ANA CLAUDIA ROSSI

REPRESENTAÇÃO DO COMPONENTE DE SOFTWARE NA FARCSOFT: FERRAMENTA DE APOIO À REUTILIZAÇÃO DE COMPONENTES DE SOFTWARE

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção de Título de Mestre em Engenharia.

São Paulo

2004

ANA CLAUDIA ROSSI

REPRESENTAÇÃO DO COMPONENTE DE SOFTWARE NA FARCSOFT: FERRAMENTA DE APOIO À REUTILIZAÇÃO DE COMPONENTES DE SOFTWARE

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção de Título de Mestre em Engenharia.

Área de Concentração:
Sistemas Digitais

Orientador:
Profa. Dra. Selma Shin Shimizu Melnikoff

São Paulo
2004

FICHA CATALOGRÁFICA

Rossi, Ana Claudia

Representação de Componentes de Software na FARCSoft-Ferramenta de Apoio à Reutilização de Componentes de Software / Ana Claudia Rossi -- São Paulo, 2004.

236p.

Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Softwares (programa (documento)) 2. Reutilização de Software 3. Componente de Software 4. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t

Aos meus Pais e ao Roberto por toda a dedicação e amor.

AGRADECIMENTOS

À minha amiga e orientadora Profa. Dra. Selma Shin Shimizu Melnikoff, pelas diretrizes, orientações e estímulo, que foram fundamentais para a conclusão deste trabalho.

Ás minhas irmãs Lídia e Catarina que sempre me apoiaram e incentivam os meus estudos.

Aos meus filhos, Carolina e Bruno.

Ao CNPq que apoiou o desenvolvimento deste trabalho.

RESUMO

Atualmente, as organizações estão cada vez mais dependentes de sistemas de informação para a realização de seus negócios. Com isso, uma das preocupações, na área de desenvolvimento de software, é a obtenção cada vez mais rápida de sistemas que atendam as necessidades atuais e que sejam flexíveis para acompanhar as mudanças de tecnologia e práticas de negócio. A reutilização de componentes de software tem sido considerada uma das formas para obter redução dos custos e do tempo de desenvolvimento e aumento da produtividade e da qualidade do produto de software. A implantação da reutilização de componentes é baseada em três elementos principais que consistem de um processo de desenvolvimento voltado para reutilização, de uma ferramenta adequada e de uma cultura de projeto. A ferramenta, por sua vez, deve ter a capacidade de armazenar os componentes e de fornecer recursos para uma recuperação eficiente. O objetivo deste trabalho é definir uma representação de componentes em um repositório, a qual permita armazenar diferentes tipos de componentes de software. Para isso, foi especificada a **F**erramenta de **A**poio à **R**eutilização de **C**omponentes de **Soft**ware, denominada de – FARCSoft, que deve fornecer suporte à reutilização de componentes de software. Esta ferramenta apresenta recursos para armazenar, gerenciar, buscar e recuperar os componentes do seu repositório. A capacidade de representação foi avaliada por meio de um conjunto de componentes de tipos, porte e tecnologia diversos, os quais foram modelados e catalogados.

ABSTRACT

Nowadays, organizations increasingly depend on information systems to carry out their business. Thus, one of the preoccupations in the software development area is the need to obtain systems faster and faster, attending to current needs and sufficiently flexible to accompany changes in technology and business practices. Software component reuse has been considered one of the ways to reduce costs and development time and increase productivity and software quality. The implantation of component reuse is based on three main elements, which consist in a development process oriented towards reuse, an adequate tool and a project culture. The tool, in turn, must be able to store the components and to supply resources for the sake of efficient recovery. This study aims to define a component representation in a repository, which allows for the storage of different kinds of software components. For this purpose, a Software Component Reuse Support Tool was specified, called FARCSoft, which should support the reuse of software components. This tool presents resources to store, manage, search and recover the components of a repository. Representation capacity was evaluated by means of a set of components with different types, sizes and technologies, which were modeled and catalogued.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADS	- *Architectural Description Standard*
CASE	- *Computer-Aied Software Engineering*
CBSE	- *Component-Based Software Engineering*
CHAIMS	- *Compiling High-level Access Interfaces for Multisite Software*
COM	- *Component Object Model*
CORBA	- *Common Object Request Broker Architecture*
EJB	- *Enterprise Java Beans*
ESPRIT	- *European Software Program for Research on Technology*
FARCSoft	- Ferramenta de Apoio à Reutilização de Componentes de Software
IDL	- *Interface Description Language*
LTS	- Laboratório de Tecnologia e Software
MIL	- *Module Interconnection Language*
OMG	- *Object Management Group*
PCTE	- *Portable Common Tools and Enviroments*
POL	- *Problem Oriented Languages*
REBOOT	- *Reused Based en Object Oriented Techniques*
RCR	- Repositório de Componentes Reutilizáveis
RAS	- *Reusable Assets Specification*
RIG	- *Reuse Interoperability Group*
RUP	- *Rational Unified Process*
SEI	- *Software Engineering Institute*
STARS	- *Software Technology for Adaptable Reliable Systems*
UDM	- *Uniform Data Model*
UML	- *Unified Modelling Language*
VHLL	- *Very High Level Language*
XML	- *eXtended Marked Language*
W3C	- *World Wide Web Consortium*
WBS	- *Work Breakdown Structure*

1. INTRODUÇÃO

Este capítulo tem o intuito de apresentar o objetivo desta dissertação e a justificativa para o tema. A estrutura da dissertação é também apresentada no final deste capítulo, com uma síntese de cada de capítulo.

1.1 Objetivo

Objetivo deste trabalho é definir uma representação e um padrão de documentação de diferentes tipos de componentes de software em um repositório. Os componentes de software podem variar com relação ao porte, à complexidade, à tecnologia aplicada para a sua construção e à quantidade diversificada de artefatos para a sua descrição.

A estrutura do repositório de componentes reutilizáveis foi definida para estabelecer um conjunto de informação para representar e documentar estes diferentes tipos de informação associadas ao componente de software.

Para isso, foi especificada uma ferramenta que permita o compartilhamento de componentes de software para reutilização no processo de desenvolvimento de sistemas de software, denominada de **F**erramenta de **A**poio à **R**eutilização de **C**omponentes de **Soft**ware, – FARCSoft.

A FARCSoft deve fornecer suporte à reutilização de software no processo de desenvolvimento de software baseado em componentes, através de recursos para armazenar a informação, em um repositório de componentes de software reutilizáveis, para gerenciar, buscar e recuperar esta informação, e também auxiliar

no entendimento da funcionalidade e na integração do componente de software para construção de um novo sistema de software.

Segundo Pressman (2000), um repositório de deve possuir um meta modelo, ou seja, um modelo de informação, que descreva a estrutura, relacionamentos e semântica dos dados nele armazenados. Desta forma, foi definido um meta modelo para representar o componente de software e toda sua informação relacionada para o repositório de componentes de software da FARCSoft.

Para a avaliação do meta-modelo que descreve o componente de software, foi aplicado um estudo de caso para exercitar a capacidade do repositório em representar e documentar diferentes tipos de componentes de software, juntamente com o respectivo conjunto de informação adicional. Os dados destes componentes foram mapeados conforme a proposta da FARCSoft e o resultado deste experimento foi analisado para avaliar a adequação do repositório.

Este trabalho se baseia em dois conceitos principais: reutilização de software e componente de software.

Na literatura existem muitas definições para o conceito de reutilização de software [(PRIETO-DÍAZ; FREEMAN, 1987); (BASILI; CALDIERA; CANTONE, 1992); (KRUEGER, 1992); (FRAKES; TERRY, 1996); (MCCLURE, 1997); (LIM, 1998)]. Neste trabalho é utilizada a definição conforme a apresentada por McClure (1997), onde a reutilização de software é o processo de construção ou montagem de sistemas de software a partir de componentes de software que foram projetados para reutilização.

Assim como o conceito de reutilização de software, o conceito de componente de software apresenta muitas definições na literatura de engenharia de software, onde inicialmente era tratado como parte de código fonte, sub-rotinas até as definições mais recentes.

E em 1992, Basili, Caldiera e Cantone (1992) definiram componente de software como sendo a coleção de artefatos, que disponibiliza, para organização do projeto, o que é necessário para integrar o componente a um sistema de aplicação e fornecer suporte durante o ciclo de vida para este sistema.

Uma outra definição, apresentada por alguns autores, define componente de software como tipo, classe ou qualquer outro produto que foi gerado especificamente para ser reutilizável (JACOBSON; GRISS; JONSSON, 1997). Ainda, componente de software, segundo Hopkins (2000), é definido como um pacote físico de executável de software com uma interface bem definida e pública.

Pode-se notar que, componente de software é um elemento de sistemas com forte tendência à reutilização e que disponibiliza um ou mais serviços que devem ser de conhecimento dos projetistas que os utilizam.

Para isso, o componente de software deve ser descrito através de um conjunto de informação que auxilie no entendimento de sua funcionalidade e sua finalidade, e por um conjunto de artefatos de software que descreva o seu processo de construção e que disponibilize a informação necessária para integrar o artefato a um sistema de aplicação, e fornecer suporte durante o ciclo de vida do sistema.

Artefato de software é um termo utilizado na nomenclatura do Processo Unificado e UML (Unified Modeling Language) (JACOBSON; RUMBAUGH; BOOCH, 1998). Neste contexto, artefato de software é qualquer parte tangível de informação que é criada, alterada, e utilizada pelo projetista durante o processo de desenvolvimento de software. De acordo com esta definição, um artefato de software pode ser um documento de especificação de requisitos, arquitetura, programa, partes de programa, projeto, modelo, ou qualquer outro documento associado ao software.

Foram avaliados os principais processos existentes na literatura sobre reutilização de componentes de software, para definir o mecanismo de suporte necessário e investigar os principais aspectos relacionados para o estabelecimento das

características de um sistema de classificação, busca e recuperação de componentes de software. Também foram estudadas propostas de documentação de componentes de software, para definir o conjunto de informação para auxiliar no entendimento da funcionalidade e em como integrar o componente de software reutilizável para a construção de um novo sistema.

Além disso, foram analisados os modelos de processo de desenvolvimento de software baseado em componentes, a fim de determinar os artefatos necessários para a representação dos componentes de software, para definir a informação a ser armazenada no repositório.

Para McClure (1997), um repositório é uma ferramenta para definição, armazenamento, acesso, e gerência da informação que descreve um empreendimento e seus sistemas de software, durante cada fase do ciclo de vida de software.

O repositório de componentes de software tem a finalidade de centralizar, em um único lugar, os componentes de software desenvolvidos para reutilização e, também, a informação e os artefatos de software relacionados a ele.

No contexto deste trabalho, repositório de componentes de software foi definido como uma base de dados, que deve permitir classificação e armazenamento dos componentes de software reutilizáveis e da informação relacionada ao componente, e disponibilizar recursos para permitir a busca e a recuperação do componente e da informação relacionada ao componente de software, tais como: especificações, modelos relacionados, código, planos e casos de teste, auxiliando o processo de entendimento do componente de software.

Uma ferramenta como a FARCSoft, que oferece recursos para manipular o repositório de componentes de software, é utilizada tanto por produtores de bens reutilizáveis, que tem um recurso para divulgar e compartilhar seus produtos de software com outros projetistas, quanto por consumidores de bens reutilizáveis, que centraliza o conhecimento relacionado aos bens reutilizáveis. Além disso, auxilia o

projetista na atividade de gerência do repositório, disponibilizando os recursos para catalogação, classificação e divulgação dos bens contidos no repositório.

1.2 Justificativa

Durante as fases do processo de desenvolvimento de software, uma grande quantidade de esforço, em geral, é desperdiçada. Pode-se observar que alguns artefatos de software gerados, durante as fases do processo de desenvolvimento (especificações, arquitetura, projeto, ou parte de código), são similares ou até mesmo iguais, a outros confeccionados em outros projetos.

Estes artefatos de software desenvolvidos para certos projetos, podem ser utilizados em novos projetos, com o intuito de diminuir o retrabalho realizado durante o processo de desenvolvimento de um produto de software.

Como exemplo, pode-se observar que a similaridade encontrada entre códigos de uma aplicação é muito grande. Os dados de um estudo relativos à reutilização de código fonte mostram o seguinte (TRACZ, 1986):

- 40% a 60% de todo o código são reutilizáveis de uma aplicação para outra;
- 60% de projeto e código sobre aplicações de negócio são reutilizáveis;
- 75% de funções de programa são comuns a mais de um programa;
- 15% de código encontrado na maioria dos programas são únicos e novos a uma aplicação específica.

Segundo os resultados da pesquisa realizada em 29 organizações por Frakes e Fox (1993) apud (LIM, 1998), onde foi apresentado o percentual de subprodutos de software gerados durante o processo de desenvolvimento de software, e que consistiam de partes reutilizáveis. Os dados deste estudo indicaram que código e a documentação do usuário eram os bens que, na média, tinham o mais alto percentual de partes reutilizáveis nestas organizações pesquisadas.

A Tabela 1 apresenta os resultados da pesquisa sobre os dados levantados, referentes à porcentagem dos subprodutos reutilizáveis, em cada fase do processo de desenvolvimento de software.

Percentual dos subprodutos que consistem de partes reutilizáveis	
Subproduto de Software	**Média**
Requisitos	20,8 %
Projeto	26,1%
Código	33,0%
Planos de Teste	26,1%
Casos de Teste	24,8%
Documentação do Usuário	30,4%

Tabela 1: Percentual dos subprodutos do processo de desenvolvimento de software que consistem de partes reutilizáveis (FRAKES; FOX, 1993) apud (LIM, 1998, p. 9).

Vários autores argumentam que a melhoria da produtividade do programador, através da utilização de componentes, está intimamente relacionada com a implementação da reutilização (BASS et al., 2001). Neste contexto, Willians (1999) apud (BASS et al., 2001) apresenta os resultados de um estudo do Gartner Group, a respeito de empresas que adotaram o desenvolvimento de software baseado em componentes, as quais obtiveram uma melhoria de 50% nos custos de desenvolvimento.

Ainda, segundo outros autores, os três conceitos, que promovem a reutilização de componentes de software, são (STETS; HUNT; SCOTT, 1999):

- Independência: permite a composição de componentes de software sem a introdução de interações implícitas, que podem provocar erros sutis de programação;
- Polimorfismo: permite que componentes de software possam ser substituídos transparentemente em um sistema, mesmo com diferentes implementações da mesma interface;

- *Late Binding*: permite, a uma aplicação, escolher os componentes de software dinamicamente, durante a execução.

Considerando estes resultados, pode-se observar que a combinação de reutilização de software e da tecnologia de componentes de software, é uma alternativa para alcançar a produtividade na elaboração de um produto de software com qualidade, visando custos mais baixos no processo de desenvolvimento. Desta forma, a reutilização de software e a tecnologia de componentes de software tornam-se ferramentas importantes para o aproveitamento das partes similares dos artefatos em componentes de software reutilizáveis, os quais podem ser armazenados em um repositório.

Vários pesquisadores apontam a necessidade de ferramentas que forneçam suporte ao processo de reutilização de software [(MCCLURE, 1997), (LIM, 1998), (IEEE Std 1517-1999), (GUO; LUQI, 2000), (PRESSMAN, 2000), (GRISS, 2001)]. Um repositório de bens de software reutilizáveis é uma destas ferramentas, e a principal razão para a existência de um repositório é tornar disponíveis os componentes de software, permitindo a divulgação e o compartilhamento e possibilitando o acesso dos projetistas que buscam candidatos potenciais à reutilização em seus projetos.

Para que a utilização do repositório de componentes seja eficiente, é necessário estabelecer um processo de desenvolvimento adequado. Basicamente, os processos de reutilização de software apresentados na literatura possuem três atividades comuns que são as seguintes: seleção do artefato para reutilização, adaptação para o objetivo da aplicação e integração ao produto de software em desenvolvimento (BASILI; BRIAND; MELO, 1996).

No entanto, para obter vantagens reais com a reutilização de software, é necessário disponibilizar os mecanismos para auxiliar o projetista nas atividades de identificação, seleção e entendimento do artefato reutilizável.

Além disso, devem ser disponibilizados recursos não só para o consumo de componentes reutilizáveis, mas também para a produção de componentes reutilizáveis e para a gerencia do repositório e das informações contidas nele [(MCCLURE, 1997), (LIM, 1998), (APPERLY, 2001)]. A Figura 1 mostra esquematização destas atividades.

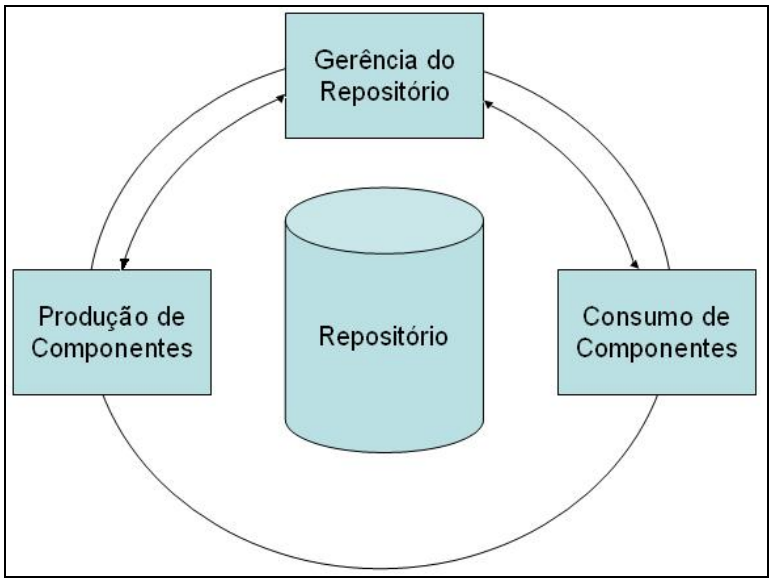

Figura 1:Consumo, produção e gerência dos componentes (APPERLY, 2001, p. 21).

Desta forma, um recurso importante para apoiar o processo da reutilização de software é uma ferramenta para gerenciar, buscar e recuperar os componentes de software armazenados em um repositório, possibilitando, desta forma, o compartilhamento e a difusão destes bens reutilizáveis. Além disso, deve disponibilizar recursos para auxiliar na tarefa de busca por componentes e compreensão da sua funcionalidade, para avaliar se é adequado à necessidade de um potencial reuso por um projetista e de como usar este componente para auxiliar no entendimento do componente de software reutilizável pelo projetista.

Para isso, é importante estudar e analisar os mecanismos de classificação de informação dos componentes de software em um repositório, as estratégias de busca

e recuperação dos mesmos e o conjunto de informação necessária para representar e documentar o componente de software dentro do repositório.

Portanto, pode-se concluir que um ambiente do tipo da FARCSoft é relevante para apoiar o processo de reutilização de componente de software, uma vez que disponibiliza uma base única de armazenamento e compartilhada para os componentes de software reutilizáveis; além disso, possibilita a busca e recuperação da informação relacionada ao componente de software.

A definição do meta-modelo do repositório de componentes reutilizáveis é a parte fundamental para definir a estrutura de representação e de documentação, da informação que descreve o componente de software.

1.3 Estrutura do Trabalho

Este trabalho está estruturado do seguinte modo.

O Capítulo 2 apresenta os principais aspectos relacionados à reutilização de software e à tecnologia de componentes de software. Para isso, são apresentados o histórico, os conceitos e as definições sobre estes dois assuntos. Além disso, são apresentadas as vantagens e desvantagens da reutilização de componentes de software.

O capítulo 3 apresenta o processo de desenvolvimento de software baseado em componentes, abordando o processo de reutilização de componentes de software, as fases do processo de desenvolvimento baseado em componentes e exemplos de metodologias. E, como conclusão do capítulo, são apresentadas as considerações sobre o impacto da utilização do processo de desenvolvimento de software baseado em componentes na produção de sistemas.

O Capítulo 4 apresenta a descrição da FARCSoft (**F**erramenta de **A**póio à **R**eutilização de **C**omponentes de **Soft**ware), o processo adotado para o seu desenvolvimento e as soluções de arquitetura.

O objetivo do Capítulo 5 é apresentar o estudo de caso que foi realizado para avaliar a capacidade de representação e de documentação do conjunto de informação relacionada ao componente de software. Para isso, foi selecionado um conjunto de diferentes tipos de componentes de software, complexidade, porte e tecnologia para serem representados, a fim de descrever a sua representação para a avaliação da capacidade do repositório de catalogar estes componentes.

O Capítulo 6 apresenta as conclusões, as contribuições e os trabalhos futuros.

2. REUTILIZAÇÃO DE COMPONENTES DE SOFTWARE

Este capítulo tem o intuito de apresentar os principais aspectos relativos à reutilização de software e à tecnologia de componentes, para constituir a base para a reutilização de componentes de software.

Desta forma, inicialmente, é apresentado um histórico sobre a reutilização de software que acaba incluindo a tecnologia de componentes. A seguir, são apresentados os conceitos, as definições e os aspectos relevantes sobre a reutilização de software e a tecnologia de componentes de software.

2.1 Histórico

A solução de um problema, muitas vezes, é baseada na aplicação de uma solução similar que já foi desenvolvida para resolver um outro problema. Em alguns casos, a solução similar deve ser adaptada para ser adequada ao novo problema.

O mesmo raciocínio pode ser aplicado na engenharia de software, para solução de um problema computacional. Desta forma, partes de software já desenvolvidas podem ser utilizadas novamente para auxiliar na solução de um novo problema. Segundo Prieto-Díaz (1993), a reutilização de software tem ocorrido desde que a programação foi inventada.

Em 1949, o uso de sub-rotinas e de biblioteca de sub-rotinas já era realizado pelo EDSAC na Universidade de Cambrigde, sendo que, por volta de 1950, existiam mais de 200 sub-rotinas armazenadas na biblioteca do EDSAC (LIM, 1998). Desta forma, pode-se notar que a reutilização de código-fonte já ocorria na década de 50.

O conceito formal de reutilização de software foi introduzido por Dough McIlroy (1968) apud (KRUEGER, 1992), em uma conferência da Organização do Tratado do Atlântico Norte. McIlroy propôs uma indústria de componentes de prateleira, componentes de código-fonte padronizados, vislumbrando a utilização de pequenos blocos de construção, disponíveis através de catálogos, para a construção de sistemas complexos.

Segundo Lim (1998), a idéia sobre reutilização de software apresentada por McIlroy foi fundamental para o conceito de fábrica de software, sendo que a versão de fábrica de software implantada pela Hitachi, em 1969, foi a primeira tentativa de integração e automação do processo de desenvolvimento de sistemas, utilizando o compartilhamento de processos e subsistemas padronizados.

Durante a década de 70, algumas empresas se empenharam em aumentar o nível de produtividade de seus processos de desenvolvimento, adotando os conceitos modernos de engenharia e a abordagem de fábrica de software.

Como referência das técnicas deste período, pode-se citar o conceito de modularização de programas apresentado por Dijkstra (1968) apud (CLEMENTS, 1995) e o de informação escondida, introduzido por Parnas (1972) apud (CLEMENTS, 1995). Dijkstra propunha o princípio da divisão de problemas, ou seja, a divisão de um sistema em partes (subsistemas) que poderiam ser desenvolvidas independentemente. Parnas propunha a decomposição de um sistema em partes que escondiam detalhes de implementação, enfatizando a importância de interfaces, para que um subsistema pudesse ser trocado por um outro diferente que disponibilizasse a mesma interface. Estas técnicas apontaram o caminho para a tecnologia de componentes de software.

Em 1974, a *System Development Corporation* registrou o termo fábrica de software. A proposta desta empresa estabelecia um conjunto de procedimentos bem definidos para um processo de software repetível e consistente, porém a reutilização de software era implícita (CUSUMANO,1991) apud (PRIETO-DIAZ, 1993).

Neste mesmo ano a NEC Fuchu enfatizou os conceitos modernos de engenharia de operação de fábrica, tais como processos padronizados, controle de qualidade, reutilização e ferramentas automatizadas, como meio de aumentar a produtividade no desenvolvimento de sistemas de software. Ainda por volta de 1977, a Toshiba estabeleceu uma fábrica de software para a criação de software de controle de tempo-real para aplicações industriais, enfatizando a reutilização de componentes (LIM, 1998).

O primeiro relatório de pesquisa sobre a reutilização de software, que apresentava resultados relevantes, foi divulgado por Lanergan (1979) apud (ISODA, 1994). A *Raytheon Company* implementou com sucesso um projeto de reutilização de software, onde foi obtido de 15% a 80% de taxa de reutilização no desenvolvimento de aplicações de negócio em COBOL.

Nesta época, a reutilização de software era focada no código-fonte. Desta forma, os componentes de software, também denominados de subsistemas, eram considerados como fragmentos de código que poderiam ser empacotados em sub-rotinas e executados. Certos autores consideraram dois níveis de produção de software, sendo o de pequenos componentes ou subsistemas de programação, e de grandes componentes conectando estes subsistemas ou componentes menores (DE RENER et al., 1976) apud (HALL, 1999).

O projeto DRACO, no final da década de 70, enfocava os componentes nos vários níveis de abstração. A partir desta visão, o componente emergiu como um pacote logicamente ligado de código de programa, no qual existiam interfaces bem definidas disponibilizadas pelo componente para distribuir suas funções para o software que o utilizava (HALL, 1999).

Durante a década de 80, a reutilização de software teve um grande apelo na discussão sobre o aumento da produtividade do processo de desenvolvimento de sistemas. A reutilização de software passou a ser considerada chave para alcançar produtividade e qualidade no desenvolvimento de sistemas [(TRACZ, 1986), (BIGGERSTAFF;

RICHTER, 1987), (LENZ; SCHMID; WOLF, 1987), (PRIETO-DÍAZ; JONES, 1987), (BARNES et al., 1988)].

Neste período, vários trabalhos foram divulgados, tais como, o trabalho do *European Software Program for Research on Technology* – ESPRIT, em 1984. Através do ESPRIT foram criados os programas para incentivo da reutilização de software, que foram (PRIETO-DÍAZ, 1993):

- PCTE – *Portable Common Tools and Enviroments*: pesquisa sobre ferramentas de reutilização;
- REBOOT – *Reused Based en Object Oriented Techniques*: fornecimento de métodos e ferramentas para dar suporte à criação e ao uso de componentes orientados ao domínio.

Ao final deste período, as pesquisas realizadas tiveram maior intensidade nas áreas direcionadas aos sistemas de bibliotecas, às técnicas de classificação, criação e distribuição de componentes reutilizáveis, aos ambientes de suporte à reutilização e aos programas de reutilização corporativos [(BURTON et al., 1987); (LENZ; SCHMID; WOLF, 1987); (PRIETO-DÍAZ; FREEMAN, 1987); (MAAREK; SMADJA, 1989); (FRAKES; GANDEL, 1990); (DEVANBU et al., 1991)].

Este crescimento ocorreu, mesmo com os resultados que demonstravam que a reutilização não estava cumprindo a promessa de aumento significativo na produtividade e qualidade.

Segundo Garlan, Allen e Ockerbloom (1995), a causa disto foi que, durante a década de 80, houve um aumento do uso das abordagens composicionais para software, que realizam a construção de sistemas de software a partir de partes pré-existentes. Entretanto, a construção sistemática de aplicações de software em larga escala, através deste tipo de abordagem, não apresentou o resultado esperado.

Segundo estes autores, isto ocorreu devido à falta de partes reutilizáveis e à inabilidade para localizá-las (quando existiam), como também devido aos problemas

de incompatibilidade entre as arquiteturas das partes escolhidas (incompatibilidade na linguagem de programação, nas plataformas operacionais, ou nos esquemas de banco de dados) (GARLAN; ALLEN; OCKERBLOOM, 1995).

Em 1986, Cox (1986) apud (HALL, 1999), apresentou uma visão na qual os componentes de software eram denominados de circuitos integrados de software. Nesta visão, a reutilização de software e a tecnologia orientada a objetos estavam direcionadas sobre a reutilização de componentes de software, que eram as classes e os objetos.

