias e ferramentas que possam auxiliar no desenvolvimento *ad hoc*. O crescimento dramático da *Web* tem demandado SIs de grande porte, freqüentemente distribuídos e contendo componentes altamente dinâmicos e interativos. A engenharia de software para a *Web*, chamada de "Engenharia para *Web*", é desafiada a fornecer abordagens sistemáticas para o desenvolvimento e manutenção de *WbIS* (GELLERSEN; WICKE; GAEDKE, 2001).

Quanto ao ciclo de vida do software, LIMA e PRINCE (2001) apresentam alguns aspectos que devem ser modelados de forma consistente para se conseguir um projeto completo de um *WbIS*, conforme segue:

a) modelagem de documentos: hipertextos com componentes estáticos e dinâmicos;

b) modelagem de banco de dados: além de documentos, os dados podem ser armazenados em bancos de dados por motivos de desempenho e segurança;

c) modelagem de transações: o usuário utiliza as funcionalidades do sistema através da interação com seus documentos, que disparam as transações apropriadas. Portanto, para modelar completamente uma transação, é necessário ter em mente os documentos envolvidos, caminhos de navegação, categorias de usuários, dados de entrada e resultados;

d) modelagem de navegação: muitos documentos podem estar envolvidos em uma transação, e a modelagem da navegação deve especificar os caminhos possíveis ao usuário, as funções executadas em cada documento, e como a base de dados é acessada;

e) modelagem de aplicação: muitas aplicações podem compor um sistema de informação para *Web*, como *applets Java* (Sun Microsystems, 2001) e

programas *CGI* (The Common Gateway Interface, 2001). A modelagem deve especificar suas funcionalidades e a localização de cada código executável.

Os tradicionais métodos de desenvolvimento de SI, não contém abstrações suficientes para facilitar a tarefa de especificar os requisitos dos *WbIS*. A necessidade de um método para o projeto e desenvolvimento de *WbISs* está relacionada ao fato de que o processo de desenvolvimento deve ser estruturado e permitir reutilização. Alguns métodos que sugiram recentemente tratam das especificidades dos *WbIS* como *OOHDM* (SCHWABE; ROSSI, 1998) e o *RMM* (ISAKOWITZ; STOHR; BALASUBRAMANIAN, 1995) (ISAKOWITZ; STOHR; BALASUBRAMANIAN, 2001), além dos DBCs que também apresentam recursos para modelagem dos *WbIS*.

A forma de armazenamento dos dados, em *WbIS*, é um ponto importante a ser tratado pela Engenharia para *Web*. A maior parte dos *WbIS* armazenam dados em arquivos que não são adequados para a manipulação de grande volume de dados. A decomposição de um *WbIS* em recursos baseados em arquivo não provê uma boa representação dos requisitos para algumas tarefas da engenharia como reutilização e manutenção (GELLERSEN; WICKE; GAEDKE, 2001). Assim, os *WbIS* que utilizam banco de dados vêm tomando um grande espaço, e preenchendo lacunas antes não solucionadas pelos arquivos na *Web*.

A integração dos *WbIS* e dos Sistemas Gerenciadores de Banco de Dados (SGBDs) é de fundamental importância, visto que o volume de dados disponíveis vem aumentando a cada dia. Com isso, aumenta a complexidade de armazenamento e gerência de objetos na *Web*. Os SGBDs, diferentemente dos arquivos comuns, possuem mecanismos que suportam várias organizações de arquivos e estruturas de acesso auxiliares, processam e otimizam consultas, controlam autorizações de acesso aos dados, dão suporte a transações em ambientes multi-usuários e utilizam linguagens específicas de consulta que independem do SI (LIMA; LIFSCHITZ, 2001).

Esta dissertação busca oferecer uma contribuição para a Engenharia para *Web*, apresentando uma arquitetura para *WbIS* que facilitará o seu desenvolvimento.

2.4 CONSIDERAÇÕES FINAIS

Este capítulo explorou como são disponibilizadas as informações na *Web*, indicando uma grande preocupação com a falta de utilização de um método específico para o desenvolvimento de *WbIS*, além da apresentação de alguns requisitos básicos que devem ser considerados para o seu desenvolvimento. Estes conceitos serão aplicados na definição da proposta deste trabalho. Em seguida conceitos básicos sobre *WbIS* foram apresentados, segundo esses conceitos, o estilo de arquitetura três camadas pode ser utilizado para a *Web*, separando os níveis de dados, lógico e de apresentação. Ainda seguindo estes conceitos, foi definida uma categorização para estes SIs. Esta categorização permitiu identificar uma família de SIs, importante para a definição de uma arquitetura.

Por fim, foi apresentada a engenharia para a *Web*, que tem como função propor abordagens estruturadas para o desenvolvimento e manutenção de *WbIS*.

O próximo capítulo apresentará uma conceitualização sobre arquitetura e método de DBC. Conceitos importantes para a proposta apresentada.

3 ARQUITETURA DE SOFTWARE

Este capítulo apresenta a definição de arquitetura de software (seção 3.1). Através do estudo das arquiteturas de software existentes foi possível identificar e definir a arquitetura utilizada para *WbIS* e do estilo de arquitetura aplicado para *WbIS*. Na seção 3.2 é descrito o estilo de arquitetura em camadas, dentre os estilos de arquitetura estudados este é o mais utilizado para *WbIS*. Também é necessário conceituar componentes e DBC (seção 3.3), elementos fundamentais para a arquitetura *WbIS* proposta nesta dissertação. Finalmente é descrito, na seção 3.3.2, o método usado para o desenvolvimento da arquitetura, o *UML Components*.

3.1 DEFINIÇÃO DE ARQUITETURA DE SOFTWARE

O conceito de arquitetura de software surgiu com a necessidade de padronizar a forma de construção de um software por meio de estruturas que pudessem auxiliar a produção de outros produtos semelhantes aos já desenvolvidos. O crescimento do tamanho e da complexidade dos SIs são os principais fatores das pesquisas realizadas para o desenvolvimento de arquiteturas de software para os mais diversos domínios. Neste contexto, fica visível a importância do desenvolvimento de artefatos bem estruturados e reutilizáveis. O resultado deste processo são SIs construídos em menor tempo, com mais qualidade e de fácil manutenção.

Abstratamente, arquitetura de software envolve a descrição de elementos a partir dos quais o sistema é construído, as interações entre esses elementos, padrões que guiam sua composição e as restrições desses padrões. Em geral, um determinado sistema é definido em termos de uma coleção de componentes e a interação entre estes componentes (SHAW; GARLAN, 1996) (BASS; CLEMENTS; KAZMAN, 1998).

Pode-se dizer ainda, que a definição da arquitetura de um produto de software facilita a comunicação entre os desenvolvedores de software e permite um entendimento maior da evolução desse produto (LAZILHA, 2002).

Os modelos arquiteturais existentes apresentam diferenças estruturais e semânticas entre componentes e interações. Esses modelos arquiteturais podem freqüentemente ser composto para definir grandes sistemas. Idealmente, elementos individuais da descrição da arquitetura são definidos independentemente, desse modo eles podem ser reutilizados em contextos diferentes (SHAW; GARLAN, 1996). Para a representação de uma arquitetura são usados estilos de arquitetura que representam a forma como os componentes do SI se comunicam e interagem.

SHAW e GARLAN (1996) descrevem vários estilos de arquitetura, entre eles: Tubos e Filtros (*Pipers and Filters*); Abstração de Dados e Organização Orientada a Objetos (*Data Abstraction and Object-Oriented Organization*); Invocação implícita baseada em eventos; Repositório; Interpretadores (*Interpreters*) e Sistemas em Camadas. Dentre os estilos existentes o estilo de arquitetura Sistemas em Camadas é o mais utilizado para representar os *WbIS*.

3.2 ARQUITETURA *WBIS* EM CAMADAS

A FIGURA 1, no capítulo 2, apresenta de forma simplificada o estilo de arquitetura três camadas utilizado para o desenvolvimento de *WbIS*. Nesta seção, será apresentado um detalhamento deste estilo de arquitetura, o qual será aplicado para o desenvolvimento desta dissertação.

Na arquitetura em camadas, os artefatos são distribuídos em camadas diferentes que representam níveis de abstração e de serviços diferentes. As camadas são organizadas hierarquicamente, e componentes em diferentes camadas têm restrições de comunicação, sendo que o componente de uma camada só pode referenciar componentes das camadas inferiores de acordo com as regras estabelecidas. Uma hierarquia em camadas apresenta vantagens como o desenvolvimento incremental e a reutilização.

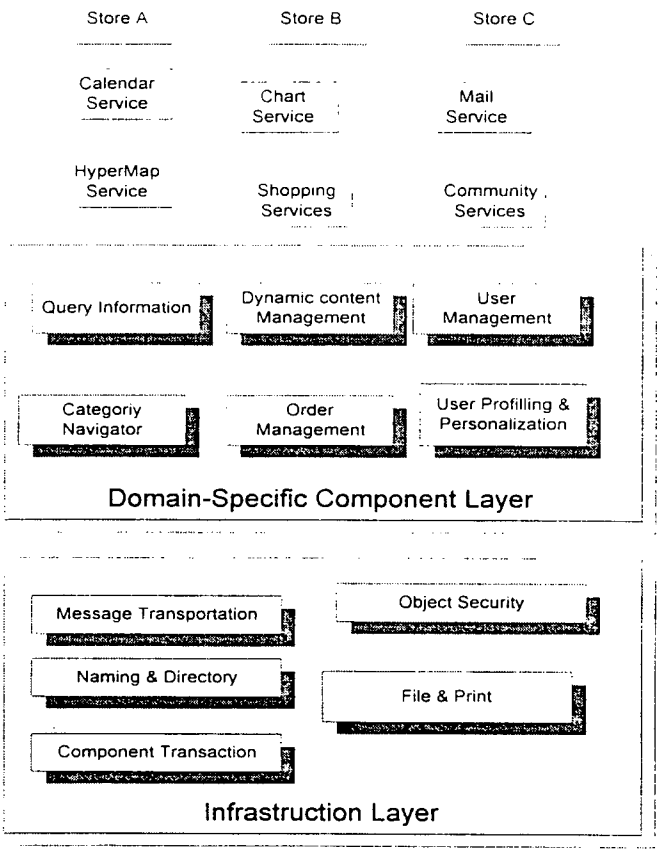
Baseado no estilo de arquitetura cliente/servidor – três camadas - a representação da arquitetura *Web* pode ser vista como apresentada na FIGURA 2 (ALENCAR; COWAN; LUO, 2002), onde:

a) a camada de infra-estrutura (*Infrastruction Layer*) fornece serviços de software unificados para a camada superior através de interfaces comum;

b) a camada de componentes específicos do domínio (*Domain_Specific Component Layer*) representa a lógica do negócio no domínio de SI; e,

c) a camada da aplicação de sistemas de informação (*CIS Application Layer*) está no topo da hierarquia. As aplicações são soluções desenvolvidas baseadas em exigências específicas do domínio e entregue para o usuário final.

FIGURA 2 - APLICAÇÃO *CIS FRAMEWORK*



FONTE: ALENCAR; COWAN; LUO, 2002.

3.3 DESENVOLVIMENTO BASEADO EM COMPONENTES

O DBC tem por objetivo a reutilização de software, e por conseqüente o aumento da qualidade dos produtos de software desenvolvidos e a redução do custo de produção. Nesta seção serão apresentados os conceitos relacionados a componentes e ao método de DBC *UML Components* (CHEESMAN; DANIELS, 2000).

Segundo D'SOUZA e WILLS (1999), um componente pode ser definido como uma unidade de software independente, que encapsula dentro de si seu projeto e implementação, e oferece interfaces bem definidas para o meio externo, para que este componente possa se unir a outros componentes e dar origem aos sistemas baseados em componentes.

A técnica de DBC visa fornecer um conjunto de ferramentas, técnicas e notações que possibilitem que, ao longo do processo de software, ocorra tanto a produção de novos componentes quanto a reutilização de componentes existentes. O amadurecimento dessas técnicas contribui para o desenvolvimento dos *WbIS*, oferecendo mecanismos para especificar e construir componentes de forma a permitir variações entre estes e que possam facilitar a geração de aplicações com características diferentes com menor esforço de desenvolvimento (LAZILHA, 2002).

Segundo BOSCH (2000), componentes podem ser classificados em 3 tipos: componentes desenvolvidos pelo próprio cliente, componentes adquiridos para um determinado domínio e os componentes *Commercial-Of-The-Shelf* (*COTS*), que são componentes genéricos encontrados no mercado. Durante o desenvolvimento de um SI a partir da arquitetura, esses três tipos de componentes podem ser selecionados para preencher a arquitetura de um SI.

É possível desenvolver software baseado em componentes *COTS*. Considera-se, neste caso, como princípio básico adquirir produtos para a maior parte da funcionalidade desejada. O processo de desenvolvimento, de uma forma geral, corresponde à identificação dos produtos disponíveis no mercado e seleção de um ou mais componentes *COTS* considerados adequados e a integração deles ao

sistema final. Dessa forma, a maioria dos modelos de ciclo de vida para sistemas baseados em componentes *COTS* considera as atividades essenciais de identificação, seleção e integração. Existem poucas variações entre os diversos modelos propostos na literatura (ALVES, 2001).

Muitos dos métodos utilizados para DBC são métodos relacionados a *UML (Unified Modeling Language)* (BOOCH; RUMBAUGH; JACOBSON, 1999) (FOWLER; KENDALL, 1997) (RUMBAUGH; BOOCH; JACOBSON, 1999) como *Catalysis* (D'SOUZA; WILLS, 1999), *Rational Unified Process (RUP)* (JACOBSON; BOOCH; RUMBAUGH, 1999) e *UML Components* (CHEESMAN; DANIELS, 2000). Porém, a OO pura não apresenta todas as abstrações necessárias para o DBC.

O *Catalysis* é uma abordagem de DBC que fornece apoio específico aos conceitos de componentes. Tem seu foco na especificação e na arquitetura de componentes e introduz algumas extensões para melhorar o projeto de componentes. No *Catalysis*, os componentes são obtidos pelo refinamento das especificações nos diferentes níveis: Domínio do Problema, Especificação dos Componentes e Projeto Interno dos Componentes.

O *RUP* segue a *UML*, mas introduz vários conceitos para descrever métodos, como *worker, artefact, activity* e *workflow*. Esses conceitos capturam o que deveria ser feito, como e quando. O *RUP* fornece alguma orientação para desenvolver e modelar componentes, mas esta orientação é bastante limitada e fica quase ao nível de empacotar código fonte. O DBC é visto somente como uma variação do desenvolvimento da aplicação geral (RATIONAL SOFTWARE CORPORATION, 2001).

Neste trabalho optou-se por utilizar *UML Components* por ser este um processo que inclui recursos para identificação e especificação de componentes, assim como de suas interfaces e por ser um método recente. Este método também considera a arquitetura do software como elemento central para determinação dos componentes bem como oferece flexibilidade necessária entre componentes e arquitetura (CHEESMAN; DANIELS, 2000).

3.3.1 O Método *Uml Components*

CHEESMAN e DANIELS (2000) refinaram os modelos conceituais de outros métodos existentes, como *Syntropy* (COOK; DANIELS, 1994) e *Catalysis* entre outros, ampliaram as idéias de processo, e as aliaram com *workflow* e a terminologia do *RUP*, assim surgiu o *UML Components*.

A principal diferença entre este método e os métodos tradicionais é que os componentes seguem o princípio do paradigma OO de combinar funções e dados relacionados em uma única unidade. Componentes estendem estes princípios fortalecendo o papel da interface somando à noção de especificação de componente.

Componentes são unidades de software estruturadas de acordo com alguns princípios específicos. Os princípios fundamentais do método suportam a tecnologia de objeto como:

a) unificação de dados e funções: um objeto de software consiste de valores de dados e de funções que processam esses dados. Esta dependência entre funções e dados melhora a coesão;

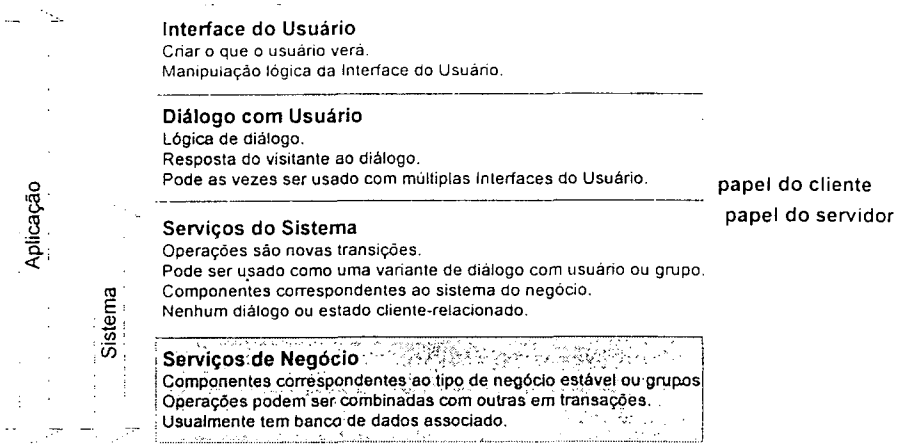
b) encapsulamento: o cliente do objeto de software está separado do processo de como o dado do objeto de software é armazenado ou como suas funções são implementadas. Clientes do objeto de software dependem de uma especificação de objeto, mas não da sua implementação;

c) identidade: cada objeto de software tem uma única identidade, indiferentemente do estado.

O *UML Components* propõe a estruturação de um sistema em quatro camadas (FIGURA 3): (1) Interface do usuário; (2) Diálogo do usuário; (3) Serviços do sistema; e, (4) Serviços de negócio, mas somente as camadas de Serviços do sistema e Serviços de negócio são tratadas pelo método. Estas camadas e a

dependência entre elas constituem a arquitetura do sistema, modelada pelo *UML Components*.

FIGURA 3 - CAMADAS DA ARQUITETURA



FONTE: CHEESMAN; DANIELS, 2000.

Neste método a arquitetura de componentes se preocupa com a estrutura e as dependências entre componentes nas camadas de serviços do sistema e de serviços do negócio. A especificação da arquitetura de componente descreve as restrições de implementação e montagem dos componentes para formar o sistema. A especificação do componente define o que será construído e que unidades existirão em tempo de execução, define o grupo de interfaces suportadas e algumas restrições em relação a como eles serão implementados.

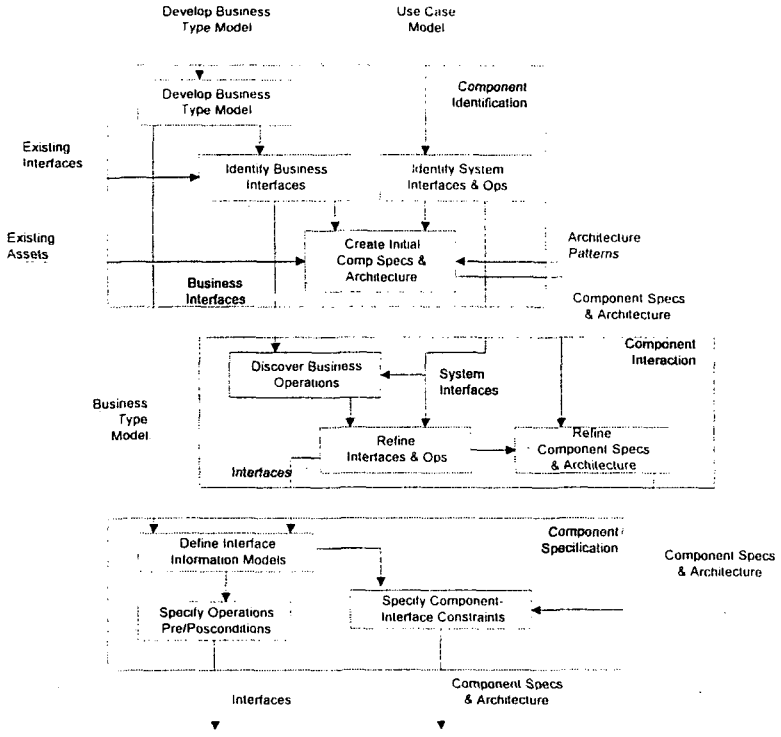
O *UML Components* é organizado da seguinte forma:

a) levantamento de requisitos: compreende o Modelo Conceitual do Negócio e Modelo de Caso de Uso;

b) especificação: compreende o Modelo de Tipo de Negócio, Especificação de Interface, Especificação de Componente e Arquitetura de Componentes.

O Levantamento de requisitos é o primeiro passo do processo, mas este não aparece como um estágio no, *Workflow* de especificação do método. Esse passo é importante, pois, define as informações que serão tratadas pelos estágios do método. Esses estágios são divididos em três, como apresentado na FIGURA 4.

FIGURA 4 - TRÊS ESTÁGIOS DO WORKFLOW DE ESPECIFICAÇÃO



FONTE: CHEESMAN; DANIELS, 2000.

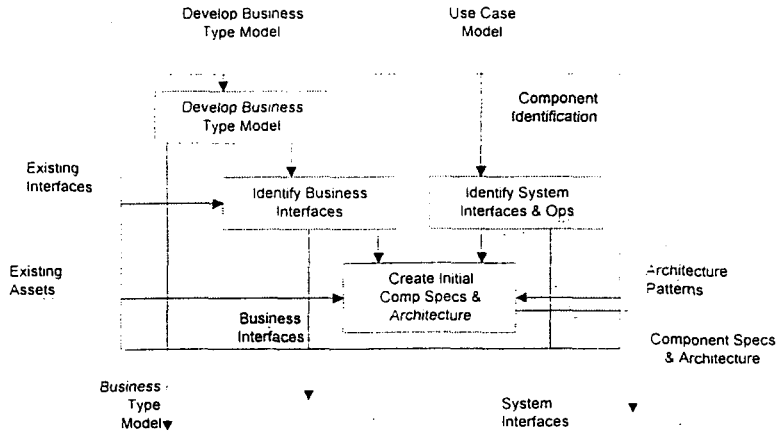
No primeiro passo, levantamento de requisitos, o modelo conceitual do negócio visa apresentar o modelo de informação que existe no domínio do problema. A proposta principal do modelo conceitual de negócio é capturar os conceitos e identificar relacionamentos.

O modelo de caso de uso fornece uma modelagem semi-formal da interação sistema-usuário, é usado para descrever as exigências de um sistema, expressada em termos da interação que tem que acontecer dentro do limite do sistema. O diagrama é composto por casos de uso e atores. O ator é uma entidade que interage com o sistema, tipicamente uma pessoa. O caso de uso é a descrição de um modelo para o comportamento que é instanciado no ambiente do sistema, realizado por um ator, que gera um estímulo.

O estágio de identificação do componente, como mostra a FIGURA 5, tem como entrada o modelo conceitual de negócio e o modelo de caso de uso. Assume uma camada de aplicação que inclui a separação de componentes do sistema e componentes de negócio. O objetivo é identificar um grupo inicial de interfaces de

negócio para os componentes de negócio e um grupo inicial de interfaces do sistema para os componentes do sistema.

FIGURA 5 - ESTÁGIO DE IDENTIFICAÇÃO DO COMPONENTE



FONTE: CHEESMAN; DANIELS, 2000.

O modelo de tipos de negócio é representado por um diagrama de classes, mas sua proposta é diferente do modelo conceitual. O modelo de tipos do negócio está preocupado com a modelagem das informações do negócio que são relevantes para o contexto do sistema examinado.

Uma tarefa importante neste estágio é a especificação de interface. Uma especificação de interface é uma interface juntamente com as especificações e acessórios necessários para definir o que um componente oferece, o que a interface tem que fazer e o que um cliente pode esperar da interface.

Uma especificação de interface consiste nas seguintes partes:

a) o tipo de interface;

b) o modelo de informação – os atributos e regras de associação da interface;

c) suas especificações de operação – assinaturas de operações e pré- e pós-condições;

d) qualquer constante adicional no modelo de informação.

Para definição das interfaces são usados dois processos, definir as interfaces do sistema e interfaces do negócio. As interfaces do sistema são definidas inicialmente fazendo a correspondência de uma interface para cada caso de uso. As interfaces do negócio são definidas identificando os elementos no negócio que são independentes da existência de outros elementos.

Alguns estereótipos são utilizados nesse estágio:

a) ***<<interface type>>*** são interfaces usadas no modelo de tipo de negócio, não possuem atributos ou associações;

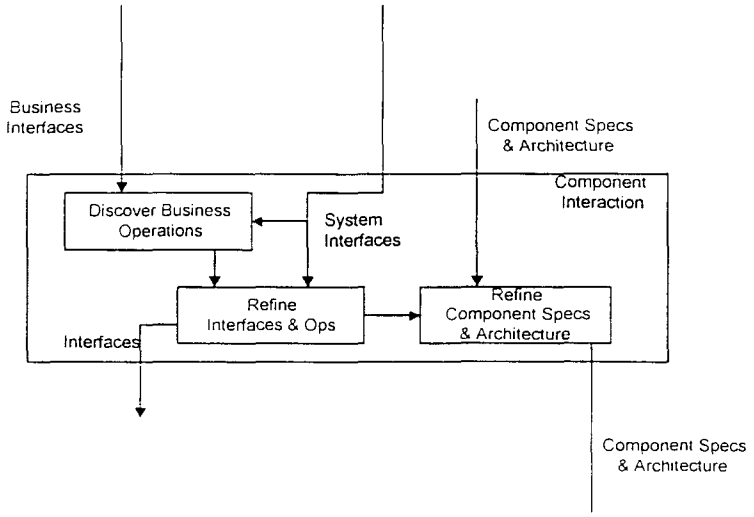
b) ***<<core>>*** um tipo de negócio "*core*" tem existência independente do negócio. São grupos de informações que não precisam de informações de outros elementos do sistema para existir, podem ser cadastradas e atualizadas sem interferir em outras partes do sistema;

c) ***<<type>>*** indica que a classe não pode ter operações porque descreve somente informações, mas também não é o projeto de banco de dados.

O estágio de identificação do componente gera um conjunto de artefatos, interfaces e componente, necessário para o desenvolvimento do estágio de interação do componente (FIGURA 6). Nesse estágio será decidido como os componentes trabalharão, para empregar as funcionalidades necessárias, usando interação.

O modelo de interação é usado para modelar o comportamento dos componentes, ele é usado aqui para: definir as várias interações que precisam acontecer dentro do sistema; refinar definições de interface existentes; identificar como as interfaces serão usadas; e, descobrir novas interfaces e operações.

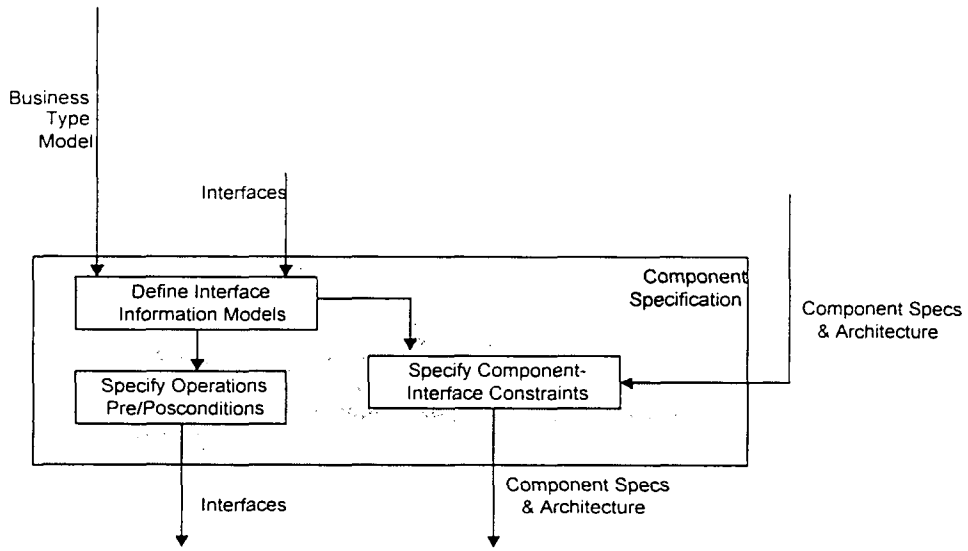
FIGURA 6 - ESTÁGIO DE INTERAÇÃO DO COMPONENTE



FONTE: CHEESMAN; DANIELS, 2000.

No estágio de especificação do componente, FIGURA 7, são detalhadas as especificações das operações e as restrições produzidas no estágio anterior. Significa que, uma determinada interface define os estados potenciais de objetos de componente em um Modelo de Informação de Interface, especificando então as pré- e pós-condições para as operações e regras de negócio capturadas como restrições.

FIGURA 7 - ESPECIFICAÇÃO DO COMPONENTE



FONTE: CHEESMAN; DANIELS, 2000.

O resultado do último estágio é a especificação de cada componente, as interfaces e a arquitetura do sistema.

3.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou conceitos importantes para o desenvolvimento desta proposta, conceituando arquitetura de software e o estilo aplicado para *WbIS*. A motivação para o estudo e para o uso de arquiteturas está no aumento da complexidade dos *WbIS*. Também é importante a definição do estilo de arquitetura a ser empregada, pois através dele será definido como os componentes que compõe a arquitetura se comunicarão.

Os conceitos sobre componentes e DBC também foram tratados aqui por sua importância na arquitetura. Afinal a arquitetura proposta nesta dissertação é composta por componentes, necessitando assim de um método que apóie o DBC.

O método de DBC, *UML Components* será utilizado para definição da arquitetura proposta, por ser um método novo e baseado em componente, a ser explorado e basear-se em métodos consolidados. Foram apresentados os três estágios propostos pelo método e suas características.

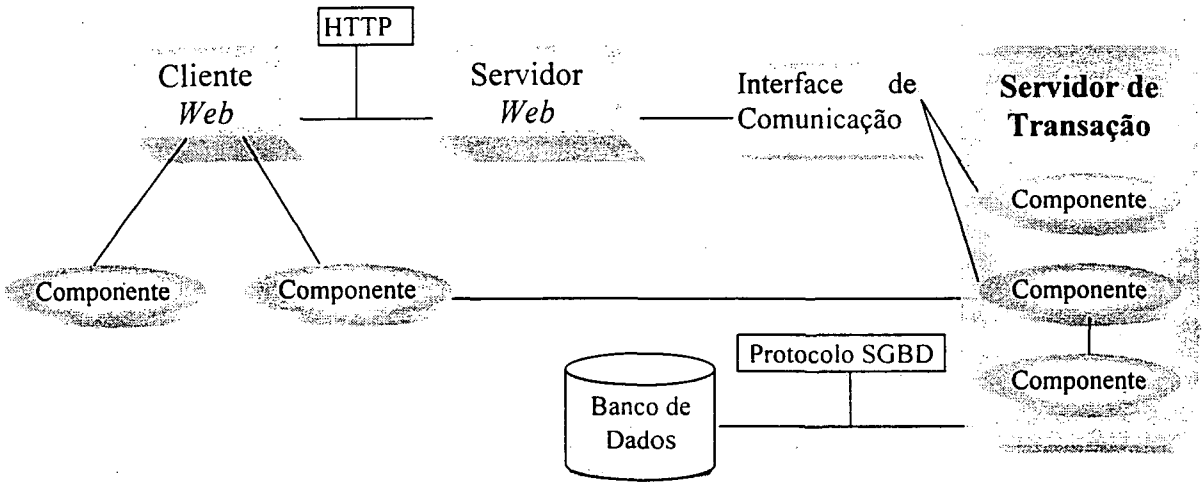
4 TRABALHOS RELACIONADOS

Este capítulo apresenta algumas arquiteturas relacionadas à proposta deste trabalho. A Arquitetura Servidor de Transação (seção 4.1), projetada utilizando componentes. A arquitetura *Enterprise Java services* (seção 4.2) é apresentada como exemplo de arquitetura proprietária. Por último, na seção 4.3, a arquitetura *CIS Framework* é apresentada, ela utiliza padrões para sanar os problemas da arquitetura de três camadas identificados por ALENCAR, COWAN e LUO (2002).

4.1 ARQUITETURA SERVIDOR DE TRANSAÇÃO

A arquitetura Servidor de Transação identificada por (LIMA; LIFSCHITZ, 1998) faz uso de componentes, que são unidades básicas do SI e são projetados para serem interoperáveis em diferentes plataformas, como mostra a FIGURA 8. Os SIs que seguem esta arquitetura são compostos por vários componentes que representam as unidades que compõem a aplicação mais o relacionamento entre estas.

FIGURA 8 - ARQUITETURA SERVIDOR DE TRANSAÇÃO



FONTE: LIMA; LIFSCHITZ, 2001.

Nesta arquitetura para *WbIS*, parte do sistema responsável pela lógica é executada no lado do cliente e parte no lado do servidor. A responsabilidade de cada parte integrante da arquitetura fica dividida da seguinte forma: (1) o cliente *Web* é responsável pela lógica da apresentação, controle da interação entre o usuário e o computador e parte do processamento da lógica do SI; (2) o servidor *Web* é o local de armazenamento do código do SI que será executado no cliente e parte da lógica do SI; e, (3) o servidor de Banco de Dados é responsável pelos dados, metadados e regras de integridade.

Esta arquitetura tenta, segundo LIMA e LIFSCHITZ (1998), cobrir uma grande preocupação que norteia as arquiteturas, segundo os pesquisadores e desenvolvedores, a garantia das propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) que afetam os bancos de dados.

O servidor de transação garante o isolamento e consistência das ações realizadas. Ao final de uma solicitação do usuário os dados são retornados via servidor *Web* ao cliente *Web*. Os componentes no lado cliente *Web* podem trocar mensagens diretamente com componentes no Servidor de Transação, sem a intervenção do servidor *Web*. Esta arquitetura considera a lógica da aplicação dividida em componentes, porém esses componentes não são definidos.

4.2 ARQUITETURAS DE TECNOLOGIAS PROPRIETÁRIAS

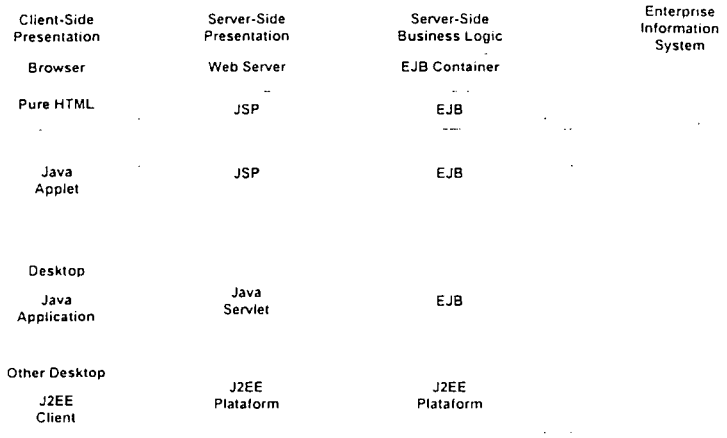
Seguindo a perspectiva do modelo arquitetural, existem muitos projetos considerados tecnologias proprietárias para *WbIS*. Por exemplo, *WebSphere Commerce suite* da IBM (IBM®, 2002), *Enterprise Java services* da *Sun* (SUN MICROSYSTEM, 2001) e o .NET *framework* da *Microsoft* (MICROSOFT CORPORATION, 2002).

Para exemplificar as especificações destas arquiteturas segue uma descrição da tecnologia *Java™2 Plataform, Enterprise Edition (J2EE),* FIGURA 9.

Várias complexidades inerentes a aplicações *Enterprise* – gerenciamento de transação, gerenciamento do ciclo de vida, *resource polling* - são construídas nesta

plataforma. Esta tecnologia permite aos desenvolvedores de aplicações e componentes liberdade para focalizar, a lógica do negócio e as interfaces de usuário.

FIGURA 9 - ARQUITETURA J2EE



FONTE: SUN MICROSYSTEMS, 2001.

O modelo de aplicação *J2EE* encapsula as camadas de funcionalidade em tipos específicos de componentes. A lógica do negócio é encapsulada nos componentes *Enterprise JavaBeans™ (EJB)*. A interação com cliente pode ser apresentada por páginas *Web HTML*, através de páginas *Web* ligadas por tecnologia baseada em *Java applets*, *Java Servlets API*, tecnologia *JavaServer Pages™*, ou através de aplicações de *Java stand-alone*. Os componentes *EJB* se comunicam transparentemente usando vários padrões: *HTML, XML, HTTP, SSL, RMI, IIOP*, entre outros.

O *J2EE* permite agrupar aplicações de um mesmo padrão de componentes comerciais e seus próprios componentes. Desde os componentes de aplicação de negócio às soluções de mercado, uma série de padrões de funcionalidades *Java Enterprise* 2 estão sendo esperados para serem disponibilizados como *off-the-shelf*. Isto significa que um *site* de *e-commerce* poderá ser construído usando uma combinação de componentes de *EJB* para uma loja de automóveis, modificando os componentes *EJB* para atendimentos aos serviços especializados do consumidor, e criando *layouts* personalizados utilizando tecnologia *JavaServer Pages*.

Esta abordagem pode proporcionar menor tempo de desenvolvimento, melhor qualidade e manutenibilidade, e portabilidade para uma série de plataformas.

4.3 *CIS ARCHITECTURE FRAMEWORK WITH PATTERNS*

O *CIS Framework* consiste de uma arquitetura genérica para sistemas de informação comunitários baseados na *Web* (*CIS – Community Information Systems*) (ALENCAR; COWAN; LUO, 2002). Um *CIS* representa um repositório de informações que precisam ser organizadas, mantidas e apresentadas para que seja possível disponibilizar serviços como notícias, eventos comunitários, atividades governamentais, negócios e transações comerciais.