Esta visão de componente de software reforça a idéia de que componente de software está associado aos conceitos de:

- Princípio de separação de responsabilidades: partes de programas que poderiam ser desenvolvidas independentemente (DIJKSTRA, 1968) apud (CLEMENTS, 1995);
- Conceito de informação escondida: decomposição de um sistema em partes que escondem detalhes de implementação atrás de interfaces (PARNAS, 1972) apud (CLEMENTS, 1995).

No final da década de 80 e início da década de 90, trabalhos apresentados por Booch (1987) e Meyer (1994) apresentavam os componentes de software utilizando a tecnologia de orientação a objetos.

O conceito de reutilização de software foi ampliado por Basili e Rombach (1988) apud (PIETRO-DÍAZ, 1993), que definiram como sendo “utilização de qualquer coisa associada com um projeto de software, inclusive conhecimento”. Desta forma, esta nova perspectiva permitiu a realização de pesquisas de outras formas de reutilização e não somente do código-fonte.

A partir desta definição, o componente de software passou a ser tratado como uma coleção de artefatos, que disponibilizam, para organização do projeto, tudo que é

necessário para integrar o componente a um sistema de aplicação e fornecer suporte durante o ciclo de vida para este sistema.

A combinação da reutilização de software e de técnicas de projeto e implementação orientados a objetos tornou-se o foco da pesquisa acadêmica e industrial por muitos anos. Neste contexto, os reutilizáveis eram as classes e os objetos que, através dos conceitos de herança e especialização, poderiam ser adaptados [(MEYER, 1987), (COX, 1990), (KRUEGER, 1992), (CHENG, 1993)].

Durante década de 90, também existia a preocupação da maneira como as organizações implementavam a reutilização de software. Muitos pesquisadores afirmavam que a melhoria na produtividade e qualidade do software seria obtida com a implementação da reutilização de modo sistemático e planejada [(PRIETO-DIAZ, 1993), (CARD; COMER, 1994), (FRAKES; ISODA, 1994), (JONES, 1994), (MCCLURE, 1997), (JACOBSON; GRISS; JONSSON, 1997), (LIM, 1998)].

As pesquisas sobre as técnicas de classificação e recuperação de componentes de software em bibliotecas para reutilização foram consideradas como fatores chave para o sucesso dos projetos de reutilização de software [(PRIETRO-DÍAZ, 1993), (MILI; MILI; MILI, 1995), (DAMIANI; FUGINI; FUSASCHI, 1997), (HENNINGER, 1997)]. Durante a década de 90, como é apresentado por Guo e Luqi (2000), muitos trabalhos relacionados à pesquisa e ao projeto de bibliotecas de componentes de software, foram desenvolvidos.

Ainda na década de 90, foi desenvolvida uma outra forma de reutilização, a de padrões de projeto. No início dos anos 90, Erich Gamma, Richard Helm, Ralph Jonhson e Jonh Vlissides organizaram uma coleção de experiências de projeto, ou seja, padrões de projeto. Os padrões de projeto são descrições de objetos e classes comunicantes que são customizados para resolver um problema geral de projeto em um contexto particular (GAMMA et al., 1995).

Neste período, os trabalhos na área de componentes de software apresentavam foco na especificação formal de modelos de componentes, na interface de componentes de software e nos processos de certificação [(BATORY; O'MALLEY, 1992), (WOHLIN; RUNESON, 1994), (CLEMENTS, 1995), (STETS; HUNT; SCOTT, 1999)]. Além disso, segundo Lim (1998), a análise de domínio e engenharia de domínio tiveram um aumento em popularidade e têm sido importantes na identificação e criação de componentes de software reutilizáveis.

Em 1998, no *First International Workshop on CBSE (Component Based Software Engineering)*, a comunidade de software procurou um entendimento maior sobre desenvolvimento de software baseado em componentes, para identificar as brechas que existiam entre as necessidades da indústria e as pesquisas acadêmicas atuais.

Uma das discussões foi sobre a definição de componente. Segundo Brown e Wallnau (1998) foram apresentadas as seguintes definições sobre componentes:

- Componente: é uma parte não trivial, quase independente, e substituível de um sistema, que cumpre uma função clara dentro do contexto de uma arquitetura bem definida;
- Componente executável de software: é um pacote dinamicamente coeso de um ou mais programas gerenciados como uma unidade e acessados através de interfaces que podem ser encontradas em tempo de execução;
- Componente de software: é uma unidade de composição com interfaces especificadas em contrato e dependências explícitas de contexto. Um componente de software pode ter distribuição independente e é elemento para composição de partes de terceiros;
- Componente de negócio: representa a implementação em software de um conceito de negócio automatizado. Componente de negócio consiste de todos os artefatos de software necessários para expressar, implementar, e distribuir o conceito como um elemento reutilizável de um grande sistema de negócio.

Apesar de cada uma destas definições descreverem o mesmo conceito, elas revelam diferentes visões de componente de software durante o processo de desenvolvimento, distribuição e manutenção. Desta forma, cada uma destas definições ressalta características dos componentes de software que são alvos de muitas pesquisas, que são:

- Relacionamento entre componentes e tecnologia de objetos;
- Relacionamento entre componentes e arquitetura de software;
- Questões relacionadas com a interface de componentes;
- Questões relacionadas com a documentação de componentes;
- Questões relacionadas com a reutilização de componentes de software;
- Questões relacionadas com a especificação de contratos de componentes;
- Questões relacionadas com a composição de componentes.

Desta forma, durante a década de 90, várias pesquisas referentes ao desenvolvimento baseado em componentes, as tecnologias envolvidas e as formas de reutilização de componentes de software foram desenvolvidas.

Segundo Brown (2000), os avanços na tecnologia baseada em computador, nos últimos anos, têm levado a indústria de software a repensar a forma como o software é desenvolvido e buscar um suporte maior para reutilização de artefatos de software. Dentro destes avanços ele destaca os seguintes aspectos:

- Rápida evolução das tecnologias de hardware: resultou em uma melhoria na razão de preço/desempenho das tecnologias de computação. O impacto nas organizações foi um aumento na capacidade computacional em todos os níveis da organização
- Distribuição da informação: o acesso remoto à informação distribuída é mais barato para desenvolvimento e manutenção de aplicações distribuídas e possuem interface mais amigável com o usuário e desempenho melhor. Isto é conseqüência dos avanços nas tecnologias de suporte à arquitetura cliente-servidor, redes e gerência de dados distribuídos;

- WWW, Internet e Intranet: avanços e difusão desta tecnologia têm feito as organizações repensarem a forma como a informação é acessada e distribuída.

Desta forma, esta heterogeneidade de ambientes de sistema tem promovido a busca pela portabilidade, que é a capacidade de mover e operar uma aplicação de software sobre diferentes plataformas, e tem sido alvo de grande interesse e pesquisas. As pesquisas nos anos 90 focam sobre padrões que reforçam a reusabilidade [(EDDON, 1999), (SIEGEL, 1999), (BACHMAN et al., 2000)].

Um exemplo disto é o padrão CORBA (*Common Object Request Broker Architecture*), especificado pela OMG (*Object Management Group*), que permite a comunicação entre aplicações, não importando a localização ou quem tenha projetado as aplicações.

A versão Corba 1.0 foi introduzida pela OMG em 1991; também foi definida uma linguagem de interfaces, a Interface Definition Language (IDL), e um conjunto de interfaces, a Aplicattion Programming Interfaces (API), que disponibilizam a integração de objetos cliente/servidor dentro de uma implementação específica de um *Object Request Broker* (ORB).

A versão CORBA 2.0, disponibilizada em Dezembro de 1994, define uma interoperabilidade maior através da especificação de como os ORBs de diferentes fornecedores podem interoperar. A versão atual é a Corba 3.0.

Outros padrões como COM (*Component Object Model*) da Microsoft e EJB (*Enterprise Java Beans*) da Sun Microsystems, reforçaram a visão de componente de software como unidades substituíveis. Seguindo esta linha, Booch, Rumbaugh e Jacobson (1998) definem componente de software como sendo uma parte física e substituível de um sistema que obedece e fornece a realização de um conjunto de interfaces.

Na visão da UML, apresentada na versão 1.3, componente de software é definido como uma parte física e substituível de um sistema que empacota a implementação e fornece a realização de um conjunto de interfaces. Um componente de software representa a peça física de implementação de um sistema, incluindo o código (fonte, binário ou executável) ou equivalentes, tais como *scripts* ou arquivos de comando (UML, 1999).

No final da década de 90, diante desta diversidade de conceitos e definições relativas à tecnologia de componentes de software, o Software Engineering Institute (SEI) elaborou um estudo, onde a preocupação foi a de estabelecer um entendimento da tecnologia de componentes de software, através da avaliação a partir das perspectivas técnicas e de negócio, considerando as tendências da indústria [(BACHMAN *et al.*, 2000), (BASS *et al.*, 2001)].

Para o SEI, componente de software é uma implementação, em software, de alguma funcionalidade, que pode ser reutilizado sem a necessidade de alteração, em diferentes aplicações, e acessado através de um padrão de especificação de componente (BASS *et al.*, 2001).

Heineman e Council (2001) definem componente de software como sendo um elemento de software que está de acordo com um modelo de componentes e pode ser distribuído independentemente e integrado sem modificação, de acordo com um padrão de composição. O modelo de componentes define as interações específicas e os padrões de composição para o componente. A implementação de um modelo de componentes é o conjunto dedicado de elementos de software executáveis, requisitados para fornecer suporte para a execução de componentes que estão de acordo com o modelo.

Diante da grande variedade de definições, muitas pesquisas ainda buscam estabelecer um consenso sobre uma definição para componente de software, conceitos envolvidos, suporte necessário para adoção desta tecnologia, processos de desenvolvimento baseado em componentes e processo de reutilização de

componentes de software [(KOZACZYNSKI, 1999), (TAKATA, 1999), (YACOUB; AMMAR; MILI, 1999), (HEINEMAN; COUNCIL, 2001), (LÜER; ROSENBLUM, 2001)]. Este esforço tem sido justificado pois a reutilização de componentes de software tem sido vista como uma forma de atingir ganhos de produtividade, qualidade e redução de custos no desenvolvimento de um produto de software.

2.2 Reutilização de Software

No processo de desenvolvimento de software, a produtividade, a qualidade e os custos são fatores críticos para o sucesso de um produto de software [(FRAKES; ISODA, 1994), (HENRY; FALLER, 1995), (BASILI; BRIAND; MELO, 1996), (JACOBSON; GRISS; JONSSON, 1997)]. Desta forma, com o intuito de alcançar eficiência no processo de desenvolvimento, os projetistas buscam alternativas para atingir tais objetivos.

Para isso, Basili, Caldiera e Cantone (1992) sugerem três metas para aumentar a qualidade e a produtividade, que consistem de: melhorar a eficiência do processo, reduzir a quantidade de retrabalho e reutilizar os artefatos de software no ciclo de vida de desenvolvimento. Portanto, a implementação de um processo de reutilização de software de modo sistemático e planejado promove estas três metas pelo fato de estar reutilizando material consolidado.

Para aprofundar os conhecimentos sobre este tema, são abordados, nesta seção, os conceitos e as definições relacionadas à reutilização de software. Além disso, são apresentados os fatores relevantes para a reutilização, os tipos de reutilização, o processo de reutilização, as ferramentas necessárias para apoiar este processo, e as vantagens e as desvantagens obtidas na aplicação de reutilização de software no processo de desenvolvimento.

2.2.1 Conceitos e Definições

Na literatura sobre engenharia de software, existem muitas definições para o conceito de reutilização de software. Foram selecionadas aquelas que foram consideradas relevantes, para a apresentação nesta seção.

Basili, Caldiera e Cantone (1992) definem como sendo o uso de um mesmo objeto mais de uma vez, sendo que objeto, neste contexto, pode ser: programas, partes de programas, especificações, requisitos, arquiteturas, casos e planos de teste.

Krueger (1992), por sua vez, define a reutilização de software como o processo de criação de sistemas de software a partir de software existente, em vez de se construir tudo de novo. Já para Frakes e Terry (1996), a reutilização de software é o uso de artefatos existentes ou o conhecimento para criar um novo software.

McClure (1997) define reutilização de software como sendo o processo de construção ou montagem de sistemas de software a partir de componentes de software que foram projetados para reutilização.

Segundo Lim (1998), a reutilização de software é o uso de bens existentes no desenvolvimento de outro software, com o objetivo de melhorar a produtividade, a qualidade e outros fatores, como por exemplo, a usabilidade. Neste contexto, bens ou componentes são os produtos ou subprodutos do processo de desenvolvimento de software, incluindo tanto elementos tangíveis (código, projeto, algoritmos, planos de teste, e documentação) e elementos intangíveis (conhecimento e metodologia).

Desta forma, pode-se observar que o conceito sobre reutilização de software tem evoluído com o passar do tempo, juntamente com evolução da área de engenharia de software. Antes, na década de 60 e 70, a reutilização era de código-fonte e algoritmos para criar novos programas. A partir disso, o conceito de reutilização de software tem sido ampliado, incluindo os outros artefatos gerados durante o processo de desenvolvimento de software.

No contexto deste trabalho, reutilização de software é definida como o processo planejado e sistemático de construção de um novo sistema, através da utilização de componentes de software reutilizáveis contidos em um repositório.

Um outro conceito importante é o de reusabilidade que estabelece o grau para o qual um artefato de software pode ser utilizado em mais do que um sistema de software, ou na construção de outros artefatos de software (IEEE STD. 1517-1999, 1999). No caso de uma biblioteca de reutilização, reusabilidade corresponde às características de um artefato que o tornam mais fácil de ser utilizado em diferentes contextos, sistemas, ou na construção de diferentes artefatos.

2.2.2 Fatores Relevantes para a Reutilização de Software

Apesar da reutilização de software conceitualmente ser simples, a sua implementação em grande escala tem enfrentado dificuldades para atingir os resultados relativos ao aumento da produtividade, à melhoria da qualidade e à redução dos custos do processo de desenvolvimento.

Muitas vezes, as organizações têm a idéia equivocada de que a reutilização é um problema de aquisição de tecnologia [(CARD; COMER, 1994), (BOEHM, 1999)], o que não é realidade. A reutilização de software é um problema de transição de tecnologia, não bastando adquirir um sistema de biblioteca de componentes de software para fazer este processo ser implantado.

Para isso, outros aspectos devem ser considerados na implementação da reutilização no processo de desenvolvimento de sistemas, podendo se citar a necessidade de mudanças no próprio processo de desenvolvimento, para fornecer suporte e promover a reutilização de software. Além disso, devem ocorrer mudanças na estrutura da organização (novos papéis e responsabilidades) para que seja disponibilizada uma infraestrutura para permitir o desenvolvimento baseado em

reutilização. Todos os membros da organização devem estar comprometidos e envolvidos, visando este objetivo comum e dispostos a investir e colaborar.

Um conjunto de fatores deve ser superado pelas organizações que visam atingir os benefícios da implementação da reutilização de software de modo sistemático e em grande escala no processo de desenvolvimento de sistemas de software. Na literatura existe um consenso de que as maiores dificuldades são devidas a fatores de engenharia, de processo, organizacionais e econômicos [(CARD; COMER, 1994), (FRAKES; ISODA, 1994), (FAFCHAMPS, 1994), (JACOBSON; GRISS; JONSSON, 1997), (BOEHM, 1999), (MORISIO; TULLY; EZRAN, 2000)].

2.2.2.1 Fatores de Engenharia

Alguns autores apontam que os fatores de engenharia incluem tanto deficiências de tecnologia e como métodos, tais como [JACOBSON; GRISS; JONSSON, 1997]:

- Falta de meios para identificar claramente os elementos do modelo que descrevem requisitos, arquitetura, análise, projeto, implementação e teste, ao longo da cadeia de desenvolvimento;
- Falta de componentes de software para reutilização;
- Falta de flexibilidade nos potenciais componentes de software reutilizáveis;
- Falta de suporte de ferramentas para executar os procedimentos de reutilização.

As dificuldades na identificação de oportunidades de reutilização são decorrentes da falta de mecanismos que forneçam suporte aos ambientes orientados à reutilização. Desta forma, este tipo de ambientes deve ser adicionado ao processo de desenvolvimento, para guiar os projetistas de componentes nas atividades de (MCCLURE, 1997):

- Identificação das oportunidades de reutilização e dos componentes de software candidatos para reutilização;
- Construção de componentes de software para reutilização;
- Localização de componentes de software reutilizáveis para montagem de novas aplicações de software.

A falta de componentes de software para reutilização pode ser causada pelos seguintes fatores:

- Falha na seleção dos componentes para reutilização;
- Falta de técnicas para empacotar, documentar, classificar e identificar componentes;
- Projeto e implementação inadequada de sistemas de bibliotecas;
- Acesso deficiente aos sistemas de bibliotecas pelos projetistas.

Outro aspecto, importante neste contexto, é quanto à qualidade do componente a ser reutilizado (JONES, 1994). Para se obter sucesso na reutilização de software, é necessário que os componentes de software reutilizáveis sejam submetidos a processos de certificação, para garantir níveis de qualidade e de confiança satisfatórios.

Além disso, também é importante considerar a falta de flexibilidade do componente de software, ou seja, se o componente de software tem pouca capacidade de adaptação a uma nova necessidade ou a uma nova arquitetura, ele é adequado a poucas, ou até mesmo a nenhuma oportunidade de reutilização.

2.2.2.2 Fatores de Processo

Como foi dito no início da seção 2.2.2, para que a implementação da reutilização de software em uma organização tenha sucesso, devem ser considerados os aspectos relacionados ao seu processo de desenvolvimento de software.

O processo de desenvolvimento deve ser sólido, sistemático e com um nível de maturidade para que a organização obtenha as vantagens reais da reutilização [(CARD; COMER, 1994), (BOEHM, 1999), (MORISIO; TULLY; EZRAN, 2000)]. Desta forma, é possível assegurar que a oportunidade de reutilização seja considerada no momento apropriado, e que os artefatos gerados, ao longo do processo de desenvolvimento, provavelmente, tenham um nível de qualidade garantido.

Nos processos tradicionais de desenvolvimento de software, em cada projeto que se inicia, o software é desenvolvido de novo. Segundo McClure (1997), a maioria dos processos de desenvolvimento utilizados pelas organizações não incluem a reutilização de software, o que ocasiona os seguintes problemas:

- Falta da definição explícita da prática da reutilização de software dentro do processo de desenvolvimento de software (onde, quando e como);
- Falha de atribuição dos papéis e responsabilidades da equipe de projeto, usuários e gerentes para implementar a reutilização;
- Falta de definição das ferramentas necessárias para fornecer suporte à reutilização no processo de desenvolvimento.

Baseado nestes problemas, pode-se dizer que, para implementar a reutilização de software, são necessárias mudanças no modo como a organização desenvolve software, ou seja, mudanças no processo de desenvolvimento de sistemas de software da organização.

Alguns autores afirmam que o processo de desenvolvimento tradicional de software não é adequado para encorajar as oportunidades de reutilização de software,pois faltam as atividades em que o projetista possa analisar e identificar as possíveis oportunidades de reutilização (JACOBSON; GRISS; JONSSON, 1997).

Para permitir a reutilização de software, o processo de desenvolvimento deve possibilitar a identificação e a criação dos artefatos para reutilização; para isso, deve prever:

- Identificação dos componentes de software reutilizáveis em termos da arquitetura do software;
- Criação, empacotamento e armazenamento dos componentes de software de modo adequado;
- Remodelagem do processo de desenvolvimento de software, de tal modo que, os projetistas possam identificar as oportunidades de reutilização e que possam utilizar os componentes de software selecionados na construção de uma aplicação de software.

2.2.2.3 Fatores Organizacionais

A implementação da reutilização de software requer mudanças na estratégia e na estrutura da organização. A estratégia se refere ao modo como a reutilização é empregada para atingir os objetivos da organização. O termo estrutura, se refere ao sistema formal de relações de trabalho que permite a divisão e a coordenação das tarefas individuais e dos grupos para atingir os objetivos da reutilização de software (LIM, 1998).

Mudanças organizacionais na estratégia e na estrutura devem promover uma infraestrutura de desenvolvimento de software baseado em reutilização. Na literatura sobre este assunto, algumas questões culturais são apontadas como mecanismos para promover a reutilização [(CARD; COMER, 1994), (FRAKES; ISODA, 1994), (MORISIO; TULLY; EZRAN, 2000)]:

- Treinamento: o melhor método para promover a aceitação de uma nova tecnologia é treinar as pessoas para obterem sucesso com esta nova tecnologia;
- Incentivos: para obter engajamento na produção e consumo de componentes reutilizáveis, uma das formas é oferecer incentivos para os que aproveitarem as oportunidades de reutilização e produzirem

componentes reutilizáveis, e para os que consumirem estes componentes em projetos;

- Medição (avaliação): coleta de informação para realizar medidas para acompanhar o progresso e demonstrar a evolução é medida necessária para justificar o investimento e obter o comprometimento do nível gerencial;
- Comprometimento gerencial: para mudar um processo requer o apoio daqueles que são capazes de modificá-lo; somente com o compromisso da gerência é possível implementar a reutilização de software.

Alguns autores apontam a falta de conhecimento como sendo base das dificuldades organizacionais no processo de implementação da reutilização de software, tanto por parte dos gerentes em organizar a reutilização, como por parte dos projetistas em implementá-la (JACOBSON; GRISS; JONSSON, 1997).

2.2.2.4 Fatores Econômicos

A reutilização de software agrega um aumento nos custos do processo de desenvolvimento de software. Estes custos estão associados à criação, ao desenvolvimento e à manutenção de componentes de software reutilizáveis. Caper Jones (1994) afirma que a construção de software reutilizável é mais cara e consome mais tempo que a construção de software convencional; o software reutilizável custa 50% a mais e consome 30% a mais de tempo do que um software normal.

O retorno do investimento para construção de componentes de software reutilizáveis e criação de uma biblioteca para armazená-los só ocorre no momento em que, os projetos que utilizam os componentes de software reutilizáveis comecem a pagar por eles, ou seja (JACOBSON; GRISS; JONSSON, 1997):

- Um componente tem de ser utilizado de 3 a 5 vezes em projetos de aplicação, para recuperar o custo inicial para criá-lo e os custos existentes para mantê-lo;
- O custo de um componente reutilizável é de 1,5 a 3 vezes maior para criar e manter do que de um componente similar de uma aplicação sem considerar a reutilização;
- Os custos são 0,25 maior para utilizar um componente reutilizável do que desenvolver um novo;
- Os benefícios da reutilização levam de 2 a 3 ciclos de produto, usualmente em torno de 3 anos, para apresentar retorno significativo.

A reutilização de software requer o suporte da alta gerência para a implementação em uma organização, pois necessita de anos de investimento antes de dar um retorno e requer mudanças na estrutura gerencial e na fundação da organização. Desta forma, um fator que dificulta a implementação da reutilização é alocação de tempo e recursos financeiros, devido às pressões de custo e cronograma na maior parte dos projetos de uma organização.

A organização deve disponibilizar e financiar recursos para acesso a software reutilizável de tanto de desenvolvimento interno como fornecedores externos, e instituir os mecanismos necessários para fornecer suporte à reutilização de software.

2.2.3 Tipos de Reutilização

Pode-se dizer que o primeiro tipo de reutilização de software, foi a de código-fonte, considerando a idéia formal da reutilização de software introduzida por McIlory (1968) *apud* (KRUEGER, 1992), que propunha a industrialização do processo de construção de aplicações de software a partir de componentes de software de prateleira na forma de código-fonte.

Na visão atual, a reutilização pode ocorrer em vários momentos durante o processo de desenvolvimento de software, e existe a possibilidade de se reutilizar diferentes tipos de artefatos gerados, tais como: especificação de requisitos, arquitetura, código-fonte, dados, projetos, documentação, estimativas, interfaces com usuários, planos e casos de teste.

Desta forma, pode-se dizer que todos os artefatos usados e produzidos durante o desenvolvimento de software podem ser elementos de reutilização de software. Diferentes pesquisadores propuseram diferentes classificações para os tipos de reutilização, mas a maior parte destas classificações está relacionada a três fatores ou a combinação deles, que são (MILI; MILI; MILI, 1995):

- Estágio de desenvolvimento na qual o conhecimento é produzido e/ou utilizado;
- Nível de abstração (por exemplo, abstrato *versus* concreto/implementado);
- Natureza do conhecimento (por exemplo, artefatos *versus* conhecimento).

Pietro-Díaz (1993) apresenta uma classificação para os tipos de reutilização, onde estes tipos são divididos em pelo menos seis perspectivas ou facetas, que são: substância, escopo, modo, técnica, intenção e produto. É importante ressaltar que pode existir composição destas perspectivas dentro de um processo de reutilização.

A Tabela 2 apresenta as perspectivas propostas e os artefatos relacionados nesta classificação.

2.2.3.1 Perspectiva de Substância

A perspectiva de substância apresenta uma visão do tipo dos itens reutilizados, que podem ser idéias ou conceitos, artefatos ou componentes e procedimentos ou habilidades.

Tipos de Reutilização						
Perspectiva	**Artefatos**					
Substância	Idéias ou conceitos	Artefatos ou componentes		Procedimentos ou habilidades		
Escopo	Vertical		Horizontal			
Modo	Sistemático		Ad hoc			
Técnica	Composicional		Gerativa			
Intenção	Caixa preta		Caixa Branca			
Produto	Especificações	Arquiteturas	Projetos	Objetos	Textos	Código-fonte

Tabela 2: Perspectivas de Reutilização (PRIETO-DÍAZ, 1993, p. 62).

Entende-se por reutilização de idéias ou conceitos, a utilização de conceitos formais, como por exemplo, reutilização de soluções genéricas para uma classe de problemas. A reutilização de artefatos ou componentes se refere à reutilização de partes de um sistema ou de um projeto em um novo sistema. A reutilização de procedimentos ou habilidades é focada na reutilização da formalização e encapsulamento de procedimentos de desenvolvimento de software, ou seja, a criação de coleções de processos reutilizáveis que podem ser conectados para instanciar um novo e mais complicado processo.

2.2.3.2 Perspectiva de Escopo

A perspectiva de escopo estabelece a forma como a reutilização ocorre dentro de uma família de sistemas ou entre famílias de sistemas, ou seja, no domínio. A perspectiva do escopo pode ser do tipo vertical ou horizontal.

A reutilização vertical estabelece o quanto o reutilizável está dentro do mesmo domínio ou de uma área de aplicação. O objetivo, neste tipo de abordagem, é gerar modelos genéricos para famílias de sistemas, os quais podem ser utilizados como gabaritos na criação de novos sistemas (PRIETO-DÍAZ, 1993).

O principal foco, neste tipo de abordagem, é a análise e a modelagem do domínio, pois é através destas atividades que vai ser possível identificar os modelos genéricos para famílias de sistemas.

A reutilização horizontal reutiliza partes genéricas em diferentes aplicações, ou seja, o objeto de reuso pode ser utilizado em diferentes domínios. Os objetos de reuso, neste tipo de abordagem, têm um custo maior devido à necessidade de implementar a flexibilidade para permitir a sua reutilização em mais de um domínio (PRIETO-DÍAZ, 1993).

2.2.3.3 Perspectiva de Modo

A perspectiva de modo estabelece o grau de sistematização do reuso, que pode ser *ad hoc* e sistemático.

A reutilização *ad hoc* é o modo mais comum. A sua principal característica é a informalidade e os objetos de reuso são escolhidos de forma não sistemática. A reutilização é conduzida pelo indivíduo e não pelo projeto, e não há processos definidos para que ela ocorra.

Na reutilização sistemática, são definidos e seguidos as diretrizes e os procedimentos e, além disso, o desempenho é avaliado através de métricas. Existe a necessidade do comprometimento da gerência e um grande investimento deve ser feito para implantar este tipo de reutilização.

2.2.3.4 Perspectiva de Técnica

A perspectiva de técnica é relacionada aos diferentes métodos de implementação da reutilização, que podem ser gerativa ou composicional. A abordagem gerativa

consiste na reutilização no nível de especificação, através do emprego de geradores de código ou de aplicações. Assim, uma vez criados os mecanismos de geração, o esforço para criação de novas aplicações é reduzido.