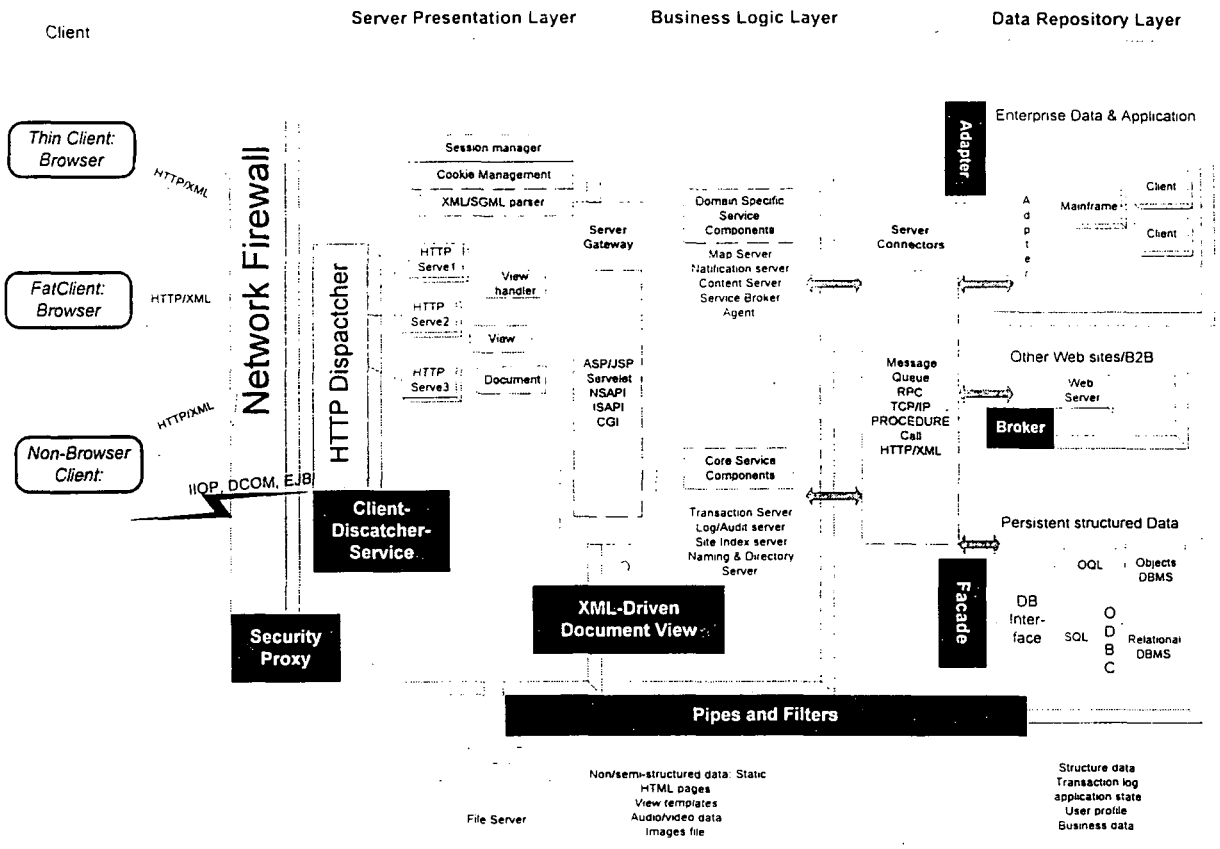
Esta arquitetura é baseada em componentes e XML e segue o estilo de arquitetura em camadas. O *CIS framework* pode ser visto de acordo com o modelo lógico em três camadas (FIGURA 10):

a) camada de apresentação (*Server Presentation Layer*): soluções estendidas do *CIS framework*, por meio de instanciação ou pela composição de seus serviços, contém o gerenciador de seção *HTML/XML* e o interpretador de *script*;

b) camada de componentes específicos do negócio (*Business Logic Layer*): contém componentes específico do domínio da lógica do negócio e processa os componentes de serviço que incluem gerenciamento de usuário e manutenção de perfil de usuário;

c) camada de infra-estrutura (*Data Repository Layer*): os serviços padrões desta camada podem estar relacionados aos sistemas operacionais e aos bancos de dados. Ela fornece, por exemplo, os serviços de armazenamento dos dados usados por outras camadas.

FIGURA 10 - CIS ARCHITECTURE FRAMEWORK WITH PATTERNS



FONTE: ALENCAR; COWAN; LUO, 2002.

Os elementos da aplicação nestas três camadas lógicas são conectados por meio de um conjunto de protocolos padrões, serviços e conectores de software. O servidor de ligação (*Server Gateway*) conecta os componentes da aplicação residindo em camadas separadas através de um conjunto de interfaces padrão, como *CGI*, *Java Servlet*, *NSAPI*, *ISAPI* e *Active Server Pages*. Os conectores representam a interação entre os componentes. A comunicação pode ser síncrona (exemplo: *database queries, RPC calls*) ou assíncrona (exemplo: *message queues, event broadcasts or pipe*).

A arquitetura global do *CIS Framework* pode ser visualizada como um grupo de camadas consistindo de componentes com o mesmo grau de generalidade. Uma aplicação *CIS Framework* administra a comunicação entre o pedido de serviço (cliente) e o provedor de serviço (servidor). Os componentes específicos do domínio da aplicação representam a lógica da aplicação, estes componentes estão localizados no nível lógico da arquitetura *CIS Framework*, identificado na FIGURA 10 como camada de componentes específicos do negócio (*Business Logic Layer*). Esses componentes de domínio são artefatos particulares de cada aplicação, no entanto em *WbIS* vários destes podem ser comuns. Segundo o *CIS Framework* estes componentes incluem:

a) consulta de informação: este componente é responsável pelas consultas que necessitam de acesso aos bancos de dados para recuperação de informações. O banco de dados acessado pode ser local ou remoto, assim o componente deve fornecer mecanismos de comunicação independente com SGBDs. Para o caso do *e-commerce* este serviço seria realizado pelo catálogo de produtos;

b) categoria de navegação: este componente é responsável pela navegação segura pelas páginas que compõem um SI. Garantindo a privacidade das operações realizadas pelo usuário, por meio de técnicas de criptografia, por exemplo. Garantir também, que o usuário navegue dentro dos limites de sua categoria, porque o SI pode estar dividido e restringir seus recursos a grupos de usuário, sendo assim, o usuário não pode visualizar informações fora da sua área de usuário. Um exemplo deste componente para *e-commerce* é permitir acesso a atividades restritas somente a um grupo de usuário, evitando que informações sigilosas sejam vistas por usuário não autorizados;

c) gerenciamento de usuário: este componente é responsável pelo cadastro e validação do usuário para uso do SI. Mantém os dados sobre seus usuários e esses dados variam de acordo com o SI, mas geralmente são dados como identificação do usuário e seus direitos de acesso. Essas informações categorizam o usuário e define o tipo de acesso que ele terá;

d) gerenciamento de conteúdo dinâmico: este componente deve disponibilizar ao usuário um mecanismo que torne possível a busca por elementos presentes no SI de maneira rápida e fácil, através da combinação de palavras chaves ou esquemas de mapa de *site*;

e) personalização e perfil do usuário: os SI devem buscar o conforto e satisfação do usuário, adaptando-se às características de personalidade de cada um e expondo o conteúdo de maneira personalizada. Para realizar este serviço, este componente deve apresentar meios para captura do comportamento do usuário, utilizando técnicas para análise de seqüências de *links* seguidos pelo usuário, serviços mais requisitados por ele, enquêtes sobre o que o usuário se interessa, entre outros. Uma outra forma de personalização é tratar aspectos referentes a grupos de usuários com perfil semelhante, apresentando as mesmas informações a cada grupo. Essas informações podem ser armazenadas no banco de dados e carregadas no início do acesso do usuário. Em um *e-commerce*, pode-se mostrar os produtos e ofertas de acordo com o perfil de cada usuário;

f) gerenciamento de estado: o gerenciamento de estado é um dos principais problemas decorrentes da integração *Web*-SGBD. Este componente realiza justamente esta tarefa, permitindo a navegação pelas páginas da aplicação sem que se perca as características de perfil do usuário, e sem a necessidade de re-conexão ao banco de dados a cada interação.

Porém, é possível existir outros componentes presentes nesta camada que podem estar presentes na maioria dos *WbIS*, principalmente na categoria dos *e-commerce*. Esta dissertação propõe um estudo para a identificação destes componentes.

A arquitetura *CIS framework* apresentada na FIGURA 10 ilustra como o *CIS framework* é estruturado, usando padrões e componentes particulares que podem suprir as limitações da arquitetura três camadas. Segundo ALENCAR, COWAN e LUO (2002) a arquitetura em três camadas tem as seguintes limitações:

a) as camadas lógicas do negócio e de infra-estrutura são firmemente acopladas. Trocando o sistema operacional ou o SGBD pode causar mudanças na camada lógica do negócio e até mesmo na camada de apresentação;

b) existe também um forte acoplamento entre a camada de apresentação e a camada lógica do negócio. Se existir uma mudança em qualquer camada, então a parte do sistema que faz referência também deve ser mudada.

4.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a Arquitetura Servidor de Transação baseada em componentes. A arquitetura proprietária *J2EE*. A arquitetura *CIS Framework*, que oferece um modelo de representação para *WbIS*, também baseado em componentes.

O estudo dessas arquiteturas foi utilizado como suporte para identificar os componentes necessários para a definição da arquitetura de componentes para *WbIS*. Essas arquiteturas têm em comum o princípio de utilizar componentes para implementar as necessidades do negócio, localizadas na camada lógicas (ou nível lógico). Somente a arquitetura *CIS Framework* identifica quais seriam os possíveis componentes que fariam parte desta camada, mas não faz um estudo aprofundado sobre estes componentes. Neste ponto esta dissertação pretende contribuir identificando e especificando os componentes necessários para o desenvolvimento dos *WbIS*.

5 UMA ARQUITETURA DE COMPONENTES PARA SISTEMAS DE INFORMAÇÃO BASEADOS NA *WEB*

Este capítulo apresenta o processo seguido para a definição da arquitetura de componentes para *WbIS* – nível lógico, segundo o modelo de componentes proposto por ALENCAR, COWAN e LUO (2002), e os definidos por meio deste estudo.

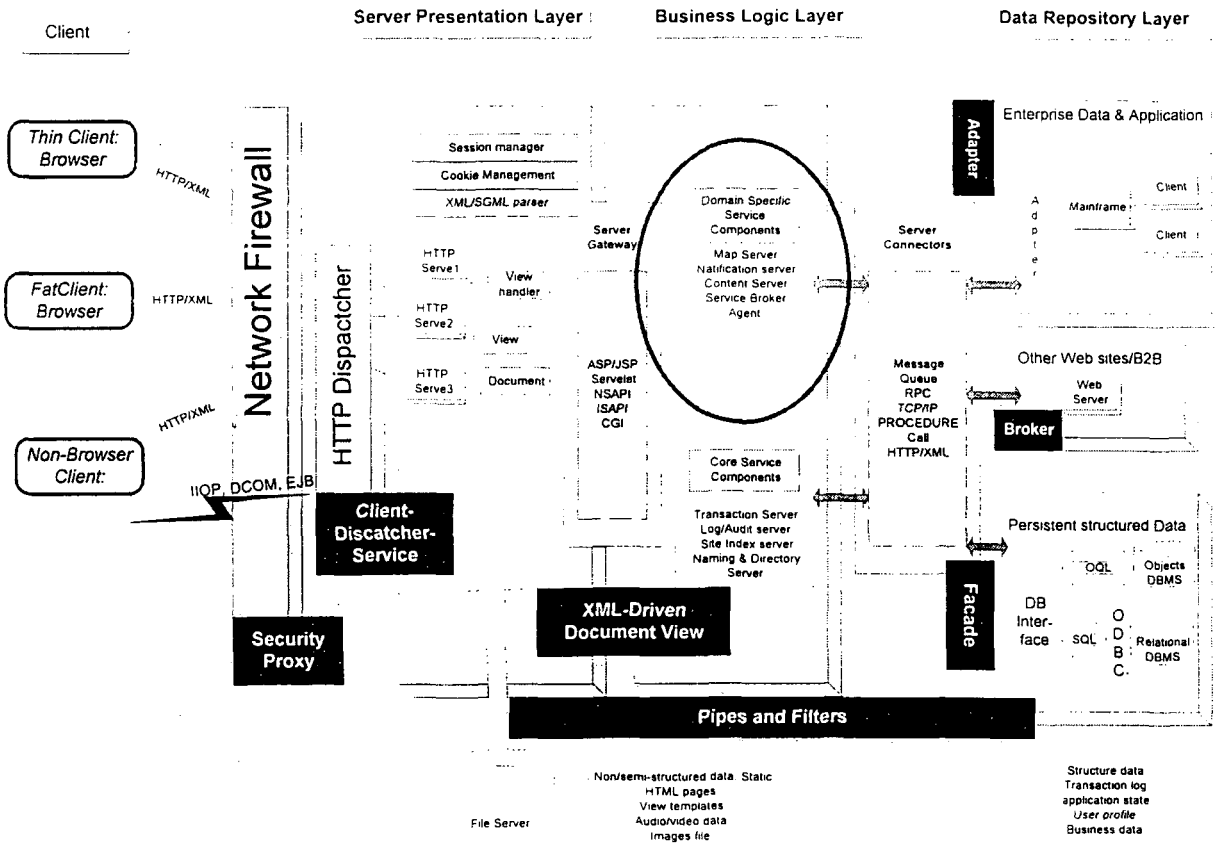
No capítulo 2 foram apresentados os principais conceitos relacionados aos *WbIS* que permitem a classificação e definição dos domínios desses sistemas, base para este estudo. Os *WbIS* são divididos, em categorias, dentro dos sistemas "*e-*", como, *e-commerce*, e-*business*, *e-government*. A categoria escolhida para estudo e definição da arquitetura foi o *e-commerce B2C*, por ser uma das categorias em maior ascensão e apresentar características semelhantes às outras categorias, facilitando assim uma futura definição de arquitetura que abranja todas as categorias "*e-*".

Para o desenvolvimento de uma arquitetura é necessário definir um estilo de arquitetura, que consiste de um vocabulário de tipos de componentes e conectores a ser utilizado. O estilo de arquitetura de sistemas em camadas é o mais utilizado entre profissionais e pesquisadores para o desenvolvimento de *WbIS*, por apoiarem projetos baseados em níveis crescentes de abstração, além da reutilização (SHAW; GARLAN, 1996).

Dentre as arquiteturas estudadas, este trabalho de dissertação optou por aprofundar a proposta de *CIS framework* de ALENCAR, COWAN e LUO (2002) propondo uma arquitetura de componentes para *WbIS* que explore o nível lógico, buscando os componentes típicos do domínio de *CIS*. Analisando o *CIS Framework* foi possível identificar as camadas e tecnologias empregadas para o desenvolvimento de cada uma delas. Por ser uma arquitetura que define todas as camadas não há um estudo aprofundado de cada uma delas, foi então que se optou por explorar o elemento da arquitetura que se refere aos "componente de serviços específico do domínio", como mostra o elemento em destaque na FIGURA 11. A ênfase está na identificação e especificação dos componentes, o que difere dos

demais trabalhos que são métodos de propósito gerais adaptado, em que o tratamento de componentes de aplicação não é devidamente realizado.

FIGURA 11 - COMPONENTE DE SERVIÇOS ESPECÍFICO DO DOMÍNIO



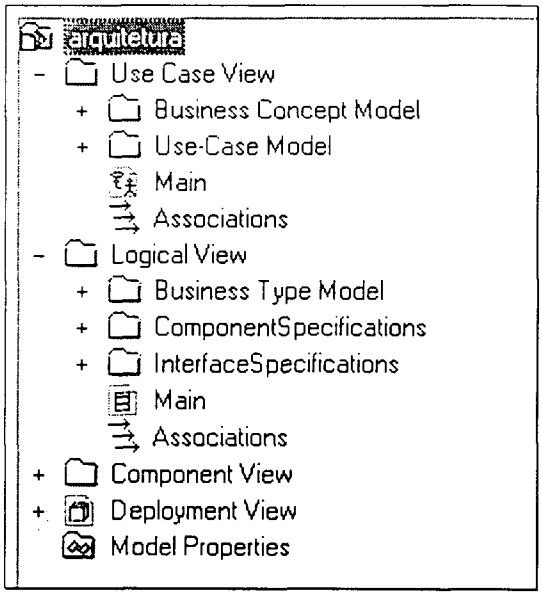
O método escolhido foi o *UML Components*, apresentado no capítulo 3. Este método considera a arquitetura do software como elemento central para determinação dos componentes bem como da flexibilidade necessária entre componentes e arquitetura (CHEESMAN; DANIELS, 2000).

5.1 PROCESSO DE OBTENÇÃO DA ARQUITETURA

Para a obtenção da arquitetura seguiu-se o método *UML Components*, conforme descrito na seção 3.3.1. Os diagramas que compõem o método foram

desenvolvidos utilizando a ferramenta case *Rational Rose* (RATIONAL SOFTWARE CORPORATION, 2002). O *framework* do processo padrão desta ferramenta foi adaptado para representar o *UML Components* como mostra a FIGURA 12. A ordem dos estágios do *UML Components* não correspondem ao da figura, pois a ferramenta coloca os itens em ordem alfabética.

FIGURA 12 - ORGANIZAÇÃO DO WORKFLOW DE ARTEFATOS



No *framework* da FIGURA 12 a visão de casos de uso compreende os artefatos do levantamento de requisitos, em que estão os diagramas de caso de uso e o modelo conceitual do negócio. Estes artefatos são as entradas para o primeiro estágio do método *UML Components,* o estágio de identificação do componente.

A visão lógica é composta pelos três estágios do método, contendo os seguintes artefatos: diagrama de modelo de tipo de negócio; o diagrama de responsabilidade das interfaces do negócio; diagrama de interação; diagrama de especificação de interface; e diagrama de especificação de componentes.

O processo de obtenção da arquitetura proposta consiste das seguintes atividades:

a) aplicação do método *UML Components* ao estudo de caso *e-magazine*;

b) generalização da arquitetura obtida na atividade anterior;

c) aplicação da arquitetura generalizada ao estudo de caso *e-rent-a-car*;

d) ajuste da arquitetura generalizada.

Essas atividades são descritas a seguir.

5.1.1 Aplicação do Método *UML Components* ao Estudo de Caso *e-magazine*

Na primeira atividade desenvolveu-se o estudo de caso *e-magazine*, cuja descrição completa é apresentada no APÊNDICE 1. Esse estudo de caso trata a venda de publicações pela *Web*, como revistas e jornais. O cliente poderá consultar as publicações comercializadas pela loja e adquiri-las, seguindo as restrições de compra da loja. O cliente recebe a publicação em seu domicílio mediante pagamento.

Outros estudos de casos foram avaliados para ajudar na definição dos componentes. Entre esses estudos de casos avaliados estão: o Projeto Intersul (PAZZINATO, 2003), classificado como *e-government*; o Modelo Café – e-B@nca (POLCELLI; SOUZA, 2002), classificado como *e-commerce*; e o Harry's Vehicle Hire (ALLEN, 2001), classificado como *e-business*. Além dos estudos realizados em GABRIEL (2001) e GABRIEL (2002) sobre *WbIS*.

5.1.2 Generalização da Arquitetura

A arquitetura obtida para o estudo de caso foi generalizada de modo a permitir a instanciação de outras aplicações da mesma categoria, obtendo desta maneira a arquitetura generalizada para *WbIS*.

Nesta seção serão apresentados os diagramas generalizados da arquitetura que foram criados a partir dos diagramas do estudo de caso *e-magazine*. Desde os diagramas que compõem a especificação até o diagrama de componentes da arquitetura, todos os estágios foram generalizadas para obtenção da arquitetura.

Antes de aplicar o primeiro estágio do *UML Components,* estágio de identificação dos componentes, é preciso fazer o levantamento de requisitos, composto pelo diagrama conceitual do negócio, FIGURA 13 e pelo diagrama de casos de uso FIGURA 14. Com estes diagramas foi possível modelar o comportamento do SI identificando quais as semelhanças do *e-magazine* com os outros estudos de caso visando a generalização da arquitetura. Estes diagramas servem de entrada para os três estágios que compreendem o método. Os diagramas generalizados (FIGURA 13 e FIGURA 14) tiveram como base os diagramas do estudo de caso apresentados na FIGURA 37 e na FIGURA 38.

FIGURA 13 - DIAGRAMA CONCEITUAL DO NEGÓCIO – ARQUITETURA

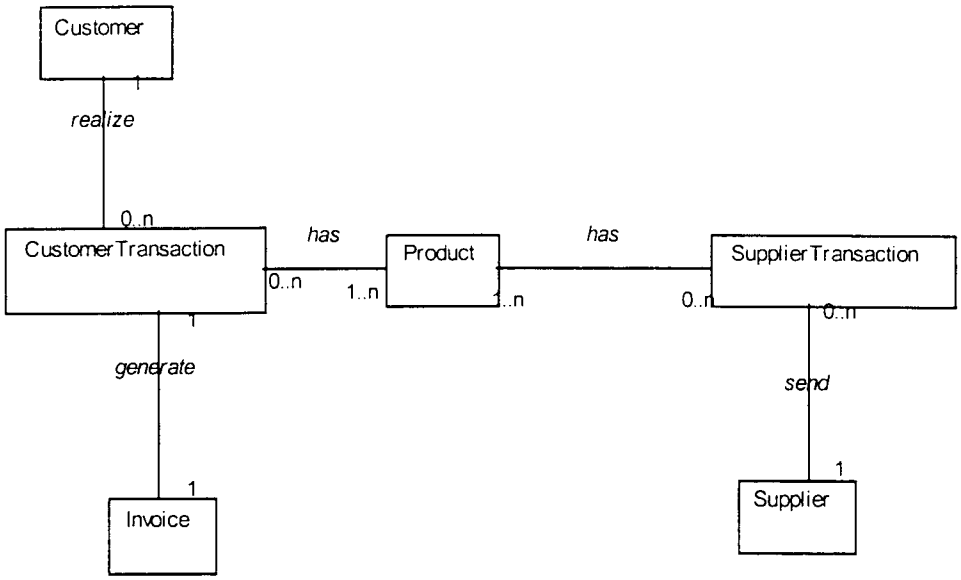
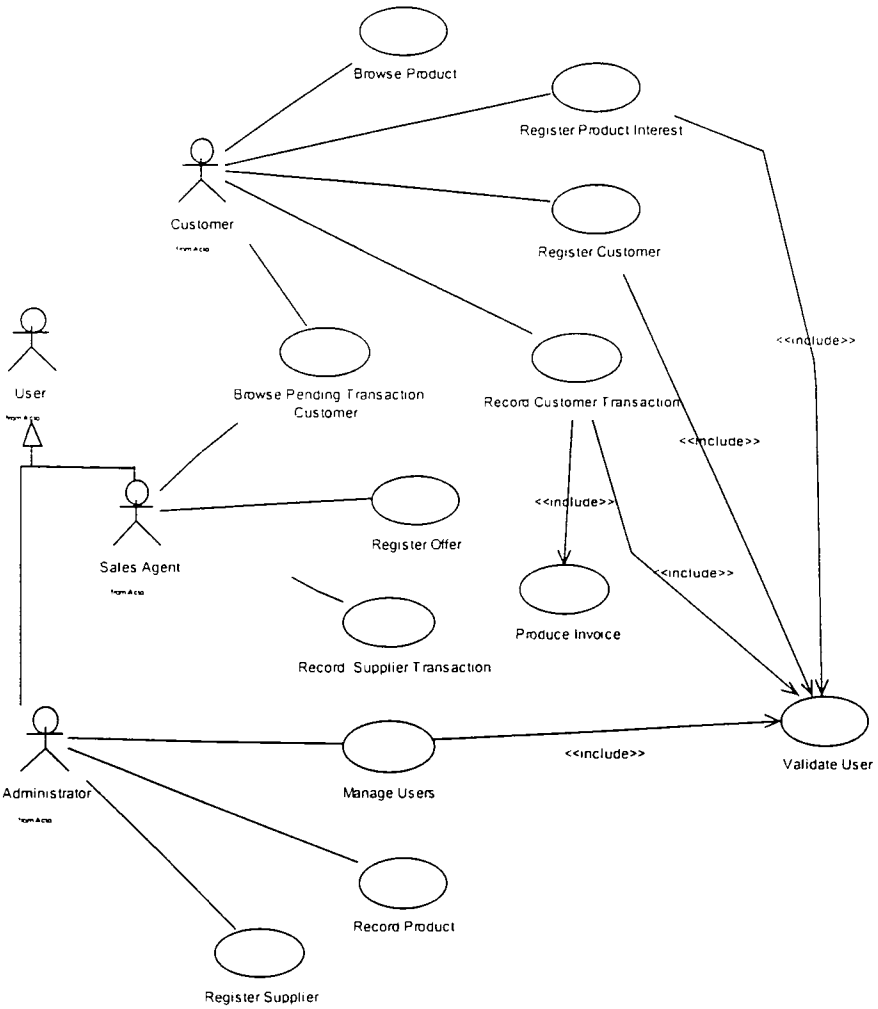


FIGURA 14 - DIAGRAMA DE CASOS DE USO – ARQUITETURA



No primeiro estágio, estágio de identificação do componente, é feita a separação das interfaces do negócio e do sistema, e dos componentes do negócio e do sistema. Por meio desta separação é possível identificar componentes independentes do negócio, indicando assim os primeiros componentes que poderão ser utilizados para compor a arquitetura. Os diagramas que fazem parte deste estágio são o diagrama de tipo de negócio e o diagrama de responsabilidade das interfaces do negócio. Os diagramas da arquitetura estão apresentados na FIGURA 15 e na FIGURA 16, respectivamente, e do *e-magazine* na FIGURA 39 e na FIGURA 40.

FIGURA 15 - DIAGRAMA DE TIPO DE NEGÓCIO – ARQUITETURA

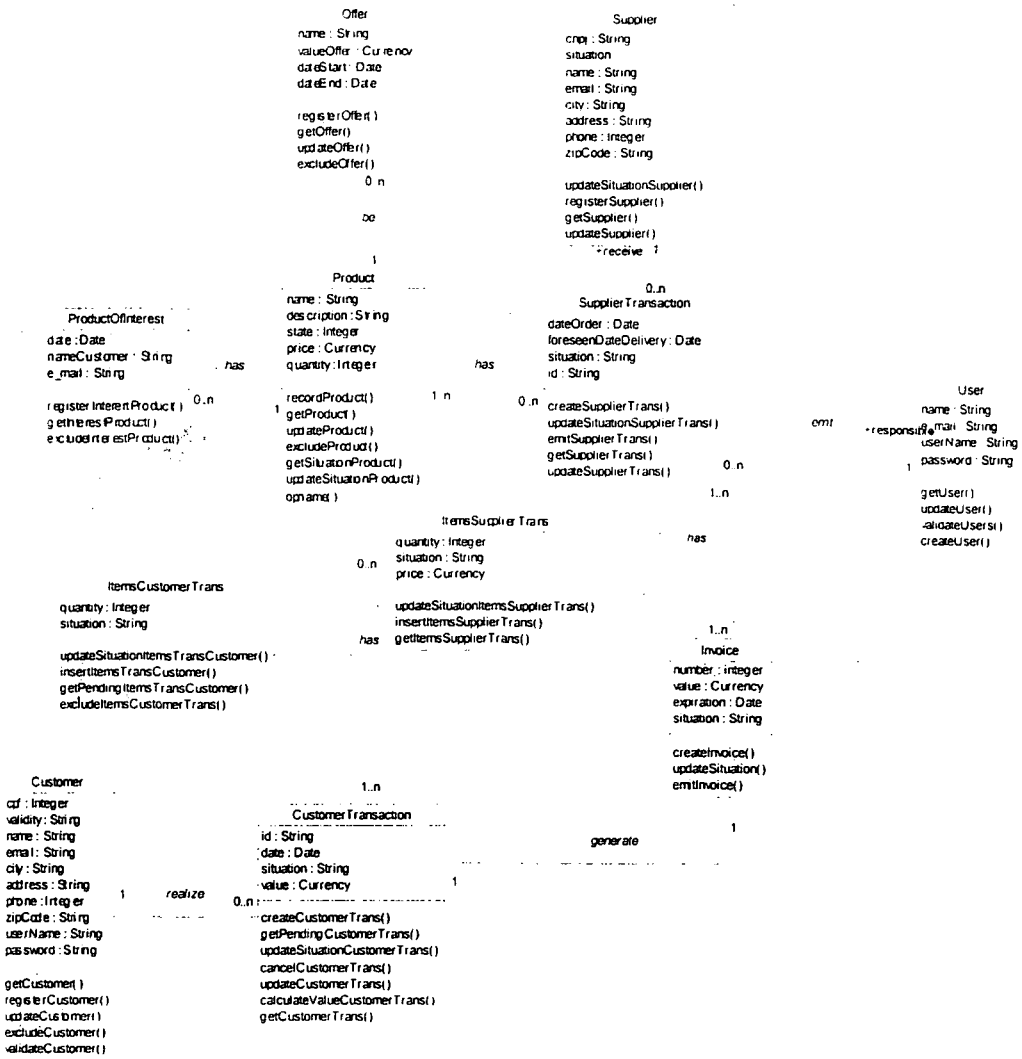
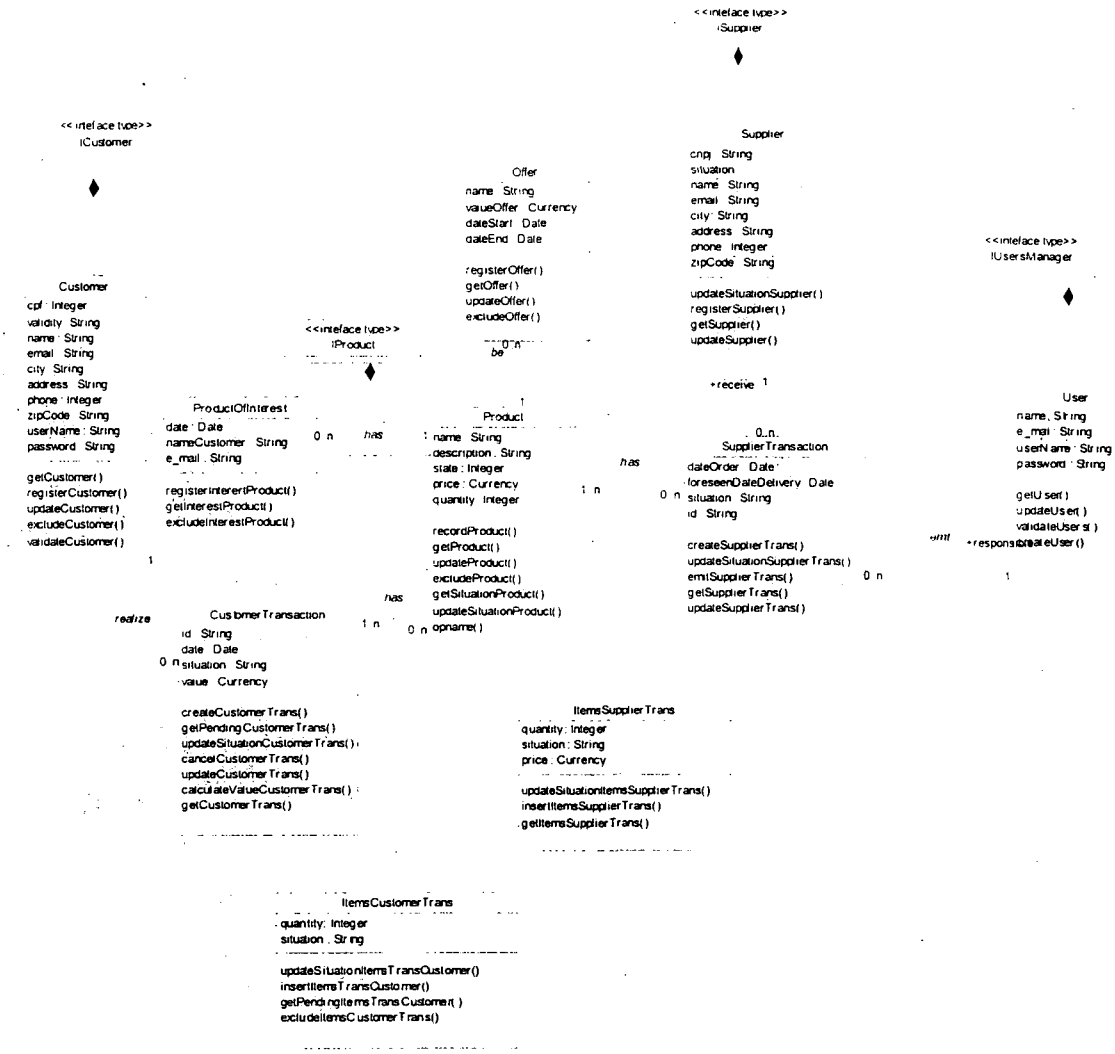


FIGURA 16 - DIAGRAMA DE RESPONSABILIDADE DAS INTERFACES DO NEGÓCIO – ARQUITETURA



O segundo estágio, estágio de interação do componente, define as interações que acontecerão entre os componentes. O diagrama utilizado neste estágio é o diagrama de interação para as interfaces do sistema e do negócio. Neste estágio, também são feitas as assinaturas das operações de cada interface.

Os diagramas de interação a seguir correspondem a interação dos métodos da interface *IProduct* do componente *Product* da arquitetura.

FIGURA 17 - DIAGRAMA DE INTERAÇÃO RecordProduct() – ARQUITETURA

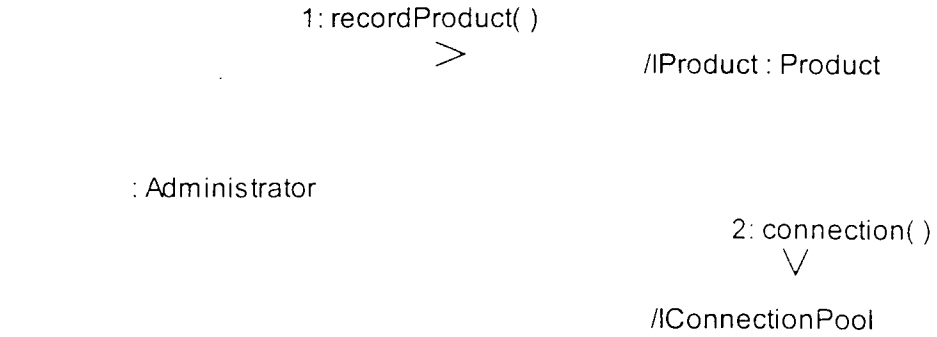


FIGURA 18 - DIAGRAMA DE INTERAÇÃO getProduct()– ARQUITETURA

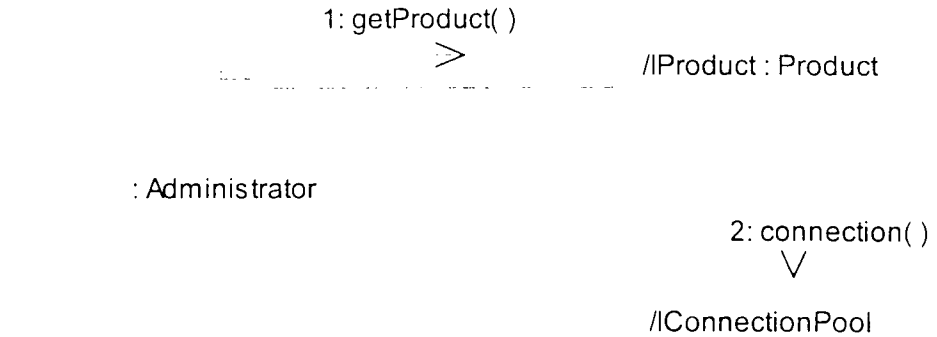


FIGURA 19 - DIAGRAMA DE INTERAÇÃO updateProduct()– ARQUITETURA

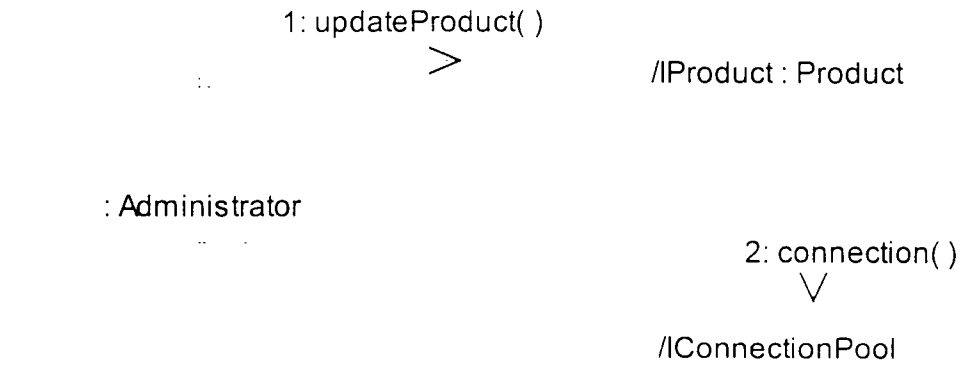


FIGURA 20 - DIAGRAMA DE INTERAÇÃO excludeProduct() – ARQUITETURA

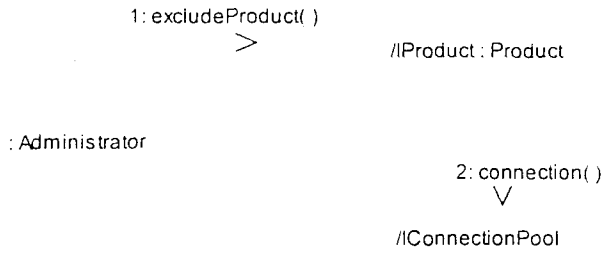


FIGURA 21 - DIAGRAMA DE INTERAÇÃO getSituationProduct()– ARQUITETURA

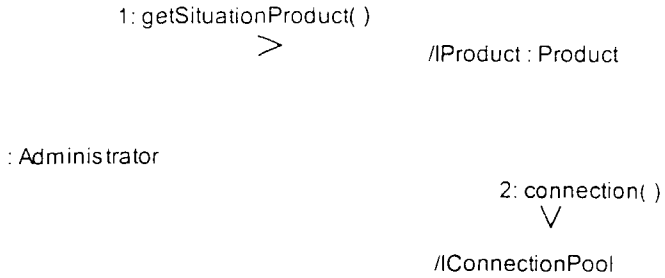
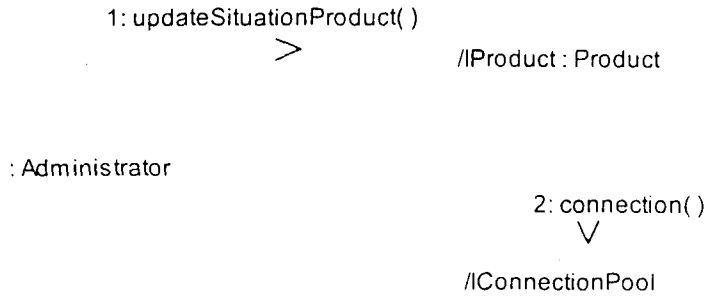


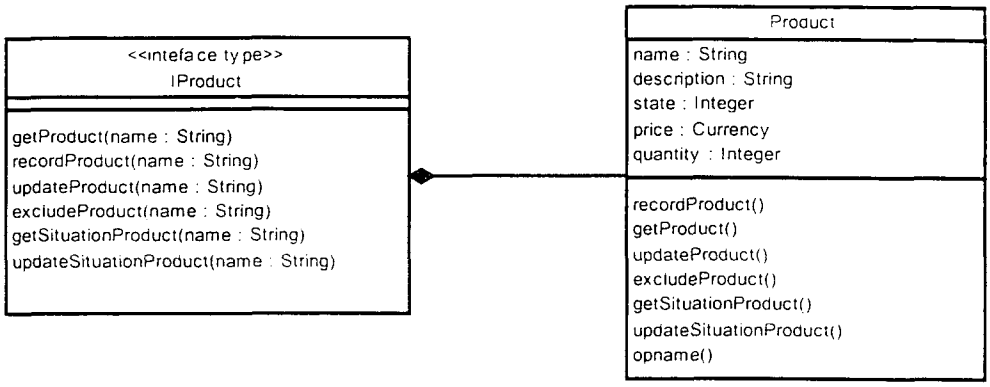
FIGURA 22 - DIAGRAMA DE INTERAÇÃO updateSituationProduct()– ARQUITETURA



No terceiro e último estágio, estágio de especificação de componente, são detalhadas as especificações das operações e as restrições produzidas no estágio anterior definindo as pré- e pós-condições dos métodos. As pré- e pós-condições são especificadas em *OCL* (*Object Constraint Language*) (WARMER; KLEPPE, 1999), uma linguagem declarativa usada para construir expressões lógicas. Os diagramas que compõem este estágio são os diagramas de especificação de interface (FIGURA 23) e de especificação de componentes (FIGURA 24), resultando na arquitetura do SI.