A reutilização composicional é a utilização de componentes existentes como blocos para construção de um novo sistema. Prieto-Díaz (1993) define o uso de componentes existentes como blocos de construção para um novo sistema. O código-fonte é o artefato de reutilização mais comum neste tipo de abordagem.

2.2.3.5 Perspectiva de Intenção

A perspectiva de intenção refere-se ao aspecto de alterabilidade do artefato de reuso e pode ser caixa-branca e caixa-preta.

Entende-se, por reutilização caixa-branca, onde o componente reutilizado é modificado e adaptado. A principal dificuldade neste tipo de reutilização é o controle de alterações dos componentes reutilizáveis, que implica em um controle de versões mais formalizado.Uma forma de contornar esta dificuldade é tornar os componentes reutilizáveis mais genéricos, mas a conseqüência desta solução pode refletir em um aumento de complexidade dos componentes e na perda de eficiência da aplicação que os reutiliza, uma vez que um componente de software, para ser capaz de adaptar às novas necessidades ou arquitetura, exige algoritmos mais complexos e mais parâmetros de configuração.

Diferentemente, na reutilização caixa-preta, o componente é reutilizado sem que qualquer alteração seja feita. Deste modo, os reutilizáveis são encapsulados e suas funcionalidades são acessadas através de interfaces padronizadas. A principal vantagem neste tipo de reutilização é a garantia de maior qualidade e confiabilidade, porém o custo para a criação dos reutilizáveis é mais alto do que na reutilização caixa-branca, uma vez que a complexidade de construção deste componente é maior.

2.2.3.6 Perspectiva de Produto

A perspectiva de produto estabelece os produtos de um projeto que podem ser reutilizados. A relação de produtos reutilizáveis é a seguinte:

- Código-fonte: é a prática de reutilização mais comum;
- Projeto: utiliza as especificações para auxiliar na identificação das partes do projeto que podem ser reutilizadas. O emprego da técnica orientada a objetos promove este tipo de reutilização;
- Especificação: apresenta maior retorno e é utilizada pela reutilização gerativa;
- Objetos: permite a reutilização dentro de um mesmo sistema ou entre sistemas diferentes, devido aos conceitos de abstração, encapsulamento e hierarquia;
- Textos: permite a reutilização dos documentos do projeto de sistema;
- Arquitetura: é um elemento fundamental para composição de um sistema na abordagem composisional.

É importante ressaltar que, além dos produtos citados anteriormente, qualquer outro subproduto de software poderá ser utilizado, em princípio.

2.2.4 Processo de Reutilização

O processo de reutilização refere-se a um conjunto de fases que incluem métodos utilizados para produção, intermediação e consumo de componentes reutilizáveis para o desenvolvimento de sistemas de software (LIM, 1998).

Desta forma, uma organização que deseja promover a reutilização, deve entender seu processo de desenvolvimento para aplicar as mudanças necessárias para suportar incorporação das fases necessárias para a reutilização.

O processo de reutilização consiste de quatro fases principais, apresentadas na Figura 2 [(JACOBSON; GRISS; JONSSON, 1997), (LIM, 1998)]:

- Gerência da infra-estrutura de reutilização: planejar e executar os processos de reutilização; para isso, inclui um conjunto de atividades relacionadas ao planejamento, iniciação, fornecimento de suporte, monitoração e coordenação das outras fases do processo de reutilização;
- Produção de componentes reutilizáveis: identificar e fornecer os artefatos reutilizáveis apropriados para as necessidades dos projetistas de software;
- Intermediação (suporte) dos componentes reutilizáveis: apoiar a reutilização, através de qualificação e certificação, configuração, manutenção e divulgação dos componentes de software reutilizáveis;
- Consumo dos componentes reutilizáveis: utilizar a biblioteca e as ferramentas associadas, identificar e recuperar os componentes necessários, integrar os componentes no sistema que está sendo construído, fornecer retorno e requisitar novos componentes para responsável pela biblioteca de componentes.

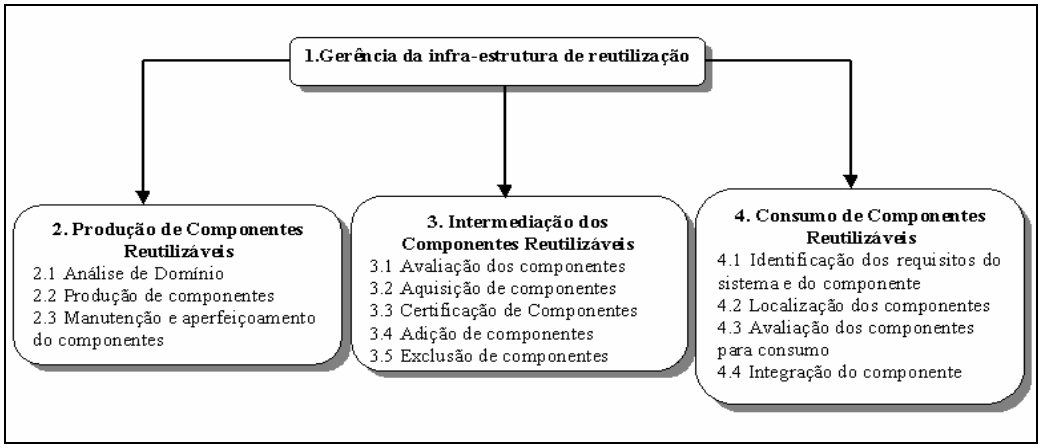

Figura 2: O processo de reutilização extraído de Lim (1998, p. 399).

Em cada uma destas fases são realizados os serviços para que a reutilização de software aconteça dentro do processo de desenvolvimento. Segundo Apperly (2001), a relação de serviços que devem ser oferecidos para fornecer suporte à gerência, produção e consumo de componente de software são os apresentados na Tabela 3.

Segmento	Serviços
Produção	Divulgar os componentes projetados informalmente ou não documentados
	Utilizar especificações de componentes com ponto de início para projeto
	Divulgar componentes com especificações
	Re-divulgar componentes e especificações
	Notificar consumidores de novos componentes ou problemas
Gerência	Gerenciar a biblioteca do repositório
	Gerenciar os usuários do repositório
	Gerenciar o catálogo
	Assegurar a qualidade dos componentes
	Gerenciar os componentes
	Tornar os componentes disponíveis
	Gerenciar as versões dos componentes
Consumo	Buscar componentes que necessita
	Reportar falhas identificadas
	Especificar componentes para desenvolvimento
	Utilizar e reutilizar especificações de componentes
	Utilizar e reutilizar componentes
	Dispor componentes
	Registrar interesse em componentes
	Receber novas notificações de componentes
	Revisar novos componentes
	Atualizar especificações de componentes e componentes

Tabela 3: Serviços de gerência de componentes segundo Applerly (2001, p. 515).

Pode-se observar que, através da tabela, a maioria dos serviços foca sobre os processos de divulgação, busca, e utilização dos componentes, não apenas na gerência estática dos artefatos.

2.2.5 Ferramentas de Apoio à Reutilização de Software

No contexto deste trabalho, ferramentas são recursos que tem a finalidade de apoiar a reutilização de software. Muitos autores afirmam que somente as ferramentas não são suficientes para assegurar o sucesso de um programa de reutilização, mas elas

facilitam a produção, difusão e consumo dos componentes de software reutilizáveis [(MCCLURE, 1997), (LIM,1998)].

São apresentadas, a seguir, as visões de alguns autores de destaque na área de reutilização.

Para Pressman (2000), uma relação de recursos deve fazer parte de um ambiente de apoio à reutilização de componentes de software, que são:

- Base de dados de componentes capaz de armazenar os componentes de software e classificar a informação necessária para recuperá-los;
- Sistema de gerência de biblioteca que forneça acesso à base de dados de componentes;
- Sistema de recuperação de componentes de software que possibilite uma aplicação cliente recuperar os componentes e os serviços a partir do servidor de biblioteca;
- Ferramentas de engenharia de software baseada em componentes que forneçam suporte à integração de componentes reutilizáveis em um novo projeto ou implementação.

Ainda segundo este autor, cada um destes recursos interage ou está incorporado a uma biblioteca de reutilização. A biblioteca é um elemento de um grande repositório, que fornece facilidades para armazenamento, busca e recuperação de componentes de software reutilizável e toda a informação relacionada a ele em uma base de dados.

Para Lim (1998), a biblioteca de software é definida como uma coleção de recursos de software e documentação relacionada, projetada para auxiliar no desenvolvimento, uso (reuso) e manutenção de software. Já o repositório de software é a estrutura da base de dados que armazena a coleção de bens de software, que deve estar disponibilizada permanentemente, na qual o bem de software é arquivado juntamente com sua a documentação relacionada.

Para McClure (1997), a biblioteca de reutilização fornece mecanismos para gerenciar os componentes reutilizáveis e os tornar disponíveis para os projetistas de sistemas de software de uma organização, e tem as seguintes funções principais:

- Organizar os componentes de software reutilizáveis;
- Armazenar os componentes de software reutilizáveis;
- Gerenciar os componentes de software reutilizáveis;
- Permitir a existência de múltiplas versões dos componentes de software reutilizáveis.

Desta forma, a biblioteca de reutilização é um elemento essencial para a formalização da prática da reutilização de software e, para isso, deve contemplar cinco tipos de recursos para fornecer suporte à criação e ao uso da biblioteca de reutilização, que são (MCCLURE, 1997):

- Repositório: ferramenta para definição, armazenamento, acesso, e gerência da informação que descreve um empreendimento e seus sistemas de software, durante cada fase do ciclo de vida do software;
- *Browser*: ferramenta para examinar o conteúdo de uma biblioteca para obter informação sobre qualquer componente contido na biblioteca, ou seja, auxilia na recuperação de componentes reutilizáveis e fornece uma variedade de diferentes visões dos componentes;
- Gerência de configuração: processo de identificação, organização e controle de modificações do software, isto é, gerência das mudanças do software durante o desenvolvimento, o desenvolvimento baseado em reutilização e a manutenção;
- Catálogo: ferramenta que pode ser utilizada para examinar uma biblioteca ou arquivo de um componente existente para extrair alguma informação descritiva a respeito do mesmo.

A Tabela 4 apresenta a descrição e a utilização de cada um destes recursos.

No contexto deste trabalho, é adotado o conceito proposto por McClure (1997), onde o repositório é um dos elementos de uma biblioteca de reutilização, ou seja,

repositório de componentes de software é uma base de dados, na qual as informações relacionadas aos componentes de software serão armazenadas de modo organizado, com a finalidade de permitir classificação, busca e recuperação da informação relacionada aos componentes de software.

Recursos de uma Biblioteca de Reutilização		
Recurso	**Descrição**	**Utilização**
Repositório	armazena componentes reutilizáveis	projetistas selecionam a partir do repositório componentes para reutilização e/ou adicionam novos componentes
Browser	examina o repositório para busca por componentes disponíveis	projetistas e gerentes utilizam para examinar o conteúdo da biblioteca
Gerência de Configuração	gerencia versões e configuração de software	projetista responsável pela biblioteca de reutilização utiliza para controlar versões dos componentes de software
Catálogo	armazena uma referência para componentes reutilizáveis que são armazenados em outro lugar	projetistas selecionam componentes para possível reutilização

Tabela 4: Recursos de uma biblioteca de reutilização segundo McClure (1997, p. 241).

A seguir, são apresentados alguns exemplos de bibliotecas de reutilização desenvolvidas por diferentes organizações (LIM, 1998):

- **STARS (Software Technology for Adaptable, Reliable Systems)**: Biblioteca de reutilização que disponibiliza funções para busca, fornecimento, avaliação e catálogo de componentes reutilizáveis, que incluem código, relatórios técnicos, requisitos e documentos de projeto;
- **Programa Alvey ECLIPSE – *Software Component Catalog***: Catálogo de componentes de software que tem a finalidade de armazenar e recuperar componentes de software;
- **GTE**: Biblioteca de reutilização e também fornece recursos para entendimento e adaptação de artefatos reutilizáveis;
- **NEC America**: Sistema de repositório de reutilização de software que armazena entidades reutilizáveis, que incluem subsistemas de software e documentos;

- **NASA Encyclopedia of Software Components (ESC)**: Sistema de catálogo e recuperação que utiliza hipertexto, hipergráficos e uma enciclopédia de metáforas;
- **Westinghouse**: ReuSE é uma ferramenta de gerenciamento de informação de bens com recursos de hipertexto, que são: submissão à biblioteca, administração, busca e recuperação;
- **Reuse SoftBoard**: Ferramenta que serve como uma biblioteca automatizada para bens reutilizáveis; inclui capacidade para importar, gerenciar, classificar, catalogar, consultar e exportar bens reutilizáveis.

Lim (1998) ainda apresenta outras ferramentas que auxiliam as atividades de produção, difusão e consumo de bens reutilizáveis, que são:

- ***Templates* de aplicação**: são modelos padronizados de software que servem como ajuda no desenvolvimento de software, que são comumente utilizados em partes;
- **CASE (Computer-Aided Software Engineering)**: é um sistema para automação de desenvolvimento de software. Fornece apoio para o ciclo de vida, permitindo que princípios gerais de engenharia de software sejam utilizados de modo prático e coordenado. O conceito fundamental, por trás das ferramentas CASE, é a utilização de bens reutilizáveis tais como, conhecimento do domínio e artefatos;
- **Geradores de aplicação**: ferramentas ou conjunto de ferramentas que recebem de entrada um conjunto de especificações e geram o código de uma aplicação através de uma linguagem de implementação. Desta forma, geradores de aplicação são pacotes de software utilizados para criação de sistemas ou componentes a partir de linguagens de alto nível, não-procedurais;
- **Orientação a objetos**: existe um consenso de que a orientação a objetos é uma abordagem de desenvolvimento que propicia a criação de componentes de software intercambiáveis e reutilizáveis (MILI; MILI; MILI, 1995). As propriedades da orientação a objetos que promovem a reutilização de

software incluem a herança, polimorfismo e encapsulamento (informação escondida) (LIM, 1998);

- **Sistemas parametrizados:** a aplicação adequada às necessidades do usuário é gerada a partir das respostas do usuário a um questionário, onde responde as questões por meio de “sim” ou “não” em um menu de escolhas;
- **Arquitetura de software**: é definida como *frameworks* e subsistemas de alta granulidade que capturam a estrutura global de um projeto de sistema de software (KRUEGER, 1992);
- **Esquemas de software**: constituem uma extensão para componentes de software reutilizáveis, mas a sua ênfase é sobre a reutilização de algoritmos abstratos e estruturas de dados em vez de reutilização de código;
- **Bibliotecas de sub-rotinas**: existem em duas formas. Em uma delas, a biblioteca contém um conjunto de funções (ou procedimentos) que são considerados de uso geral na área de computação. Na outra, as sub-rotinas da biblioteca podem ser combinadas para formular soluções para problemas em uma área única de aplicação;
- **Sistemas transformacionais**: transforma uma especificação de uma operação em alto nível, através de uma conversão incremental, em programas eficientes. Desta forma, um sistema transformacional produz um programa mais eficiente através de refinamentos sucessivos de cada função, onde podem ser introduzidas novas variáveis e estruturas de dados.

A IEEE *Societies*, através do comitê de coordenação de padrões da IEEE *Standards Association* elaborou um conjunto de documentos que descrevem padrões para a área de engenharia de software. Um destes padrões foi elaborado para o processo de reutilização, que é o IEEE Std 1517-1999 - IEEE *Standard for Information Technology—Software Life Cycle Processes -Reuse Processes*.

Este padrão disponibiliza um modelo para extensão do processo de ciclo de vida de software para incluir a prática sistemática da reutilização de software; para isso, especifica o processo, suas atividades, e suas tarefas para serem aplicadas durante cada fase de um ciclo de vida para permitir:

- A construção de um produto de software a partir de partes reutilizáveis;
- A identificação, construção, manutenção e gerência das partes reutilizáveis.

Desta forma, uma outra relação de ferramentas nesta linha pode ser encontrada no padrão Std 1517-1999. Estas ferramentas fornecem suporte à reutilização durante o processo de desenvolvimento de software que inclui a prática sistemática de reutilização. A relação destas ferramentas é apresentada na Tabela 5.

2.2.6 Métodos de Classificação, Armazenamento e Recuperação

Em uma biblioteca ou em um repositório, é necessário que os componentes de software sejam armazenados de modo que facilite sua busca, sua seleção e sua recuperação.

Desta forma, é importante discutir os métodos de classificação e recuperação da informação de reutilizáveis em um repositório ou biblioteca, já que são recursos essenciais para a implementação e aceitação do processo de reutilização. Para promover este entendimento foram analisados alguns métodos mais citados na literatura sobre reutilização de software, que são: métodos de biblioteca tradicional, métodos baseados no conhecimento e método de hipertexto.

2.2.6.1 Métodos Tradicionais de Classificação e Recuperação

Os métodos tradicionais de classificação e recuperação de informação são classificados de acordo com o tipo de vocabulário, conforme o apresentado na Figura 3. Segundo a taxonomia apresentada por Frakes e Gandel (1990), estes métodos podem ser de vocabulário controlado ou vocabulário não controlado.

Categoria	Tipo da Ferramenta	Funções executadas pelas ferramentas
Análise e Projeto	Análise e projeto de domínio orientado à reutilização	Assistir engenheiros de domínio a reconhecer similaridades entre os elementos do domínio e os elementos de teste de ajuste de modelos e arquiteturas existentes. Assistir projetistas a estender e melhorar seus inventários de modelos e arquiteturas de domínio.
	Análise de recuperação de bens legados	Analisar bens legados de modo a determinar padrões estruturais e funcionais de similaridade.
	Análise de requisitos de aplicações	Realizar cruzamento de requisitos para bens existentes de modo a minimizar diferença entre o que é disponível e o que é necessário.
	Projeto de aplicação orientado à reutilização	Validar a arquitetura de domínio selecionado de modo a apresentar aos projetistas uma lista de opções de instanciação de arquiteturas de componentes.
Construtores de Bens	Editores inteligentes	Encontrar bens apropriados e analisá-los gramaticalmente;desta forma, um projetista pode instanciá-los para um contexto particular.
	Geradores	Construir bens através da combinação de especificações de projeto, com informações de domínio contidas na ferramenta.
	Montadores	Construir bens através da combinação de especificações de projeto, com bens externos à ferramenta.
	Recondicionadores de bens legados	Empacotar padrões necessários em bens, extraídos através de ferramentas de análise de recuperação
Testadores de bens	Testadores de adaptabilidade	Assistir engenheiros de domínio na determinação e na melhoria da facilidade de reutilização de certos bens.
	Testadores de generalidade	Assistir engenheiros de domínio na determinação e na modificação do domínio de aplicabilidade de dados bens.
Gerência de Reutilização	Medidas de custo/benefício: - Reutilização de bens no ciclo de vida - Ciclo de vida da aplicação	- Determinar custos para gerenciar os mecanismos de armazenamento e recuperação de bens para a amortização dos bens sobre o tempo e sobre produtos de software. Determinar os custos relativos e os benefícios de possuir vários bens. - Determinar tempo e esforço de desenvolvimento de projeto e manutenção gasto/evitado devido à reutilização
	Configuração de bens e Gerência de versão	Monitorar o acesso aos bens necessários, as propriedades, e as responsabilidades pelo serviço, e quais versões dons bens aplicam-se para o produto de software.
	Análise de impacto de modificação de bens	Monitorar onde os bens são reutilizados, e as dependências entre os bens.
	Análise de inventário de bens	Determinar a ortogonalidade (duplicação e/ou sobreposição) do inventário, e a idade e a situação do inventário de itens.
	Catalogação de bens	Registrar formalmente os bens em vários mecanismos de armazenamento e recuperação, incluindo atualização das ferramentas de busca e recuperação com apropriados critérios de descrição e procura.
	Busca e recuperação de bens	Procurar e acessar os bens, possivelmente, permitindo os projetistas a informar determinados parâmetros de configuração tanto em tempo de construção e adaptação como em tempo de execução.
	Certificação de bens	Fornecer suporte à certificação de uma situação de um bem em termos de seu escopo.

Tabela 5: Categorias de ferramentas para fornecer suporte à reutilização segundo o IEEE STD 1517-1999 (1999, p. 41).

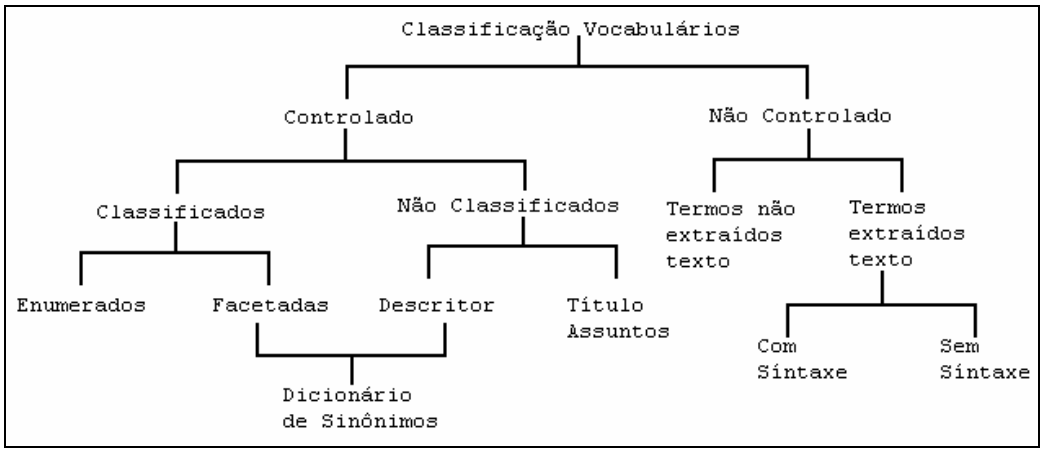

Figura 3: Taxonomia comum de vocabulários para classificação (FRAKES; GANDEL, 1990).

Entende-se por vocabulário controlado, quando os termos utilizados por indexadores e pesquisadores são restritos a um conjunto comum; deste modo, é assegurado que os termos usados pelos participantes dos projetos são os mesmos. Ao contrário, vocabulário não controlado ocorre quando pouca ou nenhuma restrição é imposta em relação à seleção ou sínteses de termos, e o relacionamento semântico entre termos não é limitado.

Os métodos de vocabulário controlado podem ser classificados ou não classificados. Os métodos classificados possuem sua estrutura apresentada através da forma enumerada e de facetas. Os métodos não classificados são menos estruturados e são baseados em palavras-chave que podem ser descritores e títulos-assunto.

Os métodos de vocabulário não controlado apresentam metodologias que permitem flexibilidade tanto na recuperação quanto na classificação, como no método de recuperação de texto-livre.

No método de classificação enumerada, um dado ou área de interesse é dividido em sucessivas classes mais restritas e assim, resultando em uma árvore de hierarquia estruturada (FRAKES; GANDEL, 1990). Pode-se citar, como exemplo de aplicação deste método, o sistema decimal de Dewey, que é um sistema de classificação de

livros ou outras publicações, onde as classes são designadas por um número de três dígitos e as subdivisões são mostradas por números depois do ponto decimal.

O método de classificação por facetas é baseado na análise de um assunto em função de seus termos básicos para a organização dos assuntos em facetas ou classes elementares básicas. Desta forma, o assunto em um domínio é descrito pela síntese destas classes ou facetas básicas. As facetas são consideradas como perspectivas, visões ou dimensões de um domínio de aplicação em particular (PRIETO-DÍAZ; FREEMAN, 1987).

O método de classificação em descritores utiliza listas, ordenadas alfabeticamente, de palavras simples que possam ser usadas para descrever a informação sobre um item. Já o método de classificação de título-assunto utiliza listas, ordenadas alfabeticamente, de termos em linguagem natural e frases que podem ser usadas para descrever a informação sobre um item.

Os métodos classificação em descritores e de título-assunto são muitos relacionados, pois, em ambos os casos, o vocabulário de termos é limitado a uma lista prescrita. A principal diferença entre os métodos é que o método de classificação de descritor tende a utilizar freqüentemente palavras simples, enquanto que, método de classificação de título-assunto utiliza frases.

O recurso de dicionários de sinônimos é um mecanismo para descrever a variedade de relacionamentos entre termos dentro de uma listagem alfabética, ou seja, sua finalidade é restringir vocabulário agrupando os termos que possuam o mesmo significado em um termo (FRAKES; GANDEL, 1990).

No método de não controlado ou análise de texto-livre, as palavras de texto são analisadas pela sua freqüência no texto e as palavras-chaves relevantes são derivadas automaticamente, através de suas propriedades estatísticas e posicionais. Deste modo, os resultados são índices automáticos constituídos de uma associação dos termos escolhidos para representar ou definir um item (PRIETO-DÍAZ, 1991).

Neste método, os índices podem ser definidos manualmente, ou podem ser extraídos a partir do texto descritivo do documento relacionado ao componente, removendo as palavras que pertencem a "*stop list*", tais como artigos, conjunções, pronomes e outros (e, a, o, um, uma, se, qual, que, por que, etc.), o usuário especifica a consulta usando palavras-chaves que são aplicadas aos índices para encontrar uma combinação com um documento (HENNINGER, 1997).

2.2.6.2 Métodos Baseados em Conhecimento e Hipertexto

Os métodos baseados em conhecimento apresentam adequação representacional e heurística poderosa. Como adequação representacional entende-se quanto um método pode expressar com a representação e, como heurística poderosa, entende-se que tipos de inferências o método pode fazer com a representação (FRAKES; GANDEL, 1990).

A seguir serão apresentados, resumidamente, os métodos para a reutilização mais relevantes, baseados em conhecimento, segundo (FRAKES; GANDEL, 1990), que são:

- **Redes Semânticas**: consistem de grafo direcionado no qual os nós correspondem aos objetos conceituais e os arcos correspondem aos relacionamentos entre os objetos. Este método tem boa adequação representacional e adequação heurística pobre;
- **Regras**: são regras de produção que consideram o formalismo para representação de conhecimento mais eficiente e é utilizado em muitas *shells* de sistemas especialistas. As regras de produção são usadas para classificar os componentes reutilizáveis baseados nos valores dos seus atributos;
- ***Frames***: são estruturas de dados, compostas de distribuidores e filtros, utilizados para representação de conhecimento.

O método de hipertexto é uma estrutura de representação que permite um usuário mover-se de um lugar para outro no corpo do texto através de *links* (FRAKES; GANDEL, 1990).

Alguns autores afirmam que, o uso apropriado da estrutura de representação por meio de hipertexto, pode melhorar a capacidade dos projetistas em encontrar e compreender componentes de software, uma vez que o hipertexto fornece meios para reunir os artefatos de software relacionados a um módulo, tal como o código de fonte, testes, documentação e notas do projeto, em uma entidade conceitual chamada de componente. Além disso, eles afirmam que o método de representação por meio de hipertexto pode ser utilizado com vários esquemas de classificação em uma biblioteca de reutilização (CREECH; FREEZE; GRISS, 1991).

2.3 Componentes de Software

O conceito de componente de software evoluiu desde a sua menção inicial por McIlroy (1968) apud (KRUEGER, 1992) como elemento importante de uma indústria que produzisse componentes de software em massa para fornecê-los para as organizações de construção de sistemas de software.

O componente de software, que na ocasião era parte de código-fonte, evoluiu conceitualmente, podendo ser um produto de qualquer fase do processo de desenvolvimento de software. No contexto deste trabalho, o componente de software pode ser desde um fragmento de código-fonte até um sistema completo, que pode ser apresentado tanto na forma de especificação ou projeto,como um código para ser integrado em um sistema.

Nesta seção, são discutidos os conceitos e as definições relacionadas com componente de software, modelo e *framework* de componentes, especificação de interface, contrato e padrões de documentação de componentes de software.

2.3.1 Definições e Conceitos

Na literatura sobre engenharia de software pode-se encontrar uma diversidade de definições de componente de software que foram sendo modificadas ao longo do tempo. Estas mudanças na definição, muitas vezes, ocorreram em função da tecnologia em evidência na época.

Conforme foi discutido na seção 2.1 (Histórico), as definições apresentadas ressaltam um determinado aspecto de componente de software e as diferenças entre as visões que eram significativas em cada época, o que mostra a evolução do conceito de componentes na área de engenharia de software. A partir destas diferentes definições podem-se extrair os conceitos em comum a respeito do que um componente deve ser, que são:

- Componente é um elemento de um sistema candidato à reutilização [(MCCLURE, 1997), (BASS *et al.*, 2001)];
- Componente descreve e cumpre uma função clara e bem definida [(ARNOLD, 1988) apud (TAKATA, 1999), (BROWN; WALLNAU, 1998)];
- Componente deve existir na forma de código (código-fonte, código binário ou executável) (UML, 1999);
- Componente está em conformidade com um conjunto de interfaces que deve ser bem definidas e públicas [(BROWN; WALLNAU, 1998), (UML, 1999)];
- Componente está inserido dentro de uma arquitetura de software, que direcione o processo de composição do componente em um sistema

[(BROWN; WALLNAU, 1998), (CHEESMAN; DANIELS, 2001), (HEINEMAN; COUNCIL, 2001)]

- Componente possui uma documentação adequada para auxiliar na busca, recuperação, adaptação e integração no seu novo ambiente e que ateste o seu grau de maturidade e confiabilidade [(MCCLURE, 1997), (KOZACZYNSKI, 1999), (YACOUB; AMMAR; MILI, 1999), (HEINEMAN; COUNCIL, 2001)];

O componente de software é um dos principais artefatos de software que pode ser utilizado e reutilizado para a construção de novos sistemas. A reutilização de software pode ser obtida por meio do desenvolvimento baseado em componentes, ou seja, um novo sistema pode ser construído utilizando partes (componentes) que foram criadas para compor um outro sistema.

Desta forma, um outro aspecto importante é a característica de um componente cumprir uma função clara e bem definida, o que ressalta o aspecto do componente ser uma estrutura modular e coesa que oferece um conjunto determinado de serviços estabelecidos em contrato. Isto significa que o componente de software encapsula os detalhes de como estes serviços são implementados e deve existir, para a sua utilização, um código, que pode ser fonte, binário executável, de acordo com o tipo de reutilização que é feita.

A estrutura modular do componente de software, encapsulando serviços que são disponibilizados através de interfaces, baseava-se inicialmente no princípio de separação de responsabilidade proposto por Dijkstra e no conceito de informação escondida proposto por Parnas, e teve sua evolução com a tecnologia orientada a objetos.

Os princípios que definem a tecnologia orientada a objetos são elementos importantes para a evolução do conceito de componentes de software, uma vez que estabelecem (MEYER, 1999):

- Informação escondida e abstração de dados: separação da implementação do componente a partir da interface do componente;
- Polimorfismo e ligação dinâmica: possibilidade da adaptação dinâmica do componente para as necessidades atuais do cliente;
- Projeto por contrato: garantia de que o componente é corretamente especificado e validado;
- Herança: organização dos componentes em hierarquias racionais.

Outros autores, como Cheesman e Daniels (2001), também ressaltam a aplicabilidade dos conceitos de que a tecnologia de orientação a objetos se baseia para o desenvolvimento de componentes de software:

- Unificação de dados e funções: um objeto de software consiste de valores de dados (estados) e funções que processam aqueles dados. Esta colocação natural de dependências entre funções e dados melhora a coesão;
- Encapsulamento: o cliente de um objeto de software é isolado da forma como os dados do objeto de software são armazenados ou como suas funções são implementadas, ou seja, o cliente depende da especificação do objeto, mas não de sua implementação. Esta separação da descrição da implementação é chave para a gerência de dependências entre os componentes de software e reduz o acoplamento;
- Identidade: cada objeto de software tem uma identidade única, indiferentemente do estado.

O conceito de interface de componente de software, decorrente do conceito de orientação a objetos, é um aspecto importante, uma vez que cada componente pode disponibilizar e solicitar serviços pré-definidos a partir de outros componentes.

A interface de componente é uma coleção de operações que são usadas para especificar os serviços disponibilizados por um componente (KRUCHTEN, 1998), ou seja, a interface tem a função de nomear a coleção de operações e especificar suas assinaturas e protocolos. A finalidade da interface é, portanto, apresentar o foco

sobre o comportamento do componente e não sobre a estrutura e a implementação das operações.

Desta forma, segundo Digres (1998), os componentes podem ser expressos em termos de interfaces e de semânticas externamente visíveis, e não da implementação, ou seja, as interfaces são mecanismos pelos quais os componentes se comunicam passando informações entre eles.

A arquitetura é um outro conceito que possui forte relação com a tecnologia de componentes de software. Segundo Stafford e Wolf (2001), a arquitetura de de software estabelece a disposição dos componentes que compõe o sistema, em uma ou mais estruturas definidas pelo papel funcional desempenhado pelos componentes, e também estabelece os relacionamentos de interação entre estes componentes.

A arquitetura de software define o conjunto de componentes de software que fazem parte do sistema, seus relacionamentos do ponto de vista estrutural e as dependências comportamentais.

É também importante ressaltar a relevância da documentação do componente de software para auxiliar o seu entendimento, pois fornece maiores facilidades para o seu uso na construção de sistemas baseados em componentes. A documentação do componente de software deve ser completa e precisa para auxiliar as atividades de seleção, comparação, análise e composição do componente de software.

O componente de software é um elemento de sistema, ou seja, uma seqüência de comandos de programa que descrevem o processamento a ser executado por uma máquina (HEINEMAN; COUNCILL, 2001), podendo estar na forma de código, pseudocódigo, ou até mesmo projeto. Além disso, a documentação pode ser resultado de qualquer método de construção do componente de software, desde os métodos mais tradicionais até os mais recentes da engenharia de software.

Considerando estas diferentes visões que existem na literatura de engenharia de software, é importante apresentar o conceito de componente de software no contexto deste trabalho.

Desta forma, o componente de software é definido como um elemento reutilizável de sistema que contém um artefato que oferece e executa a realização de um conjunto de serviços definidos e conhecidos e um conjunto de artefatos que disponibiliza a informação necessária para integrar o artefato, que pode ser executável ou não, para um sistema de aplicação, e fornecer suporte durante o ciclo de vida do sistema.

2.3.2 Modelo e Framework de Componentes de Software

Segundo Bachmann et al. (2000) não existe na literatura um entendimento único a respeito dos termos modelo e *framework* de componentes. Nesta seção são discutidas as definições mais relevantes da literatura.

Na visão do SEI (*Software Engineering Institute*), o modelo de componentes especifica os padrões e as convenções impostos sobre os projetistas de componentes. Desta forma, uma das propriedades de componentes que os distinguem de outros pacotes de software é eles estarem de acordo com o modelo de componentes.

A finalidade do modelo de componentes é especificar os padrões e convenções impostas aos projetistas de componentes, ou seja (BACHMANN et al., 2000):

- Composição uniforme: estabelecer que dois componentes podem interagir, se e somente se, eles compartilharem suposições consistentes a respeito do que cada componente fornece e necessita do outro, ou seja, qual função é executada pelo componente, como os componentes são localizados, como o fluxo de controle é sincronizado, qual protocolo de comunicação é utilizado, como os dados são codificados e assim por diante;

- Atributos de qualidade apropriados: estabelecer uma padronização dos tipos de componentes utilizados em um sistema, seus padrões de interação e estilo arquitetural, assegurando desta forma que a composição, a partir de componentes terceiros, irá atingir os atributos de qualidade desejados;
- Distribuição de componentes e aplicações: estabelecer uma pré-condição para composição, ou seja, quais componentes podem ser distribuídos a partir do ambiente do projetista para o ambiente de composição e quais aplicações construídas através da composição destes componentes podem ser distribuídas do ambiente de composição para o ambiente do cliente.

Da mesma forma, para Weinreich e Sametinger (2001), um modelo de componentes define um conjunto de padrões para implementação, nomeação, interoperabilidade, customização, evolução e disposição de componentes. Além disso, define também os padrões para modelo de implementação dos componentes associados e o conjunto de entidades de software executável, que é necessário para fornecer suporte à execução dos componentes que estão de acordo ao modelo.

Na Tabela 6 são apresentados os elementos básicos de um modelo de componentes segundo esta visão.

O conceito de *framework* de componentes, segundo o SEI, é a implementação de serviços que fornecem suporte ou fazem cumprir o modelo de componentes Desta forma, o *framework* gerencia os recursos compartilhados pelos componentes e fornece mecanismos básicos para permitir a comunicação (interação) entre eles (BACHMANN et al., 2000).

Vários autores fazem uma analogia entre *frameworks* de componentes com sistemas operacionais. Um *framework* é comparado a um mini sistema operacional, uma vez que, como um sistema operacional, é ativo e atua diretamente sobre um componente de modo a gerenciar o ciclo de vida de componentes ou outros recursos, como por exemplo iniciar, suspender, resumir, ou terminar a execução de um componente [(BACHMANN et al., 2000), (WEINREICH; SAMETINGER, 2001)].

Elementos Básicos de um Modelo de Componentes	
Padrões para	**Descrição**
Interfaces	Especificação do comportamento e propriedades do componente; definição de linguagem de descrição de interface (Interface Description Language - IDL)
Naming	Nomes únicos e globais para interfaces e componentes
Meta dados	Informação a respeito dos componentes, interfaces, e seus relacionamentos; APIs para serviços disponibilizar informação
Interoperabilidade	Comunicação e troca de dados entre componentes de diferentes fornecedores, implementados em linguagens diferentes
Customização	Interfaces para customização de componentes. Ferramentas de customização com interfaces gráficas amigáveis ao usuário que irão utilizar estas interfaces
Composição	Interfaces e regras para a combinação de componentes para criar estruturas grandes e para substituição e adição de componentes a estruturas existentes
Suporte a Evolução	Regras e serviços para substituição de componentes ou interfaces por versões mais novas
Empacotamento e Distribuição	Implementação e recursos de empacotamento e distribuição necessários para instalação e configuração de um componente

Tabela 6: Elementos básicos de um modelo de componentes (WEINREICH; SAMETINGER, 2001, p. 36).

2.3.3 Especificação de Componentes

Segundo Brown (2000), um componente necessita de uma descrição dos serviços que oferece para funcionar como um contrato entre os clientes e os fornecedores destes serviços. Para isso, a especificação de componente deve ser mais que uma lista das operações disponíveis e também deve descrever os seguintes aspectos:

- Comportamento esperado do componente em situações específicas;
- Estados permitidos do componente;
- Regras para o cliente interagir apropriadamente com o componente.

Para funcionar como um contrato entre clientes e fornecedores dos serviços de um componente, a especificação de componentes é definida como uma descrição de uma unidade de software que apresenta o comportamento de um conjunto de instâncias de um componente instalado e define uma unidade de implementação (CHEESMAN; DANIELS, 2001).

Desta forma, a especificação de componente forma um contrato mais abrangente envolvendo o realizador, implementador e o testador do componente, enquanto que a interface forma um contrato com o componente cliente. A Tabela 7 apresenta um resumo das diferenças entre interface e especificação de componentes (CHEESMAN; DANIELS, 2001).

Interface versus especificação de componentes	
Interface	**Especificação de componente**
Uma lista de operações	Uma lista de interfaces suportadas que descrevem o comportamento do componente
Definição de uma base lógica do modelo de informação	Definição dos relacionamentos entre o modelo de informação de diferentes interfaces
Representação do contrato com o cliente	Representação do contrato com o programador (implementador)
Especificação de como as operações afetam ou lidam sobre o modelo de informação	Representação da unidade de implementação e execução
Descrição dos efeitos locais somente	Especificação de como as operações devem ser implementadas em termos de uso de outras interfaces

Tabela 7: Comparação entre interface e especificação de componentes (CHEESMAN; DANIELS, 2001, p. 21).

2.3.4 Contratos

O conceito de contrato, de uma forma genérica, é definido como um conjunto de regras que especificam obrigações mútuas entre as partes que participam do mesmo. No caso de contrato de componentes, o seu conceito está relacionado a aspectos relativos à confiabilidade e ao comportamento inesperado no caso de falha ou erro.

Portanto, podem-se estabelecer algumas relações entre as características de componentes e contratos; no caso de componentes de software, as partes que participam do contrato são software (BACHMAN et al., 2000):

- Co-dependência: os contratos são entre duas ou mais partes;
- Composição: as partes freqüentemente negociam detalhes de um contrato antes de tornarem-se assinantes;
- Certificação: os contratos prescrevem regras e comportamento padronizados para todos os assinantes;
- Padronização: contratos não podem ser alterados a menos que todos os assinantes concordem.

Para Cheesman e Daniels (2001) existem dois tipos diferentes de contrato: o contrato de uso e o contrato de realização. O contrato de uso está relacionado com a execução, e o contrato de realização com o projeto. A Figura 4 apresenta as diferenças entre dois tipos de contrato.

O contrato de uso tem a finalidade de descrever o relacionamento entre a interface de componente e seus clientes e é especificado na forma de uma interface. Desta forma a especificação da uma interface deve incluir o seguinte (CHEESMAN; DANIELS, 2001):

- Operações: a lista de operações que a interface disponibiliza, tanto suas assinaturas e definições;
- Modelo de informação: a definição abstrata de qualquer informação ou estado que é retido entre pedidos de cliente por um objeto que fornece suporte a interface, e qualquer restrição daquela informação.

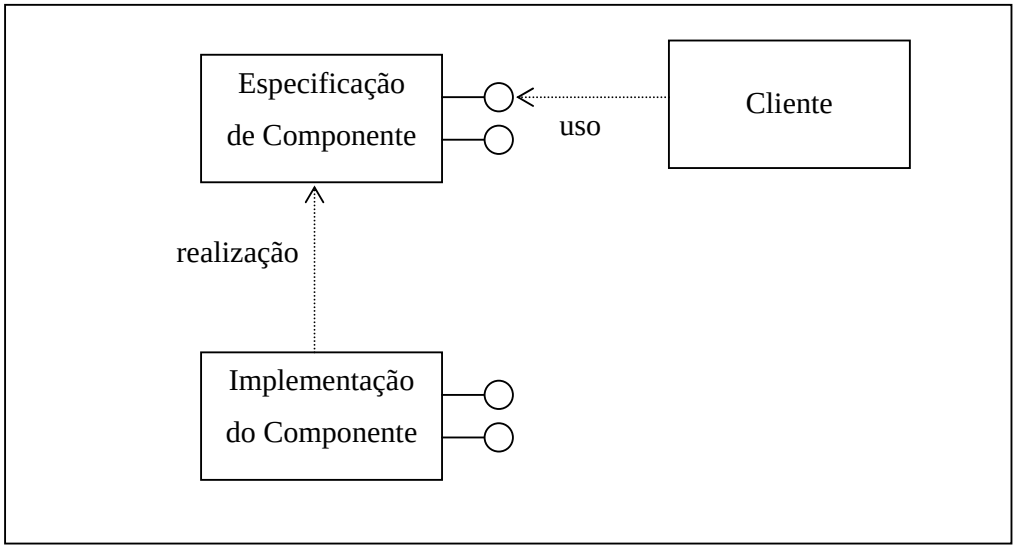

Figura 4: Diferentes tipos de contrato (CHEESMAN; DANIELS, 2001, p. 18).

Da mesma forma que uma assinatura, cada operação é definida através de pré-condição e pós-condições, onde:

- Pré-condição: uma definição das circunstâncias nas quais uma operação é válida, ou seja, descreve o que deve acontecer antes que a operação se inicie;
- Pós-condição: uma descrição dos efeitos daquela operação em seus parâmetros e o modelo de informação, ou seja, descreve o que deve acontecer quando a operação termina.

Segundo Cheesman e Daniels (2001), a utilização de pré e pós-condições não implica automaticamente um alto grau de precisão, mas especificações precisas são muito valiosas na definição de componentes que possam ser substituídos com um grau menor de dificuldade.

Através de descrição de pré-condições e pós-condições é possível especificar quais são as necessidades e restrições de uma operação antes de ser executada e quais os resultados gerados após sua execução. Este tipo de descrição auxilia na especificação de contratos entre componentes de software, uma vez que se pode especificar as

exigências de um componente para oferecer um serviço e os resultados que ele produz.

O contrato de realização é entre uma especificação de componentes e sua implementação, e deve ser seguido pela pessoa que está criando a implementação (realizador ou programador). Desta forma, o contrato de realização é incorporado na própria especificação de componentes (CHEESMAN; DANIELS, 2001).

2.3.5 Padrões de Documentação de Componentes

O projetista de software, para construir uma aplicação através da utilização de componentes de software reutilizáveis, necessita de identificar, entender e ter confiança nos atributos de qualidade do componente que está reutilizando.

McClure (1997) aponta, como principal inibidor da reutilização, o entendimento do componente de software. Para que a reutilização seja aceita pelo projetista, ele deve ser capaz de compreender o que o componente faz a partir do momento em que consulta um componente para reutilização. Desta forma, o conjunto de informação, que descreve o que o componente faz, suas propriedades, as necessidades de implementação e o ambiente em que opera, é extremamente importante para a auxiliar a reutilização de um componente.

Vários autores apontam diferentes propostas em relação ao conjunto de informação da representação do componente de software, para auxiliar no seu entendimento e para informar como integrá-lo na construção de um novo sistema de software.

McClure (1997) sugere seis tipos de documentação, na forma narrativa e/ou gráfica, que são necessárias para a documentação de um componente de software para a reutilização:

- Informação de especificação: apresentar o propósito do componente, ou seja, descrição sucinta da funcionalidade do componente, descrevendo o que o componente faz;
- Classificação: definir um esquema de classificação, para auxiliar o processo de busca e recuperação do componente de software a partir de uma biblioteca de reutilização;
- Informação declarativa: documentar as pré-condições e as pós-condições para utilização, as restrições de uso relacionadas ao componente, as asserções para correta especialização, modificação, adaptação ou instanciação do componente (restrições sobre o preenchimento de partes variáveis) e os eventos ou as condições que causam as mudanças de estado do componente;
- Informação de qualidade: incluir documentação que ateste o grau de qualidade do componente, tais como, documentação de métricas de complexidade, informações de confiabilidade e garantias que descrevam a política para determinação de responsabilidades para correção de erros e mecanismos para resolução de qualquer problema com o componente;
- Informação de reutilização: documentar o histórico de reutilização do componente representado por número de vezes que foi reutilizado, projetos e sistemas onde foi reutilizado, projetista que reutilizou, para estabelecer o grau de confiança do mesmo. Além disso, deve possuir regras de reutilização do componente, tais como, valores limites para parâmetros de substituição, opções de eficiência de desempenho, extensões de comportamento do componente, e também uma explicação de como usar, adaptar e integrar o componente reutilizável no sistema que está sendo construído;
- Documentação detalhada: documentar de modo detalhado o componente, de forma a apresentar a funcionalidade do componente, suas entradas e saídas, os dados de desempenho, os requisitos de interface (software e hardware), os algoritmos utilizados, os manuais de usuário, as decisões de projeto, os resultados, as questões relevantes (aspectos que surgiram durante o desenvolvimento, as alternativas que foram consideradas e as

> razões para escolha das alternativas), as exigências para a utilização do componente (por exemplo, DBMS, ferramentas CASE, linguagens de programação), as informações de criação (autor, data, projeto, domínio de aplicação), as limitações (por exemplo, licença e restrições legais), a documentação de testes, a responsabilidade pelo suporte e os relacionamentos do componente.

A proposta por McClure (1997) contém a informação que descreve um componente reutilizável apresentada por Sametinger (1996) em um artigo sobre documentação de componente reutilizável.

Em uma outra proposta, apresentada por Yacoub, Ammar e Mili (1999), existe um conjunto de características de um componente de software, que podem auxiliar no entendimento dos diferentes tipos de componentes, como eles são envolvidos no ciclo de vida de desenvolvimento, como eles são reutilizados e quais tecnologias são necessárias para adotar um componente no desenvolvimento de uma aplicação.

Segundo estes autores, as características de um componente podem ser classificadas em três categorias, que são: descrição informal, características exteriores e interiores (YACOUB; AMMAR; MILI, 1999).

A descrição informal apresenta a descrição resumida do componente de aspectos relacionados à funcionalidade, ao contexto que se aplica a sua solução e a que momento do processo de desenvolvimento ele deve ser utilizado. Nesta categoria o componente poderia ser considerado como blocos de construção de aplicação mentalmente dirigidos e, portanto, eles devem ter uma prescrição de solução para problemas de desenvolvimento de aplicação que aquele componente resolve, relacionando as características relativas a:

- Idade: reflete a estabilidade e maturidade;
- Origem: corresponde à origem de seu fornecimento;
- Nível de reutilização: reflete a fase do ciclo de vida de desenvolvimento em que o componente pode ser reutilizado;

- Contexto: reflete a situação na qual o componente pode ser reutilizado;
- Intenção: reflete a finalidade para o qual o componente é documentado;
- Componentes relacionados: indica os componentes que resolvem os mesmos problemas ou similares.

As características exteriores definem suas interações com outros artefatos e com plataformas sobre as quais o componente reside, compreendendo os aspectos relativos a:

- Interoperabilidade: estabelece como o componente interage com outros componentes no mesmo sistema;
- Portabilidade: reflete como o componente interage com a plataforma base -hardware, sistema operacional, sistema de comunicação;
- Papel: caracteriza o papel que identifica o que é fornecido para outros componentes da aplicação e o que exige. Pode ser ativo (afeta e é afetado pelos outros componentes) ou passivo (afetado ou acionado por outros componentes);
- Fase de integração: reflete o momento em que o componente é integrado em uma aplicação;
- *Framework*s de integração: especifica do *framework* base para a execução do componente;
- Tecnologia: reflete as dependências e restrições tecnológicas para o desenvolvimento e reutilização do componente em outra aplicação;
- Características não funcionais: requisitos de segurança, desempenho e de confiabilidade do componente.

As características interiores definem os aspectos internos que são importantes para o entendimento do componente, que são:

- Natureza: determina quando o componente pode ser utilizado no processo de desenvolvimento. O componente pode ser uma função, um dado, pacote ou uma estrutura de sistema, como também pode ser uma classe, um fragmento de código, um fragmento de projeto ou um módulo

executável; desta forma, pode ser classificado como componente de projeto, componente de especificação, executável e código;

- Granularidade: classificação baseada no tamanho e complexidade ou fase em que o componente pode ser reutilizado;
- Encapsulamento: um componente pode esconder uma ou mais decisões.A decisão pode ser uma decisão de projeto, de implementação ou de especificação;
- Aspectos estruturais: descrevem os participantes internos do componente e como eles colaboram; no caso de um componente projetado utilizando orientação a objetos, os participantes são as classes que o compõe;
- Aspectos comportamentais: descreve as respostas do componente em relação a um conjunto específico de entradas e as respostas do componente a uma seqüência específica de ações;
- Acessibilidade para o código fonte: especifica o tipo de reutilização, se caixa preta ou branca.

Outra proposta de documentação de componentes é apresentada por Houston e Norris (2001), que propõe aplicação de um modo padrão para documentar os componentes através da utilização da UML. Para eles, a falta de um padrão para modelagem de componentes de software afeta a capacidade de documentação de componentes de software de modo completo e consistente, o que atrapalha a sua reutilização do mesmo.

Para isso, segundo esta proposta, deve-se considerar dois aspectos separados para a modelagem de componentes, que são:

- Modelagem para a representação do projeto e da implementação de um componente;
- Modelagem para documentar componentes para reutilização.

Para a documentação de componentes são sugeridas regras através da inclusão do seguinte meta dado (HOUSTON; NORRIS, 2001):

- O que o componente é e faz : interfaces realizadas e funcionalidade fornecida. A descrição do componente deve permitir a combinação de um problema com a solução de modo fácil. Para isso, deve existir uma descrição clara do problema, do contexto e da solução;
- Características de execução: desempenho, tratamento de erros, concorrência (camada de segurança) e outras características devem ser claramente declaradas;
- Restrições ou limitações: descrição das limitações que invalidam o potencial de reutilização; por exemplo, o componente pode ser específico para um ambiente que não é mesmo da aplicação que irá reutilizá-lo ou pode exigir muita adaptação;
- Qualquer requisito sobre o que o componente necessita do ambiente para operar: dependências sobre componentes externos. Um componente importante é auto-contido enquanto que um componente superficial pode depender da necessidade de recursos que devem ser satisfeitas pelo ambiente para a sua operação adequada;
- Instruções para uso: texto descritivo, exemplos de diagramas de interação, de estados, ou de classes. Isto é necessário para que os usuários do componente possam integrá-lo apropriadamente com outros componentes. Este conjunto de instruções fornece uma medida de complexidade do componente;
- Como o componente executa suas responsabilidades: interfaces e realizações.

Além da descrição dos componentes, é necessário também descrever as interfaces, pois elas definem o contrato para uso dos componentes. Para isso, são definidas as regras para a sua documentação, contemplando os seguintes dados (HOUSTON; NORRIS, 2001):

- Nome da Interface: descreve de modo sucinto a funcionalidade que a interface oferece;

- Descrição da interface: descreve as responsabilidades da interface. A descrição não deve simplesmente declarar o nome da interface, mas deve esclarecer o papel que a interface tem para sistema;
- Definição da operação: deve refletir o resultado da operação. A definição da operação deve descrever o que a operação faz, incluindo algoritmos chaves e valores retornados, assim como o significado ou a semântica destes valores. Os nomes dos parâmetros da operação devem indicar qual informação é passada para a operação e também o tipo dos parâmetros;
- Documentação da interface: o comportamento definido pela interface é especificado como um conjunto de operações;
- Documentação de teste: deve incluir casos de teste e roteiros; os quais testam o comportamento de qualquer elemento do modelo realizando a interface.

Outra proposta baseada em UML é apresentada pela Rational Software (2001), que está desenvolvendo uma especificação para fornecer diretrizes para a descrição, desenvolvimento e aplicação de diferentes tipos de bens reutilizáveis.

A especificação é baseada nos conceitos da UML e do processo unificado da Rational (RUP - *Rational Unified Process*), e nos conceitos descritos no padrão de descrição de Arquitetura (ADS - *Architectural Description Standard*) [(RATIONAL SOFTWARE, 2000a), (RATIONAL SOFTWARE, 2000b)] apud (HOUSTON; NORRIS, 2001).

A Tabela 8 apresenta resumidamente a estrutura genérica para descrição de um bem reutilizável segundo esta especificação.

Elementos da especificação de um bem reutilizável		
Elementos	**Conteúdo**	**Descrição**
Resumo	Escopo/Motivação	Coleção de itens que descreve o objetivo e intenção do bem.
Classificação	Contexto	Classifica o bem dentro de um contexto em que ele se aplica.
	Domínio	Contexto que descreve a área de aplicação de negócio que o em atende.
	Desenvolvimento	Contexto que descreve o ambiente de desenvolvimento para o qual o bem foi projetado.
	Teste	Contexto que descreve as configurações de teste para o qual o bem deveria ser testado.
	Distribuição	Contexto que descreve o ambiente de execução e distribuição no qual o bem é aplicado.
Descritores	Perfil	Perfil para o qual o bem foi criado, exemplo: “componente”.
	Autor	Autor da versão específica do bem.
	Id	Identificação do bem.
	Usos conhecidos	Aplicações, sistemas, organizações que utilizaram o bem.
	Nome	Nome atribuído ao bem.
	Versão	Versão do bem.
Utilização	Atividades	Coleção de atividades descrevendo como aplicar os artefatos do bem.
	Instruções de utilização dos artefatos	Dado um artefato, conjunto de atividades específicas para sua utilização.
	Contexto das atividades	Dado um contexto referenciado, conjunto de atividades para aplicação deste bem.
Solução	Artefatos	Conjunto de artefatos que compõe a solução de um bem.
	Pontos de variabilidade	Descreve o conjunto de parâmetros sobre cada artefatos e que tipo de informação e dado necessita para ser fornecido pelo bem cliente para customizar o artefato.

Tabela 8: Elementos da especificação de um bem reutilizável proposto pela Rational Software (2001)

2.4 Vantagens e Desvantagens da Reutilização de Componentes de Software

Para a implementação da reutilização de software no processo de desenvolvimento, é importante considerar suas vantagens e suas desvantagens. Desta forma, são analisadas, inicialmente, as vantagens obtidas com a reutilização de software e, posteriormente, as desvantagens.