A FIGURA 23 exemplifica o diagrama de especificação de interface para a arquitetura *WbIS*, da interface *IProduct* e as restrições de pré- e pós-condições para essa interface.

FIGURA 23 - DIAGRAMA DE ESPECIFICAÇÃO DE INTERFACE: Iproduct – ARQUITETURA



context IProduct : :getProduct ()

pré:

product = **exists** (product | name = name)

post:

let theProduct – product **-> select** (product | product.name = name)

result.name = theProduct.name **and**

result.description = theProduct.description **and**

result.state = theProduct.state **and**

result.price = theProduct.price **and**

result.quantity = thePublication.quantity

Finalmente, um grupo de componentes semelhantes entre o *e-magazine* e os outros estudos de caso, foram identificados possibilitando assim a generalização da arquitetura.

A partir da generalização da arquitetura de componentes proposta é possível criar *e-commerce* com grau de reutilização e de fácil manutenção, permitindo a extensão dos componentes para atender as necessidades da aplicação específica.

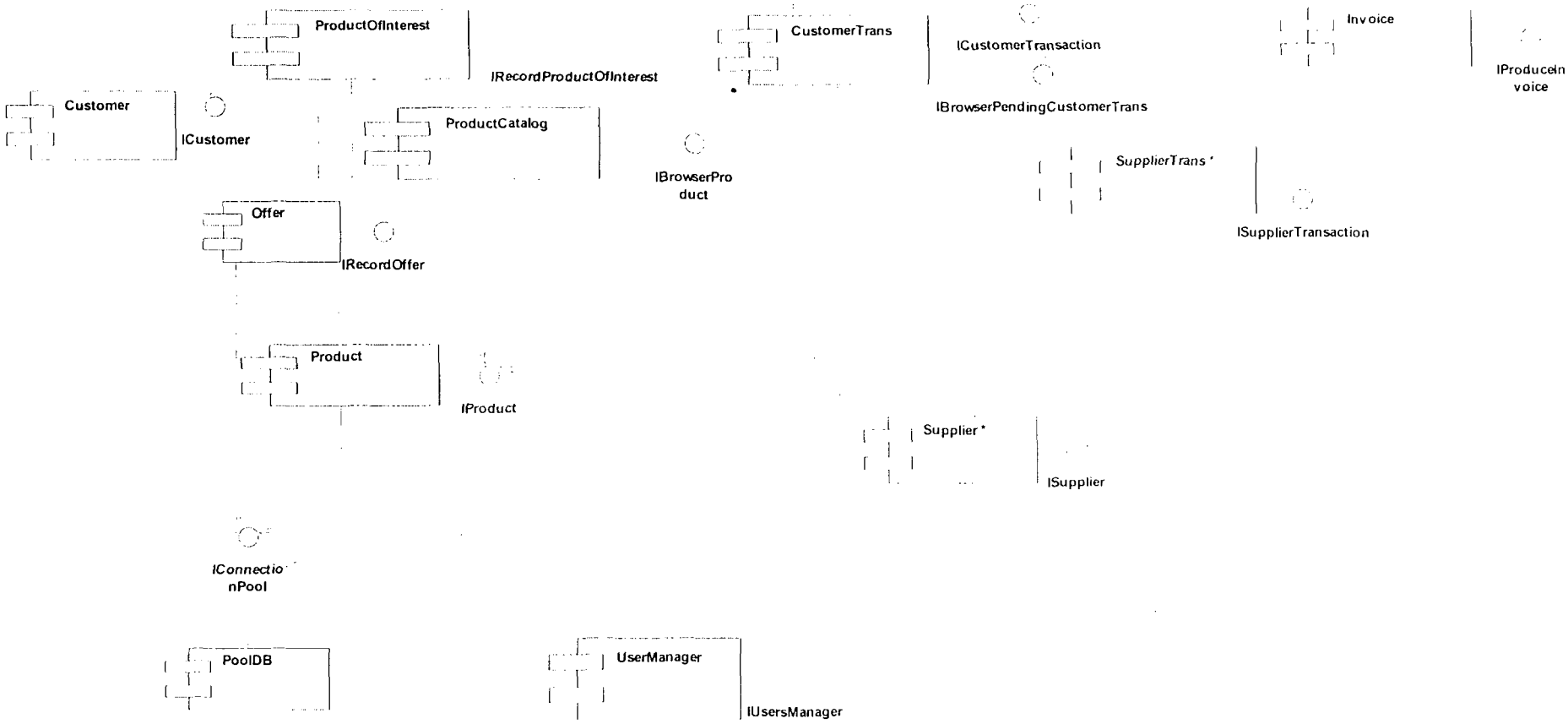
A FIGURA 24 apresenta a proposta inicial da arquitetura de componentes para *WbIS*. Os componentes com um asterisco (*) representam os componentes de um relacionamento *B2B*, os componentes com dois asteriscos (**) representam componentes externos ao sistema, ou seja, serão adquiridos de terceiros e os componentes sem asteriscos representam os componentes da arquitetura *WbIS*.

5.1.3 Aplicação da Arquitetura Generalizada

A arquitetura generalizada foi aplicada ao estudo de caso *e-rent-a-car*, apresentado no APÊNDICE 2. A partir da arquitetura proposta foram identificados os componentes necessários para o desenvolvimento do *e-rent-a-car*, e novos componentes específicos do negócio foram acrescentados à arquitetura do *e-rent-a-car*, assim a arquitetura foi revisada. Com este estudo de caso foi possível refinar os componentes inicialmente generalizados para a arquitetura, e verificar a comunicação dos componentes na arquitetura.

Esse estudo de caso trata a locação de carros pela *Web*. O cliente poderá consultar os carros disponíveis para locação e reservar ou locar os carros da loja. Após a definição do problema foram criados o modelo conceitual de negócio e o diagrama de casos de uso. Analisado os casos de uso da arquitetura (FIGURA 14) foi possível identificar os comportamentos semelhantes, assim o diagrama de casos de uso do *e-rent-a-car* (FIGURA 49) foi definido baseado no modelo da arquitetura *WbIS*.

FIGURA 24 – PROPOSTA INICIAL DA ARQUITETURA DE COMPONENTES PARA *WbIS* – NÍVEL LÓGICO



Tendo definido os dois primeiros modelos, o primeiro estágio do *UML Components* foi aplicado para definição do *e-rent-a-car*. Baseando-se no diagrama de tipo de negócio da arquitetura (FIGURA 15) o diagrama de tipo de negócio do estudo de caso foi definido como mostra a FIGURA 50, nela é possível identificar as classes da arquitetura (classes sem hachurado) e as classes específicas do domínio (classes com hachurado). Ainda neste estágio foi elaborado o diagrama de responsabilidade das interfaces do negócio (FIGURA 51) seguindo a mesma regra do diagrama de tipo de negócio.

O segundo estágio do método define a interação dos componentes. Os componentes da arquitetura que foram utilizados no estudo de caso continuam com as mesmas definições da arquitetura, somente para os componentes específicos do negócio foram feitas as novas interações.

No terceiro estágio do método, denominado especificação de componente, foram definidos os diagramas de especificação de interface e o diagrama de especificação de componente, além da definição das pré- e pós-condições. A mesma análise feita nos estágios anteriores foram aplicadas neste estágio, os diagramas da arquitetura foram a base para a definição dos diagramas do estudo de caso, sendo que, os componentes da arquitetura são definidos sem hachurado e os componentes específicos do estudo de caso com hachurado.

A modelagem do estudo de caso também foi feita na ferramenta *Rational Rose*. O mesmo arquivo da modelagem da arquitetura foi usado como base para o estudo de caso, os componentes que não foram utilizados foram apagados do modelo e os específicos do domínio foram acrescentados.

Nem todos os comportamentos definidos na arquitetura foram aplicados ao *e-rent-a-car*, por exemplo, os casos de uso *Record Transaction Supplier* e *Register Supplier*. Todas as classes e intefaces correspondentes as estes casos de uso foram eliminados do modelo do *e-rent-a-car*, assim como, os componentes que continham estes artefatos.

Outra necessidade encontrada foi a alteração do nome dos artefatos da arquitetura para facilitar o entendimento do modelo, por exemplo, o caso de uso da

arquitetura *Record Product* foi renomeado para *Record Car*. Como citado no APÊNCIDE 2, nos quadros denominados de aplicação da arquitetura proposta, foi feito a correspondência dos artefatos da arquitetura para os artefatos do *e-rent-a-car*, para que não houvesse a descaracterização dos componentes os nomes dos atributos e métodos continuaram os mesmos da arquitetura.

Também foi necessário acrescentar componentes a arquitetura do *e-rent-a-car* para obter às especificidades do estudo de caso, mantendo o nome do componente correspondente da arquitetura proposta mais a palavra *Specification*, por exemplo, *Car Specification*.

5.1.4 Ajustes da Arquitetura Generalizada

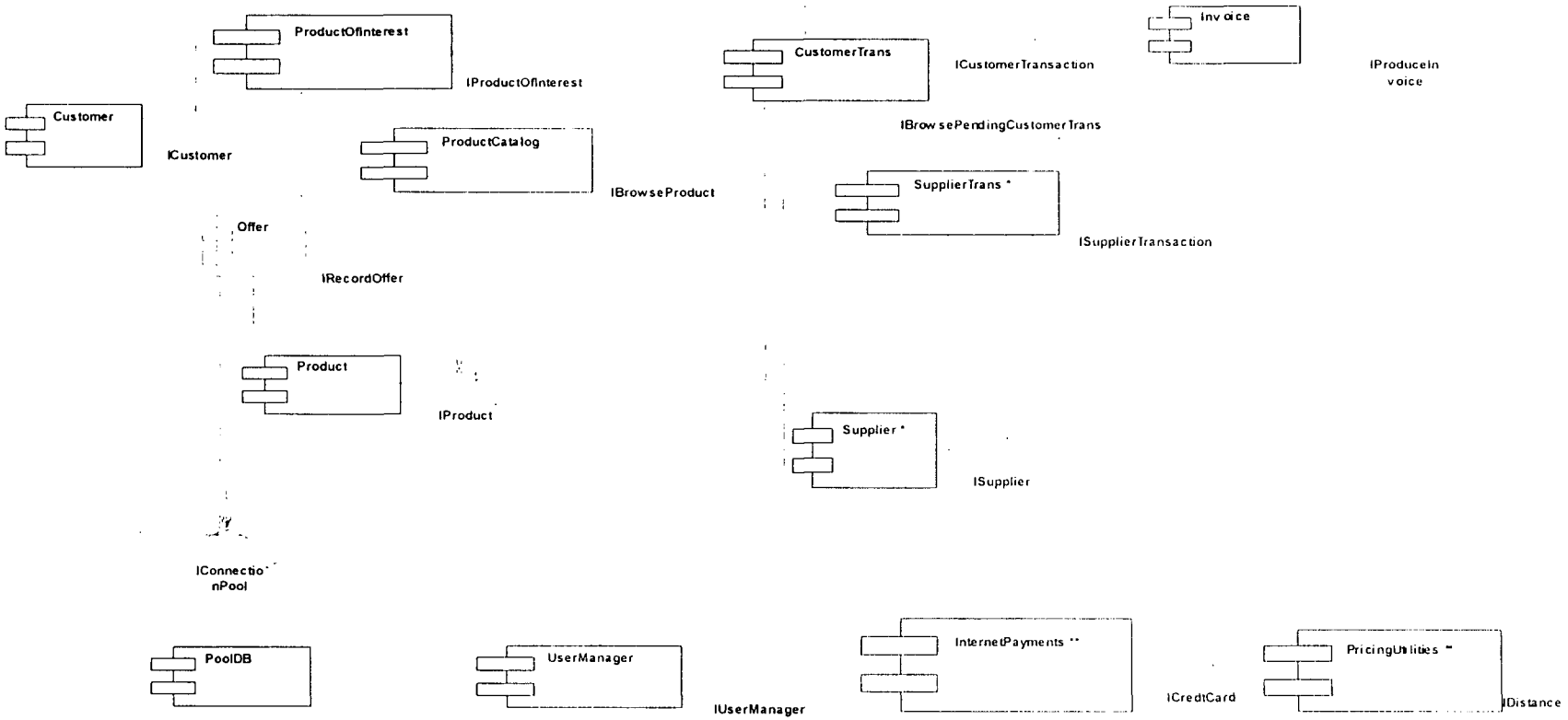
A arquitetura apresentada na FIGURA 25 contém os ajustes feito durante o desenvolvimento do estudo de caso *e-rent-a-car*.

De modo geral os componentes da arquitetura inicialmente proposta não sofreram grandes alterações durante o processo de aplicação da arquitetura ao estudo de caso *e-rent-a-car*. A aplicação dos componentes foi satisfatória permitindo desenvolver o estudo de caso com maior agilidade.

Os componentes que sofreram maiores alterações foram *Invoice* e *UserManager*. Isso aconteceu porque o tratamento dado a esses componentes no *e-magazine* era diferenciado do tratamento dado no *e-rent-a-car*, os ajustes foram feitos visando o aproveitamento dos componentes nos dois estudos de casos.

Outros dois componentes foram acrescentados, *InternetPayments* e *PrincingUtilities*. Estes componentes foram identificados por ALLEN (2001). No decorre do estudo da arquitetura foi identificada a importância destes componentes para a proposta da arquitetura.

FIGURA 25 - ARQUITETURA DE COMPONENTES PARA *WbIS* – NÍVEL LÓGICO



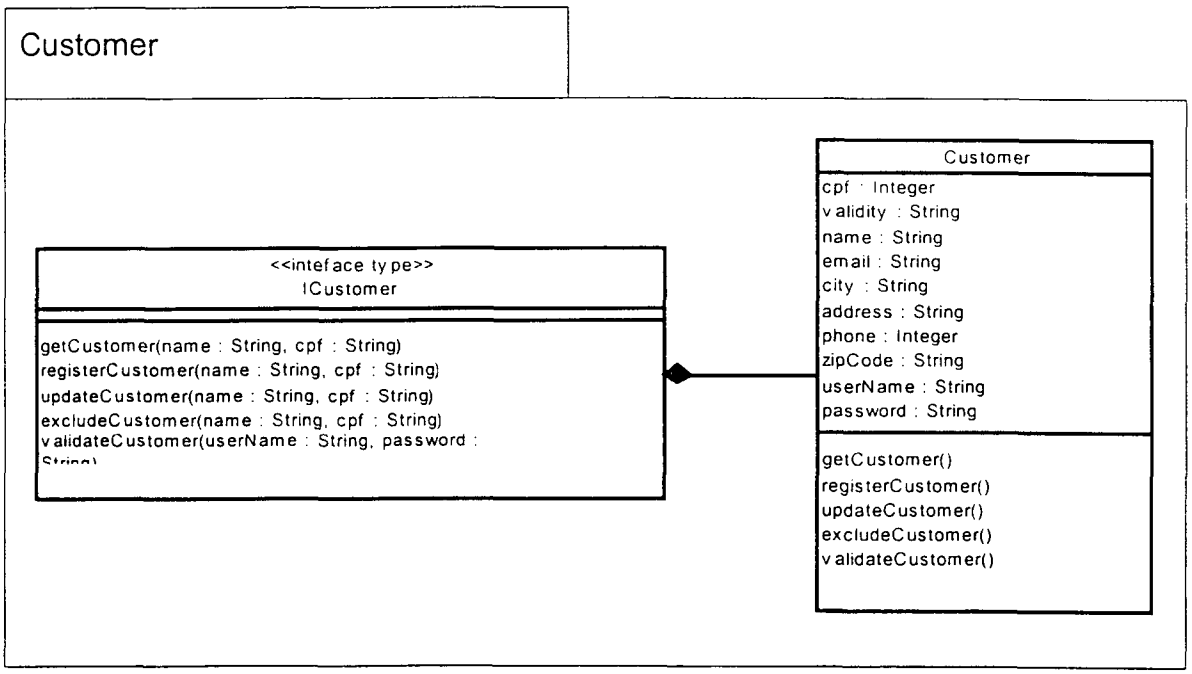
5.1.4.1 Descrição dos componentes identificados

A arquitetura de ALENCAR, COWAN e LUO (2002) não identifica todos os componentes necessários para a construção de um *WbIS*, e também não faz a especificação destes componentes. Nesta seção, serão apresentados os componentes necessários para complementar a arquitetura, para um sistema de *e-commerce B2C*.

Após uma análise dos *WbIS*, da arquitetura *CIS Framework* e do desenvolvimento e avaliação do estudo de caso *e-magazine*, foram identificados os seguintes componentes para compor a arquitetura:

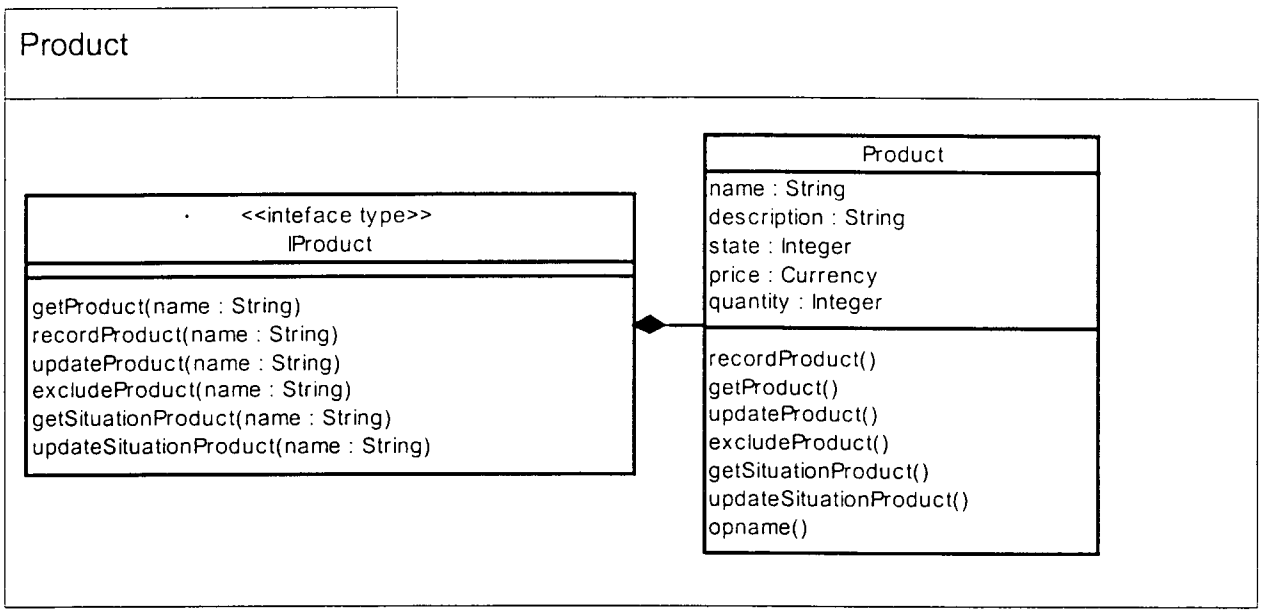
a) componente gerenciador de cliente (*Customer*) (FIGURA 26). Gerencia os dados referentes aos clientes. Para que o cliente possa efetivar uma transação ele deverá ser cadastrado e seus dados validados pelo sistema. Este componente requer serviços do componente *PoolDB*, responsável pela conexão com o banco de dados para armazenar, alterar, excluir e validar cliente. O serviço fornecido por este componente é o de consulta aos dados do cliente para que uma transação possa ser efetuada

FIGURA 26 - COMPONENTE GERENCIADOR DE CLIENTE



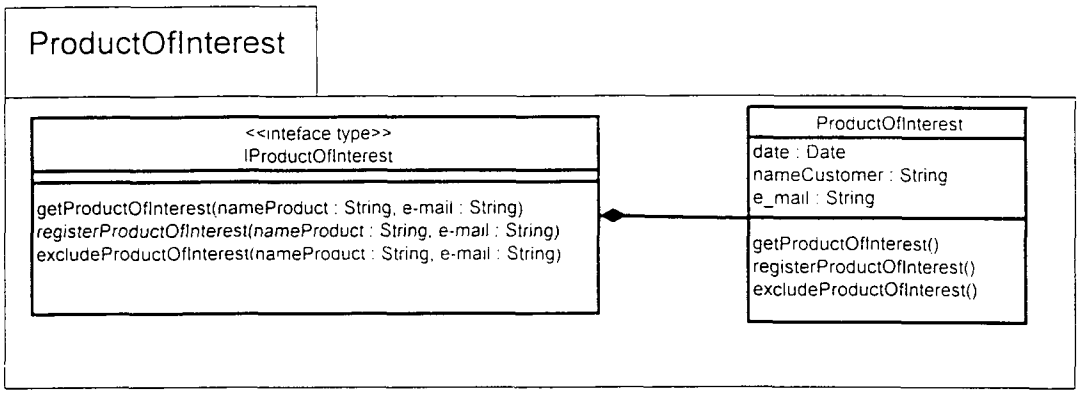
b) componente gerenciador de produtos (*Product*) (FIGURA 27). Este componente é responsável pelo gerenciamento dos produtos comercializados pelo sistema, pelo controle de estoque e pelos produtos disponíveis no catálogo de produtos. Ele requer serviços do componente *PoolDB*, responsável pela conexão com o banco de dados para armazenar, atualizar e excluir os produtos. O serviço fornecido por este componente é o de consulta aos dados do produto para que uma transação com o cliente ou fornecedor possa ser efetuada, para registro da promoção de um produto e para registrar o interesse de um cliente por um determinado produto;

FIGURA 27 - COMPONENTE GERENCIADOR DE PRODUTO



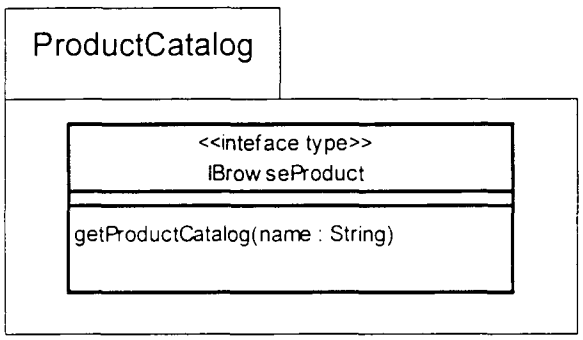
c) componente reserva de produto (*ProductOfInterest*) (FIGURA 28). Responsável pelo registro de interesse por um determinado produto. Este componente requer serviços do componente *PoolDB*, responsável pela conexão com o banco de dados para armazenar e excluir o interesse por um produto. Requer também o serviço de informações sobre os produtos cadastrados e que estejam indisponíveis, o componente responsável por este serviço é o *Product*;

FIGURA 28 - COMPONENTE RESERVA DE PRODUTO



d) componente catálogo de produto (*ProductCatalog*) (FIGURA 29). Este componente oferece o serviço de busca por produtos no sistema, através dele poderá ser verificado se um produto é comercializado e consultar os dados do produto. Este componente requer serviços do componente *Product* para obter informações sobre os produtos cadastrados;

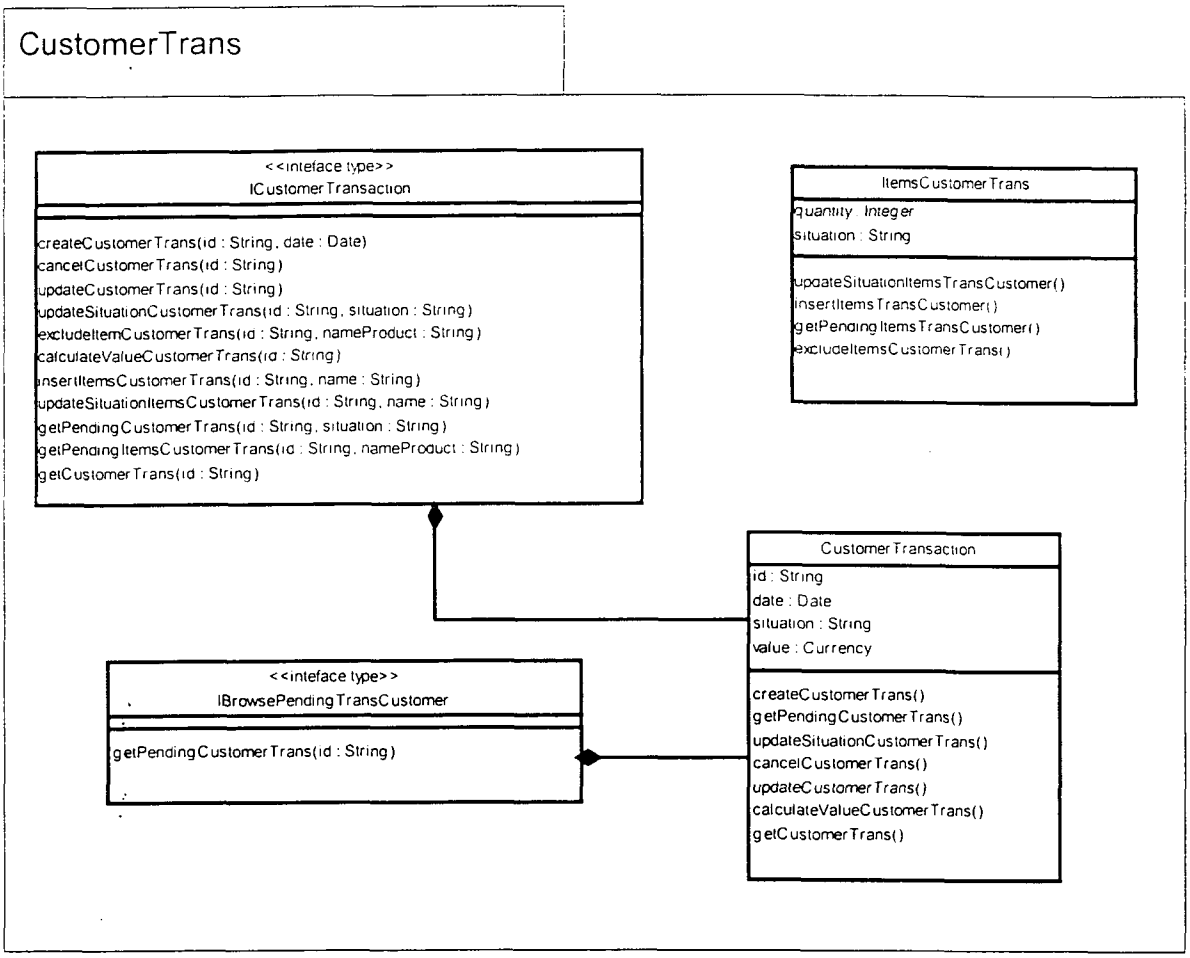
FIGURA 29 - COMPONENTE CATÁLOGO DE PRODUTO



e) componente transação com cliente (*CustomerTrans*) (FIGURA 30). Este componente representa o "carrinho de compra", onde os itens comprados são armazenados para no final realizar a transação com o cliente. Este componente requer serviços do componente *PoolDB*, responsável pela conexão com o banco de dados para armazenar, alterar, excluir a transação com o cliente. Requer também os serviços de: informações sobre os produtos cadastrados, cujo componente responsável pelo serviço é o *Product*; informações sobre os clientes cadastrados, o componente responsável é o *Customer;* e informações sobre o valor do frete, o componente responsável é

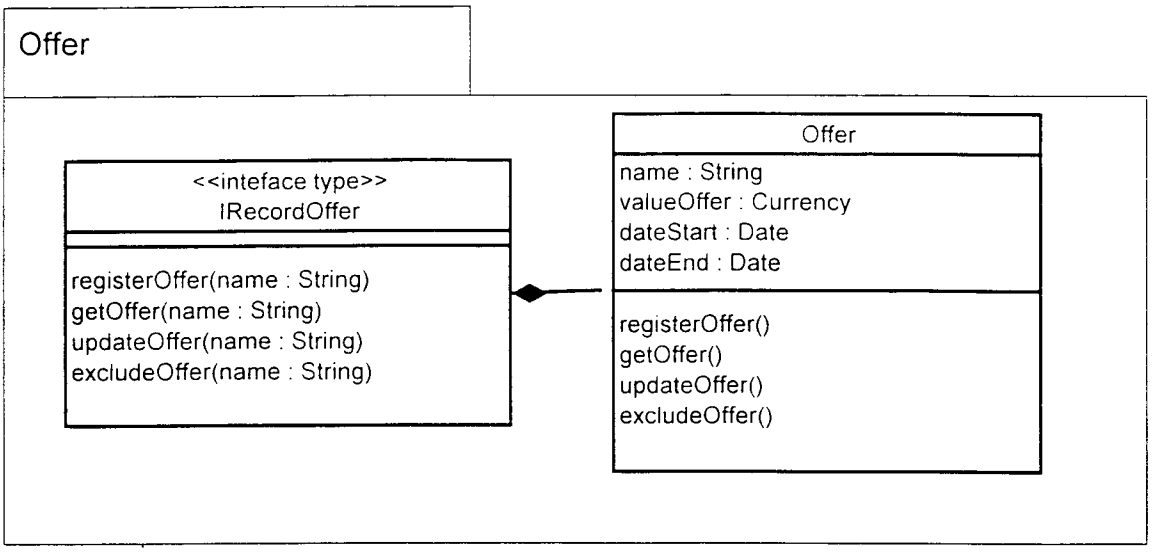
o *PrincingUtilities*. O serviço fornecido por este componente é o de consulta aos dados da transação com o cliente para que uma fatura possa ser emitida;

FIGURA 30 - COMPONENTE TRANSAÇÃO COM O CLIENTE



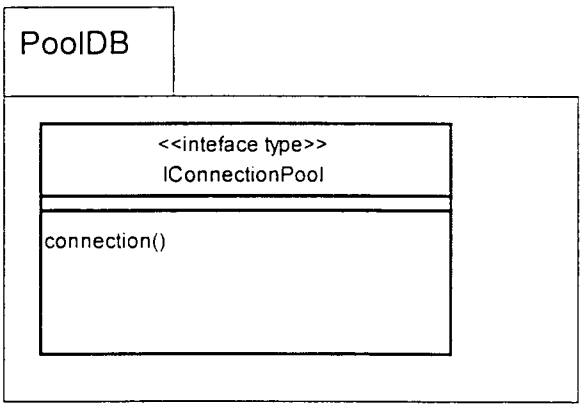
f) componente gerenciador de promoções (*Offer*) (FIGURA 31). Este componente é responsável pelo armazenamento das promoções/ofertas oferecidas pelo SI. Este componente requer serviços do componente *PoolDB*, responsável pela conexão com o banco de dados para armazenar, alterar ou excluir os dados da promoção. Requer também informações sobre os produtos cadastrados, o componente responsável por este serviço é o *Product*;

FIGURA 31 - COMPONENTE GERENCIADOR DE PROMOÇÕES



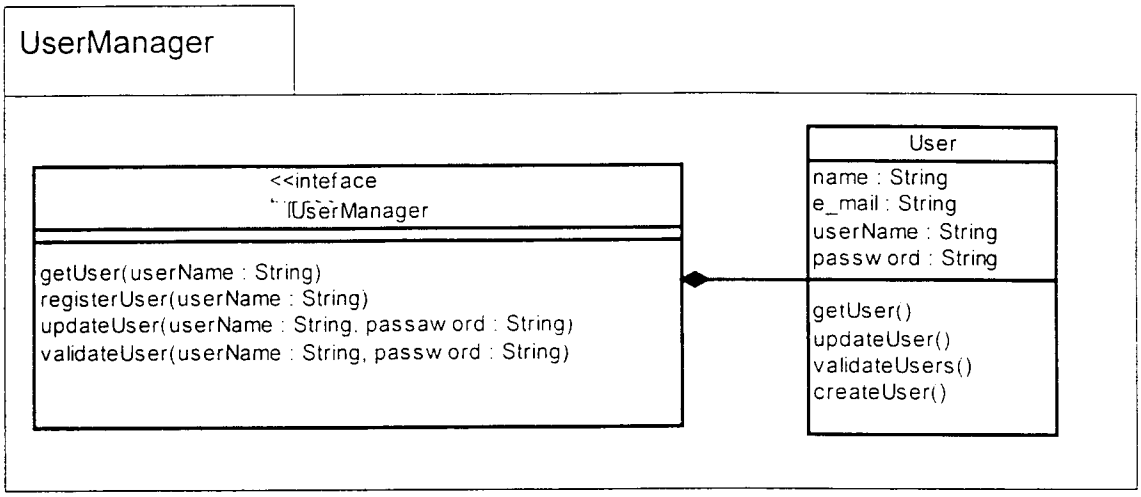
g) componente conexão com banco de dados (*PoolDB*) (FIGURA 32). Este componente de infra-estrutura é responsável pela comunicação do sistema com o banco de dados. Oferece o serviço de conexão com o banco de dados que permite armazenar, consultar, alterar ou excluir dados. Os componentes que utilizam estes serviços são: *Customer, Offer, ProductOfInterest, Product, CustomerTrans, Supplier, SupplierTrans* e *UserManager*;

FIGURA 32 - COMPONENTE CONEXÃO COM BANCO DE DADOS



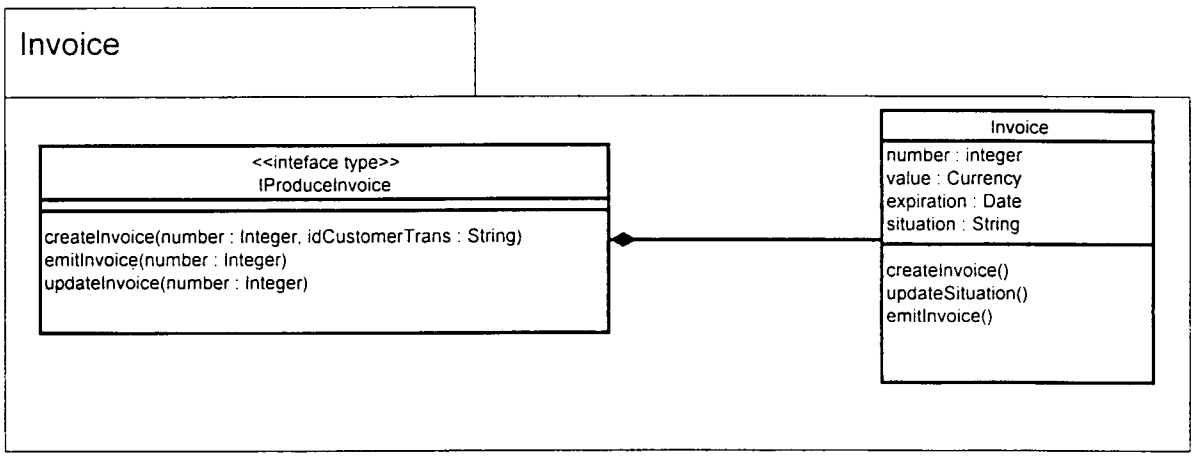
h) componente gerenciador de usuário (*UserManager*) (FIGURA 33). Este componente é responsável pelo gerenciamento de usuários. É através deste componente que o sistema faz a validação dos usuários para acesso a determinadas operações e transações. Este componente requer serviços do componente *PoolDB*, responsável pela conexão com o banco de dados para armazenar ou alterar os dados dos usuários;

FIGURA 33 - COMPONENTE GERENCIADOR DE USUÁRIO



i) componente gerenciador de fatura (*Invoice*) (FIGURA 34). Este componente é responsável pela criação e emissão das faturas referentes às transações realizadas com os clientes. Este componente requer serviços do componente *PoolDB*, responsável pela conexão com o banco de dados para armazenar, alterar ou emitir os dados das faturas. Requer também informações sobre as transações com clientes concluídas e informações sobre o pagamento, os componentes responsáveis, respectivamente por estes serviços são os *CustomerTrans* e *InternetPayments*.

FIGURA 34 - COMPONENTE GERENCIADOR DE FATURA



Os dois componentes a seguir, identificados por ALLEN (2001), representam componentes adquiridos de terceiros, ou seja, são necessários para a maioria dos *e-*

commerce, mas são encontrados prontos, não necessitando que os mesmos sejam implementados:

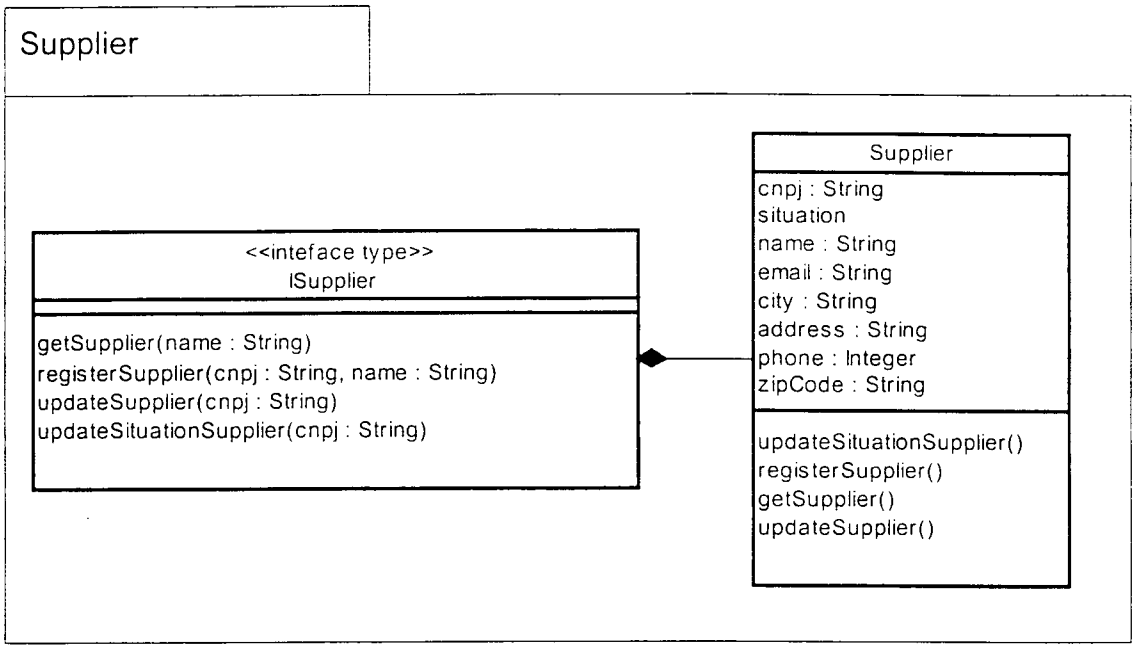
a) componente estimativa de frete (*PrincingUtilities*). Este componente é responsável pela validação do CEP do endereço do cliente e cálculo do frete para entrega do produto;

b) componente pagamento Internet (*InternetPayments*). Este componente é responsável pela validação da transação bancária nas emissões das faturas com cartão de crédito ou pagamentos *on-line*. Este componente se refere a sistemas existentes no mercado que podem ser adquiridos para efetivar o pagamento da transação comercial entre eles estão: o *NetCheque*, um sistema de pagamentos eletrônico desenvolvido para Internet pelo Instituto de Ciências da Informação da Universidade do Sul da Califórnia; o *eCheck* (*Electronic Check*) ou Cheque Eletrônico, foi introduzido numa iniciativa do *FSTC* (*Financial Services Technology Consortium*), atualmente está sob a responsabilidade do *CommerceNet*, um consórcio sem fins lucrativos que reúne empresas, governo e universidades; o projeto *ACTION* (*ACH Credit Transactions Initiated Online*) da NACHA; entre outros (MARTINS, 2002).

Alguns outros componentes foram identificados, porém, estes componentes não fazem parte de um *e-commerce - B2C.* Eles abrangem uma outra parte do negócio, o relacionamento entre empresas, o *e-commerce* chamado *B2B*. Entre os componentes *B2B*, pode-se identificar os seguintes componentes:

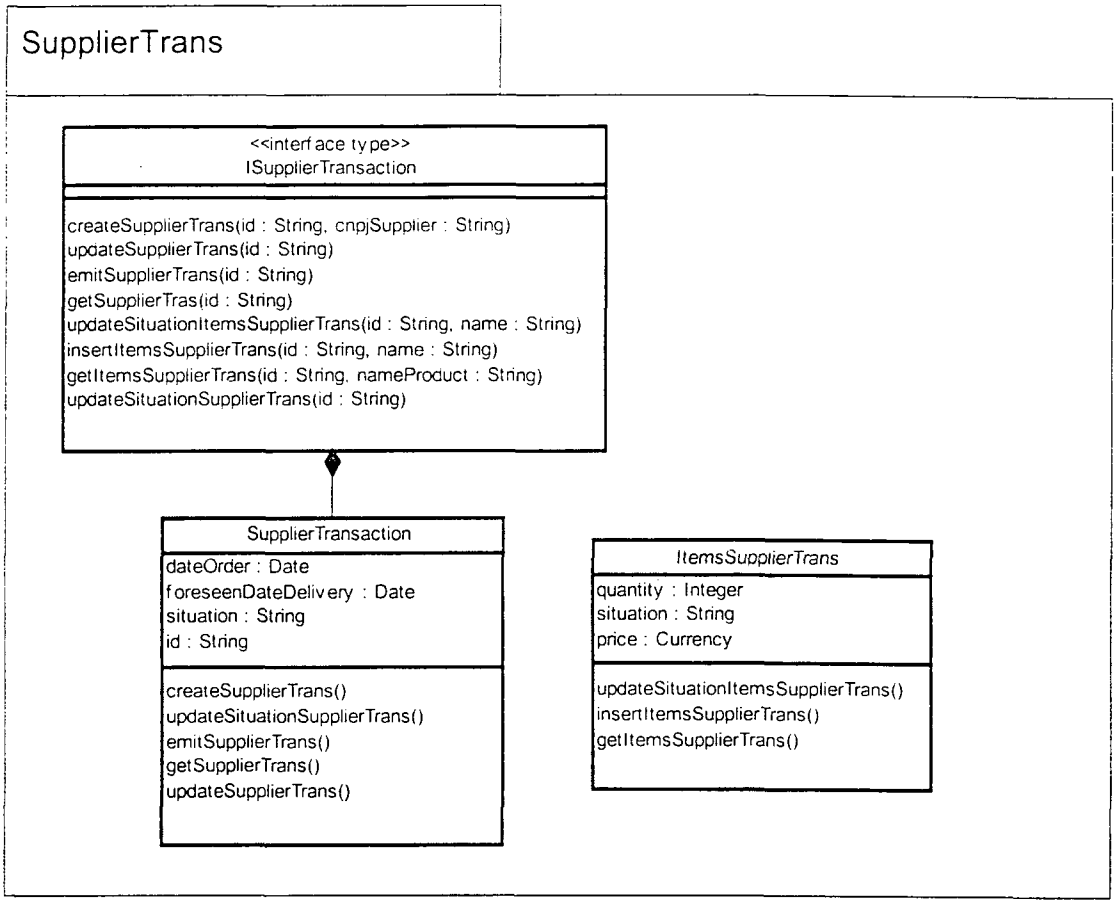
a) componente gerenciador de fornecedor (*Supplier*) (FIGURA 35). Este componente é responsável pelo gerenciamento dos dados dos fornecedores dos produtos comercializados pelo sistema. Este componente requer serviços do componente *PoolDB*, responsável pela conexão com o banco de dados para armazenar e alterar;

FIGURA 35 - COMPONENTE GERENCIADOR DE FORNECEDOR



b) componente transação com fornecedor (*SupplierTrans*) (FIGURA 36). Este componente é responsável pela criação e emissão de pedidos de produtos, para serem comercializados pelo sistema, aos fornecedores. Este componente requer serviços do componente *PoolDB*, responsável pela conexão com o banco de dados para armazenar ou alterar os dados da transação. Ele requer também informações sobre os produtos cadastrados, o componente responsável por este serviço é o *Product*, e informações sobre os fornecedores cadastrados, o componente responsável por este serviço é o *Supplier*.

FIGURA 36 - COMPONENTE GERENCIADOR DE TRANSAÇÃO COM O FORNECEDOR



Avaliando os componentes descritos anteriormente foi possível fazer o mapeamento dos componentes sugeridos pela arquitetura *CIS Framework* para a arquitetura *WbIS*, identificando e especificando esses componentes. Alguns destes componentes foram unidos em um único componente na arquitetura. Os componentes relativos à especificação de ALENCAR, COWAN e LUO (2002) são:

a) componente *UserManager* corresponde à implementação dos componentes: "gerenciamento de usuário", "personalização e perfil do usuário" e "categoria de navegação";

b) componente *CatalogProduct* corresponde à implementação dos componentes: "gerenciamento de conteúdo dinâmico" e "consultas de informação";

c) componente *PoolDB* corresponde à implementação do componente "gerenciamento de estado". Mas além do gerenciamento de estado ele tem como objetivo principal implementar os recursos necessários para acesso ao banco de dados sem que os outros componentes que precisam realizar a busca de informações no banco de dados se preocupem com a conexão.

Para relacionar a arquitetura proposta a arquitetura *CIS Framework* é preciso identificar os componentes que precisam se relacionar com as camadas. Começando com o componente *PooDB*, do nível lógico definido nesta dissertação, ele faz a comunicação com o banco de dados, realizando a comunicação entre a camada de componente específico do negócio e a camada de infra-estrutura. No estudo de caso *e-magazine* foi utilizado o banco de dados *Interbase*® 6.0 (BORLAND®, 2001), este componente foi projetado para comunicação com banco de dados relacionais, não tratando assim a comunicação dos componentes lógicos com um banco de dados OO.

A comunicação entre a camada de componente específico do negócio e a camada de apresentação é realizada por meio das interfaces dos componentes. Para as interfaces lógicas dos componentes será gerada interface de apresentação, muitas vezes os serviços de uma interface lógica podem estar agrupados aos serviços de uma outra interface lógica em uma única interface de apresentação. No *e-magazine* a tecnologia empregada para esta comunicação foram as tecnologias *Java Server Page (JSP)* (SUN MICROSYSTEMS, 2002(a)) e *Servlet* (SUN MICROSYSTEMS, 2002(b)), as páginas de apresentação do SI foram feitas unindo *HTML* com estas duas tecnologias, o *HTML* apresenta somente páginas estáticas e o *JSP* e *Servlet* permite a criação de páginas dinâmicas.

5.2 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a arquitetura de componentes para *WbIS*, nível lógico, proposta. O processo seguido para a definição da arquitetura de componentes, os componentes identificados e sua aplicação na arquitetura *CIS Framework*.

O processo proposto neste trabalho envolveu a exploração do domínio dos sistemas para *Web*, o método *UML Components*, a aplicação do estudo de caso para generalização da arquitetura e a aplicação de outro estudo de caso para avaliação da arquitetura.

Com este trabalho foi possível identificar um grupo de componentes semelhantes entre os *e-commerce* e propor a arquitetura, esta identificação tem um grande valor, pois possibilitam o desenvolvimento de SI para *Web* com segurança, confiabilidade e menor tempo de trabalho. Desenvolver a partir de componentes validados e testados economiza tempo e contribui para a produção de SIs com maior qualidade gerando menos manutenção.

O método seguido foi importante para a definição dos componentes, pois enfatiza a verificação e validação das interfaces e a comunicação entre os componentes.

6 CONCLUSÕES E TRABALHOS FUTUROS

Os *WbIS* estão em contínuo crescimento e vem sendo explorados cada vez mais nos mais diversos negócios realizados na *Web*. É evidente a necessidade existente nas empresas de aumentarem seus negócios e reduzir custos. A engenharia de software vem buscando desenvolver métodos, técnicas, ferramentas e notações que sirvam de suporte para alcançar estes objetivos. Uma técnica importante para melhorar a qualidade e a produtividade do processo de software é a reutilização de soluções.

A arquitetura de software surgiu com a necessidade de padronizar a forma de construção de um software. Criando estruturas que pudessem ajudar a construção de outros produtos semelhantes aos já desenvolvidos. Conceitos como arquitetura de software, desenvolvimento baseado em componentes, padrões e *frameworks* estão sendo estudados e aprimorados para que possam ser aplicados no desenvolvimento de SIs.

Esta dissertação propôs uma arquitetura de componentes para *WbIS*. O desenvolvimento tomou como base a arquitetura *CIS Framework* e os estudos de caso relacionados na seção 5.1.1. O projeto seguiu o processo sugerido pelo método *UML Components* (CHESSMAN; DANIELS, 2000). O processo é composto dos estágios: identificação dos componentes, interação dos componentes e especificação dos componentes. O *UML Components* foi escolhido por ser uma abordagem geral de DBC, baseado em *UML*, que envolve conceitos de arquitetura de software.

O método foi aplicado ao estudo de caso *e-magazine*, ao final do processo foi definida a arquitetura do *e-magazine* que serviu como base para a generalização e obtenção da arquitetura proposta e de seus componentes. O método também foi aplicado a arquitetura proposta para uma nova generalização da arquitetura.

Para avaliação da arquitetura proposta fez-se a aplicação da arquitetura inicial obtida ao estudo de caso *e-rent-a-car* que é um *e-commerce B2C*, com características diferentes dos outros estudos de caso analisados. A aplicação da arquitetura permitiu avaliar cada componente individualmente e verificar suas

características para aplicação em outros negócios. No estudo de caso, a partir da aplicação dos componentes da arquitetura proposta, novos componentes foram criados para cobrir as especificidades do negócio o que implica na reutilização e ajuste da arquitetura generalizada.

A principal contribuição deste trabalho é a arquitetura de componentes para *WbIS*. Esta arquitetura pode ser usada para facilitar o processo de desenvolvimento de diferentes *e-commerce* que possuam características comuns, mas que também tem aspectos diferentes de acordo com as necessidades do negócio. A exploração do método *UML Components* também pode ser vista como contribuição deste trabalho, pois é um método novo, e a exploração de seus estágios permitiu refinar os componentes de forma satisfatória.

Método para o desenvolvimento de *WbIS*, bem como técnicas de reutilização neste domínio ainda é recente. A necessidade de construir *WbIS* para não ficar atrás da concorrência, tem gerado SIs desestruturados e de difícil manutenção. Esta área é promissora, as empresas não querem perder mercado, e o futuro do comércio está na *Web*. Por isso há a necessidade de criar uma arquitetura e identificar métodos que auxiliem o desenvolvimento dos *WbIS*.

Dentre os trabalhos futuros provenientes deste esforço pode-se destacar: a aplicação da arquitetura a outros estudos de caso, permitindo um aprimoramento da arquitetura; a generalização do componente responsável pela comunicação para trabalhar com banco de dados OO; a extensão da arquitetura para as demais categorias de *e-commerce*.

REFERÊNCIAS

ALENCAR, P. S. C.; COWAN, D. D.; LUO, M. **A Framework for Community Information Systems**. Annals of Software Engineering 13(1-4): 381-411, June, 2002.

ALLEN, P. **Realizing e-Business with Components**. Great Britain: Addison-Wesley, 2001.

ALVES, C. F. **Seleção de Produtos de Software Utilizando uma Abordagem Baseada em Engenharia de Requisitos.** Recife, 2001. Dissertação (Mestrado em Ciência da Computação) – Centro de Informática, Universidade Federal de Pernambuco.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Pratice**. Massachusetts: Addison Wesley, 1998.

BEM-NATAN, R. **Objects on the Web: Designing, Building, and Deploying Objects-Oriented Applications for the Web**. New York: McGraw-Hill, 1997.

BOOCH, G. RUMBAUGH, J; JACOBSON, I. **The Unified Modeling Language Users Guide**, [S.l]: Addison-Wesley Publishing Company, 1999.

BORLAND®. **Interbase**. Disponível em: <http://www.borland.com/interbase/> Acesso em: 20/09/2001.

BOSCH, J. **Design & Use of Software Architecture: Adopting and Evolving a Product-line approach**. Great Britain: Addison-Wesley Publishing Company, 2000.

CHEESMAN, J.; DANIELS, J. **UML Components – A Simple Process for Specifying Component –Based Software**. [S.l.]: Addison Wesley, 2000.

COOK, M. E.; LAVIGNE, M. F.; PAGANO, C. M.; DAWES, S. S.; PARDO, T. A. **Making a Case for Local E-Government.** Center for Technology in Government University at Albany, SUNY, 2002. Disponível em: < http://www.ctg.albany.edu/egov/making_a_case.pdf> Acesso em: 10/04/2003.

COOK, S.; DANIELS, J. **Designing Object Systems**. [S.l.]: Printice Hall, 1994.

D'SOUZA, D.; WILLS, A. **Objects, Components and Frameworks with UML – The Catalysis Approach**. [S.l.]: Addison Wesley Publishing Company, 1999.

FOWLER, M.; KENDALL, S. **UML Distiled – Applying the Standard Object Modeling Language**. [S.l.]: Addison-Wesley, 1997.

GABRIEL, G. **Estudo de Sistemas de Informação para Web.** Maringá, 2001. 116 f. Monografia (Oficina de Engenharia de Software) – Departamento de Informática, Universidade Estadual de Maringá.

GABRIEL, G. **Proposta de uma Arquitetura de Componentes para Sistemas de Informação Baseados na Web – Nível Lógico.** Maringá, 2002. 44 f. Projeto de Qualificação (Mestrado em Informática) - Departamento de Informática, Universidade Estadual de Maringá.

GELLERSEN, H.; WICKE, R.; GAEDKE, M. **WebComposition: An Object-Oriented Support System for the Web Engineering Life Cycle**. Sixth International World Wide Web Conference, California: USA, 1997. Disponível em <http://www.scope.gmd.de/info/www6/technical/paper232/paper232.html> Acesso em: 21/09/2001.

IBM®. **WebSphere**. Disponível em: http://www.ibm.com/websphere Acesso em: 18/08/2002.

ISAKOWITZ, T. STOHR, E. A., BALASUBRAMANIAN, P., **RMM: A Methodology for Structured Hypermedia Desing.** Disponível em: <http://rmm-java.stern.nyu.edu/rmm> Acesso em 20/09/2001.

ISAKOWITZ, T. STOHR, E. A., BALASUBRAMANIAN, P., **RMM: A Methodology for Structured Hypermedia Desing.** Communications of the ACM, August 1995.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The Unified Software Development Process**. [S.l.]: Addison-Wesley, 1999.

LAZILHA, F. R. **Uma Proposta de arquitetura de Linha de Produto para Workflow Management Systems**. Dissertação. Universidade Federal do Rio Grande do Sul. Instituto de Informática. Porto Alegre. Janeiro de 2002.

LIMA, F. A.; PRINCE, R. T. **Towards an Integrated Design Methodology for Internet-based Information System**. Fifth Intenational Workshop on Engennering Hypertext Funcionality, Internacional Conference on Software Engennering. Kyoto, 1998. Disponível em: http://www.inf.ufrgs.br/~flima/paper/paper.html Acesso em 26/08/2001.

LIMA, I. N.; LIFSCHITZ, S. **Arquiteturas de Integração Web SGBD: um Estudo do Ponto de Vista de Sistemas de Banco de Dados**. XXV Software and Hardware Seminar (SEMISH'98). Belo Horizonte, 1998.

MARTINS, P. R. D. **Sistemas de Pagamento Eletrônico.** Trabalho Acadêmico (Tecnologias para Comércio Eletrônico) - Mestrado em Computação, Instituto de Computação, Universidade de Campinas. Disponível em: < http://www.dcc.unicamp.br/~rdahab/cursos/mp205/notas_links_trabalhos.html >

Acesso em: 11/11/2002.

MATTISON, R.; KILGER-MATTISON, B. **Web Warehousing and Knowledge Management**. [S.l.]: McGraw-Hill, 1999.

MICROSOFT CORPORATION. **.NET Framework**. Disponível em: <http://msdn.microsoft.com/netframework> Acesso em: 20/09/2002.

O'BRIEN, A. J. **Sistemas de Informação e as Decisões Gerenciais na Era da Internet** . Tradução Cid Knipel Moreira. São Paulo: Saraiva, 2001.

ORACLE CORPORATION. Disponível em: <http://www.oracle.com> Acesso em: 22/09/2002.

PARDO, T A.. **Realizing the promise of digital government: it's more than building a web site**. Disponível em: <http://www.cisp.org/imp/october_2000/10_00pardo.htm> Acesso em: 10/04/2003.

PAZZINATO, M. R. **Princípios para o Projeto de Arquitetura para Sistemas de Informação baseados na Web**. Maringá, 2003. 58f. Trabalho de Graduação. Curso de Ciência da Computação, Centro de Tecnologia, Universidade Estadual de Maringá.

PETERS, J. F.; PEDRYCZ, W. **Engenharia de Software**. Tradução: A. P. Garcia. Rio de Janeiro: Campus, 2001. Tradução de: An engineering approach.

POLCELLI, A., SOUZA, N. N. E. de. **Modelagem para Comércio Eletrônico & Business Intelligence**. Campinas 2002. Trabalho Acadêmico (Tecnologias para Comercio Eletrônico) - Mestrado em Computação, Instituto de Computação, Universidade de Campinas. Disponível em: < http://www.dcc.unicamp.br/~rdahab/cursos/mp205/notas_links_trabalhos.html > Acesso em: 11/11/2002.

POLTRONIERI, A.; OLIVEIRA FILHO, D.; NUNES, R. B. **Métodos para Desenvolvimento Baseado em Componentes.** Monografia. Universidade Federal do Espirito Santo, Mestrado em Informática. 2002.

RATIONAL SOFTWARE CORPORATION. **Rational Rose.** Disponível em <http://www.rational.com> Acesso em: 20/08/2002.

RATIONAL SOFTWARE CORPORATION. **Rational Unified Process: Best Practices for Software Development Teams.** Disponível em: < http://www.augustana.ab.ca/~mohrj/courses/2000.winter/csc220/papers/rup_best_practices/rup_bestpractices.html> Acesso em: 2001.

RUMBAUGH, J.; BOOCH, G.; JACOBSON, I. **The Unified Language Reference Manual**. [S.l]: Addison-Wesley Publishing Company, 1999.

SCHWABE, D., ROSSI, G., **An Object Oriented Approach to WebBased Application Desing.** Theory and Practice of Object Systems 4(4). 1998. Wiley and Sons. New York. Disponível em: <http://www.inf.puc-rio.br/~schwabe/papers> Acesso em: 20/08/2001.

SHAW, M.; GARLAN, D. **Software Architecture – Perspectives on an Emerging Discipline**. New Jersey: Prentice-Hall, 1996.

SILVA, A.; SILVA, M, M. da; DELGADO, J. **Sistemas de Informações para a *Web*: Arquitecturas Aplicacionais**. Disponível em: <http://berlin.inesc.pt/alb/papers/1998/eei98.pdf> Acesso em: 10/11/2002.

SUN MICROSYSTEMS. **Java** . Disponível em: <http://www.java.sun.com> Acesso em: 20/09/2001.

SUN MICROSYSTEMS. **Java Server Page ™.** Disponível em: < http://java.sun.com/products/jsp/ > Acesso em: 22/06/2002(a).

SUN MICROSYSTEMS. **Java Servlet Tecnology.** Disponível em < http://java.sun.com/products/servlet/ > Acesso em: 22/06/2002(b).

TANEMBAUM, A. S. **Redes de Computadores**. Rio de Janeiro: Campus, 1997.

The Common Gateway Interface. Disponível em: <http://hoohoo.ncsa.uiuc.edu/docs/CGI/ overview.html> Acesso em: 12/09/2001.

TIGRE P. B.. **Aspectos Estratégicos da Política Comercial Brasileira: Inserção no Comércio Eletrônico Global**. Estudo elaborado para o IPEA, 2000. Disponível em: <http://www.aladi.org/nsfaladi/ecomerc.nsf/wvestudios/F2E1048BE519A6BD03256ADA00697C94/$File/Paulo+B+Tigre.doc?OpenElement> Acesso em: 10/04/2003.

WARMER, J.; KLEPPE. A. **The Objetc Constraint Language – Precise Modeling with UML**. [S.l.]: Addison Wesley, 1999.

APÊNDICE 1 ESTUDO DE CASO *E-MAGAZINE*

1 OBJETIVO

Desenvolver um SI para *e-commerce* que realize a venda de publicações e periódicos, denominado *e-magazine*. Este sistema representa uma loja virtual cujo único produto comercializado são as publicações. O cliente poderá ter acesso às publicações por meio do serviço de busca, disponível em qualquer local do *site*. O cliente poderá realizar a compra selecionando os itens de interesse, colocando-o em um "carinho" de compras, e em caso de confirmar o pedido, o cliente ainda poderá especificar uma data que lhe seja mais apropriada para a entrega e a forma de pagamento.

2 DEFINIÇÃO DO PROBLEMA

Existem seis personagens no *e-magazine*:

a) o comprador: representa o cliente que acessa a loja para pesquisar preços e produtos e fazer compras;

b) o administrador: responsável pelo gerenciamento do sistema;

c) o vendedor: insere e atualiza as publicações disponíveis no sistema e acompanha o andamento das compras realizadas pelos clientes;

d) o funcionário: responsável pelo controle das faturas emitidas pelo sistema que serão enviadas aos clientes e pelas faturas emitidas pelos fornecedores.

O atributo estado na entidade publicação pode assumir seis estados, conforme segue:

a) Indisponível: a publicação será comercializada pela loja, porém ainda não liberada pela editora;

b) Disponível: a publicação está no estoque, e já pode ser adquirida;

c) Alocado: a publicação estava no estoque e foi selecionada por algum cliente e colocado em seu "carrinho de compras";

d) Reservado: a publicação esta indisponível, pois foi reservada para algum cliente;

e) Vendido: a publicação foi efetivamente adquirida por algum cliente;

f) Excluído: A publicação foi extraviada ou perdida, ou seja, ela não consta mais no estoque e não foi vendida.

Tendo em vista a descrição da loja e os estados possíveis em que cada item pode estar são identificadas as seguintes ações:

a) Informar: o cliente requisita informações sobre um item;

b) Alocar: um item disponível é escolhido pelo cliente para posterior aquisição e é "reservado" para ele;

c) Desalocar: o cliente desiste de comprar um item e o mesmo se torna disponível para venda;

d) Comprar: o cliente efetivamente compra um item;

e) Cancelar: o cliente abandona a reserva de um item, e este volta a ser disponível;

f) Reservar: o cliente seleciona um item indisponível para ser entregue futuramente e este é reservado para ele;

g) Alterar: o administrador altera informações a respeito do item, como seu preço;

h) Excluir: o administrador exclui um item do estoque por ter sido extraviado ou perdido;

i) Vender: um item reservado anteriormente para um cliente é confirmado e vendido pela loja;

j) Disponibilizar: o administrador coloca o item no estoque para venda.

A regra para compra de uma publicação consiste de:

a) a publicação está disponível se a quantidade de itens em estado indisponível para aquela publicação for menor que a quantidade de publicações em estoque;

b) não é permitida a compra de uma publicação quando a quantidade de itens em estado indisponível para aquela publicação for maior ou igual à quantidade da mesma.

A regra de preço é definida conforme segue:

a) o preço para a compra de publicação periódica: o preço para realizar uma compra durante um período de tempo é determinado pelo valor de cada publicação vezes a quantidade de entregas durante o período determinado pelo cliente. Esse preço será somado ao preço do transporte vezes a quantidade de entregas daquele período;

b) o preço para compra única: o preço para compra única é o valor de cada publicação somado com o valor do transporte.

2.1 DESCRIÇÃO DOS SERVIÇOS

Os vários serviços a serem oferecidos pela loja são descritos abaixo.

Serviço Página Inicial

Este serviço fornece informações gerais sobre a loja, sendo o ponto de vista/partida do cliente. Em termos do produto, a única ação que pode ser eventualmente realizada é informar, com o objetivo de disponibilizar informações a respeito de produtos em destaque. Essas informações consistem de atributos como nome, data da edição, conteúdo e são acessados apenas para leitura, por exemplo, informações sobre as principais e mais recentes notícias de uma publicação.

Este serviço é simples (tendo em vista que a ação informar é simples) e possui requisitos de atomicidade e durabilidade simples, uma vez que os atributos acessados são estáticos e replicáveis.

Serviço de Busca

O serviço de busca disponibiliza informações sobre produtos a partir de critérios de busca fornecidos pelos usuários. No caso do *e-magazine*, um produto satisfaz a uma busca quando os seus campos de nome, edição e conteúdo contêm a cadeia de caracteres especificada na consulta.

Serviço de Navegação

O serviço de navegação é muito semelhante ao serviço de busca. A única diferença entre eles é que os critérios de navegação são pré-definidos (os produtos são divididos em categorias) e não determinados pelos clientes como na busca.

Serviço de Colocação no Carrinho

O serviço de colocação no carrinho permite a um cliente registrar sua opção de compra. A única ação executada é alocar, que altera o estado do item de disponível para alocado e tem caráter temporário. Se a compra for realizada os dados serão armazenados na entidade compra juntamente com o código da compra.

Excluindo o serviço página pessoal, para todos os demais, é necessário identificar o cliente para que o produto possa ser comprado por ele. Essa identificação é feita através de um parâmetro, o identificador do cliente, que somente é disponível quando o comprador efetuou seu *username* ou seu cadastramento no sistema.

Serviço de Compra

O serviço de compra permite a efetivação de opções de compra através do serviço de colocação no carrinho. Para cada um dos itens selecionados deve ser aplicada a ação comprar (ou a ação vender no caso do item estar no estado Reservado), mudando o estado desses itens para vendido em caráter definitivo, o que também classifica o serviço como tal.

Serviço de Cadastramento

O serviço de cadastramento é responsável pela inserção de um novo cliente no sistema e permitir que ele realize compras na loja. Este serviço não executa

nenhuma ação que atinja diretamente o ciclo de vida da publicação, desta forma ele pode ser classificado como um serviço simples. O cliente fornece diversas informações pessoais, como nome, endereço e *e-mail*, escolhe um *username* (nome para acesso do sistema) e uma senha, e requisita o cadastro. O sistema então armazena tais dados e gera um número identificador do cliente. O cliente poderá fazer uma compra, somente, se estiver cadastrado no sistema.

Serviço de Validação

O serviço de *username* permite que o cliente possa ser identificado pelo sistema. Através das informações *username* e *password*, o sistema permite que o cliente realize a compra.

3 MODELAGEM BASEADA NO *UML COMPONENT*

Esta seção apresenta a modelagem do estudo de caso segundo o método *UML Components* descrito no capítulo 3, seção 3.3.1 da dissertação.

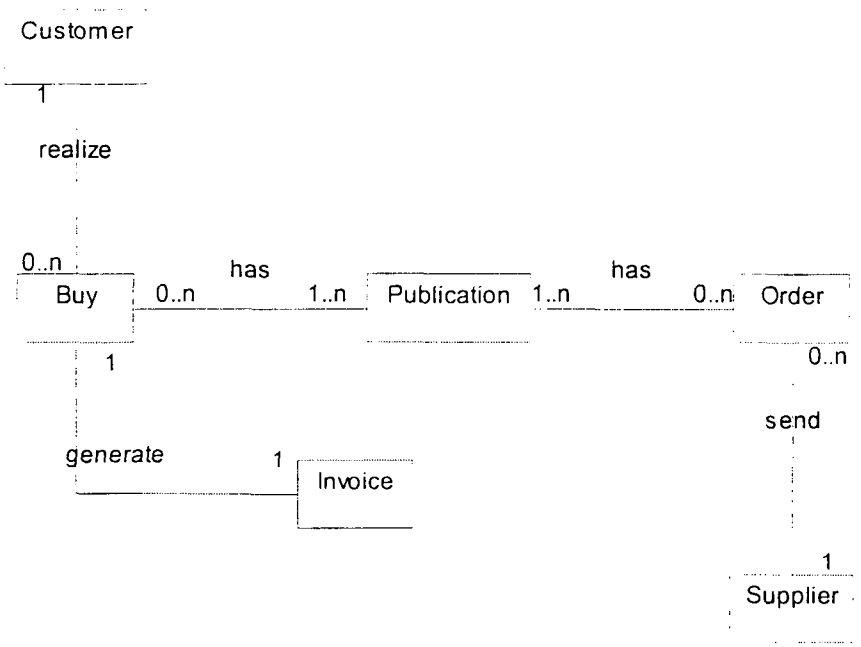
3.1 REQUISITOS

Este passo compreende o levantamento de requisitos onde o negócio é definido para modelagem dos estágios subseqüentes.

3.1.1 Modelo conceitual de negócio

O modelo conceitual do negócio identificou as principais informações que serão tratadas pelo sistema e o seus relacionamentos, apresentado na FIGURA 37.

FIGURA 37 - DIAGRAMA MODELO CONCEITUAL DE NEGÓCIO – *E-MAGAZINE*



3.1.2 Modelo de casos de uso

Este modelo extraiu do negócio os comportamentos que deverão se instanciados pelo sistema e as pessoas que interagirão com ele. Para o *e-magazine* foram identificados os seguintes atores:

a) *administrator*: agente responsável pelo controle e administração do sistema;

b) *customer*: agente que compra os produtos disponíveis no *e-magazine*;

c) *sales agent*: agente responsável pelo registro das publicações no sistema e tem permissão para acessar as informações de uma compra realizada por um cliente e obter o seu status;

d) *payments executive*: agente responsável pelo controle das faturas emitidas para envio aos clientes, e pela faturas emitidas pelos fornecedores.

Avaliando o ambiente do *e-magazine* os seguintes casos de uso foram identificados:

a) *record publication:* responsável pelo controle das publicações, abrangendo a inclusão, alteração e exclusão de publicações;

b) *register customer:* responsável pelo controle dos clientes, abrangendo a inclusão, alteração e exclusão de cliente;

c) *record Internet buy:* responsável pelo controle das compras realizadas pela Internet, abrangendo a inclusão, alteração e cancelamento das compras;

d) *create order:* responsável pela emissão dos pedidos das publicações para os fornecedores responsáveis pelas publicações vendidas;

e) *register supplier:* responsável pelo controle dos fornecedores, abrangendo a inclusão e alteração de fornecedores;

f) *validate user:* responsável pela validação dos usuários para utilização do sistema;

g) *produce invoice:* responsável pelo controle das faturas emitidas pelos fornecedores e pelas faturas emitidas para os clientes, abrangendo a inclusão e alteração das faturas;

h) *browse pending buy*: responsável por disponibilizar as informações sobre uma compra, fornecendo o estado da compra, por exemplo: atendida ou pendente;

i) *manager users*: responsável pelo controle dos usuários, abrangendo a inclusão, alteração e exclusão de usuários. Através deste caso de uso o administrador autoriza os usuários acessar áreas restritas como, por exemplo, ele autoriza o *Sales Agent* a criar os pedido para serem emitidos aos fornecedores;

j) *register offer*: responsável pelo controle das promoções oferecidas pelo *site*, abrangendo a inclusão, alteração e exclusão das promoções de publicações;

k) *browse publication*: responsável pela busca por um produto que seja de interesse do cliente e esteja disponível na loja;

l) *register interest publication*: responsável pelo cadastro do interesse que um cliente tem em relação a uma publicação, que não esta disponível no momento. Através deste recurso o sistema informa por *e-mail*, o cliente, a chegada da publicação em questão;

FIGURA 38 - DIAGRAMA DE CASO DE USO – *E-MAGAZINE*

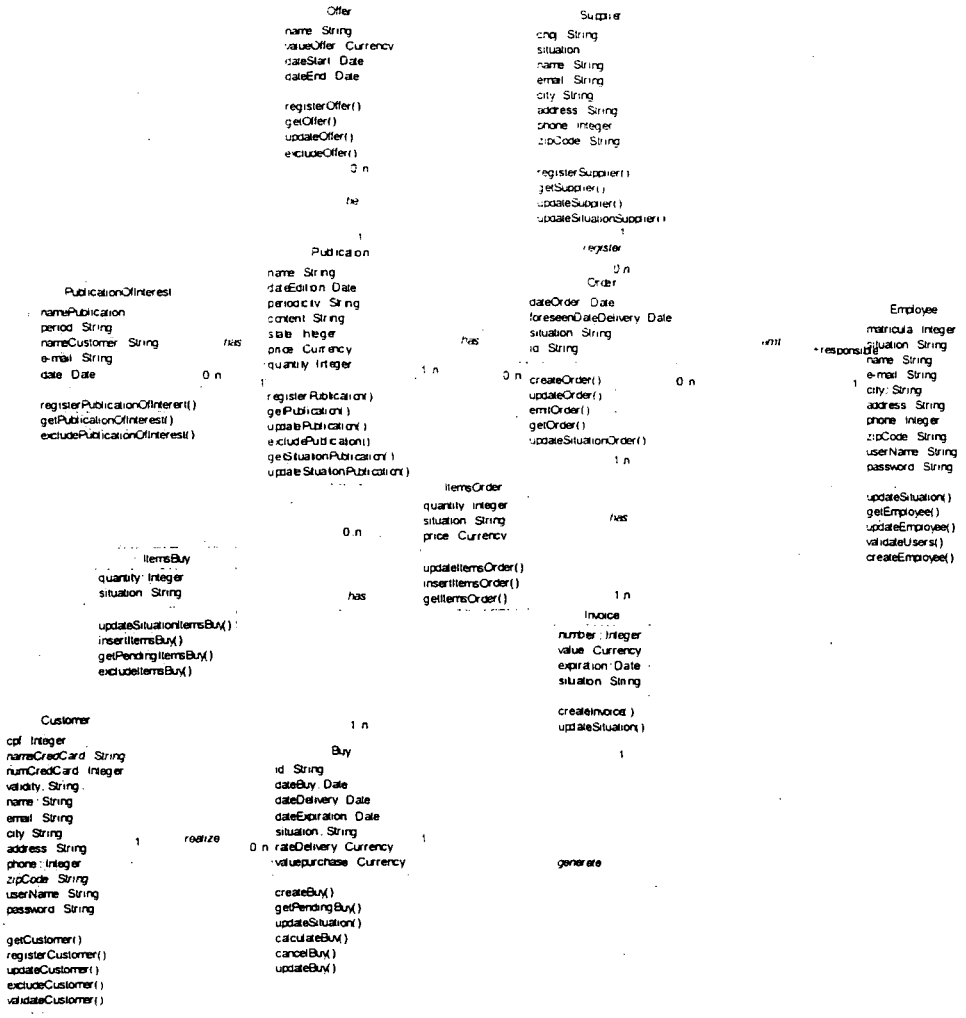
3.2 IDENTIFICAÇÃO DE COMPONENTES

A partir dos modelos definidos no levantamento de requisitos, neste estágio será definido um grupo inicial de interfaces de negócio e de interfaces do sistema.