Na literatura sobre reutilização de software são apontados muitos aspectos vantajosos decorrentes da sua implementação no processo de desenvolvimento de software [(TRACZ, 1986), (JACOBSON; GRISS; JONSSON, 1997), (MCCLURE, 1997), (RINE, 1997), (LIM, 1998)]:

- Redução do custo de desenvolvimento;
- Melhoria da qualidade de software;
- Aumento da produtividade de software;
- Melhoria da interoperabilidade entre sistemas de software;
- Melhoria do *time-to-market*, ou seja, o tempo desde o momento em que o produto é concebido até a sua entrega;
- Compartilhamento do conhecimento dos sistemas e da forma de construção;
- Compartilhamento dos componentes de software de modo mais formal.

As principais desvantagens, relativas à reutilização de software, correspondem ao alto custo inicial e a sua não inclusão explícita nas metodologias de desenvolvimento de software. Estes fatores dificultam a implementação da reutilização.

Além disso, o custo de construção de um componente de software reutilizável é mais caro e consome mais tempo para confecção. Este alto custo é devido aos seguintes aspectos [(JONES, 1994), (MCCLURE, 1997), (LIM, 1998)]:

- Processo de implantação custoso e demorado;

- Manutenção de níveis de qualidade do processo de desenvolvimento de componente de software para reutilização;
- Treinamento de pessoal;
- Necessidade de pessoas adequadas para fazer a análise de domínio.

3. PROCESSO DE DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES

O processo de desenvolvimento baseado em componentes é apontado como uma abordagem de desenvolvimento que promove a redução do custo e do tempo de desenvolvimento de sistemas de software.

Tanto a tecnologia que fornece suporte às aplicações baseadas em componentes, como os processos para desenvolvimento de aplicações baseadas em componentes tem igual importância para o sucesso do desenvolvimento baseado em componentes.

Segundo Booch (1995) apud (KRUTCHEN, 2000), um processo de desenvolvimento de software tem quatro funções, que são:

- Fornecer um guia de como estabelecer a ordem das atividades da equipe;
- Especificar os artefatos que serão desenvolvidos e quando eles serão desenvolvidos;
- Direcionar as tarefas individuais dos projetistas e da equipe como um todo;
- Oferecer critérios para monitoração e medida dos produtos e das atividades do projeto.

Este capítulo tem a finalidade de apresentar o processo de desenvolvimento de software baseado em componentes, para isso, são discutidos os processo de reutilização de componentes de software e o processo de desenvolvimento baseado em componentes.

Também são apresentados os exemplos mais relevantes de processos de desenvolvimento baseado em componentes e o impacto desta abordagem no processo de desenvolvimento de sistemas.

3.1 Conceitos e Definições

Segundo o SEI, processo de desenvolvimento baseado em componentes envolve os passos técnicos para projeto e implementação de componentes de software, montagem de sistemas a partir de componentes de software previamente construídos, e distribuição de sistemas para os seus ambientes destinos. Um outro conceito importante é a engenharia de software baseada em componentes, que envolve as práticas necessárias para executar o desenvolvimento baseado em componentes, de modo repetitivo, para construir sistemas que tenham suas propriedades conhecidas (BACHMAN et al., 2000).

Deste modo, esta abordagem tem, como premissa fundamental, a construção de um sistema de software a partir de componentes de software já existentes. Isto contrasta com o processo de desenvolvimento tradicional, em que praticamente todos os elementos são construídos.

Como qualquer processo de desenvolvimento, o processo de desenvolvimento de software baseado em componentes envolve uma seqüência lógica de fases constituídas de atividades que transformam os artefatos de entrada em artefatos de saída para construir o sistema desejado. Entretanto, esta abordagem difere da tradicional, pois um sistema é obtido através do uso de componentes de software.

Apesar de não serem iguais, os modelos de processo da abordagem baseada em componentes, encontrados na literatura de engenharia de software, possuem um conjunto de fases em comum, que são [(BROWN; SHORT, 1997), (MCCLURE, 1997), (POUR, 1998), (PRESSMAN, 2000)]:

1. Análise de Requisitos;
2. Aquisição do componente;
3. Entendimento do componente;
4. Adaptação do componente;
5. Composição do componente;
6. Certificação do Componente.

Como exemplo, a Figura 5 apresenta um processo de desenvolvimento de software baseado em componentes apresentado por Pressman (2000), onde podem ser identificadas as fases listadas anteriormente.

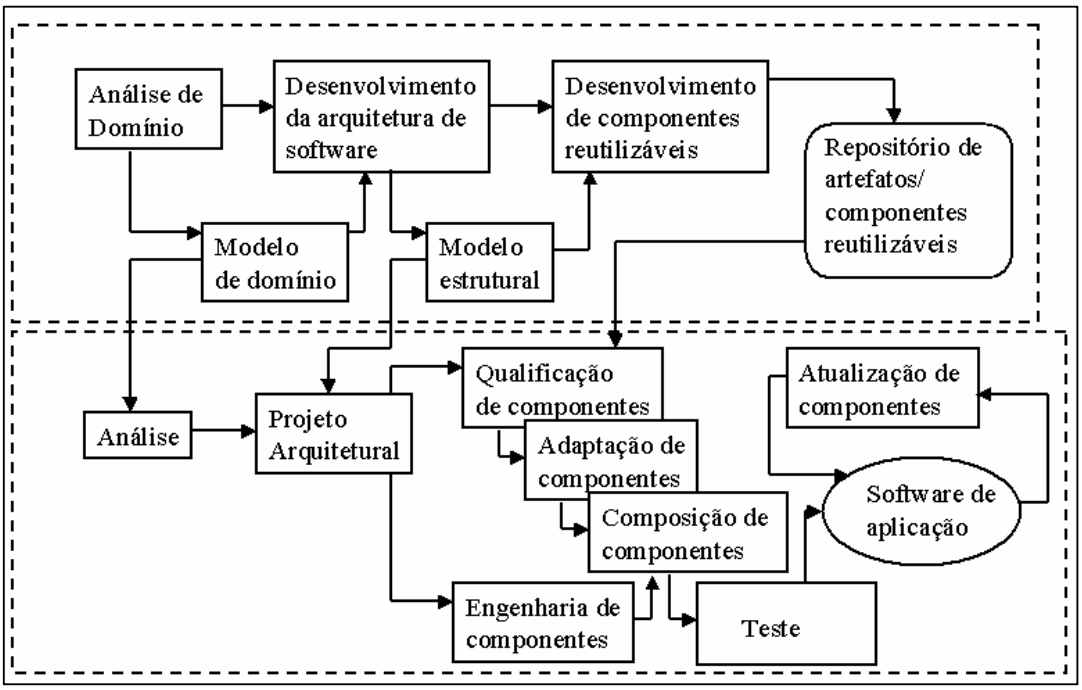

Figura 5: Modelo de processo que suporta desenvolvimento de software baseado em componentes extraído de Pressman (2000, p. 742).

Pode-se afirmar que a reutilização de software está implícita dentro desta abordagem, uma vez que um sistema de software é construído a partir de componentes de software previamente construídos, e além disso, um ou mais componentes podem ter sido utilizados para a construção de um outro sistema de software. Entretanto, é necessário estabelecer em quais momentos do processo as atividades da reutilização ocorrem.

É importante destacar que, no caso do processo de desenvolvimento baseado em componentes, existem dois enfoques a serem considerados: desenvolvimento de componentes de software para reutilização e desenvolvimento de software com componente reutilizável.

O desenvolvimento de componentes de software para reutilização trata da construção de componentes de software para serem reutilizados (RIBOT; BONGARD; VILLERMAIN, 1994). Nesta abordagem, são abrangidos os projetos para construção de modelos de domínios e arquitetura, construção de novos componentes, ou re-engenharia de componentes existentes para melhorar a sua reusabilidade.

O desenvolvimento de software com componente reutilizável, diz respeito ao uso de componentes de software reutilizáveis para a obtenção de sistemas. Suas atividades são relativas ao processo de desenvolvimento, incluindo a identificação e a busca de componentes reutilizáveis na base de reutilização, a construção de novo componente de software (se necessário) ou sua adaptação ou sua composição (RIBOT; BONGARD; VILLERMAIN, 1994).

A seguir são abordadas as fases do desenvolvimento de componentes de software para reutilização e as fases do desenvolvimento de software utilizando componentes de software.

3.1.1 Desenvolvimento de Componentes de Software para Reutilização

O desenvolvimento de componentes de software para reutilização compreende as fases que se referem à identificação e ao fornecimento de componentes de reutilizáveis apropriados às necessidades dos projetistas. Para isso, fazem parte desta abordagem as seguintes fases:

- Análise de domínio;
- Construção de componentes reutilizáveis;
- Manutenção e divulgação dos componentes;
- Certificação de componentes;
- Intermediação de componentes reutilizáveis.

3.1.1.1 Análise do Domínio

Frakes e Isoda (1994) afirmam que o conceito chave para o processo de reutilização de software de modo sistemático, é domínio de sistema. Para eles, o domínio é definido como um conjunto de sistemas que compartilham as decisões de projeto.

Desta forma, domínio é um conjunto de sistemas relacionados, que tem propriedades em comum. É interessante analisar e organizar estas propriedades comuns em uma coleção de componentes reutilizáveis, que podem ser utilizados para construir futuros sistemas neste mesmo domínio.

A análise de domínio é o processo de identificação, análise, e especificação de características comuns e relevantes em um domínio de sistema. Pode-se dizer que é uma técnica essencial para a reutilização de software, uma vez que, através deste processo, é realizada a identificação e a criação de um conjunto de componentes reutilizáveis para ser utilizado no desenvolvimento de sistemas em dado domínio.

Para isso, deve estabelecer um conjunto de componentes de software para reutilização, por meio da identificação e registro das características comuns e variáveis de sistemas em domínios, o que permite identificar os componentes de software candidatos a serem reutilizados.

Segundo alguns autores, existem muitos métodos de análise de domínio, porém, eles compartilham características em comum, que são [(MCCLURE, 1997), (POULIN, 1999)]:

- Caracterização do domínio e planejamento do projeto: análise de viabilidade e planejamento como qualquer projeto de engenharia;
- Coleta de dados: aquisição de dados brutos e informações a partir de especialistas, aplicação existentes, documentação e outros;
- Análise de dados: filtragem, esclarecimento, abstração, e organização dos dados brutos;

- Classificação: generalização dos encontrados para cobertura máxima e a especialização dos encontrados para identificar as partes específicas do problema que alteram com os vários possíveis usos;
- Avaliação do modelo de domínio: passo crítico da validação dos resultados da análise de domínio.

A Figura 6 mostra as entradas necessárias e saídas geradas pela fase de análise de domínio, segundo McClure (1997). As entradas necessárias são as seguintes:

- Informação descrevendo os sistemas existentes em um domínio (por exemplo, código fonte, documentação de sistema, plano de testes, manual do usuário, modelo de projeto, versões do sistema);
- Perícia e conhecimento a partir de especialistas no domínio;
- Informação descrevendo sistemas futuros e planejados (por exemplo, planejamento estratégico corporativo e arquitetura de informação);
- Informação a respeito de tendências tecnológicas.

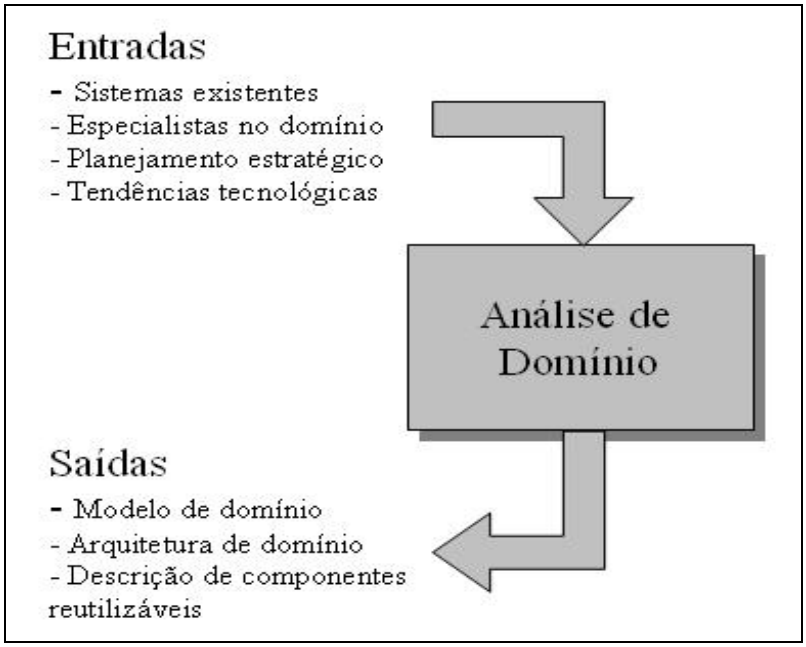

Figura 6: Entradas e saídas da análise de domínio

As saídas geradas pela análise de domínio são (MCCLURE, 1997):

- Modelo de domínio: modelo de definição de um domínio que descreve os conceitos essenciais (por exemplo, funções, características, dados, classes de objetos, requisitos) e seus relacionamentos;
- Arquitetura de domínio: modelos de projeto em alto nível para construção de sistemas em um domínio a partir de componentes reutilizáveis;
- Descrições dos componentes reutilizáveis: documentação para utilização para produção de sistemas e projeto de sistemas para aplicação em sistemas em um dado domínio.

3.1.1.2 Construção de componentes

No processo de componentes de software para reutilização, a fase de construção de componentes consiste do desenvolvimento de novos componentes e ou da reengenharia de componentes com o objetivo específico da reutilização (Lim, 1998).

Desta forma, a partir do conjunto de componentes identificados pela análise de domínio, que inclui tanto componentes como arquiteturas, inicia-se o desenvolvimento, ou seja, projeto e implementação dos componentes identificados através da análise das oportunidades de reutilização. Lim (1998) apresenta duas abordagens para a fase de construção de componentes reutilizáveis, que são: pré-fabricação e a reengenharia.

A pré-fabricação é uma abordagem que visa criação de componentes reutilizáveis com o objetivo de reutilização e inclui as atividades de: análise de requisitos, projeto, codificação, teste e manutenção. A Tabela 9 descreve as atividades da fase de construção segundo Lim (1998).

A segunda abordagem é a re-engenharia, que consiste de examinar os bens existentes, avaliar a viabilidade de transformá-lo em um componente reutilizável e, caso seja viável, implementar as alterações necessárias.

Atividades fase de construção	
Atividade	**Descrição**
Investigação	Consiste da análise de domínio, ou seja, os requisitos do usuário são coletados para uma família de sistemas/produtos existentes e planejados e, então, analisados seus aspectos comuns.
Avaliação de Risco	Inclui tanto a análise de riscos de *marketing* como de riscos técnicos. Esta atividade auxilia na decisão de se prosseguir com o desenvolvimento do produto.
Projeto	Consiste de projeto externo e interno. Projeto externo é a análise detalhada dos requisitos e definição da visão externa do produto, ou seja, a arquitetura. Projeto interno é a tradução do projeto externo em projeto detalhado do sistema e dos subsistemas.
Codificação	Consiste de uma série de atividades desde a codificação até o teste de unidade. As técnicas que apóiam a reusabilidade (exemplo: procedimentos parametrizados e *templates* de codificação) devem ser utilizados, e os projetistas devem enfatizar o bom estilo de codificação.
Teste	Corresponde o teste de integração e de sistema. O teste de componentes reutilizáveis é mais complexo, pois o plano de testes deve levar em consideração todos os contextos no qual o software será reutilizado.
Manutenção	Manutenção envolve as mudanças que o componente de software sofre depois de ter sido entregue.

Tabela 9: Atividades do processo de construção de um novo componente

3.1.1.3 Manutenção e Divulgação de Componentes

A manutenção envolve as mudanças que o componente de software é submetido após a sua entrega. Estas mudanças são realizadas tanto para melhorar ou adicionar as funções do componente de software, ou para corrigir erros ou para adaptar a um novo ambiente.

Segundo Lim (1998), o produtor de componentes realiza a manutenção para reparar falhas em componentes de software reutilizáveis e para melhorar a reusabilidade de componentes e seus atributos; ele deve realizar a manutenção tradicional, tanto

adaptativa, como corretiva ou perfectiva, mas com uma ênfase sobre a manutenção da reusabilidade do componente.

A divulgação de componentes tem a finalidade de notificar os projetistas em potencial sobre os novos componentes de software na biblioteca de componentes e sobre as atualizações de versões tanto para correção de defeitos, adaptações a novas situações ou melhorias na funcionalidade oferecida pelo componente.

3.1.1.4 Certificação de Componente

Segundo a definição apresentada no IEEE *Standard Glossary de Software Engineering Terminology* (1990) apud (FLYNT; DESAI, 2001), certificação é uma garantia escrita, demonstração formal, ou o processo de confirmação de que um sistema ou componente cumpre com seus requisitos especificados e é aceitável para uso operacional.

Um componente de software, antes de ser utilizado em aplicação de software ou incorporado em repositório de componentes, deve ser submetido a um processo de certificação, com a finalidade de garantir a qualidade e confiança deste componente.

Bachman estabelece duas ações que envolvem a certificação de componentes, que são (BACHMAN et al., 2000):

- Estabelecimento de que o software possui propriedades particulares, mais freqüentemente em conformidade com alguma especificação contra o qual o componente ou *framework* é certificado;
- Homologação do componente através de um agente ou uma organização de confiança, que ateste a verdade desta conformidade.

Voas (2000) propõe o processo de certificação do software baseado no uso, como um método para assegurar a qualidade de software. Neste caso, agências independentes,

denominadas *Software Certification Laboratories* (SCL), são responsáveis pela execução do processo de certificação do comportamento do software no ambiente do usuário.

A certificação do componente de software é uma fase muito importante no processo de desenvolvimento baseado em componentes. A adoção de um processo de certificação tem o intuito de assegurar a confiança nos componentes que são utilizados em um sistema, como também no sistema como um todo.

3.1.1.5 Intermediação dos componentes reutilizáveis

A fase de intermediação de componentes tem a finalidade de auxiliar o esforço de reutilização através da qualificação e certificação, configuração e promoção dos componentes de software reutilizáveis. Nesta fase, também estão incorporadas as atividades de classificação e recuperação de componentes de software da biblioteca (LIM, 1998).

Desta forma, esta fase é composta por um conjunto de atividades que são as seguintes:

- Avaliação dos componentes para intermediação: avalia o componente para decidir os aspectos para a sua utilização;
- Obtenção dos componentes: determina a origem do componente, ou seja, se será produzido interna ou externamente;
- Adição dos componentes: inclui a catalogação, a classificação e a documentação (descrição) do componente;
- Exclusão dos componentes: exame do inventário de componentes e exclusão daqueles que não está justificando seu armazenamento na biblioteca.

3.1.2 Desenvolvimento de Software com Componente Reutilizável

A abordagem de desenvolvimento de sistemas de software através de componentes de software reutilizáveis é um processo de construção de sistema que prevê a incorporação de partes de software previamente construídas para este fim.

O objetivo desta seção é descrever, de um modo genérico, as fases que compõe esta abordagem de desenvolvimento, sem se preocupar em refletir as características de um processo específico. As fases desta abordagem são as seguintes:

- Análise de requisitos;
- Projeto;
- Implementação;
- Aquisição de componente;
- Qualificação de componente;
- Adaptação de componentes;
- Composição de componente;
- Testes do sistema;

Algumas fases são comuns ao processo de desenvolvimento tradicional, tais como: análise de requisitos, projeto, implementação e testes; no entanto, existem algumas atividades que são específicas do processo de desenvolvimento baseado em componentes.

A fase de análise de requisitos não difere da fase de um processo convencional de desenvolvimento de software; são especificados os requisitos funcionais, informações necessárias como entrada, as informações geradas como saída, o comportamento do sistema, as restrições de projeto e as interfaces com outros elementos do sistema.

Na fase de projeto, ocorre o desenvolvimento da arquitetura do sistema e a identificação dos componentes de software necessários para compor o sistema, os quais podem existir na biblioteca ou podem ser especificados novos componentes a

serem desenvolvidos. No caso de reutilização de componentes prontos, deve-se verificar a necessidade de adaptação dos mesmos.

Durante a fase de implementação, os componentes de software, selecionados para compor ao sistema, são recuperados do repositório e é realizada a sua integração. Também são desenvolvidos os outros códigos necessários para a implementação do sistema.

Na fase de testes ocorre a validação do sistema para assegurar o funcionamento adequado do sistema e de seus componentes.

As demais fases que são relacionadas com a abordagem de desenvolvimento de software baseado em componentes estão apresentadas nas próximas seções.

3.1.2.1 Aquisição do Componente

A fase de aquisição de componentes de software se refere tanto ao conjunto de atividades para o desenvolvimento interno de um componente de software, quanto ao conjunto de atividades para a aquisição de um componente de software construído por um fornecedor externo à organização, para atender as necessidades de um projeto.

No caso do desenvolvimento interno, as atividades envolvidas para a construção dos componentes são equivalentes ao processo de desenvolvimento tradicional, que são: análise de requisitos, projeto, implementação, testes e manutenção. A vantagem deste tipo de abordagem, segundo Takata (1999), é que o componente de software construído é mais flexível e possui uma maior aderência às necessidades.

No caso de aquisição de um fornecedor externo, as atividades envolvidas obtenção dos componentes são equivalentes ao processo de aquisição de software de fonte externa, e são as seguintes: especificação dos requisitos (função, desempenho,

ambiente e restrições), seleção dos fornecedores, licitação, avaliação dos componentes encontrados e escolha do mais adequado às necessidades. A vantagem desta abordagem é o custo menor (TAKATA, 1999).

3.1.2.2 Qualificação do Componente

Uma vez que o componente foi obtido a partir de uma biblioteca de componentes reutilizáveis, desenvolvido ou adquirido a partir de um fornecedor externo, é necessário que ele seja entendido e qualificado para garantir que realiza as funções necessárias, vai se adequar à arquitetura da aplicação e possui características de qualidade (por exemplo, desempenho, confiabilidade e usabilidade) (PRESSMAN, 2000).

Segundo Brown e Short (1997), a qualificação do componente pode identificar interfaces que são possíveis fontes de conflito e sobreposição entre componentes. Para contornar este problema, o entendimento do componente deve ser apoiado pela sua documentação, que deve possuir a informação necessária para sua utilização de modo correto, evitando este tipo de conflito.

Para Brown (1996) apud (PRESSMAN, 2000), uma relação de fatores devem ser considerados durante esta fase, que são:

- Interface de programação da aplicação;
- Ferramentas de desenvolvimento e integração exigidas pelo componente;
- Requisitos de execução, incluindo uso de recursos (memória ou armazenamento), tempo ou velocidade e protocolo de rede;
- Exigências de serviço, incluindo interfaces com o de sistema operacional e apoio por outros componentes;
- Características de segurança, incluindo controles de acesso e protocolo de autenticação;

- Aquisição de projetos embutidos, incluindo o uso de algoritmos específicos numéricos ou não numéricos;
- Manipulação de exceções.

Desde que o componente possua uma documentação adequada, conforme foi apresentada na seção 2.3.5, as questões apresentadas nesta lista podem ser extraídas a partir da sua documentação.

3.1.2.3 Adaptação de Componentes

A fase de adaptação de componentes pode ser considerada como sendo o que acontece entre o momento em que é tomada a decisão de utilizá-lo e o momento em que o componente torna-se parte do produto (MILI; MILI; MILI, 1995). Isto ocorre, pois uma vez que os componentes são desenvolvidos, inicialmente, para atender as necessidades de uma determinada aplicação ou um conjunto de requisitos pré-determinados, podem necessitar de adaptações para a utilização em um novo sistema.

Durante a atividade de adaptação, são realizados ajustes baseados em regras, com o intuito de assegurar que os conflitos entre os componentes sejam minimizados. O processo de adaptação pode ser dividido em três sub-tarefas, que são (MILI; MILI; MILI, 1995):

- Seleção: caso o componente tenha uma parte variável ou, alternativas de implementação explicitamente enumeradas, deve ser selecionada a que é apropriada para o problema em questão;
- Modificação: caso componente ou qualquer uma de suas variações não for utilizada deve ser modificado para ser reutilizado;
- Integração: analisa se o componente é compatível com o ambiente.

Além disso, a atividade de adaptação de componentes pode ser classificada de acordo com o grau de acessibilidade da estrutura interna do componente (*component*

wrapping), podendo ser de três diferentes abordagens (BROWN e WALNAU, 1996) apud (PRESSMAN, 2000):

- Caixa branca: o código-fonte do componente pode ser acessado, permitindo que seja reescrito para realizar ajustes para integração com a aplicação;
- Caixa cinza: o código-fonte não pode ser modificado. A linguagem de extensão ou API permite que os conflitos sejam removidos ou mascarados;
- Caixa preta: não existe linguagem de extensão ou API e o componente é fornecido apenas na forma executável binário. Desta forma, é necessária a introdução de pré-processamento ou pós-processamento na interface do componente para remover ou mascarar os conflitos.

3.1.2.4 Composição dos Componentes

A fase de composição se refere à combinação dos componentes de software que foram identificados, adquiridos, desenvolvidos, adaptados, para compor a arquitetura de um sistema baseado em componentes (PRESSMAN, 2000).

Desta forma, considerando que um sistema baseado em componentes é formado por componentes de software de origens diferentes (internas e externas), podem ser encontrados problemas gerados por estas incompatibilidades entre componentes, chamada de arquitetura mal combinada (GARLAN; ALLEN; OCKERBLOOM, 1995).

Uma documentação dos componentes é muito importante, para fornecer informações relevantes para auxiliar na fase composição, com o intuito de evitar os conflitos na arquitetura do sistema. Para isso, conforme apresentado na 2.3.5, a documentação deve fornecer informação do tipo: API, ferramentas de desenvolvimento e integração necessária, requisitos de armazenamento secundário (*run-time*), requisitos de processador (desempenho) e de rede (capacidade), serviços de software necessários

(sistema operacional ou de outros componentes), suposições de segurança e suposições de projeto incorporado.

Outro aspecto importante, é fornecer uma infraestrutura de comunicação e coordenação através de padrões de infraestrutura (CORBA, COM/DCOM ou JavaBeans), que fornecem a base para a aplicação do desenvolvimento baseado em componentes, para construir aplicações em ambientes heterogêneos distribuídos [(BROWN; WALLNAU, 1998), (POUR, 1998), (PRESSMAN, 2000)].

3.2 Exemplos de Processos de Desenvolvimento de Software Baseado em Componentes

Nesta seção serão apresentados os processos de desenvolvimento de software baseados em componentes, relevantes para este trabalho.

O Rational Unified Process (RUP) e o Catalysis são processos bastante referenciados na literatura de engenharia de software. Já a metodologia proposta por Takata [TAKATA, 1999] orienta o desenvolvimento baseado em componentes de software, foi implantada em uma instituição financeira de grande porte e obteve resultados satisfatórios.

3.2.1 Rational Unified Process (RUP)

O Processo Unificado é processo de desenvolvimento de software baseado em componentes, ou seja, prevê que o sistema de software pode ser construído através da composição de componentes de software interconectados através de interfaces bem definidas.

Além disso, resultou da incorporação dos recursos dos métodos mais relevantes de análise e projeto orientados a objetos, e tem como característica, ser um processo muito amplo, que pode ser especializado para diferentes classes de sistema, organizações, projetos e áreas de aplicação (JACOBSON; BOOCH; RUMBAUGH, 1998).

O RUP é uma instância mais específica do Processo Unificado e as melhores práticas de processos de software incorporadas neste processo são (KRUCHTEN, 2000):

- Desenvolvimento iterativo;
- Gerência de requisitos;
- Utilização de arquitetura e componentes;
- Modelagem visual através da UML;
- Qualidade de processo e de produto;
- Gerência de configuração e versões;
- Casos de uso dirigindo muitos aspectos do desenvolvimento;
- Modelo de processo de desenvolvimento que pode ser adaptado e estendido para as características da organização;
- Necessidade ferramentas de desenvolvimento de software para fornecer suporte ao processo.