3.2.3 Modelo de tipo de negócio

Esse modelo teve como base o modelo de negócio do passo anterior, foram feitos refinamentos e acrescentados informações para definição do negócio em relação às informações de entradas, as informações processadas e as informações de saída. Como mostra o diagrama da FIGURA 39.

FIGURA 39 - MODELO DE TIPO DE NEGÓCIO COM DECISÕES DE PROJETO – *E-MAGAZINE*



3.2.4 Especificação de interface

O processo de especificação de interface baseia-se no modelo de casos de uso, a princípios todos os casos de uso se transformam em interfaces do sistema. Uma outra análise foi feita no modelo de tipo de negócio, foram verificadas quais informações poderiam ser geradas independente do negócio, assim foram definas as interfaces de negócio. Abaixo estão descritas as interfaces do sistema e depois é feito um mapeamento das interfaces do negócio para as interfaces do sistema. Pois foi verificado após a definição dos dois tipos de interfaces que as interfaces do negócio já existiam nas interfaces do sistema, por isso também, não foi feita a

descrição dessas interfaces, elas possuem as mesmas características das interfaces correspondentes.

Interface *IRecordPublication:* oferece os seguintes serviços: cadastramento das publicações no sistema; atualização das publicações; busca das publicações registradas; e, exclusão de uma publicação registrada.

Interface *ICreateCustomer:* oferece os seguintes serviços: cadastramento de clientes no sistema; atualização dos dados dos clientes; busca de informações dos clientes cadastrados; e, exclusão do cliente do sistema.

Interface *IRecordInternetBuy:* oferece os seguintes serviços: criação do registro de um compra inicial por um cliente (identificado ou não); atualização da compra (quando o cliente escolhe outra publicação); atualização de uma compra pendente no sistema; cancelamento da compra; exclusão de um item selecionado na compra.

Interface *ICreateOrder:* oferece os seguintes serviços: criação de um pedido de compra a um determinado fornecedor conforme quantidade de produtos em estoque; atualização de um pedido; emissão do pedido para o fornecedor.

Interface *ICreateSupplier:* oferece os seguintes serviços: cadastramento de fornecedores no sistema; atualização dos dados dos fornecedores; busca de informações dos fornecedores cadastrados.

Interface *IProduceInvoice:* oferece os seguintes serviços: criação das faturas dos pedidos feitos pelos clientes e das faturas correspondentes aos pagamentos de fornecedores; emissão das faturas; e, atualização das faturas.

Interface *IBrowsePendingBuy:* oferece o serviço de consulta da situação de um compra, (se já foi encaminhado e a data ou se ainda está pendente no sistema e o motivo).

Interface *IUserManager:* oferece os seguintes serviços: cadastramento de usuário para utilização do sistema que necessitam de acesso restrito; atualização dos dados dos usuários; busca de informações dos usuários cadastrados; e, validação dos usuário para uso do sistema.

Interface *IRecordPromotion:* oferece os seguintes serviços: cadastramento das promoções de venda no sistema; atualização das promoções; busca das promoções registradas; e, exclusão de uma promoção registrada.

Interface *IRecordPublicationOfInterest:* oferece os seguintes serviços: cadastramento do interesse de um cliente em adquirir uma publicação não disponível; atualização do cadastro; exclusão do cadastro; e, busca das informações de um cadastro de interesse de publicação

Interface *IBrowsePublication:* oferece o serviço de consulta as publicações disponíveis no sistema para compra.

Mapeamento entre as interfaces de sistema e de negócio

Interface de Sistema	**Interface de Negócio**
ICreateCustomer	ICustomerMgt
ICreateSupplier	ISupplierMgt
IRecordPublication	IPublicationMgt
IUserManager	IEmployeeMgt

O diagrama de responsabilidade de interface de negócio, apresentado na FIGURA 40, mostra o relacionamento entre as interfaces de negócio e as classes.

FIGURA 40 - DIAGRAMA DE RESPONSABILIDADE DE INTERFACE DE NEGÓCIO – *E-MAGAZINE*

3.3 INTERAÇÃO DO COMPONENTE

Neste estágio é definido a interação entre os componentes. Os diagrama de colaboração gerados nesse estágio estão direcionados as interfaces do sistema e as interfaces do negócio. As figuras que segue apresentam os diagramas de interação para cada método da interface ***IPublicationMgt***, os demais diagramas não foram apresentados devido ao grande volume de informações.

FIGURA 41 - DIAGRAMA DE INTERAÇÃO getRecordPublication() – *E-MAGAZINE*

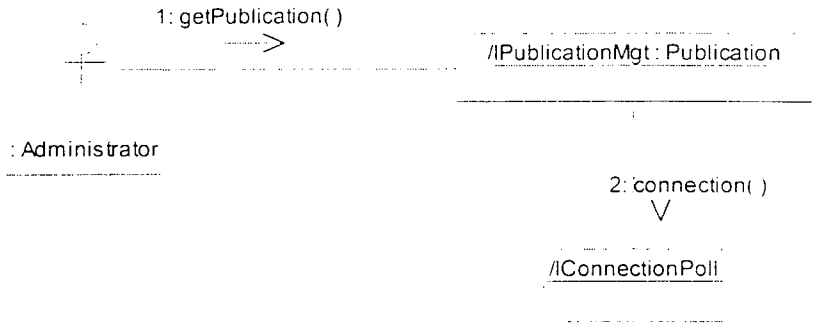


FIGURA 42 - DIAGRAMA DE INTERAÇÃO registerRecordPublication()– *E-MAGAZINE*

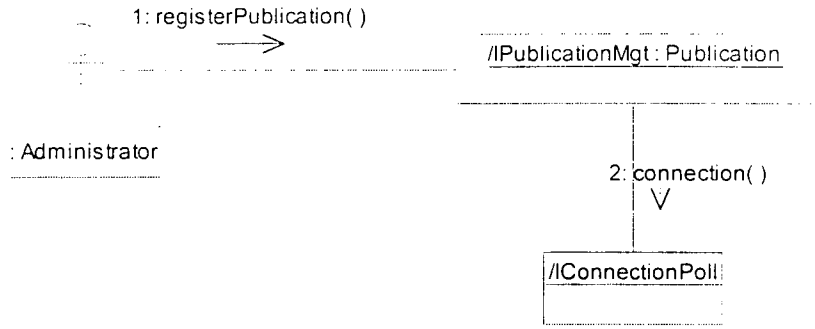


FIGURA 43 - DIAGRAMA DE INTERAÇÃO updateRecordPublication()– *E-MAGAZINE*

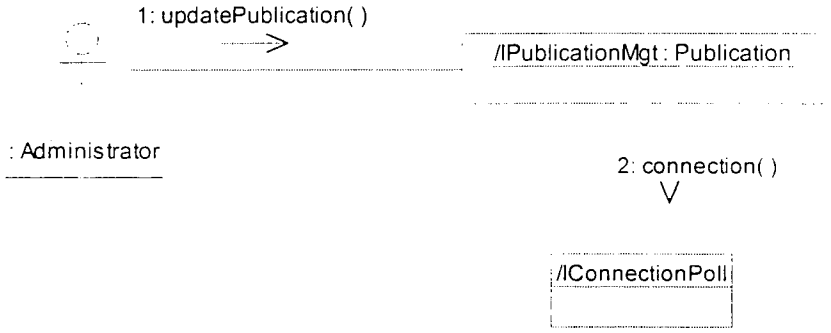


FIGURA 44 - DIAGRAMA DE INTERAÇÃO excludePublication()– *E-MAGAZINE*

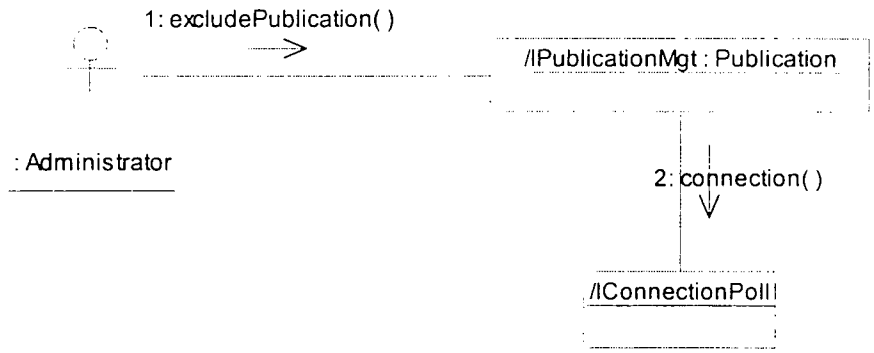


FIGURA 45 - DIAGRAMA DE INTERAÇÃO getSituationPublication()– *E-MAGAZINE*

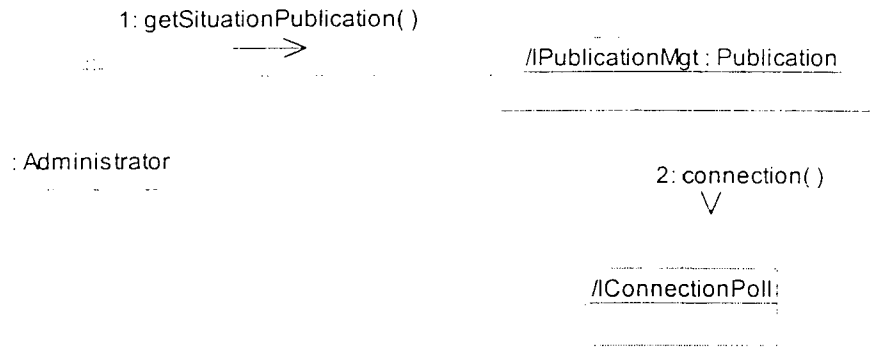
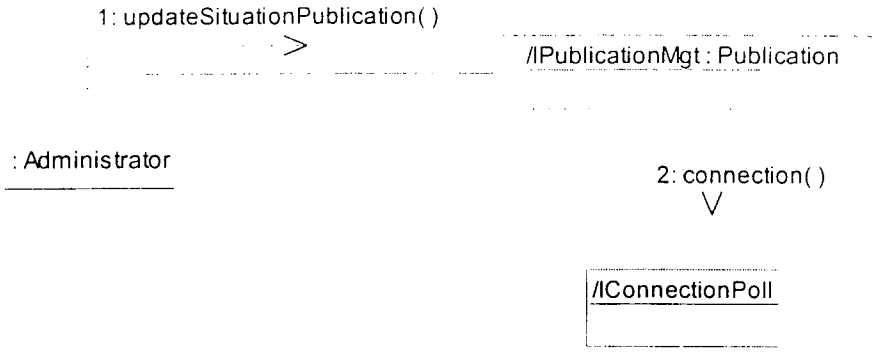


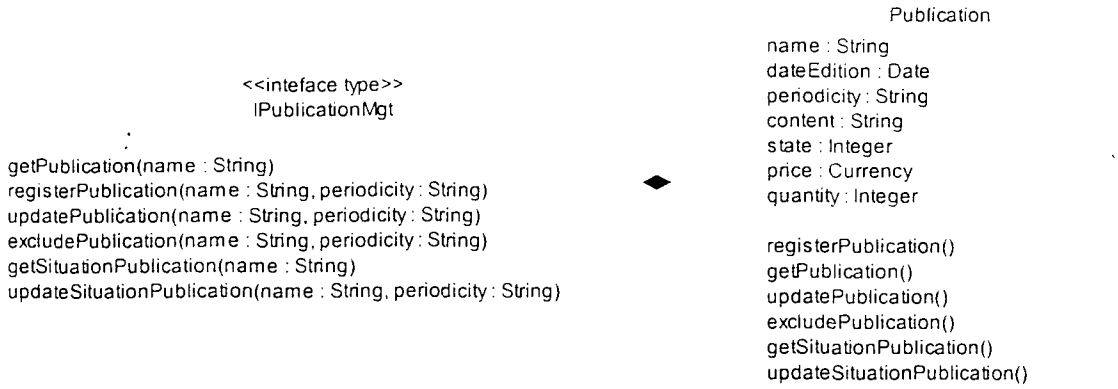
FIGURA 46 - DIAGRAMA DE INTERAÇÃO updateSituationPublication()– *E-MAGAZINE*



3.4 ESPECIFICAÇÃO DO COMPONENTE

Neste estágio foi realizado o detalhamento das operações e das restrições das interfaces gerando o diagrama de especificação de interface e definindo as pré- e pós-condições dos métodos. O diagrama de especificação de interface é apresentado a seguir para cada interface definida no estágio anterior, esse diagrama não representa os componentes do sistema, ele representa a interação entre a interface e a classe a qual ela se comunica e mostra ainda as demais classes necessárias para a instanciação de seus métodos, mesmo que estas classes façam parte de outros componentes. Posteriormente a comunicação será realizada entre os componentes. A definição das pré- e pós-condições foi apresentado para a interface *IPublicationMgt,* servindo como modelo para mostrar como este trabalho foi realizado, as demais não estão listadas devido ao grande volume de informações que seriam acrescentada a este trabalho.

Interface: *IPublicationMgt*



context IPublicationMgt : :getRecordPublication ()

pre:

publication = **exists** (publication | name = name)

post:

let thePublication – publication **-> select** (publication | publication.name = name)

result.name = thePublication.name **and**

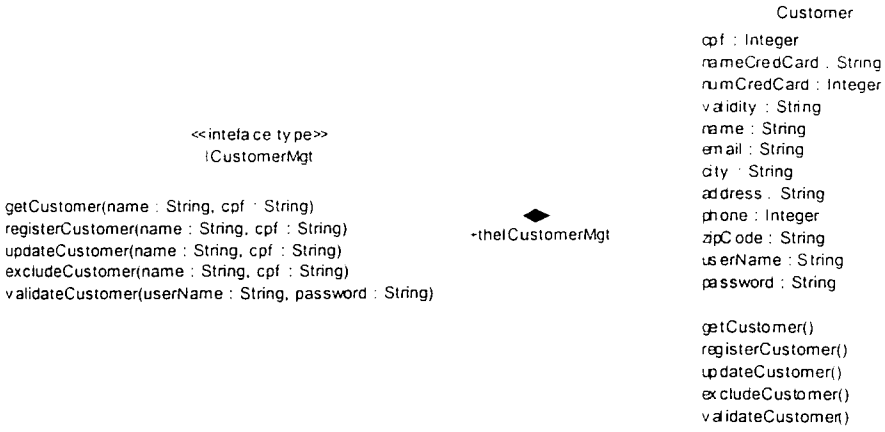
result.dateEdition = thePublication.dateEdition **and**

result.periodicity = thePublication. periodicity **and**

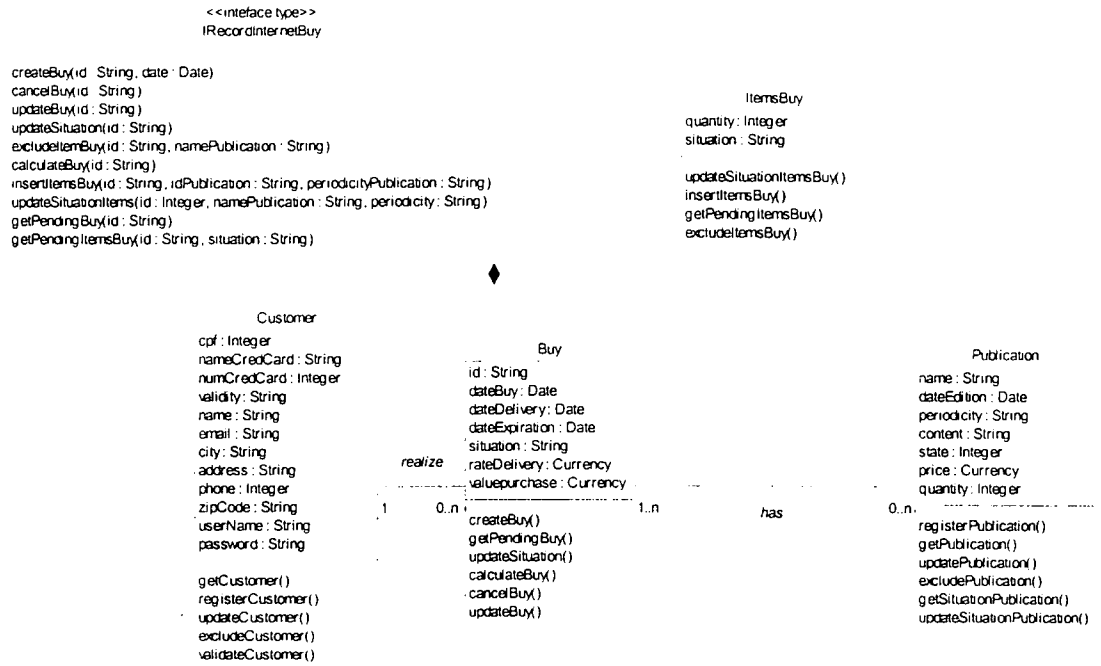
result.content = thePublication.content **and**

result.state = thePublication.state

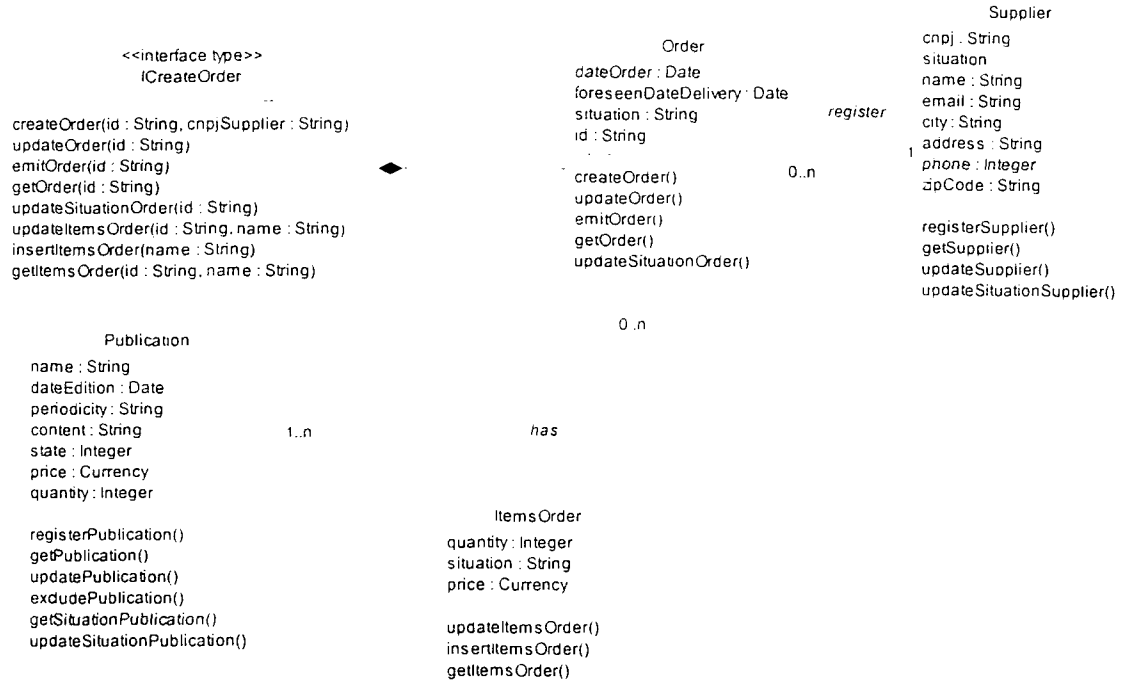
Interface: *ICustomerMgt*



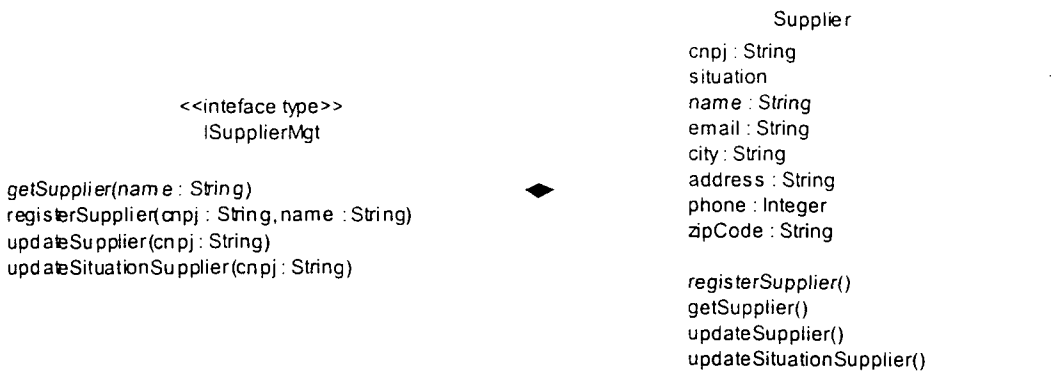
Interface: *IRecordInternetBuy*



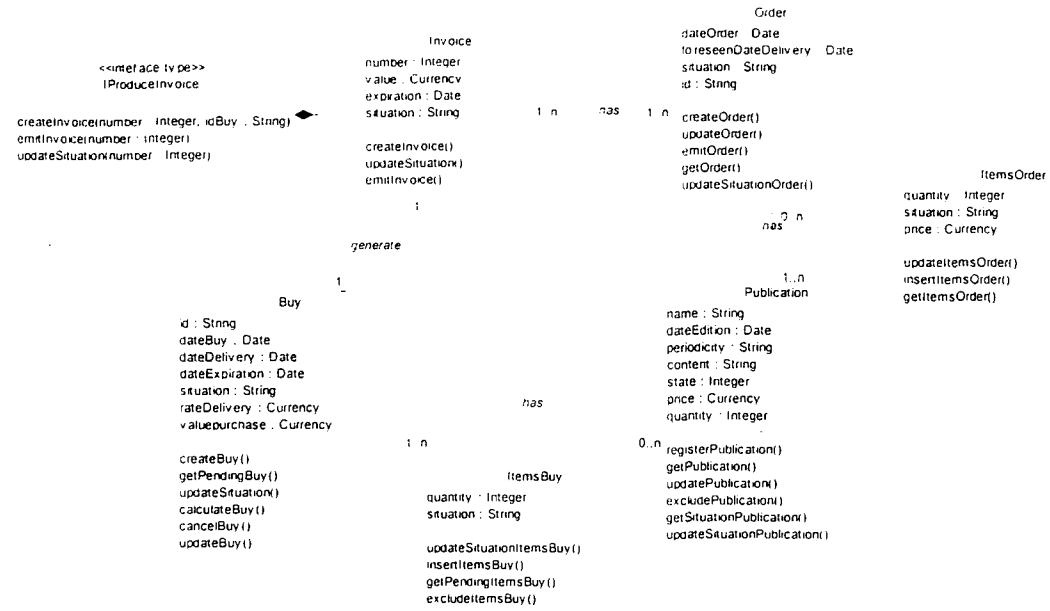
Interface: *ICreateOrder*



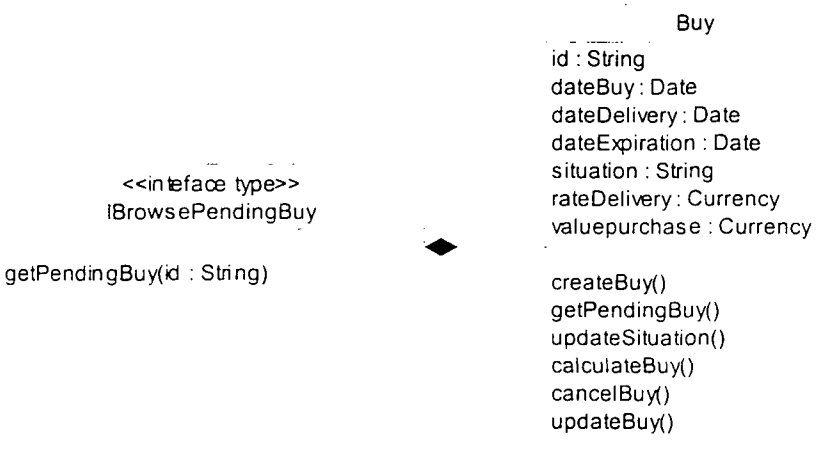
Interface: *ISupplier*



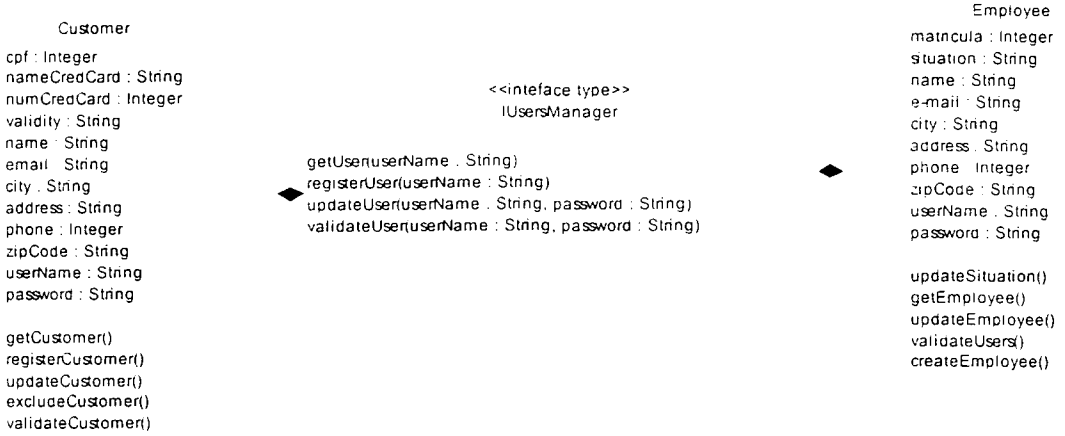
Interface: *IProduceInvoice*



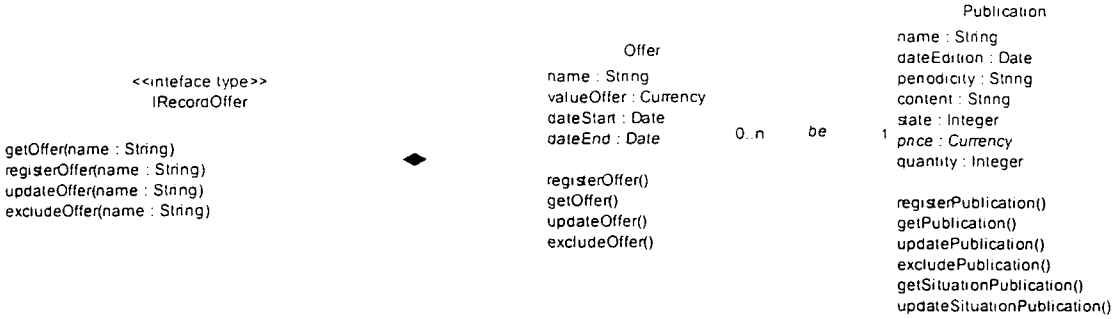
Interface: *IBrowsePendingBuy*



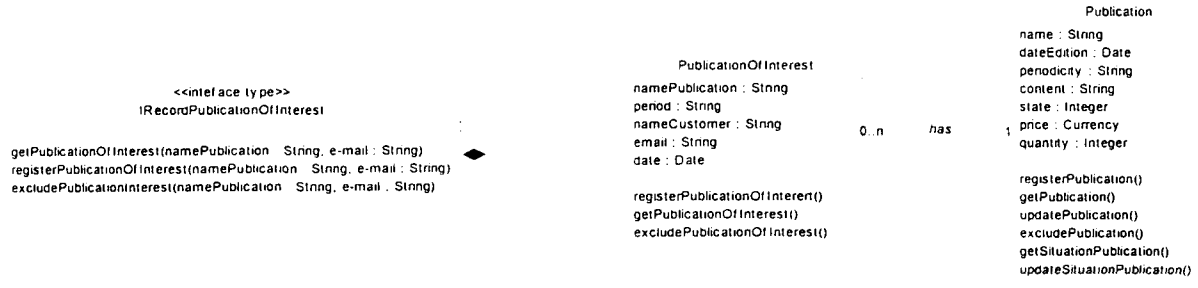
Interface: *IUserManager*



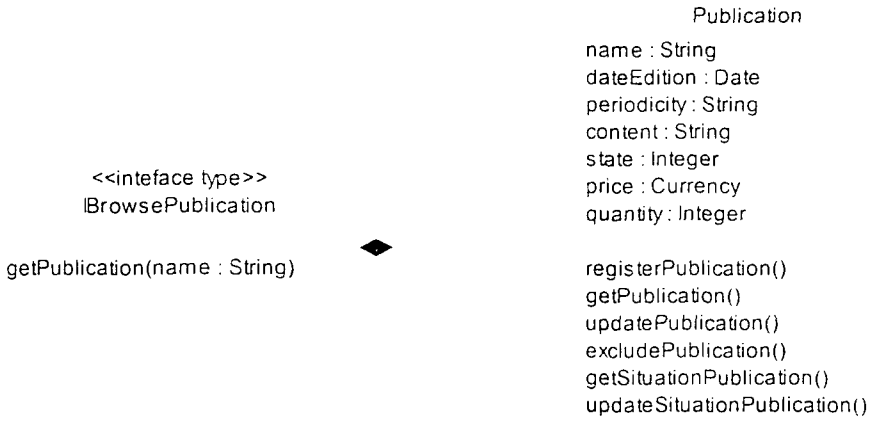
Interface: *IRecordOffer*



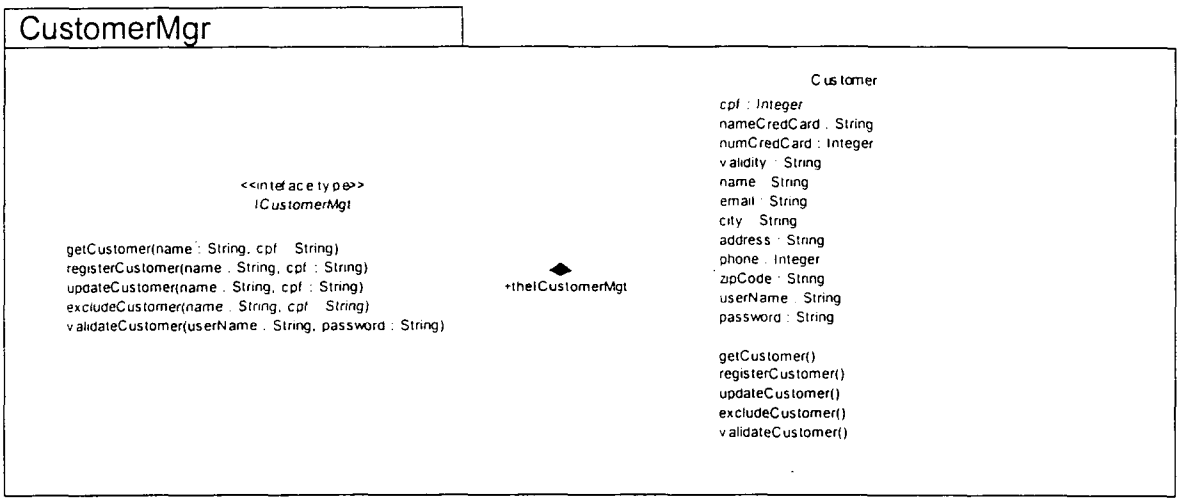
Interface: *IRecordPublicationOfInterest*

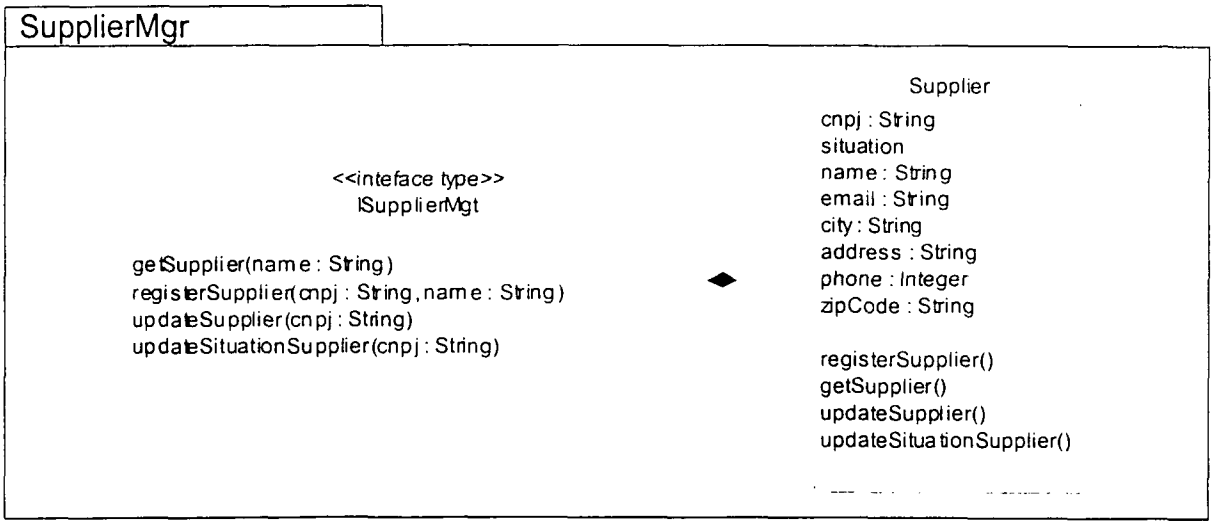


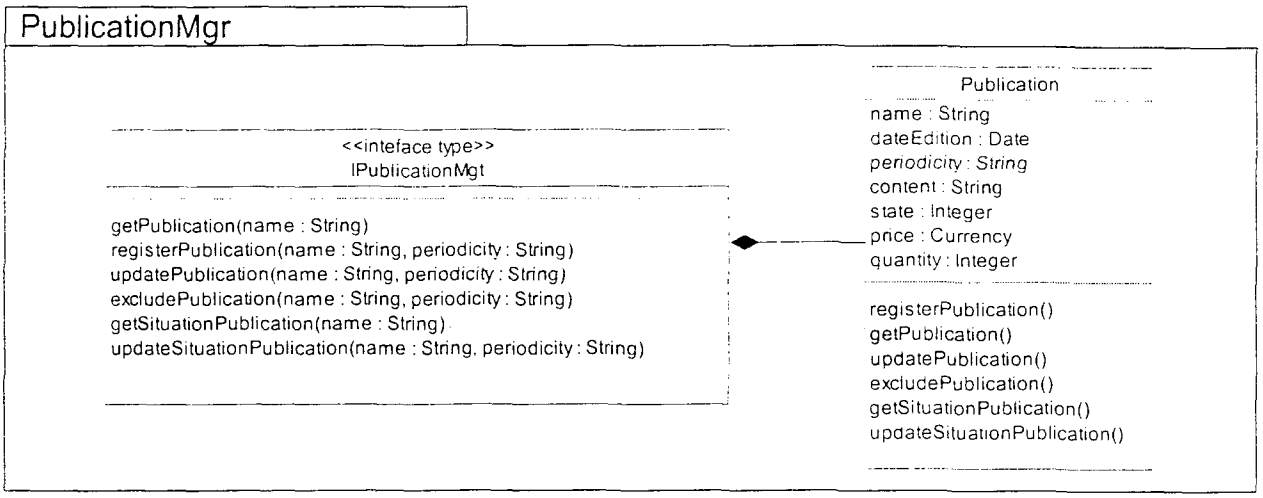
Interface: *IBrowsePublication*

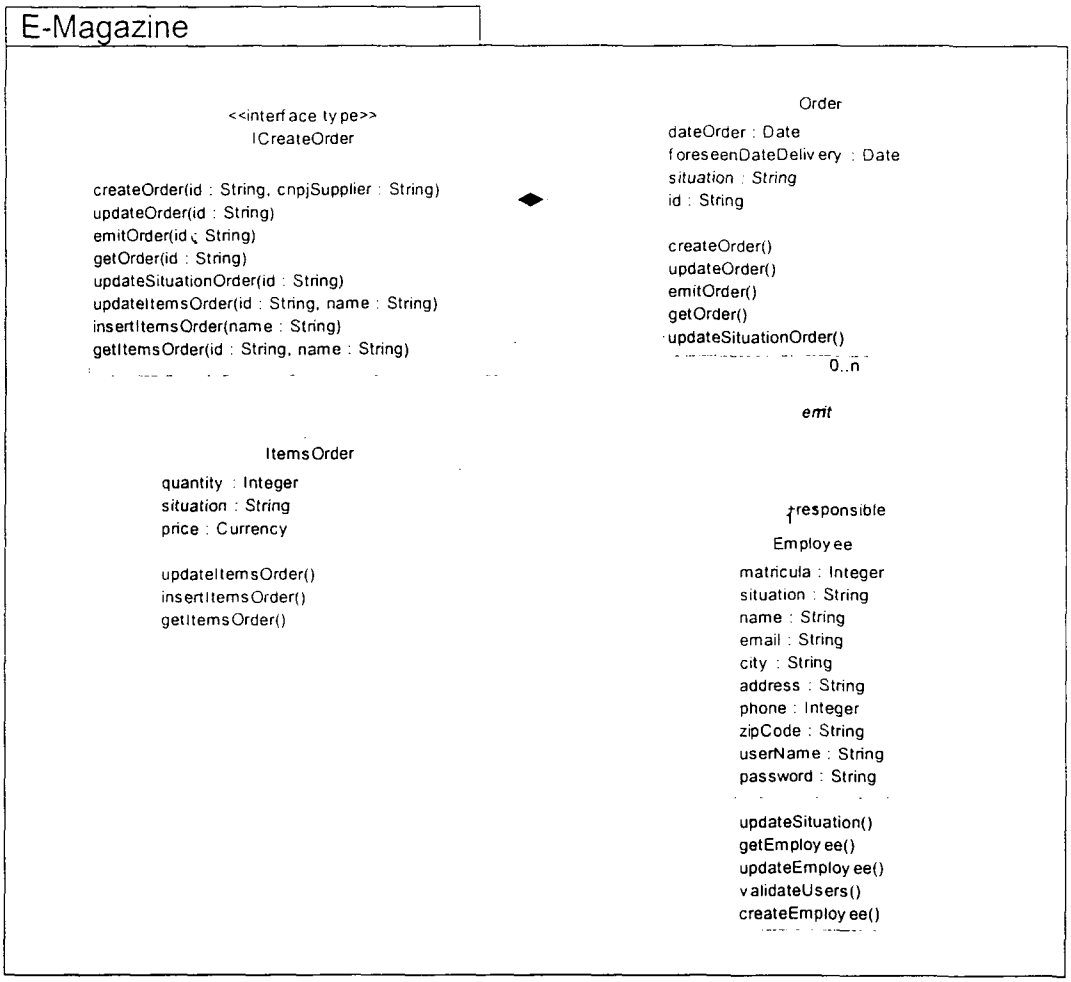


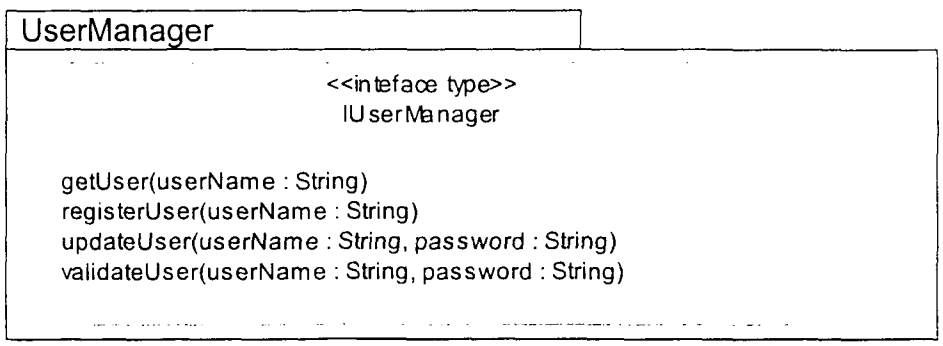
O segundo diagrama gerado neste estágio é o diagrama de especificação de componente, ele representa quais classes e interfaces farão parte do componente. Segue abaixo a representação dos componentes necessários para definição do SI.

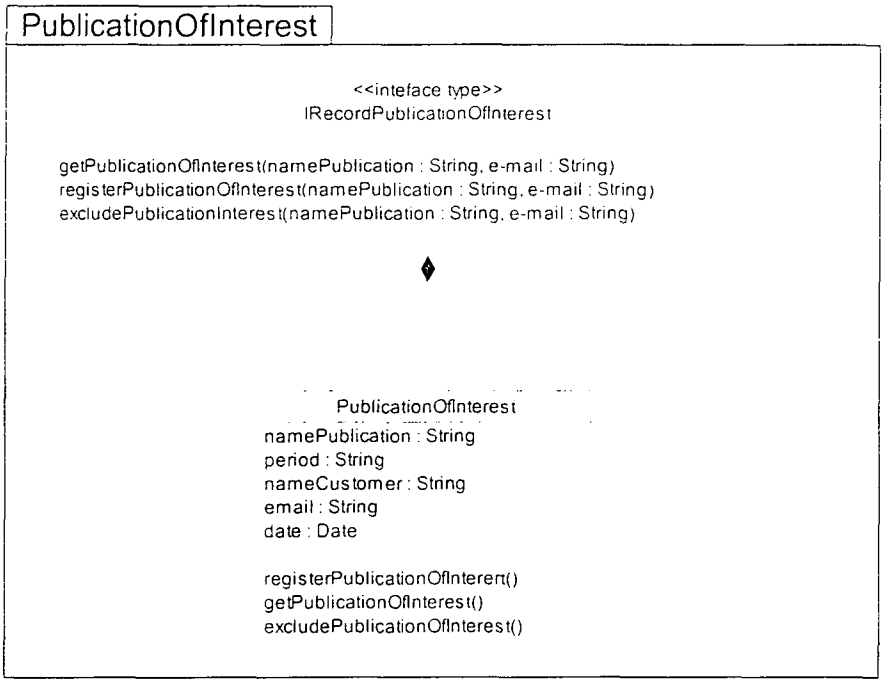


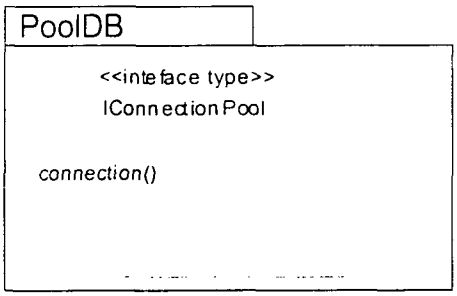


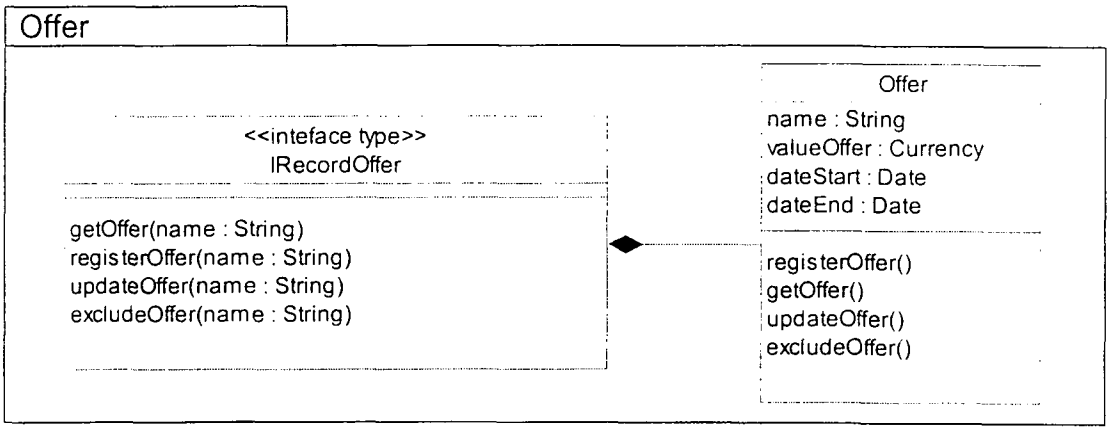

PublicationMgr
<<inteface type>>
IPublicationMgt
getPublication(name : String)
registerPublication(name : String, periodicity : String)
updatePublication(name : String, periodicity : String)
excludePublication(name : String, periodicity : String)
getSituationPublication(name : String)
updateSituationPublication(name : String, periodicity : String)
Publication
name : String
dateEdition : Date
periodicity : String
content : String
state : Integer
price : Currency
quantity : Integer
registerPublication()
getPublication()
updatePublication()
excludePublication()
getSituationPublication()
updateSituationPublication()

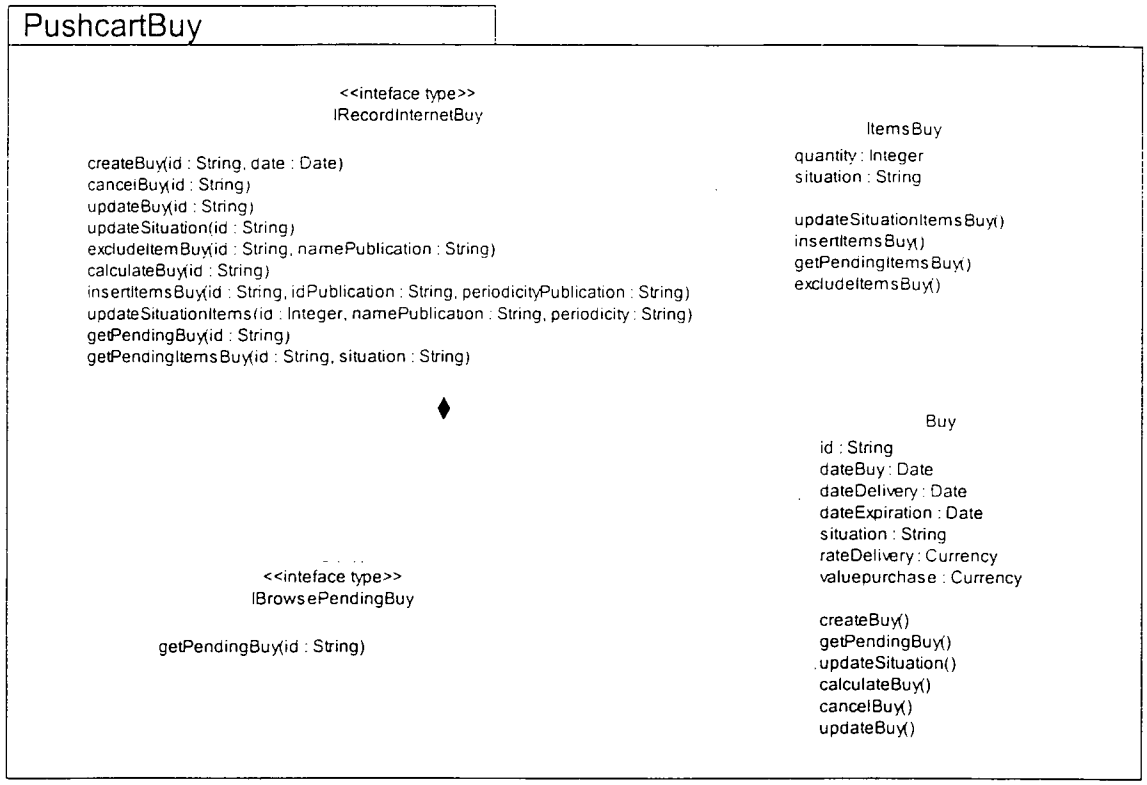

E-Magazine
<<interface type>>
ICreateOrder
createOrder(id : String, cnpjSupplier : String)
updateOrder(id : String)
emitOrder(id : String)
getOrder(id : String)
updateSituationOrder(id : String)
updateItemsOrder(id : String, name : String)
insertItemsOrder(name : String)
getItemsOrder(id : String, name : String)
Order
dateOrder : Date
foreseenDateDelivery : Date
situation : String
id : String
createOrder()
updateOrder()
emitOrder()
getOrder()
updateSituationOrder()
0..n
emit
ItemsOrder
quantity : Integer
situation : String
price : Currency
updateItemsOrder()
insertItemsOrder()
getItemsOrder()
responsible
Employee
matricula : Integer
situation : String
name : String
email : String
city : String
address : String
phone : Integer
zipCode : String
userName : String
password : String
updateSituation()
getEmployee()
updateEmployee()
validateUsers()
createEmployee()

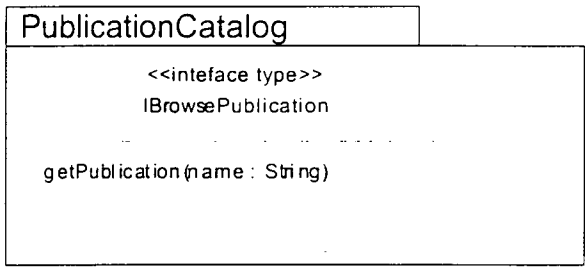

UserManager
<<inteface type>>
IUserManager
getUser(userName : String)
registerUser(userName : String)
updateUser(userName : String, password : String)
validateUser(userName : String, password : String)

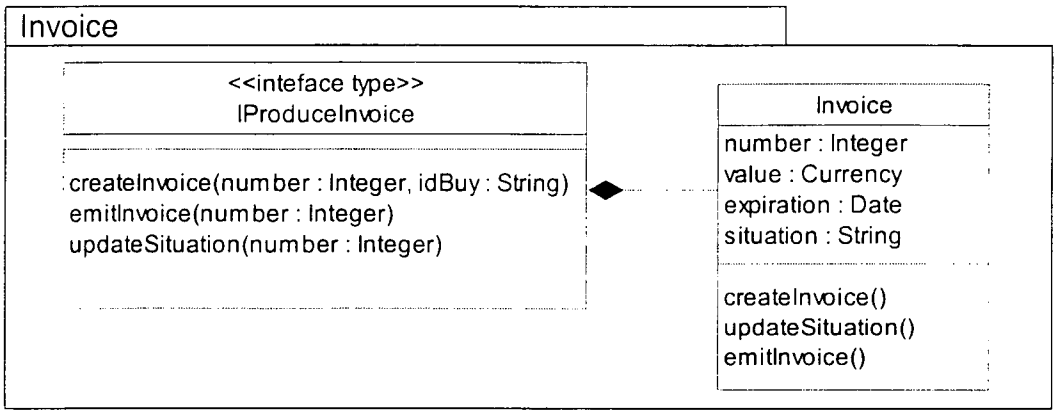

PublicationOfInterest
<<inteface type>>
IRecordPublicationOfInterest
getPublicationOfInterest(namePublication : String, e-mail : String)
registerPublicationOfInterest(namePublication : String, e-mail : String)
excludePublicationInteres(namePublication : String, e-mail : String)
PublicationOfInterest
namePublication : String
period : String
nameCustomer : String
email : String
date : Date
registerPublicationOfInterert()
getPublicationOfInterest()
excludePublicationOfInterest()


PoolDB
<<inteface type>>
IConnectionPool
connection()


Offer
<<inteface type>>
IRecordOffer
getOffer(name : String)
registerOffer(name : String)
updateOffer(name : String)
excludeOffer(name : String)
Offer
name : String
valueOffer : Currency
dateStart : Date
dateEnd : Date
registerOffer()
getOffer()
updateOffer()
excludeOffer()


PushcartBuy
<<inteface type>>
IRecordInternetBuy
createBuy(id : String, date : Date)
cancelBuy(id : String)
updateBuy(id : String)
updateSituation(id : String)
excludeItemBuy(id : String, namePublication : String)
calculateBuy(id : String)
insertItemsBuy(id : String, idPublication : String, periodicityPublication : String)
updateSituationItems(id : Integer, namePublication : String, periodicity : String)
getPendingBuy(id : String)
getPendingItemsBuy(id : String, situation : String)
ItemsBuy
quantity : Integer
situation : String
updateSituationItemsBuy()
insertItemsBuy()
getPendingItemsBuy()
excludeItemsBuy()
Buy
id : String
dateBuy : Date
dateDelivery : Date
dateExpiration : Date
situation : String
rateDelivery : Currency
valuepurchase : Currency
createBuy()
getPendingBuy()
updateSituation()
calculateBuy()
cancelBuy()
updateBuy()
<<inteface type>>
IBrowsePendingBuy
getPendingBuy(id : String)

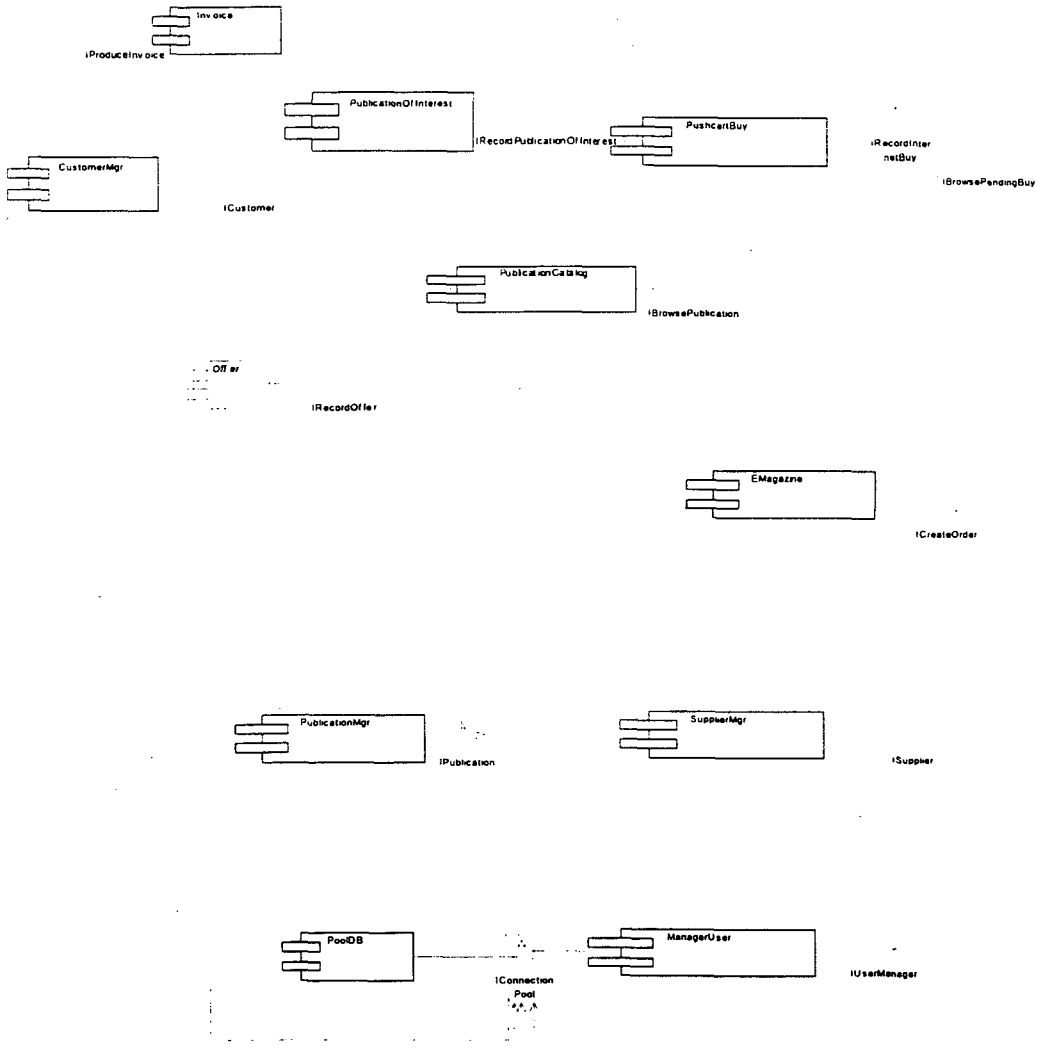

PublicationCatalog
<<inteface type>>
IBrowsePublication
getPublication(name : String)


Invoice
<<inteface type>>
IProduceInvoice
createInvoice(number : Integer, idBuy : String)
emitInvoice(number : Integer)
updateSituation(number : Integer)
Invoice
number : Integer
value : Currency
expiration : Date
situation : String
createInvoice()
updateSituation()
emitInvoice()

3.5 ARQUITETURA DE COMPONENTE

A FIGURA 47 apresenta a arquitetura do *e-magazine*, que serviu de base para a generalização a arquitetura *WbIS*.

FIGURA 47 - ARQUITETURA DO *E-MAGAZINE*



APÊNDICE 2 ESTUDO DE CASO *E-RENT-A-CAR*

1 OBJETIVO

Desenvolver um SI para *e-commerce* que realize a locação de carros, denominado *e-rent-a-car.* Este sistema representa uma loja virtual cujo único produto comercializado são os carros. O cliente poderá ter acesso aos automóveis por meio do serviço de busca, disponível em qualquer local do *site*. O cliente poderá realizar uma locação selecionando os itens de interesse, colocando-o em um "carrinho" de reserva, e em caso de confirmar a reserva, o cliente ainda poderá especificar a data, atual ou não, que ele precisará do carro locado e a forma de pagamento.

2 DEFINIÇÃO DO PROBLEMA

Existem dois personagens no *e-rent-a-car:*

a) o cliente: aluga e devolve o carro, e liquida a dívida, quando for o caso;

b) o gerente: para o qual os relatórios e consultas são enviados. Determina também as operações de cadastros e gerencia o sistema.

O atributo estado na entidade carro pode assumir cinco estados, conforme segue:

a) indisponível: o carro será comercializado pela loja, porém não está liberado por motivos de manutenção;

b) disponível: o carro está regular, e pode ser alugado;

c) alocado: o carro estava disponível e foi selecionada por algum cliente e colocado em sua "carrinho de reservas";

d) reservado: O carro esta indisponível, pois foi reservado para algum cliente;

e) locado: o carro foi efetivamente alugado por algum cliente.

Tendo em vista a descrição da loja e os estados possíveis em que cada item pode estar são identificadas as seguintes ações:

a) informar: o cliente requisita informações sobre um item;

b) alocar: um item disponível é escolhido pelo cliente para posterior locação e "reservado" para ele;

c) desalocar: o cliente desiste de alugar um carro e o mesmo se torna disponível para locação;

d) locar: o cliente efetivamente loca um carro;

e) cancelar: o cliente abandona a reserva de um carro, e este volta a ser disponível;

f) reservar: o cliente seleciona um carro indisponível para ser locado futuramente e este é reservado para ele, desde que no período solicitado o carro não esteja reservado para outro cliente;

g) alterar: o administrador altera informações a respeito do carro, como sua quilometragem;

h) excluir: o administrador exclui um carro do estoque por ter sido vendido, roubado ou batido;

i) disponibilizar: o administrador coloca o carro no estoque para locação.

As regras para locação de um carro consistem de:

a) o carro está disponível, se ele não estiver em manutenção ou locado;

b) não é permitida a locação de um carro quando a data informada para a locação coincidir com a data de uma locação já reservada.

As regras de preço são definidas como segue:

a) o carro será cadastrado com um valor por km rodado;

b) ainda poderá ser inserido no preço da locação o valor de multas efetuadas no período;

c) dependendo da forma de pagamento poderá haver descontos no valor final da locação.

2.1 DESCRIÇÃO DOS SERVIÇOS

Os vários serviços a serem oferecidos pela loja são descritos abaixo.

Serviço Página Inicial

Este serviço fornece informações gerais sobre a loja, sendo o ponto de vista/partida do cliente. Em termos do produto, a única ação que pode ser eventualmente realizada é informar, com o objetivo de disponibilizar informações a respeito de carros em destaque. Essas informações consistem de atributos como nome, marca, ano, cor e são acessados apenas para leitura.

Este serviço é simples (tendo em vista que a ação informar é simples) e possui requisitos de atomicidade e durabilidade simples, uma vez que os atributos acessados são estáticos e replicáveis. Em termos de requisitos de processamento, este serviço demanda, além da leitura de arquivos para construção da página, consultas atributo-valor.

Serviço de Busca

O serviço de busca disponibiliza informações sobre carros a partir de critérios de busca fornecidos pelos usuários. No caso do *e-rent-a-car*, um produto satisfaz a uma busca quando os seus campos nome, marca e ano contêm a cadeia de caracteres especificada na consulta.

Serviço de Navegação

O serviço de navegação é muito semelhante ao serviço de busca. A única diferença entre eles é que os critérios de navegação são pré-definidos (os carros são divididos em categorias) e não determinados pelos clientes como na busca.

Serviço de Colocação no Carrinho

O serviço de colocação no carrinho permite a um cliente registrar sua opção de locação. A única ação executada é alocar, que altera o estado do carro de disponível para alocado e tem caráter temporário. Se a locação for realizada os dados serão armazenados na entidade locação juntamente com o código da locação.

Excluindo o serviço página pessoal, para todos os demais, é necessário identificar o cliente para que o carro possa ser locado por ele. Essa identificação é feita através de um parâmetro, o identificador do cliente, que somente é disponível quando o cliente efetuou seu *username* ou seu cadastramento no sistema.

Serviço de Locação

O serviço de locação permite a efetivação de opções de reserva através do serviço de colocação no carrinho. Para cada um dos itens selecionados deve ser aplicada a ação locar, mudando o estado desses itens para locado em caráter definitivo, o que também classifica o serviço como tal.

Serviço de Cadastramento

O serviço de cadastramento é responsável pela inserção de um novo cliente no sistema e permitir que ele realize locações na loja. Esse serviço não executa nenhuma ação que atinja diretamente o ciclo de vida do carro, desta forma ele pode ser classificado como um serviço simples. O cliente fornece diversas informações pessoais, como nome, endereço e *e-mail*, escolhe um *username* (nome para acesso do sistema) e uma senha, e requisita o cadastro. O sistema então armazena tais dados e gera um número identificador do cliente. O cliente poderá fazer uma locação, somente, se estiver cadastrado no sistema.

Serviço de Validação

O serviço de *username* permite que o cliente possa ser identificado pelo sistema. Através das informações *username* e *password*, o sistema permite que o cliente realize a locação.

3 PROCESSO DE DESENVOLVIMENTO DO ESTUDO DE CASO

Após a definição e delimitação do ambiente do sistema foi feita uma avaliação na arquitetura *WbIS* com o intuito de reutilizar os componentes para o desenvolvimento deste estudo de caso. A próxima seção apresenta a modelagem do estudo de caso, nos diagramas apresentados. A arquitetura é identificada pelos artefatos sem hachurado, os artefatos específicos, necessários para resolução do problema são apresentados com hachurado.

Primeiramente, avaliou-se o diagrama de casos de usos para identificação dos processos semelhantes. Em seguida foram identificadas as classes necessárias para o estudo de caso e comparadas com as classes que compõem a arquitetura. As interfaces do sistema e as interfaces do negócio também foram avaliadas. Tendo cumprido todos estes estágios do *UML Components*, foi possível aplicar a arquitetura e identificar os demais componentes específicos da aplicação, como apresentado na próxima seção.

4 MODELAGEM BASEADO NO *UML COMPONENT*

Esta seção apresenta a modelagem do estudo de caso segundo o método *UML Components* descrito no capítulo 3, seção 3.3.1 da dissertação.

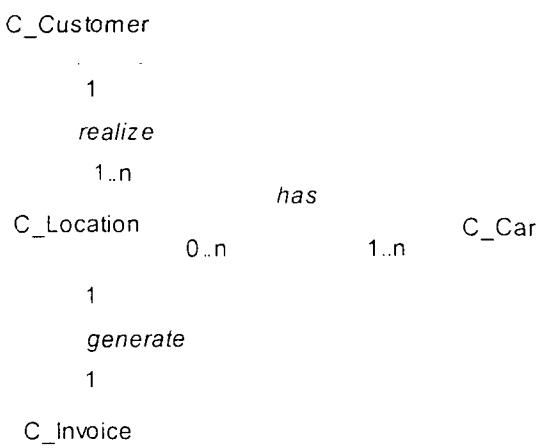
4.1 REQUISITOS

Este passo compreende o levantamento de requisitos onde o negócio é definido para modelagem dos estágios subseqüentes.

4.1.5 Modelo conceitual de negócio

O modelo conceitual do negócio identificou as principais informações que serão tratadas pelo sistema e o seus relacionamentos, apresentado na FIGURA 48.

FIGURA 48 - DIAGRAMA MODELO CONCEITUAL DE NEGÓCIO – *E-RENT-A-CAR*



4.1.6 Modelo de casos de uso

Para a criação do diagrama de caso de uso do *e-rent-a-car* foram identificados os seguintes atores:

a) *administrator*: Agente para o qual os relatórios e consultas são enviados. Determina também as operações de cadastros;

b) *customer*: Agente que aluga e devolve o carro, e liquida a dívida, quando for o caso.

E os seguintes casos de uso:

a) *Record Car:* responsável pelo controle dos carros, abrangendo a inclusão, alteração e exclusão de carros;

b) *Register Customer:* responsável pelo controle dos clientes, abrangendo a inclusão, alteração e exclusão de cliente;

c) *Record Internet Rent:* responsável pelo controle das locações realizadas pela *Internet*, abrangendo a inclusão, alteração e exclusão das locações;

d) *Record Return:* responsável pelo controle das devoluções dos carros locados pela *Internet*;

e) *Validate User:* responsável pela validação dos usuários para utilização do sistema;

f) *Produce Invoice:* responsável pelo controle das faturas emitidas para os clientes, abrangendo a inclusão e alteração das faturas;

g) *Browse Pending Rent*: responsável por disponibilizar as informações sobre uma locação, fornecendo o estado da locação, por exemplo: atendida ou pendente;

h) *Manager Users*: responsável pelo controle dos usuários, abrangendo a inclusão, alteração e exclusão de usuários;

i) *Register Offer*: responsável pelo controle das promoções oferecidas pelo *site*, abrangendo a inclusão, alteração e exclusão das promoções de carros para locação;

j) *Browse Car*: responsável pela busca das informações sobre algum carro disponível na loja, sendo realizada pelo cliente;

k) *Register Interest Car*: responsável pelo cadastro do interesse que um cliente tem em relação a um carro, que não esta disponível no momento. Através deste recurso o sistema informa por *e-mail*, o cliente, a disponibilização do carro em questão;

l) *Emit Report*: responsável pela emissão dos relatórios disponíveis no sistema.

Aplicação da arquitetura:

Não esta sendo modelados alguns casos de uso que fazem parte da arquitetura inicial: *Record Transaction Supplier, Register Supplier*. Estes dois casos de uso fazem parte de uma transação *B2B*.

O caso de uso *Record Car* corresponde ao caso de uso da arquitetura *Record Product*.

O caso de uso *Record Internet Rent* corresponde ao caso de uso da arquitetura *Record Transaction Customer*.

O caso de uso *Browse Pending Rent* corresponde ao caso de uso da arquitetura *Browse Pending Transaction Customer*.

O caso de uso *Browse Car* corresponde ao caso de uso da arquitetura *Browse Product*.

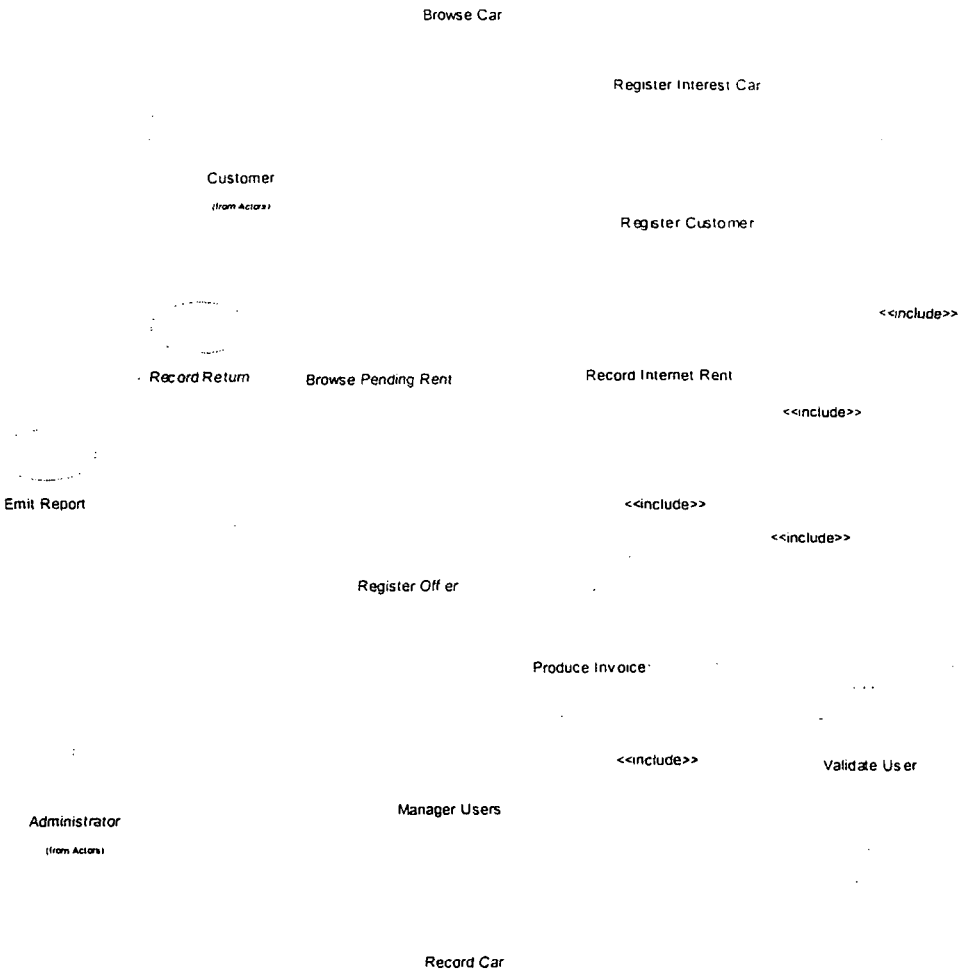
O caso de uso *Register Interest Car* corresponde ao caso de uso da arquitetura *Register Interest Product*.

Os casos de uso *Record Return* e *Emit Report* não fazem parte da arquitetura, são especificações no negócio.

Para este estudo de caso foram necessários somente dois atores que fazem parte da arquitetura o *Administrator* e o *Customer*. As funções dos atores *User* e *Sales Agent* são realizadas pelo *Administrator*.

Na FIGURA 49 são apresentados os casos de uso necessários para resolução dos problemas do negócio. Os casos de uso sem hachurado fazem parte da arquitetura, os demais, com hachurado, foram modelados seguindo as especificações do negócio.

FIGURA 49 - DIAGRAMA DE CASO DE USO – *E-RENT-A-CAR*



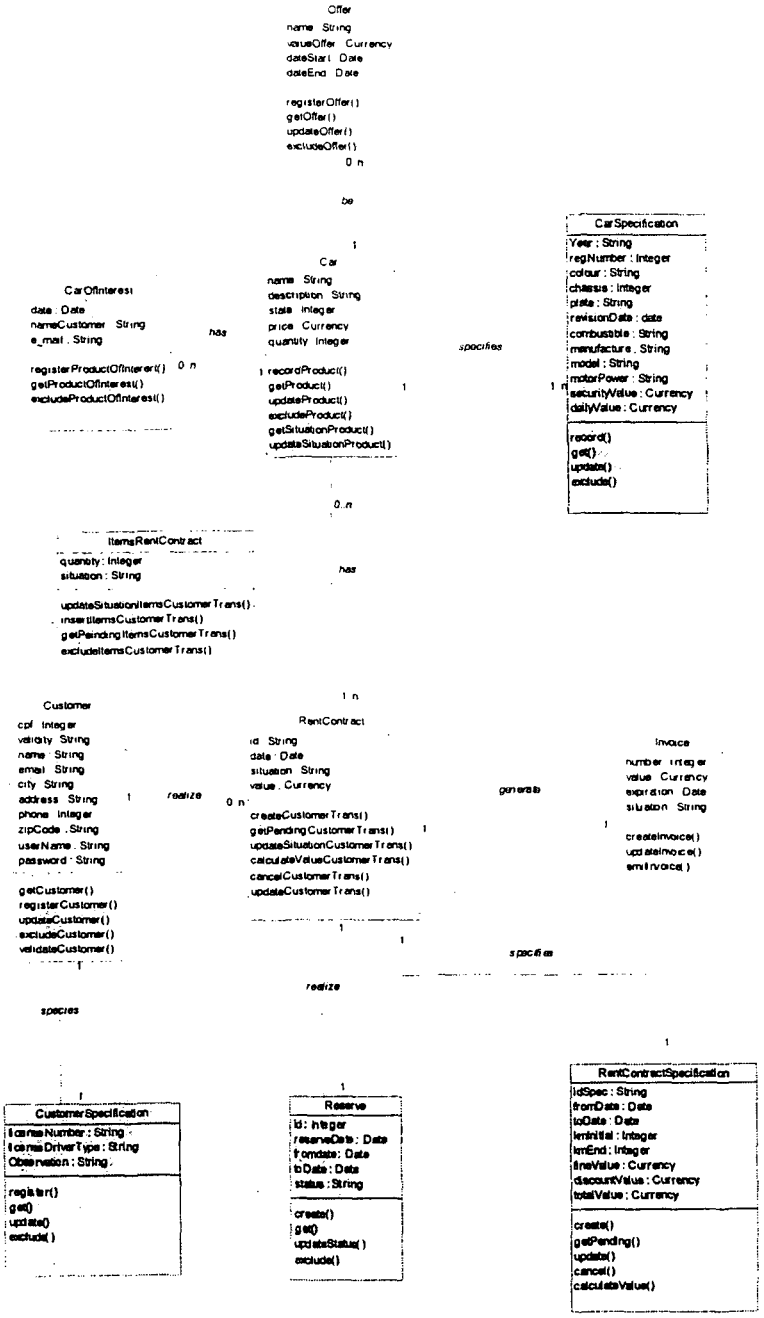
4.2 IDENTIFICAÇÃO DE COMPONENTES

A partir dos modelos definidos no primeiro passo, especificação de requisitos, neste estágio de identificação de componentes será definido um grupo inicial de interfáces de negócio e de interfaces do sistema.

4.2.7 Modelo de tipo de negócio

Esse modelo teve como base o modelo de negócio da especificação de requisitos anterior, foram feitos refinamentos e acrescentados informações para definição do negócio em relação às informações de entradas, as informações processadas e as informações de saída. Como mostra o diagrama da FIGURA 50.

FIGURA 50 - MODELO DE TIPO DE NEGÓCIO – *E-RENT-A-CAR*



Aplicação da arquitetura:

As seguintes classes não são necessárias para o projeto do negócio: *SuplierTransaction*, *ItemsSupplierTransaction* e *Supplier*.