A Figura 7 apresenta um exemplo da estrutura do RUP, suas fases e seu núcleo do fluxo de trabalho.

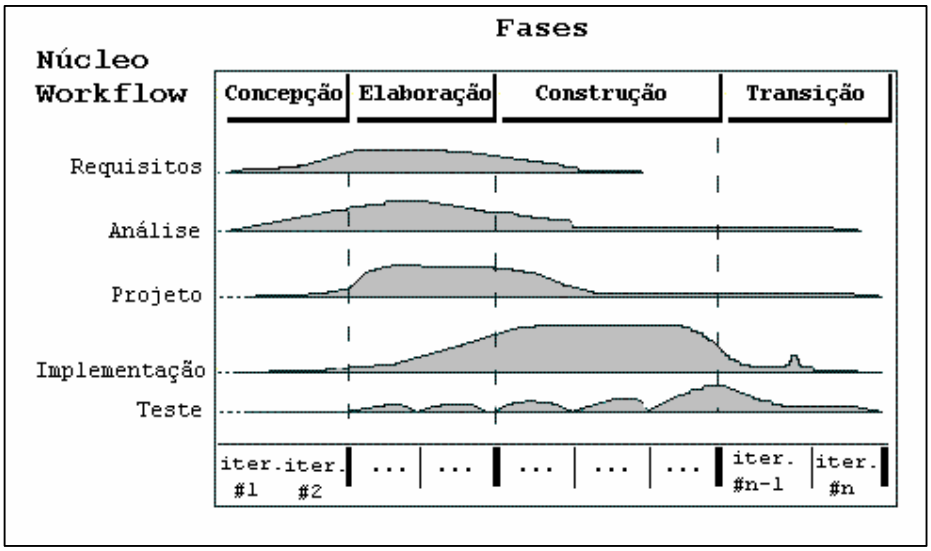

Figura 7: Estrutura, fases, iterações e o fluxo de trabalho da estrutura do RUP (BROWN, 2000, p. 23).

É importante destacar que no eixo vertical, da estrutura do RUP, está a representação dos fluxos de trabalho do processo, agrupando as atividades logicamente pela natureza (KRUCHTEN, 2000). Este processo abrange um ciclo de vida de desenvolvimento completo, consistindo basicamente de: modelagem de negócio, gerência de requisitos, análise e projeto, implementação, teste e distribuição.

É importante ressaltar que, podem existir outros elementos no ciclo de vida, e que cada elemento é apoiado por ferramentas e guias de processos descrevendo cada um deles e sua aplicação.

O eixo horizontal, segundo (KRUCHTEN, 2000), representa o tempo e mostra como os componentes do ciclo de vida do processo são desdobrados através das suas fases. Esta representação descreve os aspectos dinâmicos do processo como ele ordena, e expressa em termos de ciclos, fases, iterações e os pontos de verificação, sendo que, dentro de cada fase, gerentes ou projetistas podem dividir o trabalho em duas ou mais iterações e cada fase termina com um ponto de verificação.

Cada fase do processo é descrita a seguir:

- Concepção: definição clara do que é o produto a ser desenvolvido; para isso, devem ser estabelecidos o escopo do sistema, uma visão inicial dos casos de uso principais, as condições limites e o critério de aceitação;
- Elaboração: elaboração da especificação detalhada de mais casos de uso do produto e da arquitetura do sistema. Esta fase pode ser executada em uma ou mais iterações, dependendo do escopo, tamanho, e o nível de domínio sobre o assunto do projeto;
- Construção: processo de produção do produto, ou seja, implementação do software, integração e teste dos componentes e avaliação da versão dos produtos em relação aos critérios de aceitação. O resultado desta fase é o

produto que incorpora todos os casos de usos, definidos pelo cliente e pela gerência, para fazer parte desta versão;

- Transição: conjunto de atividades que conduzem o produto obtido ao ambiente de produção, compreendendo, por exemplo, manufatura, treinamento de usuário, suporte e correção de erros encontrados depois da entrega.

3.2.2 Catalysis

Segundo D'Souza e Wills (1999), o Catalysis é uma abordagem dirigida a modelos para construção de sistemas abertos distribuídos a partir de componentes e *frameworks*, para refletir e fornecer suporte a empreendimentos adaptativos. Para isso, esta abordagem é dividida em quatro fases, que são:

- Modelo de negócio: entender as necessidades do cliente, capturar as regras de negócio, comportamento e as restrições do problema a ser resolvido;
- Especificação do sistema: identificar as funções do sistema, e estabelecer como será a interação do sistema com pessoas e sistemas e detalhar o modelo de negócios com a finalidade de construir interações mais precisas das ações executadas pelo sistema;
- Arquitetura: é definida a arquitetura da aplicação para identificar os componentes necessários à implementação do sistema, e também, descrever as colaborações e dependências entre os componentes;
- Projeto: são especificados os componentes e o contexto de utilização; nesta fase tem-se uma preocupação com o desacoplamento dos componentes;
- Implementação: definir a implementação interna para os componentes e suas interações de forma que todos os requisitos do componente sejam satisfeitos.

Esta abordagem possui um conjunto de características que são ressaltadas por D'Souza e Wills (1999), que são:

- Processo formal que enfatiza a precisão e rastreabilidade dos modelos;
- Baseado em UML;
- Baseado em componentes;
- Reutilização de modelos e *frameworks.*

3.2.3 Metodologia para Utilização de Componentes de Software para Ambiente Cliente-Servidor

Esta seção apresenta a metodologia de desenvolvimento visando sistemas cliente-servidor e definida considerando os seguintes processos (TAKATA, 1999):

- Desenvolvimento de aplicações: relacionado com a engenharia de software;
- Desenvolvimento de componentes: relacionado com a engenharia de domínios.

Segundo Takata (1999), a existência de dois processos permitiu a segregação de funções em três grupos, que são os seguintes:

- Grupo de Análise e Projeto de Aplicação: composto por pessoas tanto do grupo de desenvolvimento de componentes como do grupo de desenvolvimento de aplicação, o que facilita a transmissão da especificação e dos requisitos entre os grupos, permitindo uma melhor utilização dos componentes existentes;
- Grupo de Desenvolvimento de Componentes: responsável pela engenharia de domínio e pela obtenção/manutenção dos componentes, que pode ser tanto por desenvolvimento interno ou externo;
- Grupo de Desenvolvimento de Aplicação: responsável pela construção de aplicações e também por fornecer consultoria na análise de domínio de negócio, já que possui uma experiência maior com o negócio do que o grupo de componentes.

Através da Figura 8, podem ser visualizadas as fases de cada um destes processos.

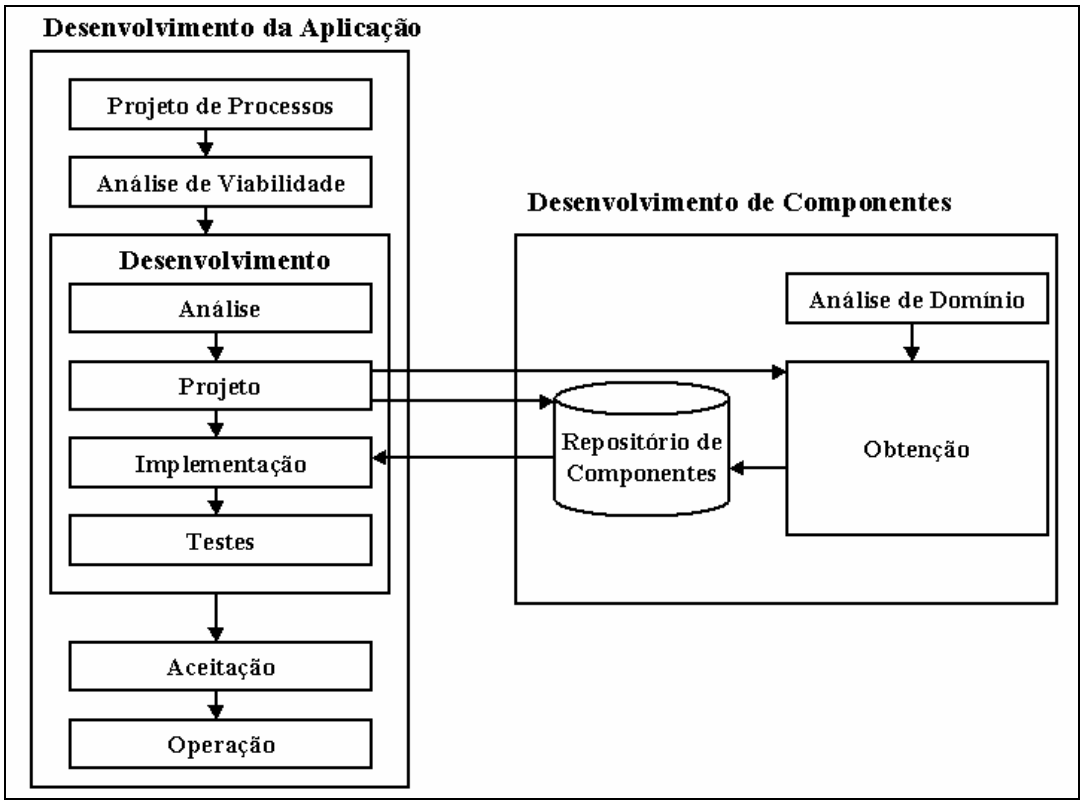

Figura 8: Fases da metodologia proposta por Takata (1999, p. 104).

3.3 Impacto dos Componentes de Software no Processo de Desenvolvimento de Software

A abordagem de desenvolvimento de software baseado em componentes tem sido considerada como uma solução para o problema do crescimento da demanda de sistemas mais complexos, de maior porte e com alta taxa de manutenção devido a mudanças nas necessidades de negócio.

Esta seção tem a finalidade de discutir o impacto da tecnologia de componentes de software no processo de desenvolvimento de sistemas de software. Para isso, são discutidas as expectativas em relação à abordagem de desenvolvimento baseados em

componentes e os seus principais fatores relevantes. Também são apresentados o conceito de fábrica de componentes e algumas considerações sobre o futuro da tecnologia de componentes de software.

3.3.1 Expectativas em relação ao Desenvolvimento Baseado em Componentes

As organizações estão cada vez mais dependentes da tecnologia de informação, que automatiza e agiliza seus processos de negócio, e o seu uso adequado é freqüentemente considerado como fator essencial para obterem vantagem competitiva.

Segundo Kalakota e Robinson (2002), o principal agente impulsionador em muitos domínios atuais de negócio é a utilização de tecnologias baseadas em Internet para abrir novas oportunidades de mercado, aperfeiçoar a entrega de produtos ou realização de serviços para clientes, e organizar processos internos de negócio. Esta nova estratégia, também chamada de *e-business*, envolve também as operações de *marketing* e comercialização realizadas por meio digitais.

Com estas perspectivas, as organizações de desenvolvimento de sistemas de software sofrem pressões para entregar as suas soluções cada vez mais rapidamente, com mais qualidade e com maior flexibilidade para mudanças de tecnologia e de práticas de negócio. Para isso, é necessário buscar alternativas para agilizar o processo de desenvolvimento de sistemas, com o objetivo de atender o crescimento do mercado de tecnologia de informação, o qual necessita de sistemas de informação cada vez mais complexos, maiores e mais vitais para as organizações.

Brown (2000) aponta que os componentes e as abordagens baseadas em componentes podem fornecem suporte ao desenvolvimento de sistemas de informação, que utilizam a Internet como uma infra-estrutura, com custo-efetivo para distribuição das aplicações. Para isso, tais ambientes devem apresentar as seguintes características:

- Serviços projetados e implementados em partes (pacotes) por equipes de projetistas;
- Distribuição destas partes (pacotes) em várias de máquinas, que podem estar separadas fisicamente e até mesmo em ambientes heterogêneos;
- Integração de funcionalidade obtida a partir de muitas origens, incluindo pacotes adquiridos e sistemas desenvolvidos anteriormente (sistemas legados);
- Aplicações com arquitetura em n-camadas envolvendo servidores web e servidores de aplicação;
- Fornecimento de suporte rápido a novas iniciativas de negócio que podem ser desenvolvidas de modo mais ágil para atender as demandas de mercado, disponibilizando flexibilidade suficiente para que o software possa ser adaptado às mudanças do ambiente de negócios.

Segundo Bass et al. (2001), a tecnologia de componentes de software é amplamente considerada como o melhor meio de obter ganhos de produtividade no desenvolvimento de aplicações, sistemas flexíveis e a qualidade necessária pela revolução da tecnologia de informação na era da Internet.

Vários autores, na literatura pesquisada, apontam a abordagem baseada em componentes de software nos processos de desenvolvimento, como potenciais provedoras de diferentes benefícios que atendem as necessidades deste tipo sistema, devido a alguns fatores, que são [(JACOBSON.; GRISS; JONSSON, 1997), (BROWN, 2000), (BASS et al., 2001), (HEINEMAN; COUNCIL, 2001), (CASANOVA; STRAETEN; JONCKERS, 2003)]:

- Redução significativa do custo e do tempo de desenvolvimento de sistemas complexos e de grande porte entregues ao mercado;
- Melhoria na manutenabilidade e na flexibilidade, através da possibilidade de substituir um componente por uma nova versão mais adequada;
- Aumento da qualidade dos sistemas baseados em componentes;
- Promessa de reutilização em alta escala.

A discussão apresentada por Bass et al. (2001) também é interessante. Um dos benefícios obtidos com a utilização de componentes de software é o aumento da produtividade do programador. Segundo um estudo apresentado pelo Gartner Group (1999) apud [BASS et al., 2001], as organizações, que adotam o desenvolvimento baseado em componentes, obtêm uma melhora nos custos de programação em 50%. Entretanto, um prognóstico de mercado sugere que somente 30% da primeira geração de investimentos corporativos em componentes poderão gerar a esperada melhoria da produtividade, devido à falha na definição dos objetivos e conflitos no contexto dos componentes que inibem a reutilização.

Atrelado ao aumento de produtividade de programação está a redução do tempo para colocar o produto no mercado, uma vez que, através da montagem de sistema de software utilizando-se blocos de construção prontos (componentes de software), é mais rápido em vez de se construir totalmente a aplicação, pois está se reduzindo o esforço de desenvolvimento.

Bass et al. (2001) afirmam que, enquanto o aumento de produtividade pode ser atribuído à reutilização de software desenvolvido anteriormente, a tecnologia de componentes de software leva à redução do tempo do produto estar disponível em mercado devido a duas razões adicionais, que são:

- Linhas de produtos de componentes para domínios verticais, como por exemplo, as aplicações de *e-commerce*, que tratam a construção de sistemas como configuração e interconexão de componentes pré-construídos dentro do contexto de uma arquitetura de aplicação bem definida;
- A tecnologia de componentes de software aumenta o nível de abstração da construção do sistema através do empacotamento de várias fontes de complexidade em um modelo de componentes separado.

A reutilização de software é um dos fatores motivadores do desenvolvimento baseado em componentes [(BROWN, 2000), (BASS et al., 2001), (WILLIANS, 2001), (STALLINGER et al., 2002)], uma vez que, um novo sistema pode ser

construído mais rapidamente e a um custo menor, através da composição de componentes de software desenvolvidos previamente.

Entretanto, apesar da reutilização ser uma idéia simples a sua implantação é difícil. Desta forma, para alcançar os seus benefícios, é necessária a existência de um processo de desenvolvimento de software estabelecido que forneça suporte à reutilização e ao desenvolvimento baseados em componentes, e ferramentas que forneçam o suporte necessário.

Para alguns autores, o conceito de componentes de software vai além da reutilização, ou seja, abrange mais que a capacidade de um mesmo elemento de software servir para a construção de diferentes aplicações; envolve também a extensibilidade, ou seja, a facilidade de adaptação de software a mudanças na especificação [(MCKINSEY,2001), (ANDREWS; GHOSH; CHOI, 2002)].

A melhoria da qualidade fornece um dos fortes argumentos para a tecnologia de componentes de software e a reutilização. Segundo Poulin (2001), vários projetos declaram o resultado de dez vezes mais qualidade (como medida por densidade de defeitos) em componentes reutilizados.

Mas para Bass et al. (2001), existe pouca evidência se a tecnologia de componentes de software melhora os atributos de qualidade de sistema, tais como, flexibilidade, escalabilidade, segurança, desempenho, confiabilidade e assim por diante.

Meyer, Mingins e Schmidt (1998), destacam que, apesar de a reutilização e o desenvolvimento baseado em componentes terem aumentado a expectativa da obtenção da qualidade dos sistemas, é necessário garantir a aplicação de técnicas para construir componentes de qualidade.

Diante das questões que motivam e inibem a adoção da tecnologia de componentes de software, é importante analisar os fatores que devem ser considerados para a implantação do processo de desenvolvimento baseado em componentes.

3.3.2 Principais fatores relevantes

O processo de desenvolvimento de sistemas baseado em componentes é uma abordagem complexa e que exige um alto nível de comprometimento por parte da organização para atingir os resultados almejados quanto à produtividade, custos e qualidade do produto de software.

Devido à sua complexidade, o processo de desenvolvimento baseado em componentes necessita de uma mudança cultural no modo como a organização gerencia e desenvolve os sistemas de software. Para isso, é necessário um comprometimento gerencial para assegurar que a organização seja estruturada, treinada e provida de pessoas com capacidade técnica para seguir o processo estabelecido e estar conforme aos padrões definidos.

Desta forma, existe a necessidade de vencer a resistência cultural e promover a nova abordagem através de programas de incentivos e de treinamento da equipe. Segundo Reifer (2001), existem alguns aspectos culturais e psicológicos de resistência cultural, que são:

- Apatia: “Por que mudar se não está errado?”, ou seja, não acreditam na necessidade da mudança e não vêem problemas no modo atual como as coisas são realizadas;
- Síndrome Não Inventado Aqui (Not Invented Here – NIH): “Apesar de funcionar em outro lugar não significa que funcionará aqui.”, ou seja, apesar dos casos de sucesso conhecidos, não acreditam que possa ser aplicado na organização;
- Medo do desconhecido: “Por que deveríamos realizar mudanças agora?”, ou seja, medo de mudanças e adoção de uma nova tecnologia que não dominam;

- Torre de Marfim: Pelo fato de se estar lidando com uma tecnologia emergente, alguns praticantes podem argumentar que é prematuro tentar utilizar alguma coisa como componentes neste momento.

Para isso, é importante o desenvolvimento de um plano de negócios bem elaborado e divulgado, que comprove a validade da tecnologia proposta. Além disso, é necessário convencer a alta e a média gerência, e também,os projetistas e os programadores de que a organização já está pronta para transformação tecnológica e que a forma mais efetiva de mudar é a implementação do processo de desenvolvimento baseado em componentes.

Vale à pena ressaltar que este tipo de processo é caro e que exige um alto nível de investimentos para a sua implantação em uma organização, sendo necessário inicialmente o desenvolvimento de um projeto piloto que comprove os resultados prometidos por esta abordagem (HEINEMAN; COUNCILL, 2001).

Segundo Reifer (2001), quanto ao aspecto técnico, muitas organizações de desenvolvimento de software têm adotado paradigmas, métodos e ferramentas que não promovem o uso e a reutilização de componentes. O processo de desenvolvimento adotado pela organização, deve ser prever explicitamente as atividades que identifiquem e incluam, no sistema a ser desenvolvido, os componentes existentes ou a desenvolver.

O processo de desenvolvimento baseado em componentes deve ter suporte de ferramentas que auxiliem os analistas, projetistas e programadores na tarefa de construir aplicações através da customização e montagem em larga escala de componentes de software.

Segundo Heineman e Councill (2001), somente as iniciativas bem fundamentadas de implementação de desenvolvimento baseado em componentes atingiram o sucesso, sendo que, devem estar fundamentadas em pessoal treinado, cronogramas razoáveis e o apoio do alto nível gerencial. Para isso, devem adotadas algumas medidas, que são:

- As equipes devem ser suficientemente treinadas, acompanhadas e experientes para produzir um cronograma inicial do projeto;
- A organização deve disponibilizar fundo suficiente para completar o projeto, incluindo todos os fatores de riscos identificados;
- Todos os papéis e posições devem ser recrutados, treinados, acompanhados e avaliados no processo de desenvolvimento baseado em componentes, antes de se passar os novos membros da equipe para um projeto maior. A monitoração e controle devem ser contínuos;
- O gerente de projeto e produto deve ter conhecimento de cada posição e fase do ciclo de vida do desenvolvimento baseado em componentes e estabelecer harmonia com os subordinados e a gerência superior.

Mas a adoção do desenvolvimento baseado em componentes implica em desenvolver ou adquirir componentes confiáveis e disponíveis para reutilização. Segundo Bass et al. (2001), existe uma grande preocupação no mercado quanto à falta de componentes disponível mercado e quanto à confiabilidade dos componentes desenvolvidos por terceiros.

A falta de componentes disponíveis no mercado pode ser atribuída a duas causas. Uma delas se refere ao aspecto que mercado de componentes, por se tratar de uma tecnologia emergente, ainda não é suficientemente amplo e profundo. A outra causa se refere ao fato de que os consumidores de componentes podem estar tendo dificuldades na localização dos componentes e, quando localizam, podem estar tendo dificuldades no entendimento da funcionalidade e de como reutilizar.

Além disso, um outro aspecto importante é quanto à confiabilidade do componente de software. Alguns pesquisadores afirmam a necessidade de entidades independentes para certificação de componentes, com o intuito de garantir a confiabilidade e qualidade de componente de software [(VOAS, 1998), (VOAS, 2000), (BACHMAN et al., 2000), (HEINEMAN; COUNCILL, 2001)].

O componente de software desenvolvido internamente pela organização para reutilização também necessita de mecanismos e processos que garantam, aos projetistas, recursos para a localização, entendimento e incorporação deste componente no sistema a ser construído. Além disso, existe a necessidade de implementação de um processo para garantir a confiabilidade e qualidade deste componente.

3.3.3 Fábrica de Componentes de Software

Desde a idéia proposta por Cusumano (1991) apud (BASILI; CALDIERA; CANTONE, 1992) de fábrica de software, o conceito evoluiu para fábrica de componentes de software.

Uma proposta apresentada por Basili, Caldiera e Cantone (1992), define a fábrica de componentes como sendo uma extensão e redefinição do conceito de fábrica de software. A fábrica de software é uma unidade de produção de código através de um ambiente de produção integrado. Já a fábrica de componentes é uma unidade de produção de código através de um ambiente integrado que manipula cada tipo de informação e experiência relacionada a código. A Figura 9 apresenta o modelo da fábrica de componentes que foi por proposta por Basili, Caldiera e Cantone (1992).

Pode-se observar que, o principal objetivo de uma fábrica de software é produzir software em massa a um custo efetivo e, segundo Herzum e Sims (2000), a abordagem de desenvolvimento baseada em componentes e de componentes de negócio são pré-requisitos necessários e fundamentais.

Para estes autores uma fábrica de componentes de software é um grupo de desenvolvimento de software com capacidade de produzir software de alta qualidade, a um custo efetivo e com maior rapidez, para um empreendimento de modo repetitivo (HERZUM; SIMS, 2000).

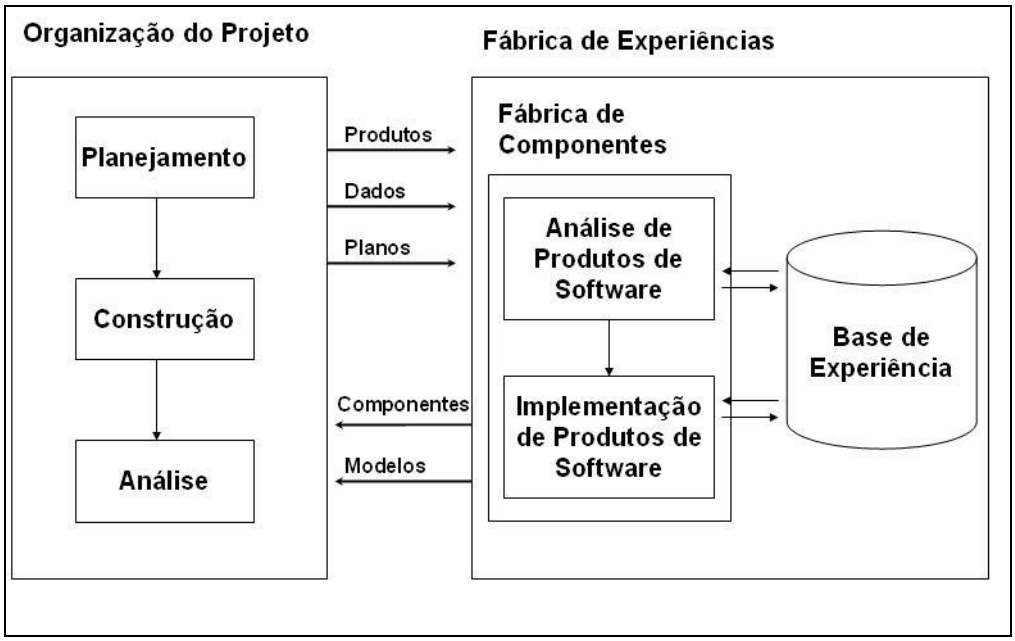

Figura 9: Fábrica de componentes proposta por Basili, Caldiera e Camntone (1992).

Para Kozaczynski (2001), uma fábrica de componentes de software estabelece um processo, organização, ferramentas e infra-estrutura de execução que separa as seguintes preocupações e responsabilidades chaves:

- Infra-estrutura de desenvolvimento de componentes;
- Projeto, integração e teste de sistemas de componentes de aplicação;
- Desenvolvimento e teste de unidade de componentes de aplicação.

Desta forma, a cooperação entre os três grupos, executando cada um suas responsabilidades chaves, assegura o desenvolvimento previsível e repetitivo sobre uma infra-estrutura estável e testada.

Kozaczynski (2001) aponta alguns aspectos chaves na configuração e sucesso de implantação de uma fábrica de componentes, que são:

- Definição do papel da arquitetura: responsável pela totalidade do projeto do produto, incluindo a decomposição da aplicação em componentes e a infra-estrutura de componentes;

- Infra-estrutura imatura de componente: força os projetistas a tomar decisões complexas sobre a seqüência na qual os componentes são desenvolvidos;
- Dimensão e empacotamento de componentes terceirizados: a comunicação inter-grupo é necessária para assegurar que a dimensão e empacotamento atendam as necessidades do produto;
- Distribuição do desenvolvimento em equipes, não é sinônimo de desenvolvimento baseado em componentes: em um ambiente de fábrica de componentes é definida claramente a divisão de responsabilidades e de preocupações, além de, ajudar a gerenciar a comunicação entre equipes; mas a fábrica de componentes não mantém as equipes, de desenvolvimento de aplicações e de desenvolvimento de componentes, unidas. As duas equipes devem compartilhar os processos, as ferramentas, o desenvolvimento, o ambiente de teste, e também, atitudes e cultura de desenvolvimento;
- Comunicação não ambígua: qualidade da comunicação entre a equipe de projeto e de desenvolvimento de componentes é crítica: qualquer ambigüidade pode resultar em falhas na fabricação e atrasos na entrega do produto;
- Dificuldades de integração em larga escala com sistemas legados: integração de sistemas legados sempre limita a liberdade do projeto e pode facilmente corromper o projeto de componentes. A componentização de sistemas legados é difícil, especialmente porque não existe uma seqüência de passo clara para atingir o resultado desejado;
- Falta de coleta de métricas com antecedência: facilita a previsão de custo e tempo de desenvolvimento de componentes, para auxiliar na coordenação das atividades e evitar conflitos entre os grupos que compõe a fábrica de componentes.

3.3.4 Futuro de componentes de software

Desde a idéia apresentada por McIlory (1968) apud Krueger (1992), em que foi proposta uma indústria de componentes de prateleira para a construção de sistemas mais complexos, a proposta apresentada por Diijkstra (1968) apud Clements (1995) a respeito da modularização de programas (princípio da divisão de problemas) e o conceito de informação escondida introduzido por Parnas (1972) apud Clements (1995), muito tem sido pesquisado e estudado sobre os temas de reutilização e desenvolvimento baseado em componentes.