Foi necessário criar as seguintes classes para armazenar características específicas do negócio: *CarSpecification*, *CustomerSpecification*, *Reserve* e *RentContractSpecification*.

A classe *Product* da arquitetura corresponde à classe *Car* no estudo de caso. O atributo *regNumber* corresponde ao RENAVAM do carro.

A classe *CustomerTrans* da arquitetura corresponde à classe *RentContract* no estudo de caso.

A classe *ItemsCustomerTrans* da arquitetura corresponde à classe *ItemsRentContract* no estudo de caso.

A classe *ProductOfInterest* da arquitetura corresponde à classe *CarOfInterest* no estudo de caso.

A classe *ProductCatalog* da arquitetura corresponde à classe *CarCatalog* no estudo de caso.

As classes que tiveram alterações nos nomes são: *Customer*, *Offer*, *Invoice* e *PoolDB*.

4.2.8 Especificação de interface

O processo de especificação de interface baseia-se no modelo de casos de uso, a princípios todos os casos de uso se transformam em interfaces do sistema. Uma outra análise foi feita no modelo de tipo de negócio, foram verificadas quais informações poderiam ser geradas independente do negócio, assim foram definas as interfaces de negócio. Abaixo estão descritas as interfaces do sistema e as interfaces de negócio.

Interface *IRentContract:* oferece os seguintes serviços: criação do registro de uma locação/reserva inicial por um cliente (identificado ou não); atualização da locação; atualização de uma locação pendente no sistema; cancelamento da locação; exclusão de um item selecionado na locação.

Interface *IProduceInvoice:* oferece os seguintes serviços: criação das faturas das locações feitas pelos clientes; emissão das faturas; e, atualização das faturas.

Interface *IBrowsePendingRentContract:* oferece o serviço de consulta da situação de uma locação.

Interface *IRecordOffer:* oferece os seguintes serviços: cadastramento das promoções de locação no sistema; atualização das promoções; busca das promoções registradas; e, exclusão de uma promoção registrada.

Interface *I CarOfInterest:* oferece os seguintes serviços: cadastramento do interesse de um cliente em locar um carro não disponível; atualização do cadastro; exclusão do cadastro; e, busca das informações de um cadastro de interesse de locação de carro.

Interface *IBrowseCar:* oferece o serviço de consulta aos carros disponíveis no sistema para locação.

Interface *ICarSpecification:* oferece os seguintes serviços: cadastramento das especificações dos carros no sistema; atualização das especificações dos carros; busca das especificações dos carros registradas; e, exclusão de uma especificação do carro registrado.

Interface *IRentCarSpecification:* oferece o serviço de criação das especificações do registro de uma locação por um cliente (identificado ou não); atualização das especificações da locação; atualização das especificações de uma locação pendente no sistema; cancelamento das especificações da locação; exclusão de um item selecionado na locação.

Interface *IReserve:* oferece o serviço de criação da reserva de um carro por um cliente (identificado ou não); atualização da reserva de um carro; atualização da

reserva de um carro pendente no sistema; cancelamento da reserva de um carro; exclusão de um item selecionado na reserva.

As interfaces de negócios são aquelas associadas a partes do sistema que são independentes. Na arquitetura existem quatro interfaces identificadas: *ICustomer, ISupplier, IProduct e IUserManager.* Para o estudo de caso em análise as interfaces identificadas são: *ICustomer* a mesma da arquitetura; *ICar,* correspondendo a interfaces *IProduct;* e *IUserManager* a mesma da arquitetura.

Interface *ICustomer:* oferece os seguintes serviços: cadastramento de clientes no sistema; atualização dos dados dos clientes; busca de informações dos clientes cadastrados; e, exclusão do cliente do sistema.

Interface *ICustomerSpecificatio:* oferece os seguintes serviços: cadastramento das especificações do clientes no sistema; atualização das especificações dos clientes; busca de informações dos clientes cadastrados; e, exclusão das especificações do cliente do sistema.

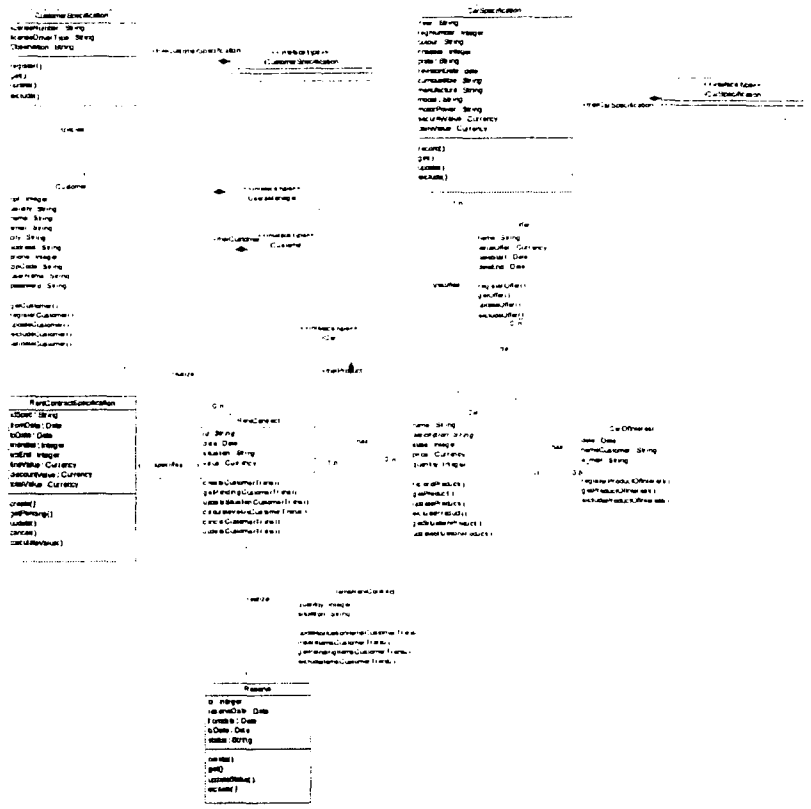
Interface *ICar:* oferece os seguintes serviços: cadastramento dos carros no sistema; atualização dos carros; busca dos carros registrados; e, exclusão de um carro registrado.

Interface *ICarSpecification:* oferece os seguintes serviços: cadastramento das especificações dos carros no sistema; atualização das especificações dos carros; busca das especificações dos carros registrados; e, exclusão das especificações de um carro registrado.

Interface *IUserManager:* oferece os seguintes serviços: cadastramento de usuário para utilização do sistema que necessitam de acesso restrito; atualização dos dados dos usuários; busca de informações dos usuários cadastrados; e, validação dos usuário para uso do sistema.

O diagrama de responsabilidade de interface do negócio, apresentado na FIGURA 51, mostra o relacionamento entre as interfaces do negócio e as classes.

FIGURA 51 - DIAGRAMA DE RESPONSABILIDADE DE INTERFACE DE NEGÓCIO – *E-RENT-A-CAR*



4.3 INTERAÇÃO DO COMPONENTE

Neste estágio é definida a interação entre os componentes. Os diagrama de colaboração gerados nesse estágio estão direcionados as interfaces do sistema e as interfaces do negócio. As figuras que segue apresentam os diagramas de interação para cada método da interface ***ICar***, os demais diagramas não foram apresentados devido ao grande volume de informações.

FIGURA 52 - DIAGRAMA DE INTERAÇÃO getProduct() – *E-RENT-A-CAR*

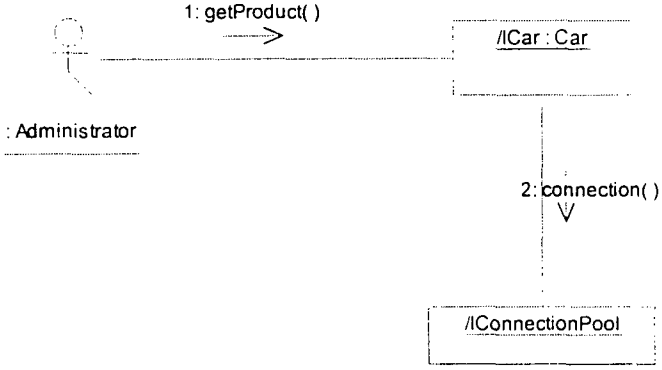


FIGURA 53 - DIAGRAMA DE INTERAÇÃO recordProduct() – *E-RENT-A-CAR*

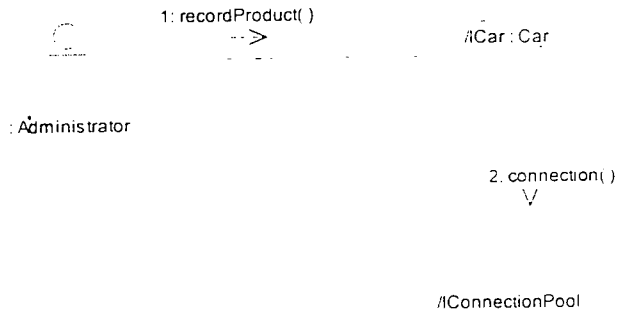


FIGURA 54 - DIAGRAMA DE INTERAÇÃO updateProduct() – *E-RENT-A-CAR*

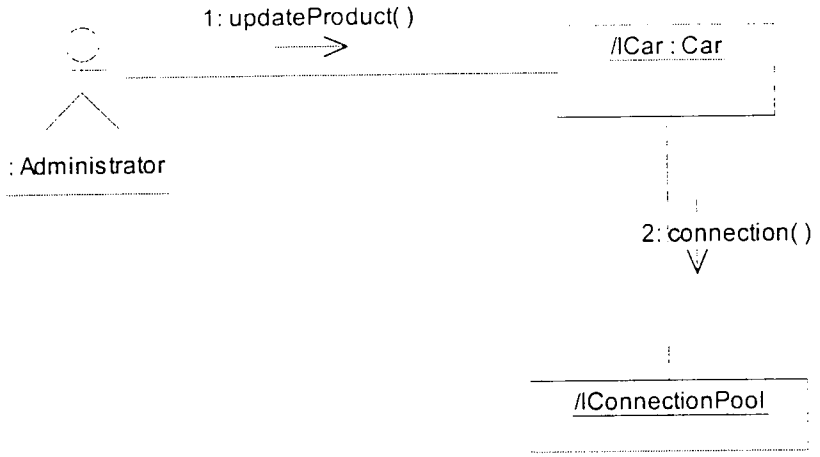


FIGURA 55 - DIAGRAMA DE INTERAÇÃO excludeProduct() – *E-RENT-A-CAR*

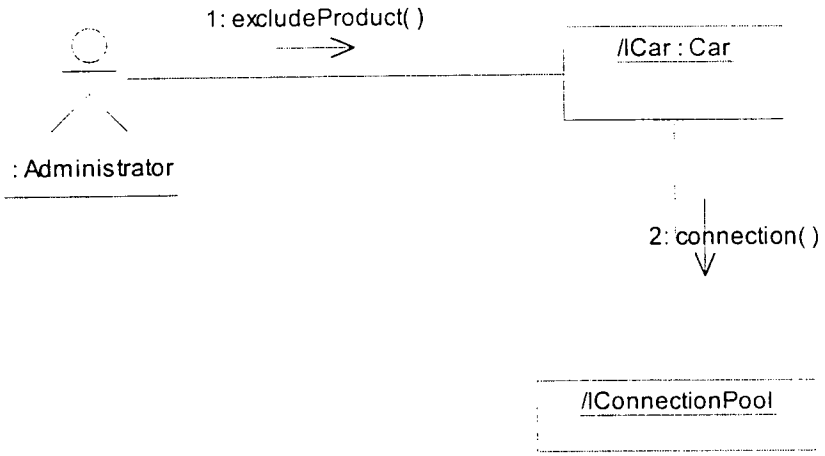


FIGURA 56 - DIAGRAMA DE INTERAÇÃO getSituationProduct() – *E-RENT-A-CAR*

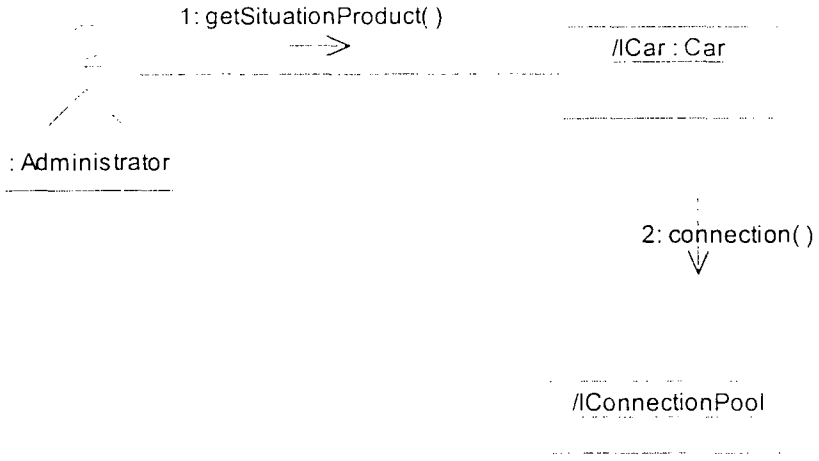
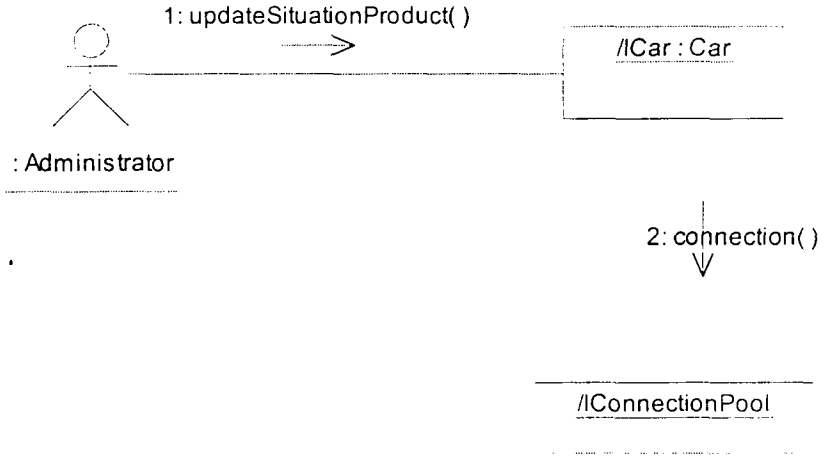


FIGURA 57 - DIAGRAMA DE INTERAÇÃO updateSituationProduct() – *E-RENT-A-CAR*

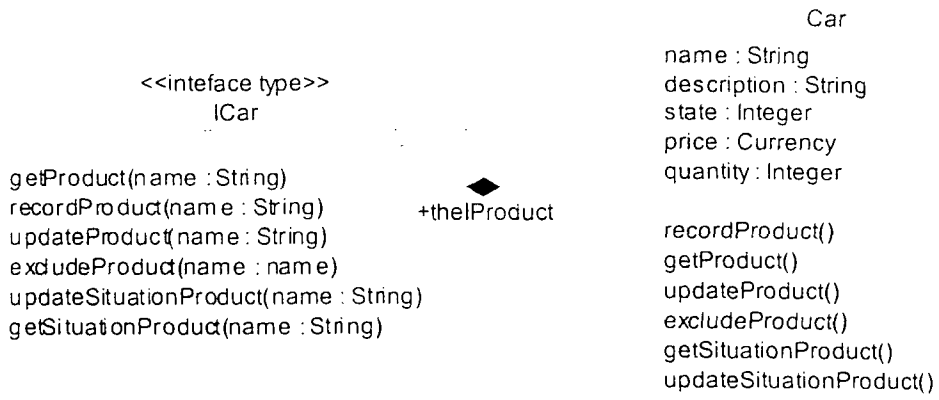


4.4 ESPECIFICAÇÃO DO COMPONENTE

Neste estágio foi realizado o detalhamento das operações e das restrições das interfaces gerando o diagrama de especificação de interface e definindo as pré- e pós-condições dos métodos. O diagrama de especificação de interface é apresentado a seguir para cada interface definida no estágio anterior, esse diagrama não representa os componentes do sistema, ele representa a interação entre a interface e a classe a qual ela se comunica e mostra ainda as demais classes necessárias para a instanciação de seus métodos, mesmo que estas classes façam

parte de outros componentes. Posteriormente a comunicação será realizada entre os componentes. A definição das pré- e pós-condições foi apresentado para a interface *ICar*, servindo como modelo para mostrar como este trabalho foi realizado, as demais não estão listadas devido ao grande volume de informações que seriam acrescentada a este trabalho.

Interface: *ICar*



context ICar : :getProduct ()

pre:

car = **exists** (car | name = name)

post:

let theCar – car **-> select** (car | car.name = name)

result.name = theCar.name **and**

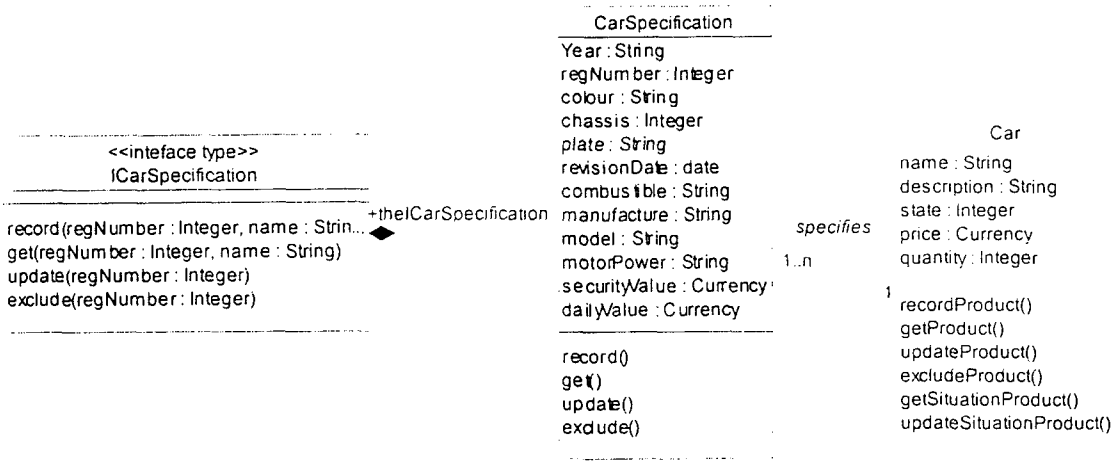
result.description = theCar. description **and**

result.state = theCar.state **and**

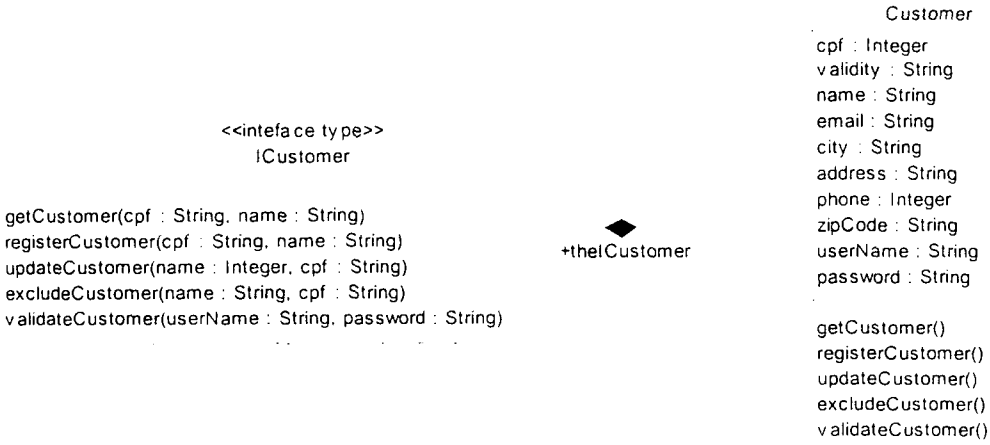
result.price = theCar.price **and**

result.quantity = theCar.quantity

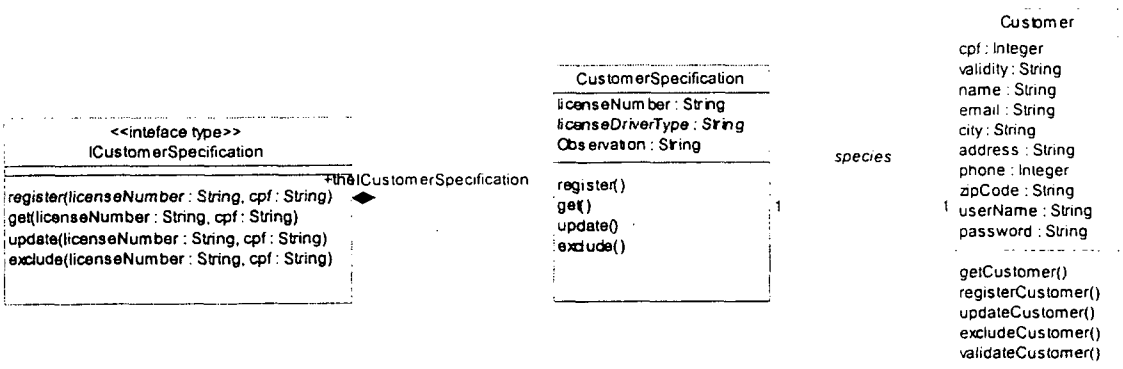
Interface: ICarSpecification



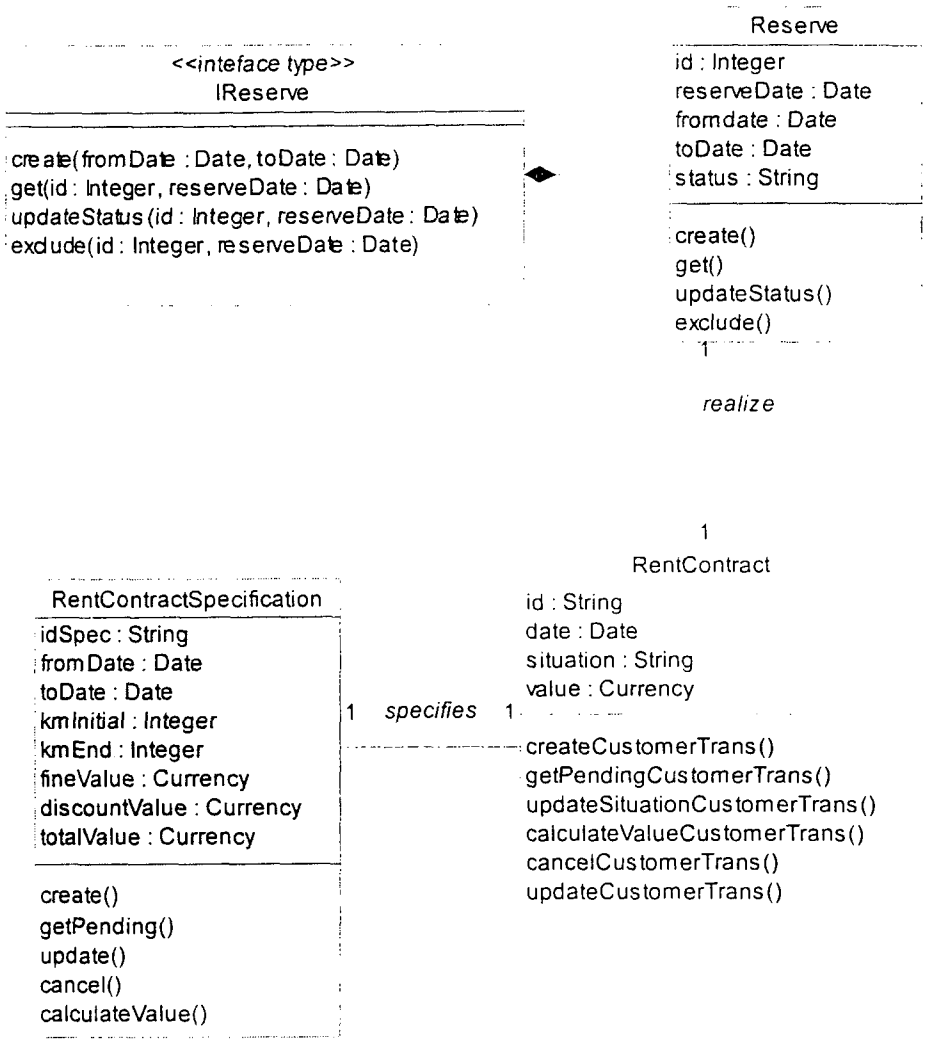
Interface: *ICustomer*



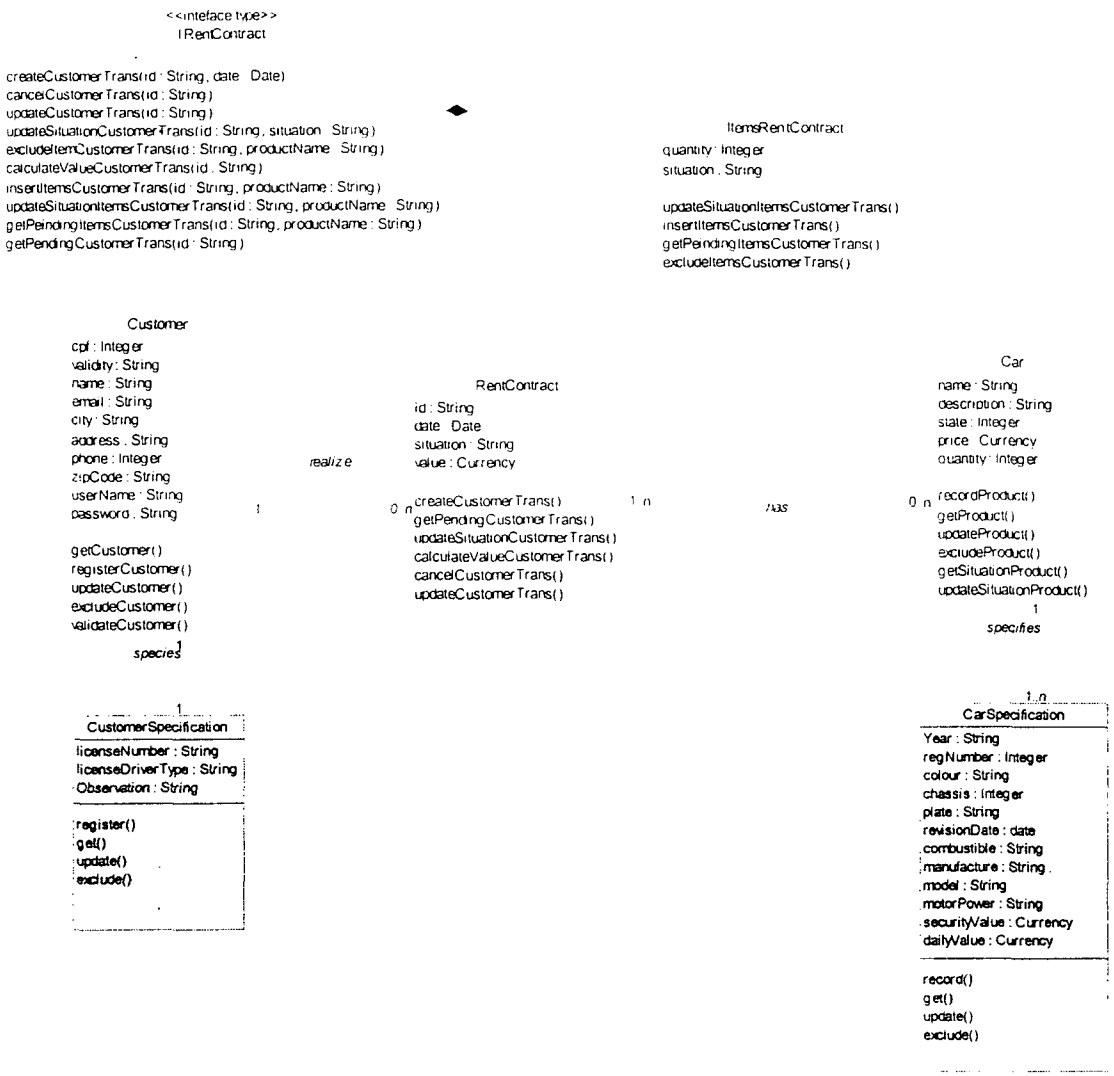
Interface: *ICustomerSpeficication*



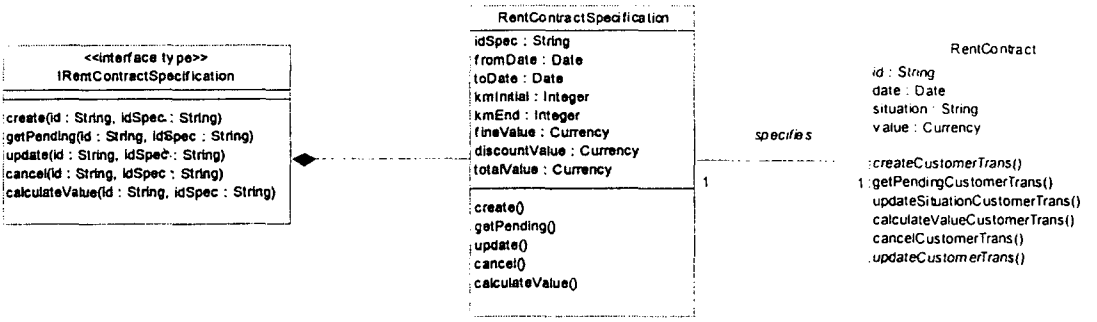
Interface: *IReserve*



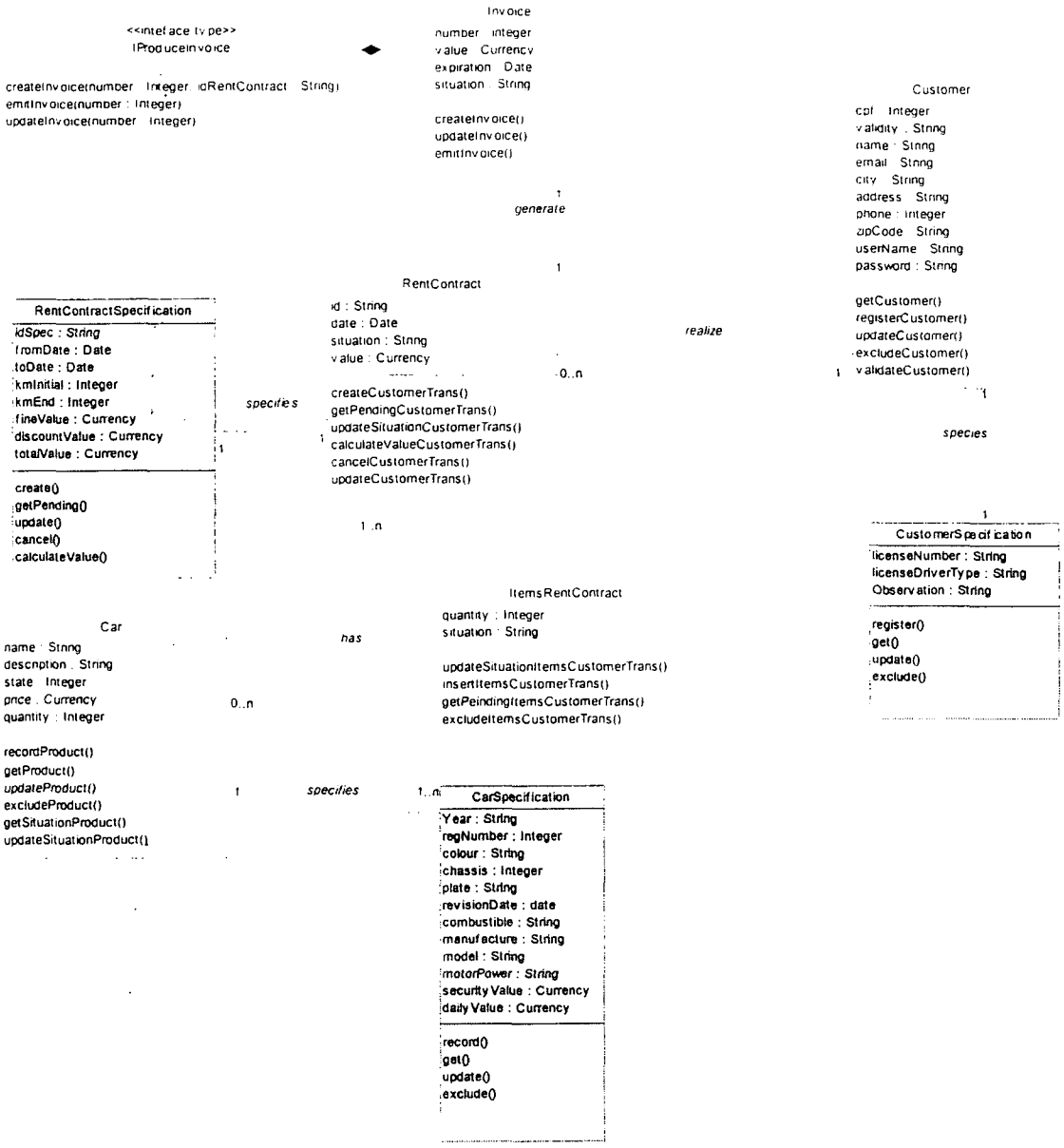
Interface: *IRentContract*



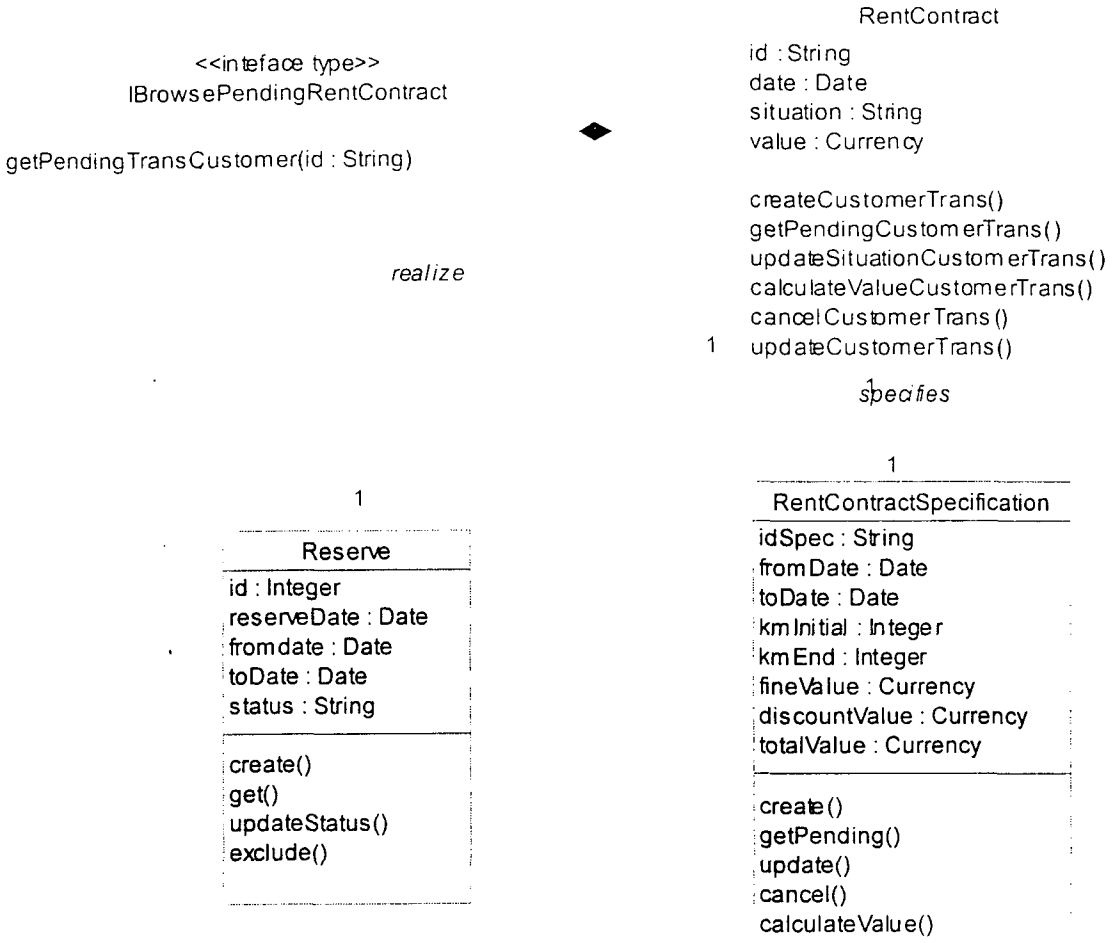
Interface: *IRentContractSpecification*



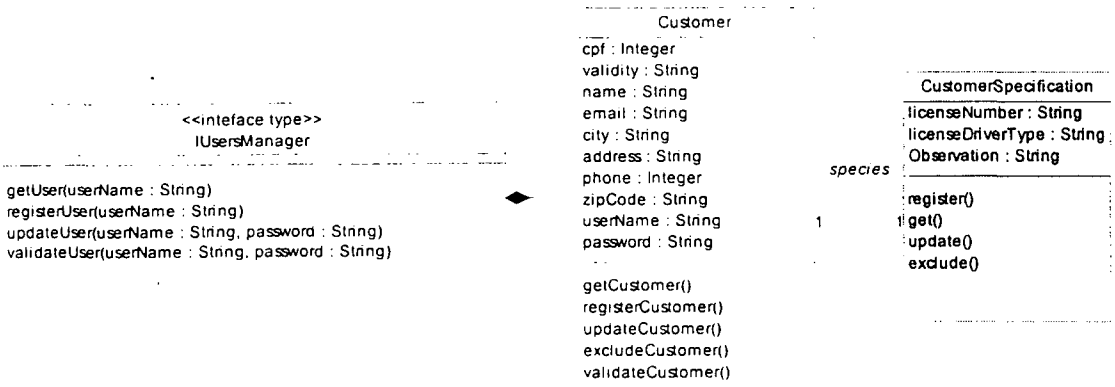
Interface: *IProduceInvoice*



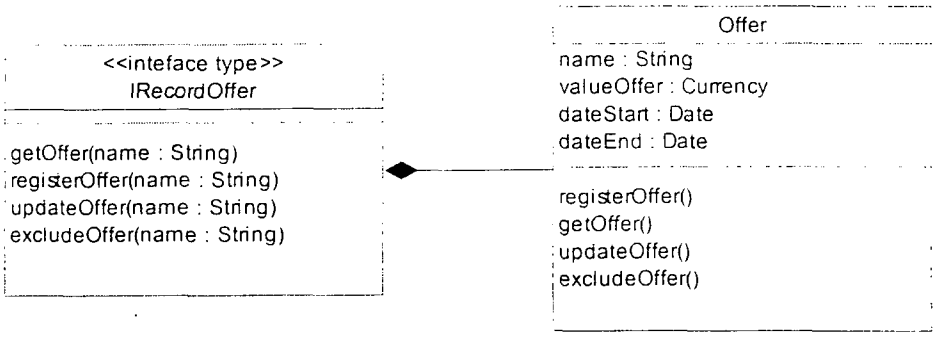
Interface: *IBrowsePendingRentContract*



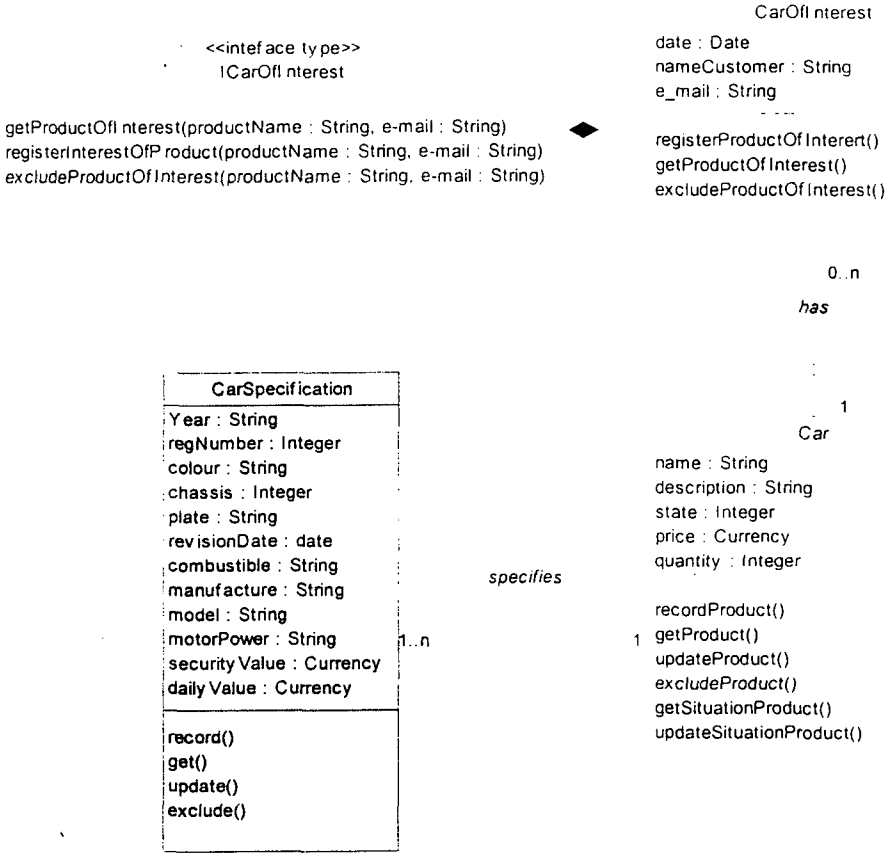
Interface: *IUserManager*



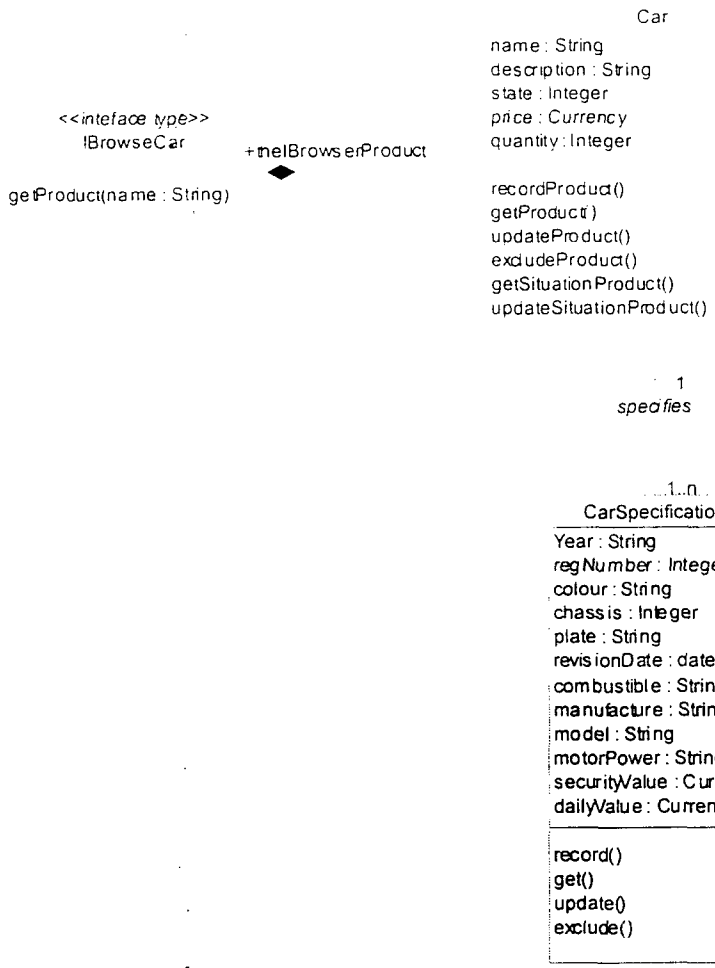
Interface: *IRecordOffer*



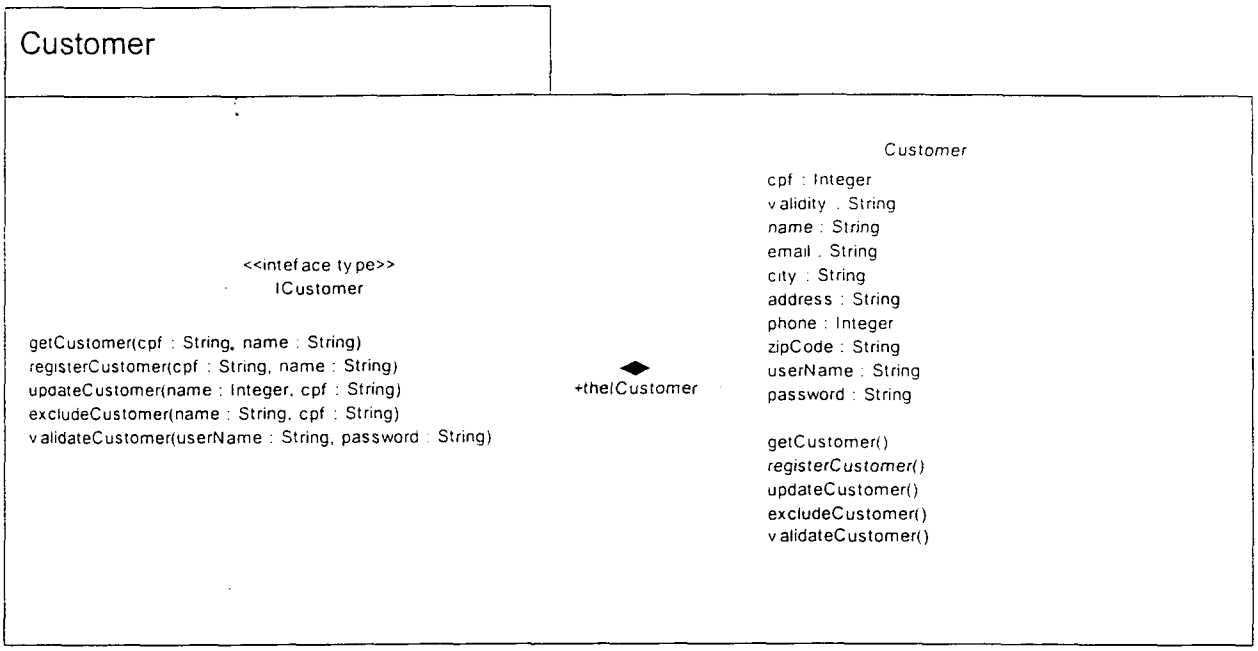
Interface: *ICarOfInterest*

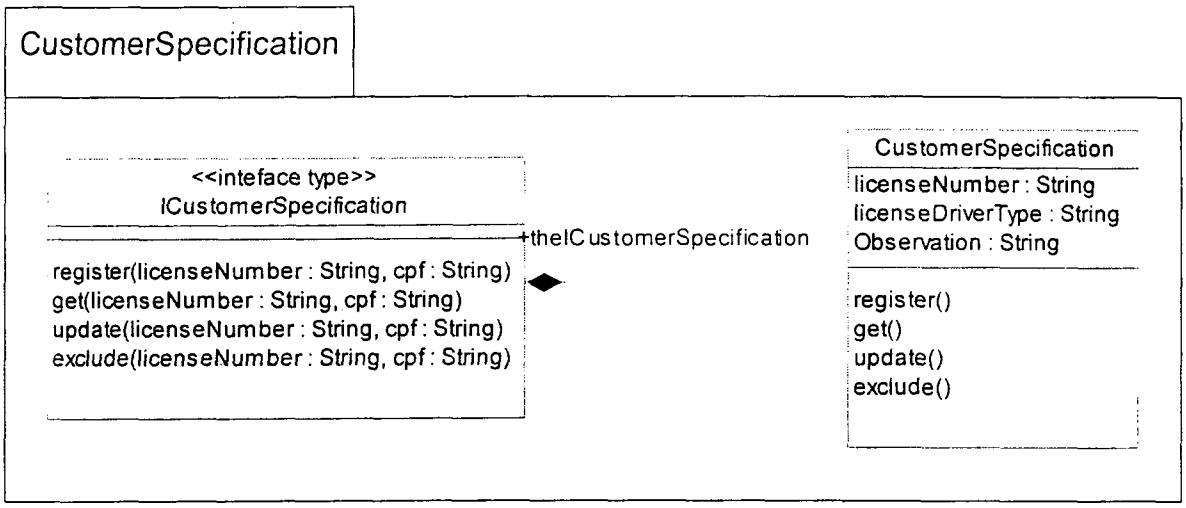


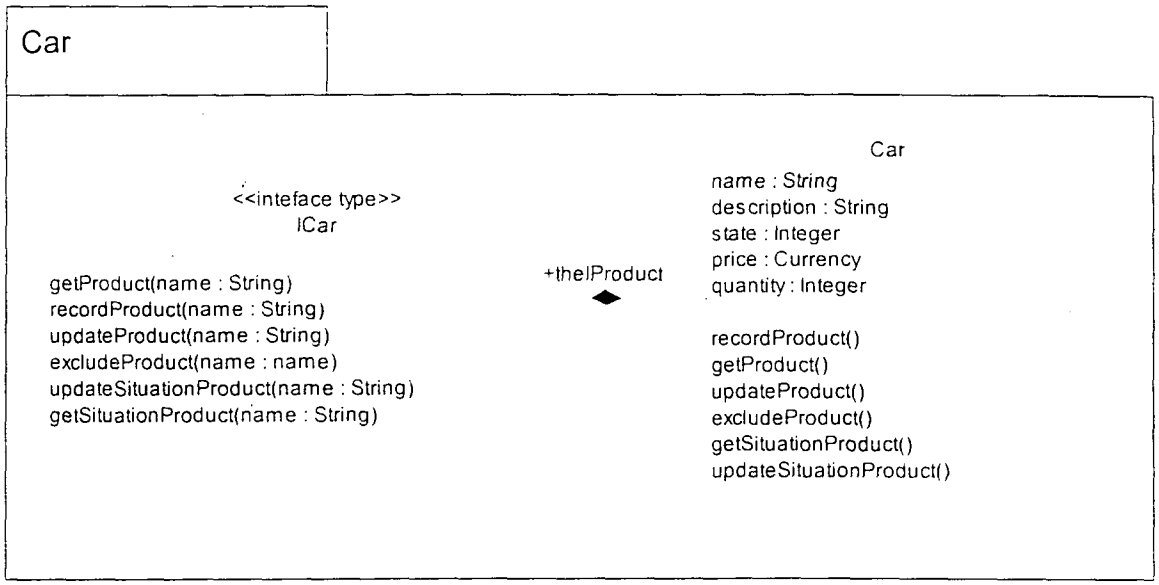
Interface: *IBrowseCar*

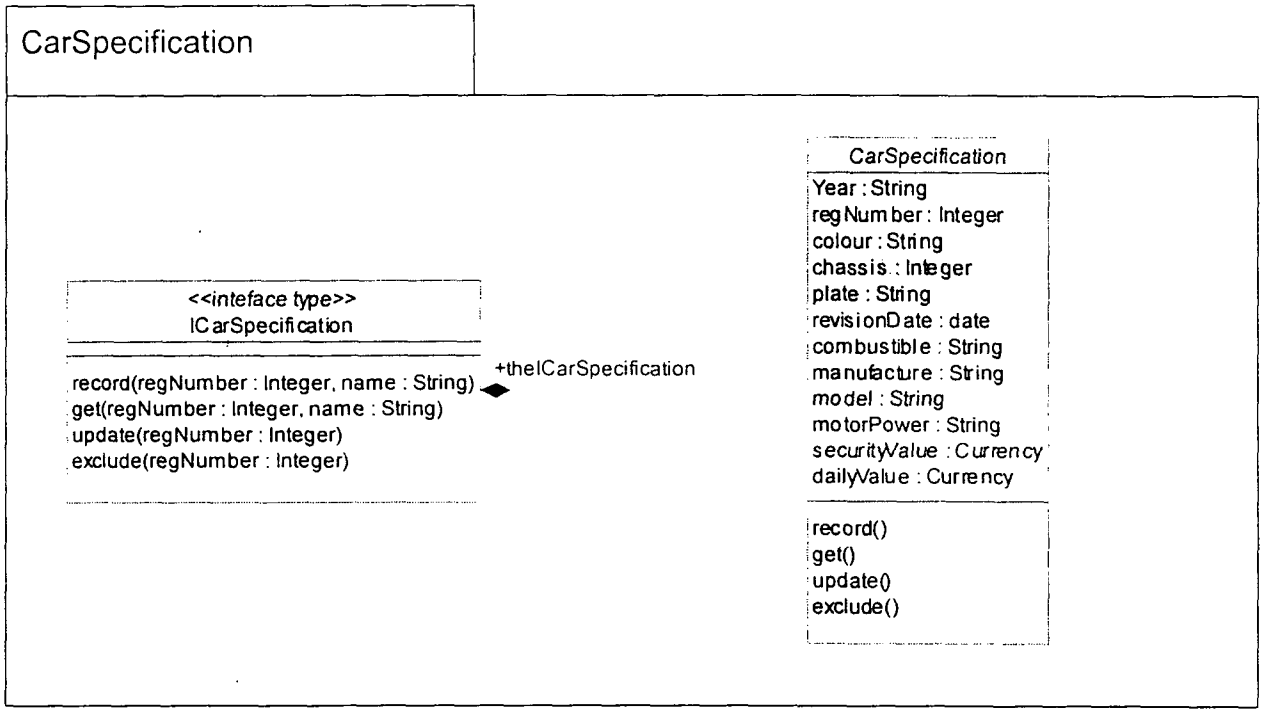


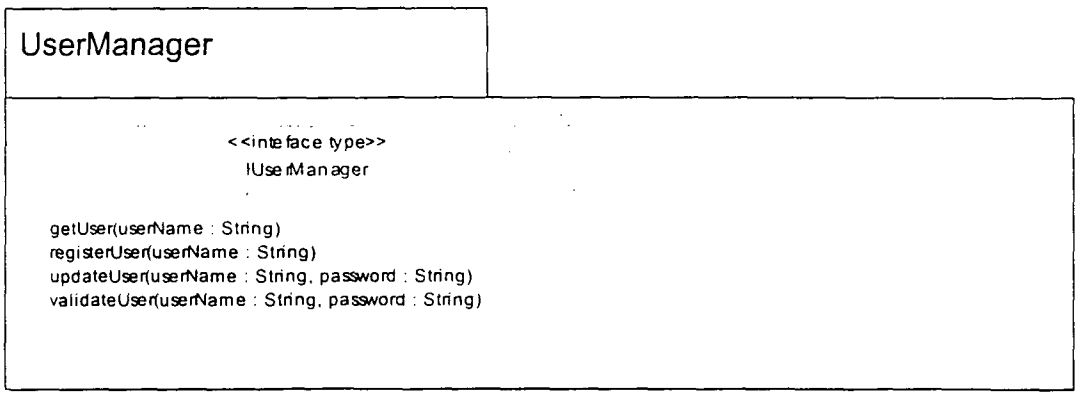
O segundo diagrama gerado neste estágio é o diagrama de especificação de componente, ele representa quais classes e interfaces farão parte do componente. Segue abaixo a representação dos componentes necessários para definição do SI.

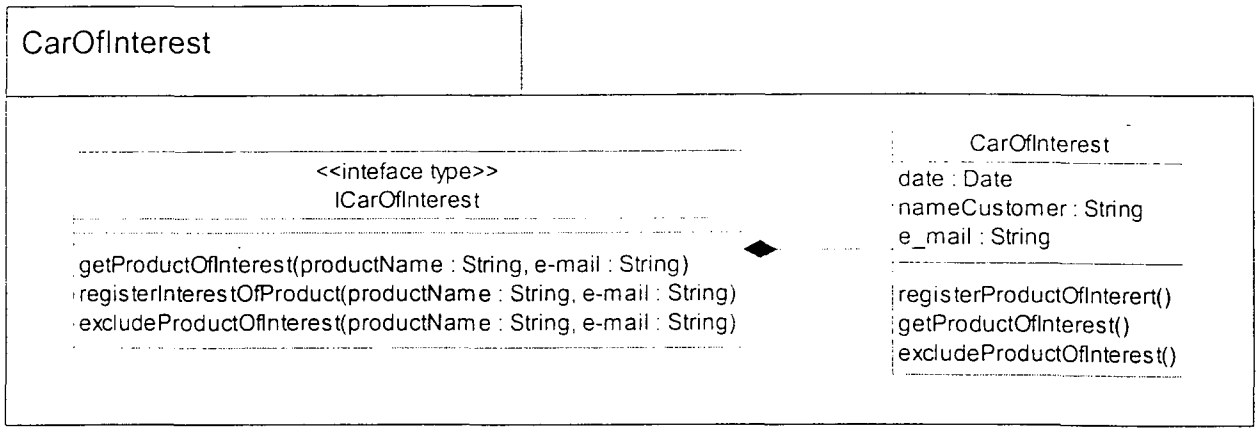

Customer
<<inteface type>>
ICustomer
getCustomer(cpf : String, name : String)
registerCustomer(cpf : String, name : String)
updateCustomer(name : Integer, cpf : String)
excludeCustomer(name : String, cpf : String)
validateCustomer(userName : String, password : String)
+theICustomer
Customer
cpf : Integer
validity : String
name : String
email : String
city : String
address : String
phone : Integer
zipCode : String
userName : String
password : String
getCustomer()
registerCustomer()
updateCustomer()
excludeCustomer()
validateCustomer()

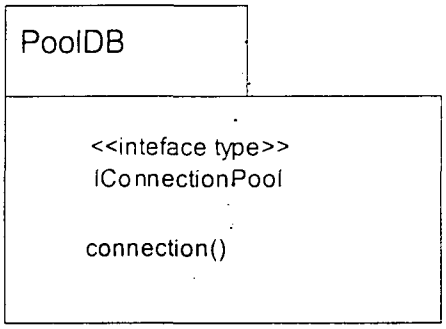

CustomerSpecification
<<inteface type>>
ICustomerSpecification
register(licenseNumber : String, cpf : String)
get(licenseNumber : String, cpf : String)
update(licenseNumber : String, cpf : String)
exclude(licenseNumber : String, cpf : String)
+theICustomerSpecification
CustomerSpecification
licenseNumber : String
licenseDriverType : String
Observation : String
register()
get()
update()
exclude()

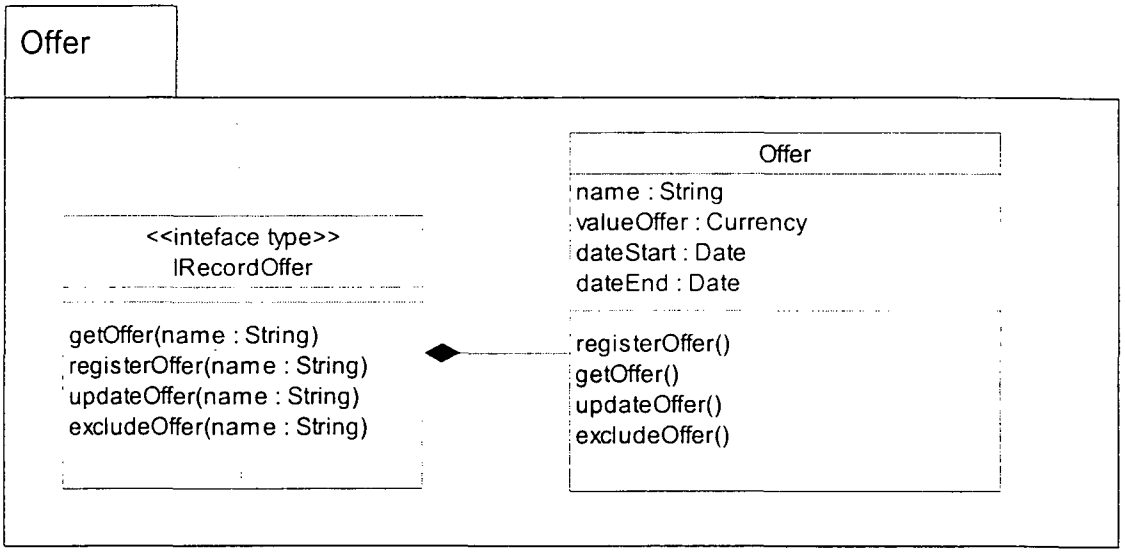

Car
<<inteface type>>
ICar
getProduct(name : String)
recordProduct(name : String)
updateProduct(name : String)
excludeProduct(name : name)
updateSituationProduct(name : String)
getSituationProduct(name : String)
+theIProduct
Car
name : String
description : String
state : Integer
price : Currency
quantity : Integer
recordProduct()
getProduct()
updateProduct()
excludeProduct()
getSituationProduct()
updateSituationProduct()

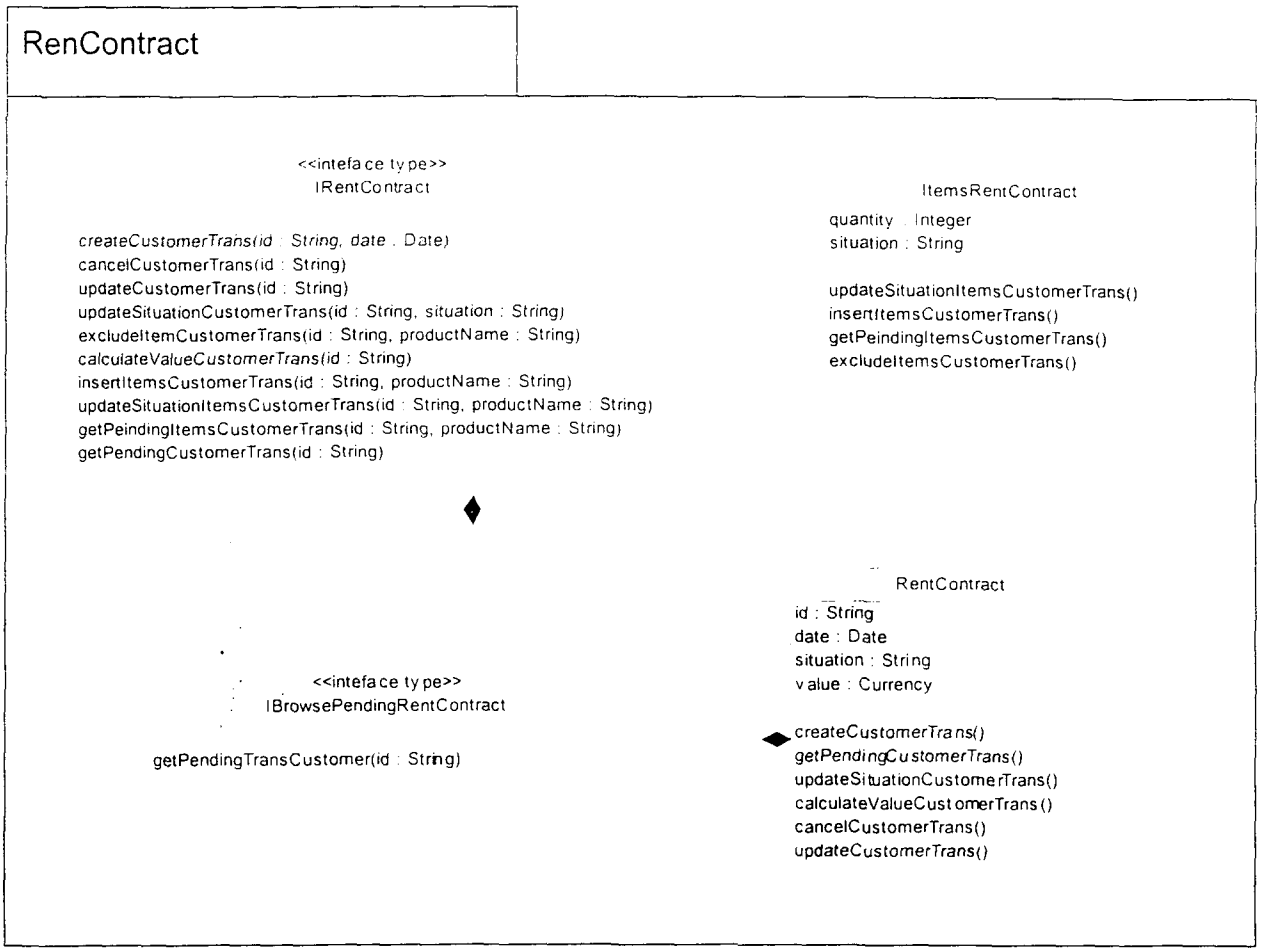

CarSpecification
<<inteface type>>
ICarSpecification
record(regNumber : Integer, name : String)
get(regNumber : Integer, name : String)
update(regNumber : Integer)
exclude(regNumber : Integer)
+theICarSpecification
CarSpecification
Year : String
regNumber : Integer
colour : String
chassis : Integer
plate : String
revisionDate : date
combustible : String
manufacture : String
model : String
motorPower : String
securityValue : Currency
dailyValue : Currency
record()
get()
update()
exclude()

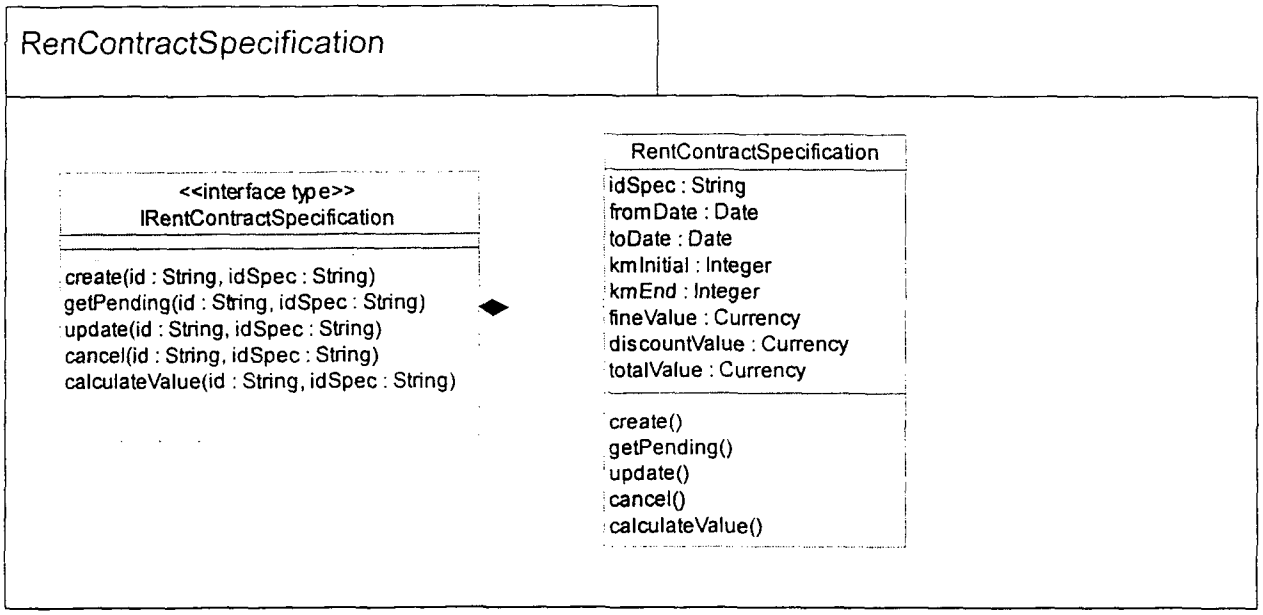

UserManager
<<inteface type>>
IUserManager
getUser(userName : String)
registerUser(userName : String)
updateUser(userName : String, password : String)
validateUser(userName : String, password : String)

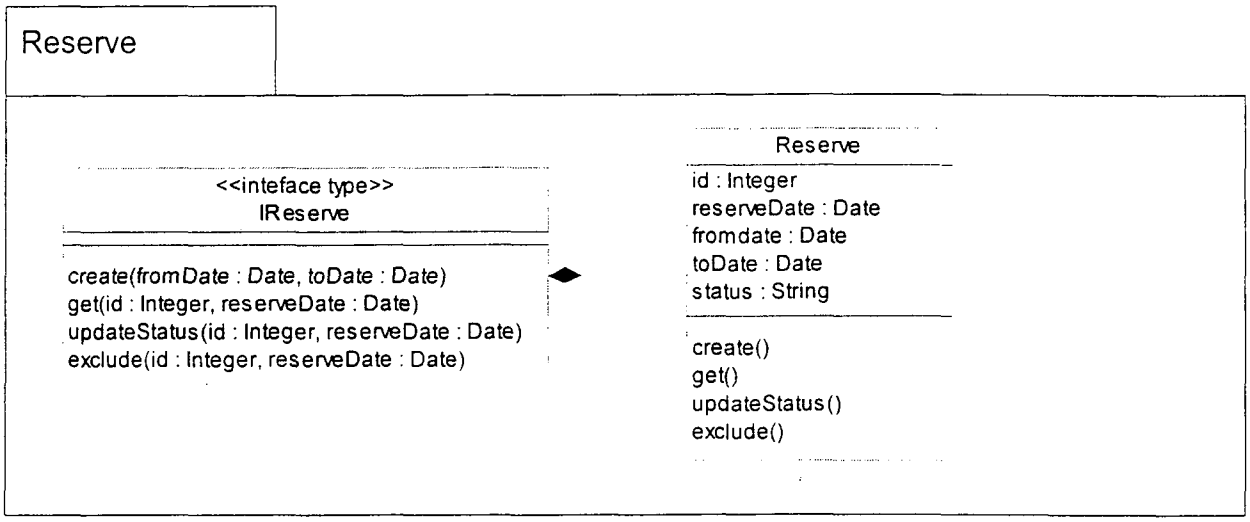

CarOfInterest
<<inteface type>>
ICarOfInterest
getProductOfInterest(productName : String, e-mail : String)
registerInterestOfProduct(productName : String, e-mail : String)
excludeProductOfInterest(productName : String, e-mail : String)
CarOfInterest
date : Date
nameCustomer : String
e_mail : String
registerProductOfInterert()
getProductOfInterest()
excludeProductOfInterest()

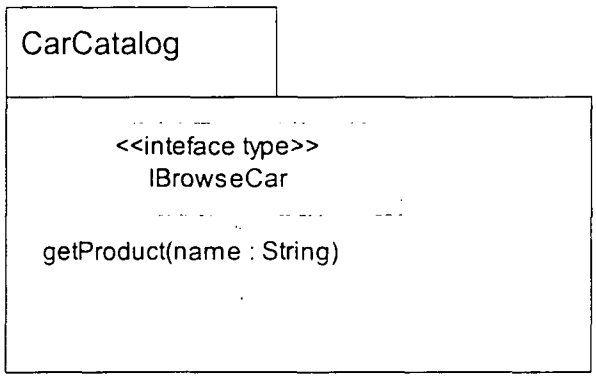

PoolDB
<<inteface type>>
IConnectionPool
connection()

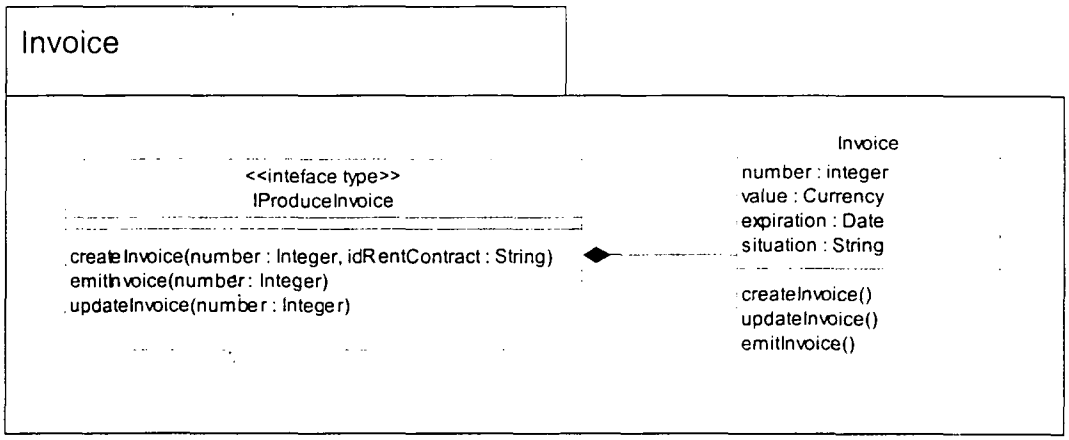

Offer
<<inteface type>>
IRecordOffer
getOffer(name : String)
registerOffer(name : String)
updateOffer(name : String)
excludeOffer(name : String)
Offer
name : String
valueOffer : Currency
dateStart : Date
dateEnd : Date
registerOffer()
getOffer()
updateOffer()
excludeOffer()


RenContract
<<intefa ce ty pe>>
IRentContract
createCustomerTrans(id : String, date : Date)
cancelCustomerTrans(id : String)
updateCustomerTrans(id : String)
updateSituationCustomerTrans(id : String, situation : String)
excludeItemCustomerTrans(id : String, productName : String)
calculateValueCustomerTrans(id : String)
insertItemsCustomerTrans(id : String, productName : String)
updateSituationItemsCustomerTrans(id : String, productName : String)
getPeindingItemsCustomerTrans(id : String, productName : String)
getPendingCustomerTrans(id : String)
ItemsRentContract
quantity : Integer
situation : String
updateSituationItemsCustomerTrans()
insertItemsCustomerTrans()
getPeindingItemsCustomerTrans()
excludeItemsCustomerTrans()
<<intefa ce ty pe>>
IBrowsePendingRentContract
getPendingTransCustomer(id : String)
RentContract
id : String
date : Date
situation : String
v alue : Currency
createCustomerTrans()
getPendingCustomerTrans()
updateSituationCustomerTrans()
calculateValueCustomerTrans()
cancelCustomerTrans()
updateCustomerTrans()


RenContractSpecification
<<interface type>>
IRentContractSpecification
create(id : String, idSpec : String)
getPending(id : String, idSpec : String)
update(id : String, idSpec : String)
cancel(id : String, idSpec : String)
calculateValue(id : String, idSpec : String)
RentContractSpecification
idSpec : String
fromDate : Date
toDate : Date
kmInitial : Integer
kmEnd : Integer
fineValue : Currency
discountValue : Currency
totalValue : Currency
create()
getPending()
update()
cancel()
calculateValue()


Reserve
<<inteface type>>
IReserve
create(fromDate : Date, toDate : Date)
get(id : Integer, reserveDate : Date)
updateStatus(id : Integer, reserveDate : Date)
exclude(id : Integer, reserveDate : Date)
Reserve
id : Integer
reserveDate : Date
fromdate : Date
toDate : Date
status : String
create()
get()
updateStatus()
exclude()


CarCatalog
<<inteface type>>
IBrowseCar
getProduct(name : String)


Invoice
<<inteface type>>
IProduceInvoice
createInvoice(number : Integer, idRentContract : String)
emitInvoice(number : Integer)
updateInvoice(number : Integer)
Invoice
number : integer
value : Currency
expiration : Date
situation : String
createInvoice()
updateInvoice()
emitInvoice()

4.5 ARQUITETURA DE COMPONENTE

A FIGURA 58 apresenta a arquitetura do estudo do caso *e-rent-a-car*. Os componentes com hachurado representam os componentes específicos do negócio, o componente com dois asteriscos (**) representa um componente externo ao desenvolvimento do sistema, ou seja, será adquirido de terceiros e os componentes sem hachurado representam os componentes da arquitetura *WbIS*.

Alguns componentes que faziam parte da arquitetura não foram necessários para a modelagem do estudo de caso. São eles: componente *Supplier*, o componente *SupplierTrans*, o componente *PrincingUtilities*.

Os componentes da arquitetura aplicados ao estudo de caso são: *Customer, CustomerTrans (RentContract), Product (Car), ItemsCustomerTrans (ItemsRentContract), ProductCatalog (CarCatalog), ProductOfInterest (CarOfInterest), Offer, Invoice, PoolDB, UserManager* e *InternetPayments*.

FIGURA 58 - ARQUITETURA DO *E-RENT-A-CAR*