Várias pesquisas e estudos têm sido desenvolvidos na área de tecnologias de componentes a respeito de técnicas específicas, práticas e processos. Todo o interesse em torno deste tema deve-se à promessa de ganho de produtividade, redução de custos e aumento da qualidade dos sistemas construídos, que são cada vez mais complexos e essenciais para as organizações.

O SEI conduziu uma pesquisa e realizou entrevistas com organizações que adotaram a tecnologia de componentes de software e o resultado desta pesquisa, extraído das notas técnicas elaborado por Bass et al. (2001), é apresentado na Figura 10.

A partir dos resultados analisados da pesquisa, Bass et al.(2001) concluíram que o mercado percebe os seguintes inibidores principais, apresentados em ordem decrescente de importância, que são:

- Falta de componentes disponíveis;
- Falta de padrões estáveis para a tecnologia de componentes de software;
- Falta de componentes certificados;
- Falta de métodos de engenharia para produzir consistentemente sistemas de qualidade a partir de componentes.

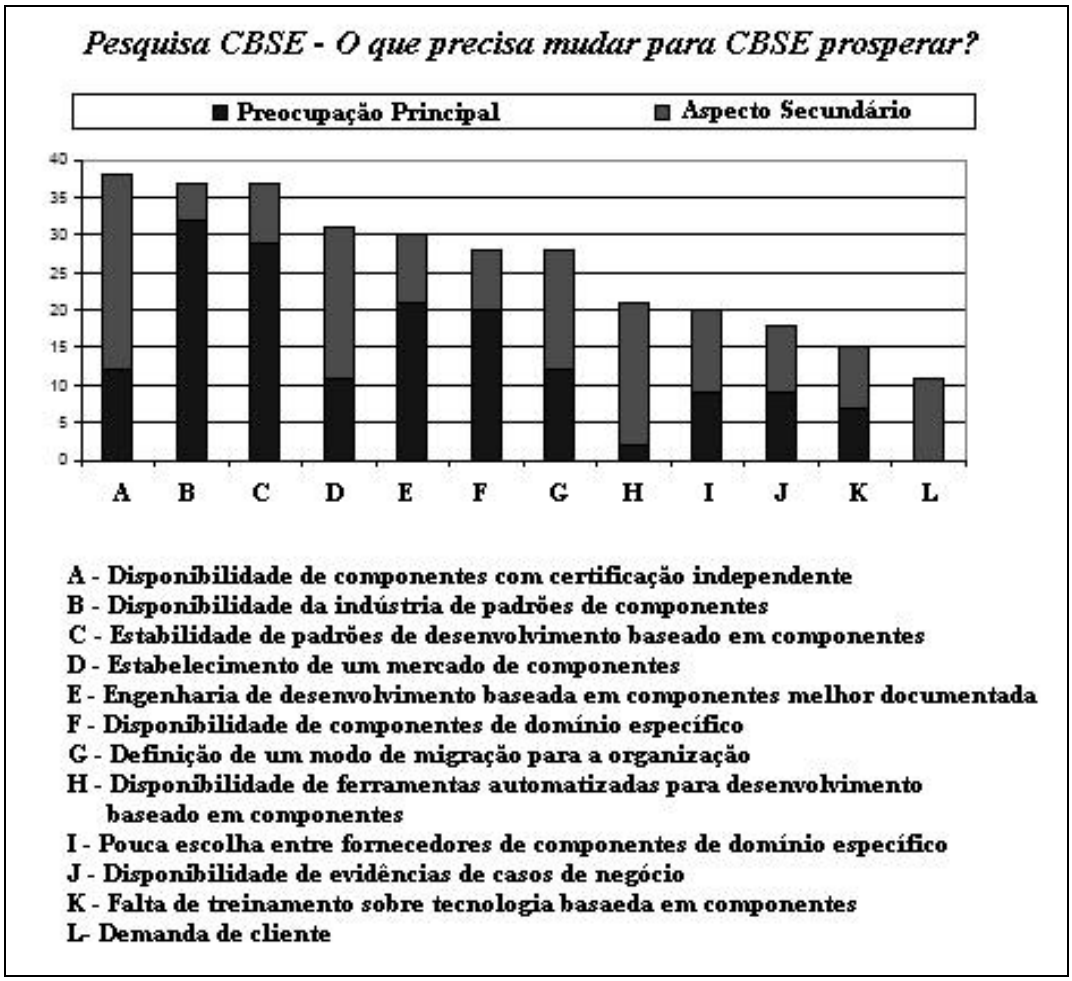

Figura 10: Resumo dos resultados da pesquisa realizada pelo SEI extraído de Bass et al. (2001).

Desta forma, para atender a estas necessidades, é importante que a tecnologia de componentes de software e processo de desenvolvimento baseado em componentes forneça suporte para integração e montagem de aplicações a partir de diferentes peças ou partes de aplicação.

Para alguns pesquisadores, uma questão que permanece em aberto é qual a abordagem de desenvolvimento mais apropriada para o desenvolvimento de sistemas baseados em componentes, uma vez que não existe um entendimento comum sobre os objetivos deste tipo de processo (HISSAN; SEACORD; LEWIS, 2002).

Um outro aspecto importante é com relação às tecnologias de plataforma e infra-estrutura que fornecem a abertura e flexibilidade necessárias para a interoperabilidade entre componentes de diferentes plataformas ou ambientes que necessitam interagir entre si para disponibilizar serviço e informação necessária para um sistema de aplicação em um ambiente corporativo.

Neste sentido, existem muitas iniciativas de estabelecimento de padrões e modelos de componentes como elementos essenciais para obter uma solução flexível e eficiente no desenvolvimento de aplicação complexas em ambientes heterogêneos. Alguns autores apontam iniciativas que buscam a padronização e a definição de modelos de componentes para aumentar a interoperabilidade e a troca de informações entre soluções empresariais, que são [(BROWN, 2000), (HEINEMAN; COUNCILL, 2001)]:

- CORBA da Object Management Group (OMG): possibilita que uma aplicação acione operações sobre objetos distribuídos sem se preocupar com a localização do objeto, linguagem de programação, sistema operacional, protocolo de comunicação, interconexões e plataformas de hardware;
- Sun Entreprise Java Beans (EJB): disponibiliza a descrição de um modelo arquitetural para sistemas distribuídos baseados em Java, na qual os serviços de infra-estrutura requisitados por qualquer aplicação são definidos de um modo padronizado;
- COM+ da Microsoft: é base de um modelo de tecnologias projetado para fornecer suporte ao desenvolvimento de aplicações distribuídas em larga escala na plataforma Windows;
- XML da World Wide Web Consortiun (W3C): é uma linguagem baseada em *tags* relativamente simples, que pode ser utilizada para descrever e comunicar informação entre remetente e destinatário.

Os padrões de modelos de componentes EJB e CORBA são provavelmente os que têm maior influência sobre as soluções em escala empresarial nos últimos anos. Segundo Brown (2000), estes padrões definem a estrutura básica e o vocabulário para um estilo arquitetural de construção de sistemas distribuídos. Alem disso, como

o mercado para sistemas distribuídos em larga escala continua em maturação, deverão surgir muito mais soluções focadas sobre o suporte destas tecnologias.

Quanto às ferramentas para apoiar o desenvolvimento baseado em componentes, pode-se dizer que são elementos essenciais e que sem a sua utilização, a tarefa de desenvolver aplicações a partir de componentes previamente construídos torna-se muito mais complexa e difícil. Heineman e Councill (2001) apresentam uma relação de avanços nesta área, que são:

- Ambiente de desenvolvimento como, por exemplo, o APPGALLERY, da Hitachi, é um ambiente de desenvolvimento baseado em componentes que disponibiliza um conjunto de componentes, de ferramentas para desenvolvimento e customização, além de um depurador visual que auxilia a construção de componentes para reutilização;
- Gerência de Componentes, como por exemplo, o Aonix Select Component Manager que fornece suporte para publicação, gerência e utilização de componentes de software em todo o empreendimento, sendo que componentes e especificações são compartilhados através de um repositório. Possui também controle de versão;
- Ferramentas para auxiliar a busca em catálogo de componentes e instanciar o componente no ambiente de desenvolvimento para testa se o componente é adequado;
- Ambientes integrados de desenvolvimento que fornecem suporte aos conceitos baseados em componente, incluindo composição em tempo de projeto, distribuição e evolução de componentes.

Muito se tem falado na literatura de engenharia de software a respeito do mercado de componentes de software. Segundo Heineman e Councill (2001), nos Estados Unidos existem aproximadamente 4.000 componentes disponíveis para venda na Internet. Mas, na maioria dos casos não existem garantias sobre a confiabilidade destes componentes.

Para isso, é necessário o estabelecimento de padrões de documentação bem estruturada que auxilie no entendimento de características, funcionalidade, utilização

e catalogação do componente. Segundo Houston e Norris (2001), não existe uma abordagem padrão documentação e catalogação de projeto de subsistemas, componentes e outros bens reutilizáveis. É importante destacar que existe uma iniciativa de empresas como Microsoft, IBM, ComponentSource e Rational para propor a OMG utilizar o RAS (*Reusable Assets Specification*) como futuro padrão de documentação.

Um outro ponto importante relacionado à confiabilidade do componente é quanto à certificação do componente. Segundo Heineman e Councill (2001), devem ser aprofundadas as questões relacionadas à homologação de componentes, através de entidades independentes de certificação, tal como é realizado pela *Underwriters Laboratories*, que realizam certificação de softwares de segurança críticos. Uma entidade de certificação avalia o processo, requisitos, projeto, implementação, verificação, teste, e validação do componente contra os requisitos iniciais.

Considerando os aspectos discutidos anteriormente sobre a tecnologia de componentes de software e a complexidade e heterogeneidade dos sistemas que são desenvolvidos da composição de componentes, pesquisas devem ser desenvolvidas no sentido de aperfeiçoar os processos de desenvolvimento baseado em componentes, as ferramentas que apóiam estes processos, padrões modelos de infra-estrutura e de documentação de componentes e questões relacionadas à certificação, como forma de garantir a confiabilidade destes sistemas desenvolvidos utilizando esta tecnologia.

4. FARCSOFT - FERRAMENTA DE APOIO À REUTILIZAÇÃO DE COMPONENTES DE SOFTWARE

O processo de desenvolvimento baseado em componentes necessita de ferramentas que auxiliem no armazenamento, busca, divulgação, compartilhamento e entendimento de componentes reutilizáveis para a construção de aplicações de software a partir destes componentes.

Este capítulo tem a finalidade de apresentar a ferramenta proposta neste trabalho; para isso, é inicialmente apresentada a descrição do ambiente da FARCSoft (**F**erramenta de **A**póio à **R**eutilização de **C**omponentes de **Soft**ware), através dos requisitos necessários para a utilização da ferramenta, da definição do escopo e dos requisitos funcionais que a ferramenta contempla e, os tipos de reutilização que implementa.

Além disso, também são apresentadas a especialização do Processo Unificado (JACOBSON; BOOCH; RUMBAUGH, 1998), adotada como processo de desenvolvimento para a construção e as considerações sobre as soluções adotadas em seu desenvolvimento.

4.1 Descrição da FARCSoft

A FARCSoft é uma ferramenta para fornecer suporte ao processo de desenvolvimento de sistemas que emprega reutilização de componentes de software, os quais são armazenados em um repositório e, por meio de mecanismos busca, recuperação e visualização da informação relativa ao componente de software, possibilita o compartilhamento, a divulgação e o entendimento destes componentes de software reutilizáveis no processo de desenvolvimento de sistemas.

É importante destacar que, a FARCSoft não é restrita a um único tipo de processo de desenvolvimento e não é imposta nenhuma restrição em relação ao método ou à técnica para o projeto e a construção do componente de software, podendo ser empregada em diferentes processos utilizando diferentes técnicas para modelar os artefatos produzidos, desde que seja atendido um conjunto de requisitos.

No contexto desta ferramenta, como foi apresentado na seção 2.3.1, um componente de software é definido como um elemento de sistema reutilizável que contém:

- um artefato que executa e disponibiliza a realização de um conjunto de serviços definidos e conhecidos;
- um conjunto de artefatos que disponibiliza a informação necessária para integrar o componente a um sistema de aplicação, artefatos estes que podem ser executáveis ou não, e fornecer suporte durante o ciclo de vida do sistema.

O componente de software, neste caso, pode ser desde um fragmento de código, uma função, uma classe, um *framework*, um pacote executável ou outro elemento de sistema, que deve disponibilizar e executar serviços através de interfaces conhecidas e definidas e documentar a informação que auxilie no entendimento de sua funcionalidade e na forma de integração do artefato reutilizável para a construção de um novo sistema.

Para isso, possui uma base de dados, onde os componentes de software reutilizáveis são armazenados e catalogados, para serem compartilhados entre os projetistas através dos recursos da WEB.

Esta base de dados é um repositório de componentes. No contexto deste trabalho, repositório de componentes de software foi definido como uma base de dados, que deve permitir classificação e armazenamento do componente de software reutilizável e da informação relacionada ao componente.

A Figura 11 apresenta um exemplo de contexto, dentro de um processo de desenvolvimento baseado em componentes, em que um repositório de componentes, como o da FARCSoft, se enquadra.

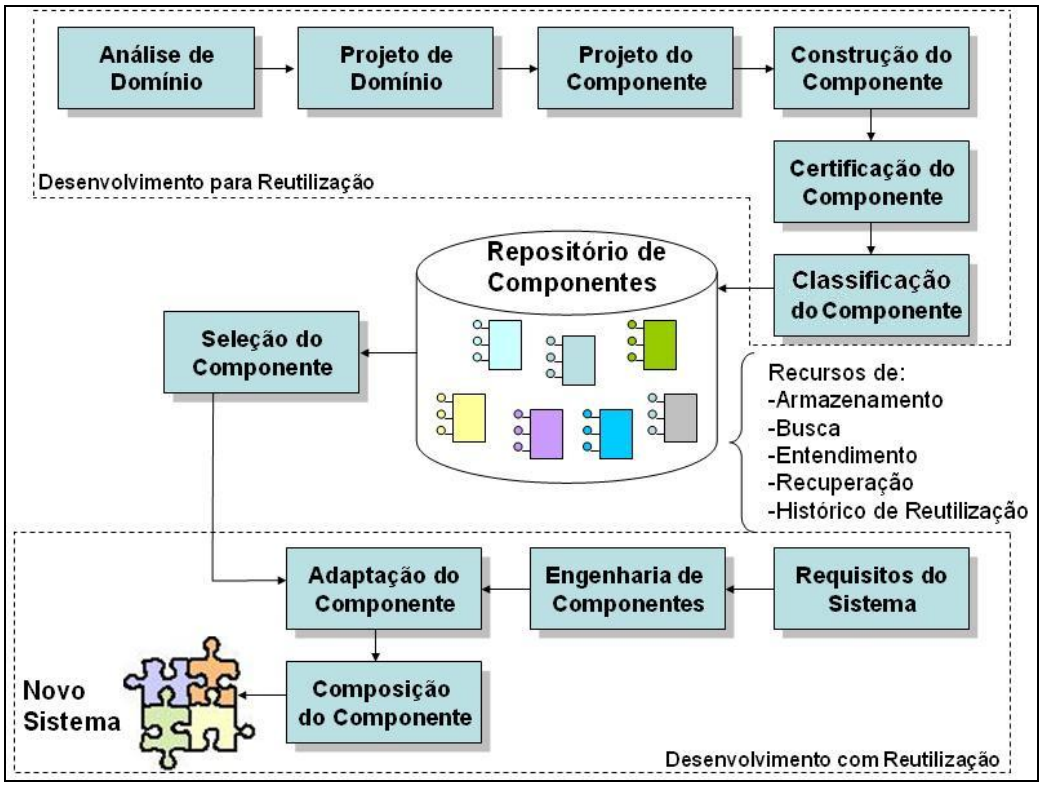

Figura 11: Esquematização da aplicação do repositório no processo de desenvolvimento baseado em componentes.

Os componentes de software são concebidos e construídos, levando-se em conta as características do domínio de aplicação, o que vai tornar a sua reutilização mais eficiente. Após a sua construção, estes componentes devem ser submetidos às atividades de certificação, para que seja garantida a sua qualidade, o que aumenta a confiança para a utilização dos reutilizáveis. Após a certificação, ocorre a classificação do componente segundo o esquema de classificação do repositório e a sua catalogação, juntamente com o conjunto de artefatos relacionados no repositório.

Os componentes de software reutilizáveis serão armazenados em um local que permita que diferentes produtores e consumidores compartilhem os componentes; para isso, o repositório de componentes é um recurso importante para promover o compartilhamento de componentes de software reutilizáveis entre os projetistas que desenvolvem componentes para reutilização e os projetistas que constroem novos sistemas com a reutilização de componentes previamente construídos para este propósito.

Para a utilização eficiente da ferramenta, o conteúdo do repositório deve ser confiável, no sentido de armazenar as versões adequadas de componentes com qualidade. Para isso, é necessária a figura do administrador do repositório, que é a pessoa responsável pela gerência do conjunto de informação e artefatos relacionados ao componente reutilizável, ou seja, responsável pela definição do esquema de classificação, catalogação de componentes reutilizáveis no repositório, e também, pelo controle dos componentes de software no repositório, reportando aos projetistas, usuários do repositório, as ocorrências de novas inclusões de componentes, atualização de versões e problemas encontrados em versões catalogadas no repositório.

Como forma de garantir a qualidade dos componentes armazenados no repositório, também é tarefa do administrador do repositório verificar se os componentes de software candidatos a fazerem parte do repositório de componentes passaram por um processo de certificação estabelecida.

Neste contexto, o objetivo da FARCSoft é armazenar, catalogar e difundir componentes reutilizáveis para construção de novos sistemas, através de recursos para busca e recuperação de componentes, entendimento da sua funcionalidade, armazenamento da documentação relativa ao componente, monitoramento do histórico das reutilizações efetuadas, e notificação de atualizações nos componentes disponíveis no repositório.

Desta forma, um ambiente para apoiar o processo de reutilização de componentes deve disponibilizar recursos para auxiliar nas tarefas de cada pessoa que compõe este processo, baseando-se nos serviços propostos por Apperly (2001), conforma apresentado anteriormente na seção 2.2.4, a FARCSoft oferece suporte aos serviços que estão em destaque na Tabela 10.

Serviços de gerência do componente	
Segmento	**Serviços**
Produção	Divulgar os componentes projetados informalmente ou não documentados
	Utilizar especificações de componentes com ponto de início para projeto
	Divulgar componentes com especificações
	Re-divulgar componentes e especificações
	Notificar consumidores de novos componentes ou problemas
Gerência	Gerenciar a biblioteca do repositório
	Gerenciar os usuários do repositório
	Gerenciar o catálogo
	Assegurar a qualidade dos componentes
	Gerenciar os componentes
	Tornar os componentes disponíveis
	Gerenciar as versões dos componentes
Consumo	Buscar componentes que necessita
	Reportar falhas identificadas
	Especificar componentes para desenvolvimento
	Utilizar e reutilizar especificações de componentes
	Utilizar e reutilizar componentes
	Dispor componentes
	Registrar interesse em componentes
	Receber novas notificações de componentes
	Revisar novos componentes
	Atualizar especificações de componentes e componentes

Tabela 10: Serviços de gerência de componentes que a FARCSoft fornece suporte basedo na Apperly (2001).

4.1.1 Definição do Escopo

A FARCSoft é uma ferramenta desenvolvida para fornecer suporte difusão de componentes de software reutilizáveis através do ambiente WEB e, desta forma, permitir o compartilhamento de um repositório de componentes de software dentro de um ambiente de desenvolvimento e promover a reutilização de componentes de software entre os projetistas.

Os usuários potenciais desta ferramenta são os pesquisadores e os projetistas do LTS (Laboratório de Tecnologia e Software) e os estudantes de pós-graduação e graduação dos cursos de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo.

Os pesquisadores, os projetistas e os estudantes do LTS poderão compartilhar os componentes de software desenvolvidos em seus trabalhos entre si e, desta forma, o repositório terá o papel de centralizar e disponibilizar os componentes reutilizáveis e os seus artefatos.

A tecnologia da WEB vai permitir a independência de plataforma, acesso distribuído, informação compartilhada e facilidade de acesso. Um aspecto importante a ser ressaltado é a segurança de acesso como um dos pontos fundamentais, pois o compartilhamento do repositório será realizado através da infra-estrutura da WEB, que é uma rede pública.

Para isso, existe a necessidade de estabelecer um controle de acesso aos recursos da ferramenta somente aos usuários autorizados que pertencem ao grupo de LTS e, além disso, controlar os recursos que vão ser disponibilizados para cada tipo de categoria de usuário. Isto se torna necessário, uma vez que os recursos de gerência do repositório devem ser utilizados somente pelo usuário que desempenha a função de administrador do repositório garantindo, assim, maior controle sobre a qualidade dos componentes e dos artefatos armazenados e catalogados.

A Figura 12 apresenta os subsistemas da FARCSoft, os diversos tipos de usuários e os respectivos recursos aos quais têm acesso.

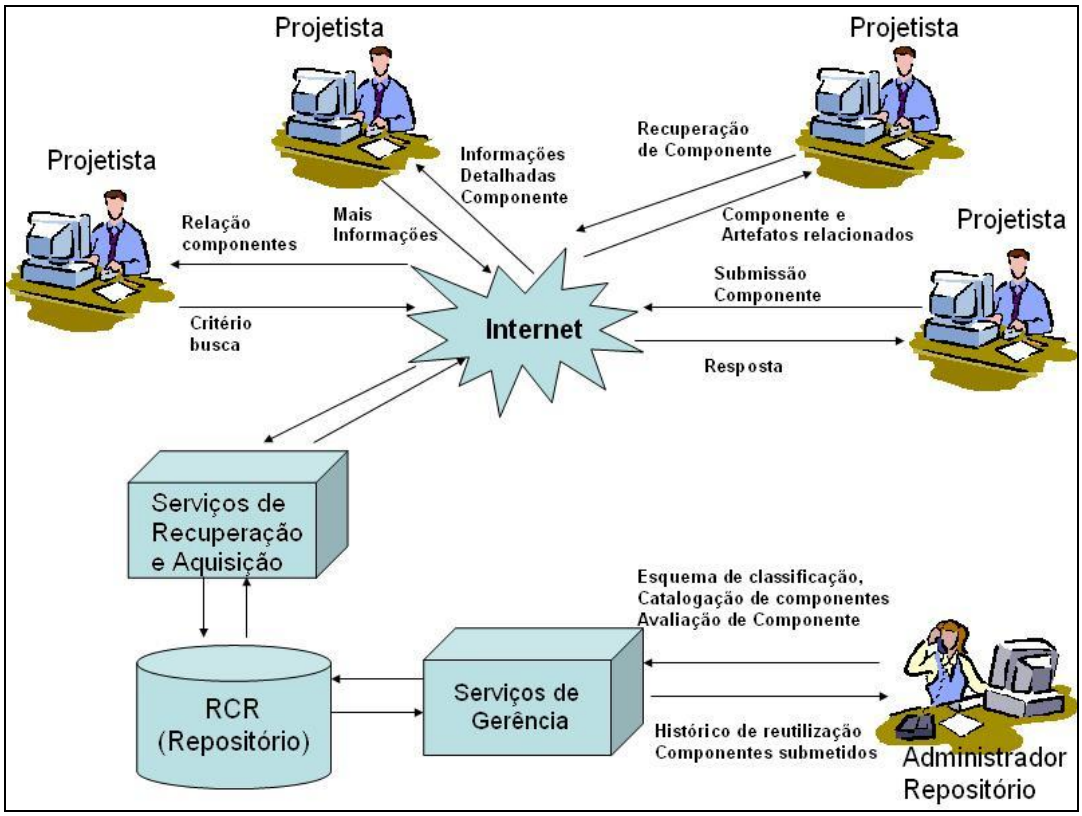

Figura 12: Ambiente de utilização da ferramenta e seus subsistemas componentes.

Através da Figura 12 podem-se observar os recursos do repositório de componentes de software disponíveis aos projetistas:

- Pesquisa: possibilita que o projetista realize buscas por componentes no repositório, através de recursos para estabelecer critérios e parâmetros sobre as características do componente que está sendo procurado. Isto é possível através da definição de um esquema de classificação para o repositório;
- Submissão: possibilita que o projetista submeta seus componentes reutilizáveis para a inclusão no repositório, ou propor atualizações de novas versões de um componente já catalogado no repositório;

- Exploração: permite que o pesquisador consulte informações mais detalhadas dos componentes de software do repositório, descrições e documentação relacionada para auxiliar no entendimento das funcionalidades do componente;
- Recuperação: disponibiliza recursos para auxiliar no processo de extrair o componente de software e os artefatos relacionados necessários para a sua reutilização na construção de uma nova aplicação.

Já o administrador do repositório vai ter disponíveis os seguintes recursos como suporte às atividades de gerência dos componentes de software reutilizáveis no repositório:

- Esquema de classificação: disponibiliza os recursos para criar e manter a estrutura do esquema de classificação dos componentes de software no repositório;
- Catalogação: disponibiliza os recursos para a inclusão do componente de software e das informações relacionadas no repositório e permitir a manutenção do mesmo e a definição do esquema de classificação do componente;
- Validação dos submetidos: disponibiliza os recursos para validar os novos componentes candidatos a fazerem parte do repositório ou novas versões de componentes que já fazem parte do repositório;
- Divulgação: disponibiliza os recursos para que o administrador do repositório divulgue, para os projetistas, informações sobre novos componentes, ou atualização de versões e, até mesmo, erros encontrados em componentes.

A FARCSoft visa atender tanto as atividades dos projetistas como as do administrador do repositório. No ambiente de reutilização que está sendo proposto, o projetista pode ser tanto o que produz componentes reutilizáveis, como o que os consome. O administrador do repositório é a pessoa responsável pela gerência dos componentes de software armazenados no repositório.

Para atender estes dois tipos de usuários optou-se pela divisão da ferramenta em dois subsistemas. Desta forma, um subsistema disponibiliza os recursos para os projetistas e o outro, para o administrador do repositório. O repositório de componentes, denominado RCR (Repositório de Componentes Reutilizáveis) é uma base de dados comum na qual as informações dos componentes de software reutilizáveis estão armazenadas para compartilhamento.

Desta forma, a FARCSoft é composta pelos seguintes elementos:

- Base de dados: armazena os componentes de software e seus artefatos. Este banco de dados é denominado de Repositório de Componentes Reutilizáveis (RCR);
- Subsistema de Gerência do Repositório: permite a manutenção e o controle da informação armazenada no repositório, através de mecanismos de classificação e armazenamento dos componentes de software e seus artefatos relacionados, a validação dos componentes candidatos a serem armazenados no repositório e a divulgação de novos componentes ou versões, ou de problemas detectados;
- Subsistema de Recuperação e Aquisição de Componentes: permite pesquisa, exploração e recuperação de componentes de software armazenados no repositório e também a submissão de novos componentes ou versões de componentes para catalogação no repositório.

4.1.1.1 Repositório de Componentes Reutilizáveis (RCR)

Como foi definido anteriormente, no contexto deste trabalho, o repositório de componentes de software é definido como uma base de dados, cujo conteúdo foi projetado para permitir classificação, armazenamento, busca e recuperação da informação relacionada aos componentes de software, tais como: especificações, modelos relacionados, código, planos e casos de teste.

O Repositório de Componentes Reutilizáveis (RCR) foi implementada através de uma base de dados. A Figura 13 apresenta o modelo entidade relacionamento do RCR.

Apesar do modelo relacional de banco de dados ser considerado mais lento que modelo hierárquico e de rede (LAUNDON; LAUNDON, 2001), sua escolha se baseia nas seguintes considerações:

- Volume de dados do repositório não será tão grande a ponto de ter problemas de desempenho; segundo Poulin (1999), uma biblioteca de software para reutilização chega a ter por volta de 100 elementos;
- Mais flexível em relação à elaboração de consultas, podendo combinar fonte de dados diferentes.

O RCR foi implementado utilizando o banco de dados WinMySQL, que é um banco de dados de software livre e apresenta boa reputação quanto à estabilidade e portabilidade. O RCR vai estar disponível em um servidor WEB e, desta forma, vai ser acessado tanto pelo subsistema de gerência do repositório, como também pelo subsistema de recuperação e aquisição de componentes, promovendo o compartilhamento das informações relacionadas aos componentes reutilizáveis entre os usuários do ambiente.

O conjunto de arquivos relacionados ao componente de software é armazenado em uma estrutura de diretório, no servidor WEB, que possibilite a sua recuperação no momento da extração do componente para reutilização. Deste modo, para o componente, vai ser criada uma pasta que vai permitir distribuir a relação de artefatos de software que o descreve em subpastas de acordo com o tipo de artefato: código, documentação, modelos e outros tipos de documentos.

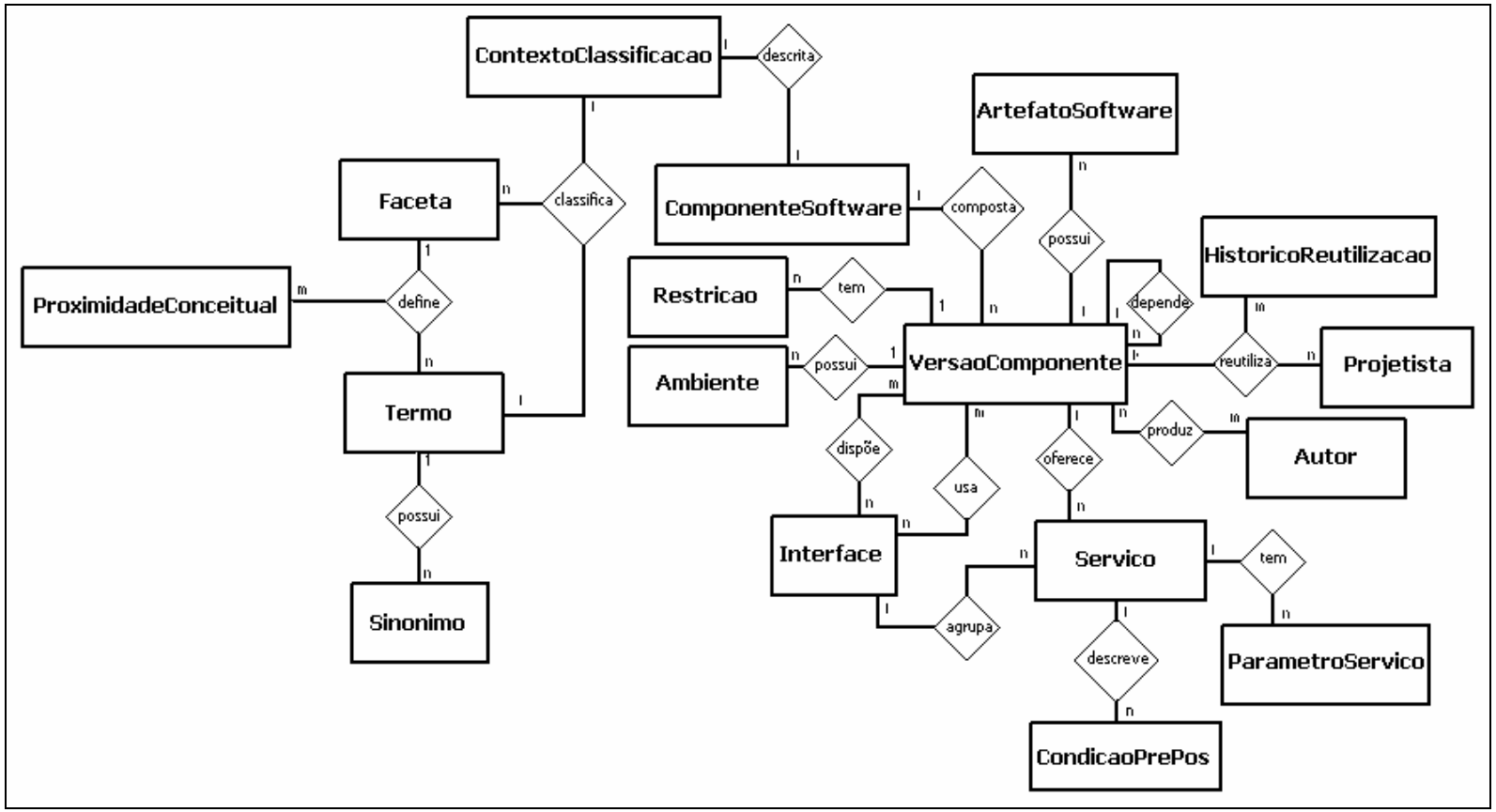

Figura 13: Modelo entidade-relacionamento do RCR.

4.1.1.2 Subsistema de Gerência do Repositório

O subsistema de gerência do repositório não é compartilhado através do ambiente WEB, uma vez que a gerência do repositório será realizada apenas por uma pessoa que exerce o papel de administrador do repositório; por essa razão não existe a necessidade de ser acessada através da WEB, o que preserva também a integridade do repositório.

Deste modo, o subsistema de gerência do repositório deve disponibilizar os recursos necessários para apoiar as atividades do administrador do repositório. Para isso, deve fornecer recursos para as atividades de:

- Manutenção dos usuários do repositório;
- Manutenção do esquema de classificação dos componentes de software reutilizáveis no repositório;
- Manutenção do componente de software no repositório;
- Consulta das informações históricas sobre reutilização de componentes de software;
- Controle de componentes submetidos, por projetistas, para inclusão no repositório, ou seja, disponibilizar recursos para que o administrador do repositório consulte, recupere e avalie os componentes submetidos e a informação relacionada;
- Divulgação de inclusão de novos componentes ou versões ou, problemas encontrados em versões de componentes aos projetistas que utilizam a ferramenta.

A proposta de arquitetura adotada para este subsistema é baseada no modelo em três camadas. A Figura 14 retrata a esquematização da arquitetura que é composta pelas seguintes camadas:

- Camada de interface: implementa as funções de interface com o usuário, composta por um conjunto de janelas que permitem a interação do

administrador do repositório com o sistema, disponibilizando uma interface amigável através dos recursos da linguagem Java;

- Camada de serviços: tem a finalidade de implementar o conjunto de serviços oferecidos pela ferramenta e que manipulam as informações armazenadas no RCR;
- Camada de armazenamento de dados: é formada por um banco de dados relacional, no caso, o WinMySQL, que armazena as informações do repositório de componentes de modo organizado.

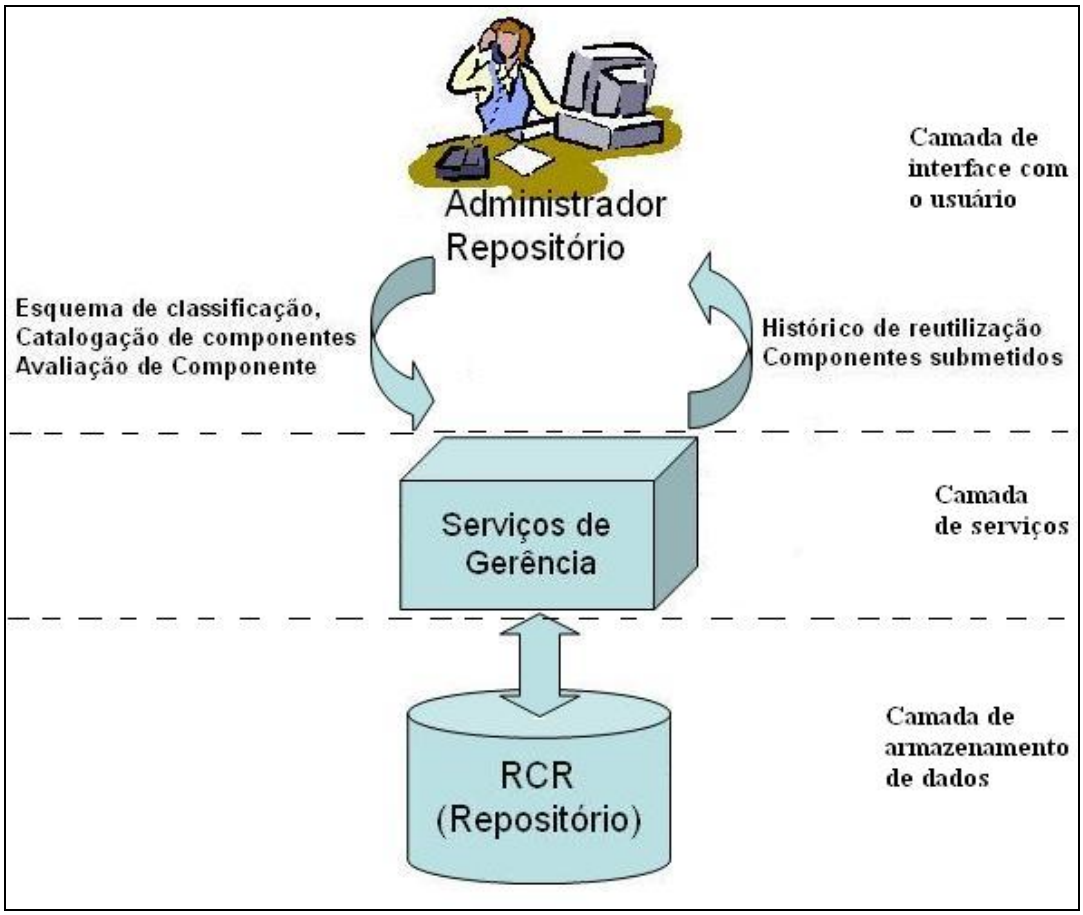

Figura 14: Arquitetura do subsistema de gerência repositório de componentes.

4.1.1.3 Subsistema de Recuperação e Aquisição de Componentes

O subsistema de recuperação e aquisição de componentes de software está sendo desenvolvido para ambiente WEB; desta forma, o acesso é disponibilizado através de navegador WEB acessado através de um *link* na página do LTS.

A finalidade deste subsistema é auxiliar o projetista, através de recursos para definir critérios de pesquisa por um determinado componente, possibilitar a recuperação de um conjunto mais restrito de componentes que satisfaçam este critério, oferecer recursos para auxiliar no entendimento da funcionalidade de um componente selecionado, e permitir a extração do componente e da sua documentação para auxiliar a sua reutilização.
A proposta de arquitetura adotada para este subsistema é baseada no modelo em quatro camadas. A Figura 15 retrata a esquematização da arquitetura que é composto pelas seguintes camadas:

- Camada de interface: implementa as funções de interface com o usuário, composta por um conjunto de páginas que permitem a interação do usuário com o sistema, disponibilizando uma interface amigável através dos recursos da linguagem HTML;
- Camada de comunicação: estabelece o meio de troca de informações entre a camada de interface com o usuário e a camada de serviços. A WEB será utilizada como meio de transmissão de informações entre o usuário e o sistema;
- Camada de serviços: tem a finalidade de implementar o conjunto de serviços oferecidos pela ferramenta e que manipulam as informações armazenadas no banco de dados;
- Camada de armazenamento de dados: é formada por um banco de dados relacional, no caso, a base de dados do RCR que armazena a informação do repositório de componentes de modo organizado.

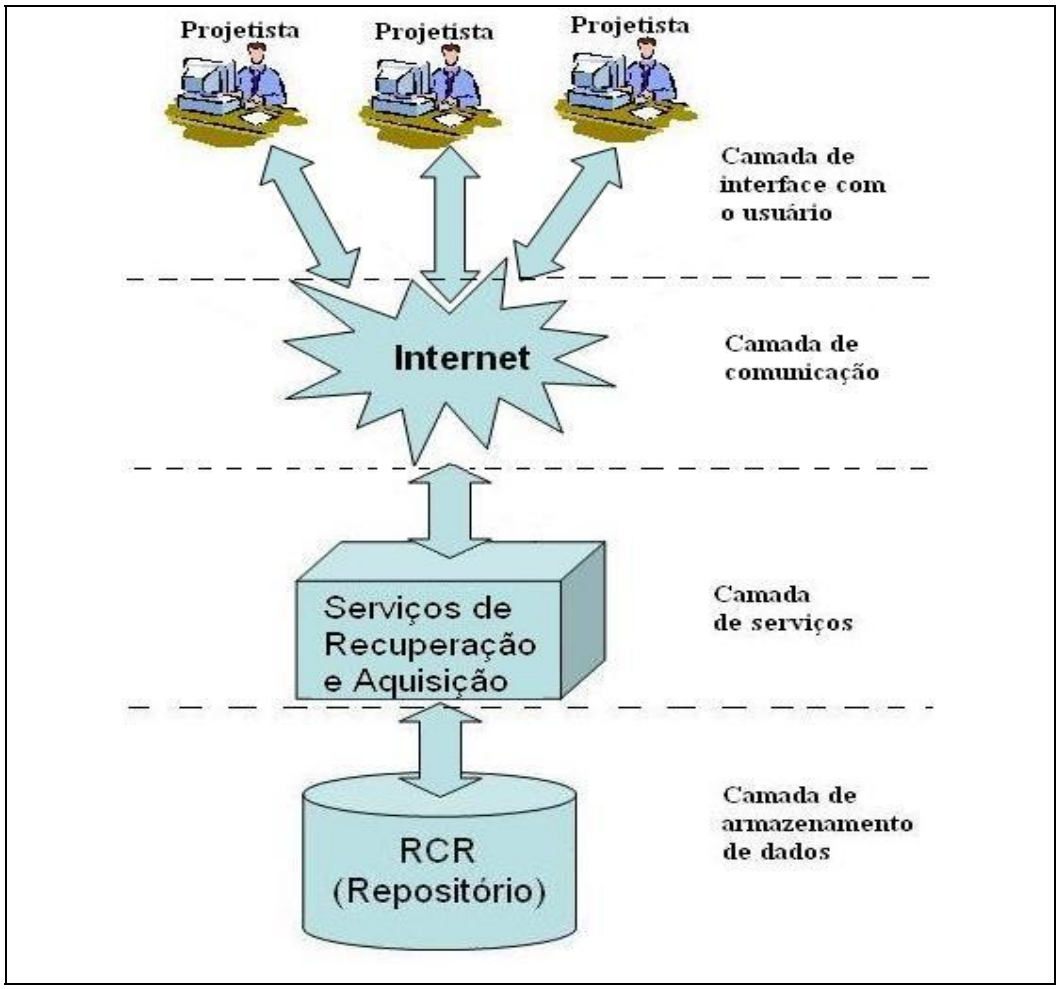

Figura 15: Arquitetura do subsistema de recuperação e aquisição de componentes de software

Para um sistema deste tipo, existem duas propostas mais adequadas: a tecnologia de *JavaServer Pages* (JSP) e a tecnologia de *Servlets*. A tecnologia de *JavaServer Pages* (JSP) possibilita a junção de código Java em páginas HTML, permitindo criar conteúdo dinâmico na página (HALL, 2002). Na tecnologia de *Servlets*, os programas são executados sobre um servidor WEB atuando, como uma camada intermediária entre uma requisição vinda de um navegador WEB ou outro cliente HTTP e a base de dados ou aplicações sobre o servidor HTTP.

A opção pela tecnologia de *JavaServer Pages* foi selecionada devido ao fato de que a tecnologia de *Servlets* requer maior habilidade com programação em Java, e o código

HTML produzido é muito misturado dentro do código Java, dificultando o entendimento. A tecnologia de *JavaServer Pages* é mais amigável para desenvolvimento de aplicações Web, uma vez que, pode-se embutir código Java dentro das páginas HTML do subsistema de recuperação e aquisição de componentes.

É também importante ressaltar que se pode combinar as tecnologias de *Servlets* (através da utilização de *JavaBeans*) e de *JavaServer Pages*, quando for necessário. Isto possibilita uma solução em que os projetistas dos serviços trabalhem com a lógica de negócio em Java, e os projetistas da GUI possam trabalhar com HTML e JSP (HALL, 2002).

Para a camada de serviços vai ser adotada a tecnologia a tecnologia de *Servlets*, para implementar as regras de negócio relacionadas ao subsistema e acesso à base de dados do repositório de componentes.

4.1.2 Tipo de Reutilização suportada pela FARCSoft

O tipo de reutilização, que a FARCSoft contempla, foi definido a partir da proposta apresentada por Prieto-Díaz (1993). A Tabela 11 apresenta as perspectivas de reutilização propostas por Prieto-Diaz (1993) e os tipos de reutilização adotados para o ambiente da FARCSoft estão sombreados para dar destaque.

Perspectiva	**Artefatos**					
Substância	Idéias ou conceitos	Artefatos ou componentes			Procedimentos ou habilidades	
Escopo	Vertical				Horizontal	
Modo	Sistemático				Ad hoc	
Técnica	Composicional				Gerativa	
Intenção	Caixa preta				Caixa Branca	
Produto	Especificações	Arquiteturas	Projetos	Objetos	Textos	Código-fonte

Tabela 11: Tipos de reutilização do ambiente da FARCSoft.

Quanto à perspectiva de substância, a FARCSoft permite a reutilização tanto de idéias ou conceitos, como também de artefatos ou componentes, ou seja:

- Idéias ou conceitos: considerando que o projetista possa consultar as informações referentes a um componente de software para auxiliar no entendimento e na solução de um problema similar;
- Artefatos e componentes: considerando que o propósito do repositório de componentes reutilizáveis é de armazenar as informações sobre o componente de software e os seus artefatos de software.

Para a perspectiva de escopo, o domínio pode ser tanto vertical ou horizontal, ou seja, não está sendo feita restrição quanto à forma como a reutilização ocorre em uma família de sistemas ou entre família de sistemas. O componente de software pode ser específico para uma área de aplicação, como também, pode ser genérico para ser reutilizado em diferentes domínios.

Para a perspectiva de modo, foi adotada a reutilização sistemática, uma vez que, a FARCSoft está sendo desenvolvida para apoiar o processo de desenvolvimento baseado em componentes que emprega reutilização e, para isso, devem ser estabelecidas e seguidas diretrizes e procedimentos, e também deve ser avaliado o desempenho através de métricas.

Para a perspectiva de técnica, foi adotado o tipo de reutilização composicional, ou seja, a reutilização de componentes existentes para a construção de um novo sistema. A reutilização do tipo gerativa implica em geração de código automática que não é o objetivo da ferramenta.

Para a perspectiva de intenção, que se refere ao aspecto da alterabilidade do artefato reutilizável, foi estabelecido o tipo caixa-branca. Na reutilização do tipo caixa-branca, o componente reutilizado pode ser modificado e adaptado; a principal dificuldade é o controle de alterações dos componentes reutilizáveis, o que exige um controle de versões mais formalizado.

O componente de software selecionado para reutilização será extraído do repositório com todo o conjunto de artefatos relacionados a ele, permitindo que o projetista tenha acesso a toda documentação armazenada referente ao componente de software, desde especificação de requisitos até o código. Desta forma, fica o projetista encarregado da decisão sobre a necessidade de alterações ou adaptações para a reutilização do componente na integração em um novo sistema.

A ferramenta não controla as alterações realizadas no componente de software extraído de seu repositório para reutilização. O projetista no momento que extrai o componente do repositório fica responsável pelo uso e por possíveis alterações que possam ser necessárias para a sua reutilização na aplicação que está sendo desenvolvida.

Quanto à perspectiva de produto, que estabelece os produtos de um projeto que poderão ser reutilizados, optou-se por permitir reutilizar todo o produto relacionado ao componente de software.

4.2 Processo de Desenvolvimento

O desenvolvimento da FARCSoft está sendo realizado através do Processo Unificado que foi adaptado às características deste projeto. Apesar da equipe inicial ser de uma pessoa, por se tratar de projeto acadêmico vinculado a uma dissertação de mestrado, a implantação da versão final vai demandar uma equipe maior, o que justifica a escolha desta abordagem de desenvolvimento.

Uma característica importante deste processo é o fato de ser iterativo e incremental, ou seja, as atividades podem ser organizadas em iterações que podem ser compostas de descoberta, entendimento, projeto, implementação e testes. Desta forma, cada iteração pode resultar em uma atualização de versão do software, através da inserção de novos requisitos, permitindo a evolução incremental do software.

Uma outra característica importante deste processo é o fato de ser dirigido a casos de uso e centrado na arquitetura do software, reforçando o desenvolvimento iterativo e incremental, o que fornece suporte ao desenvolvimento baseado em componentes.

Para a sua aplicação no desenvolvimento do projeto da ferramenta deste trabalho, as fases do Processo Unificado foram esquematizadas do seguinte modo, como pode ser mostrado na Figura 16:

- Fase de concepção: definição do escopo da ferramenta a ser desenvolvida através da elaboração da lista de características da ferramenta, de um modelo do processo de negócio e dos casos de usos principais;
- Fase de elaboração: especificação dos requisitos através do detalhamento dos casos de uso e do diagrama de classes da ferramenta e a construção de um protótipo inicial;
- Fase de Construção: construção do protótipo dos subsistemas da FARCSoft;
- Fase de Transição: Avaliação do protótipo da FARCSoft, quanto ao aspecto de representação e de documentação do componente de software, através da aplicação de um estudo de caso.

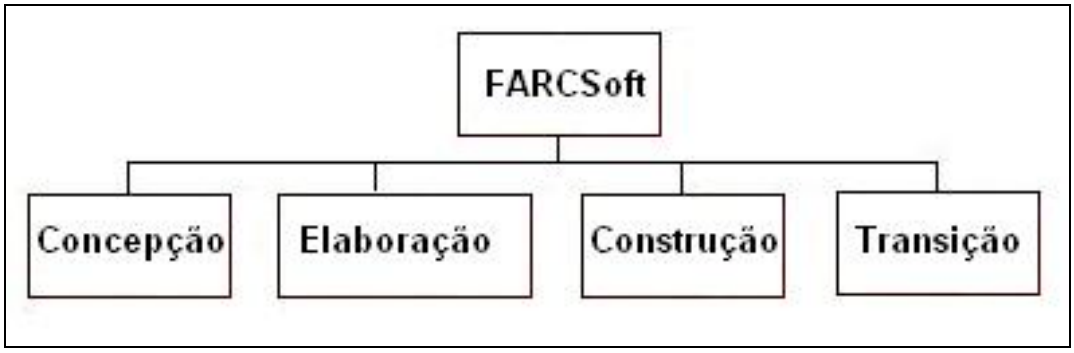

Figura 16: Fases do desenvolvimento da FARCSoft.

As Figura 17 até Figura 20 representam a estruturação do trabalho relacionado ao desenvolvimento da dissertação através de WBS (Work Breakdown Structure).

Durante a fase de concepção, como pode ser observado na Figura 17, foram realizadas as atividades para a eliciação dos requisitos da ferramenta, o planejamento inicial para a sua construção e a preparação do ambiente de desenvolvimento da ferramenta. Também foi realizada a atividade para revisão dos produtos gerados.

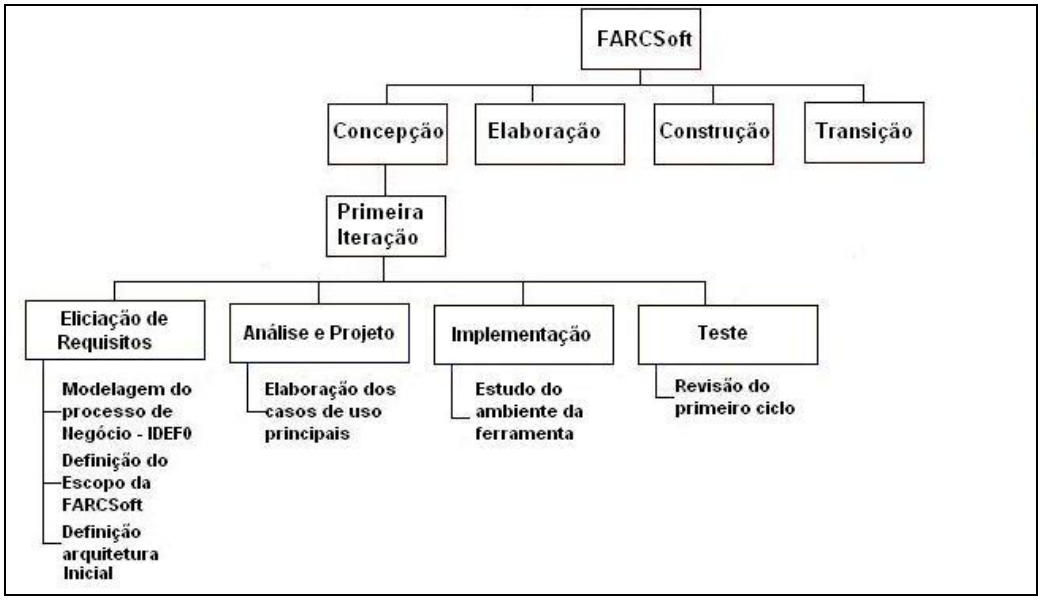

Figura 17: Atividades da fase de concepção

Por meio da modelagem do processo de negócio, foi possível compreender melhor as necessidades do processo de desenvolvimento que emprega a reutilização de componentes de software.

A fase de elaboração foi dividida em duas iterações, como é mostrado na Figura 18. Na primeira iteração foram desenvolvidos a especificação de requisitos, o detalhamento dos casos de uso e o modelo de classes. Neste primeiro momento, foram especificados os requisitos considerando as necessidades do administrador do repositório.

Do mesmo modo, na segunda iteração, foram especificados e modelados os requisitos para contemplar as necessidades do projetista, que utiliza o repositório para buscar componentes para reutilização.

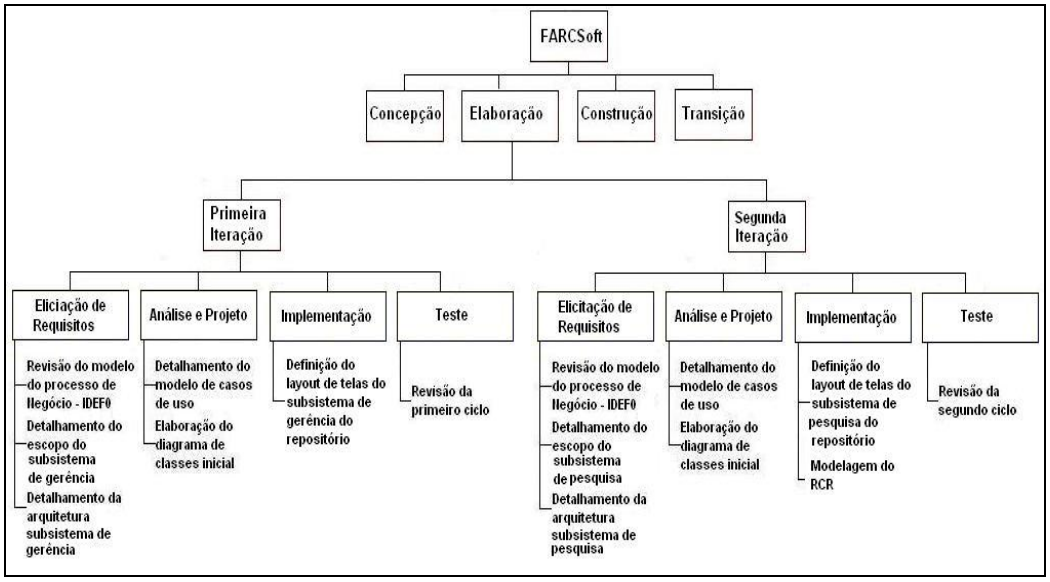

Figura 18: Ciclos e atividades da fase de elaboração.

Além disso, foram consolidados e detalhados os modelos UML como um todo, considerando as necessidades dos dois tipos de usuário da ferramenta. Também foi definida a interface gráfica da ferramenta através do *layout* das telas dos subsistemas da ferramenta, com a finalidade de validar a interface gráfica com o usuário.

A fase de construção também foi dividida em duas iterações, sendo que nesta fase foi implementado um ambiente mínimo para a execução do estudo de caso, como é apresentado na Figura 19. Na primeira iteração foi implementada a base de dados do RCR e na segunda iteração foi implementado o protótipo do subsistema de recuperação e aquisição de componentes de software.

Na fase de transição, como pode ser observado na Figura 20, foi aplicado o estudo de caso para validar a FARCSoft no aspecto de representação e documentação do componente de software.

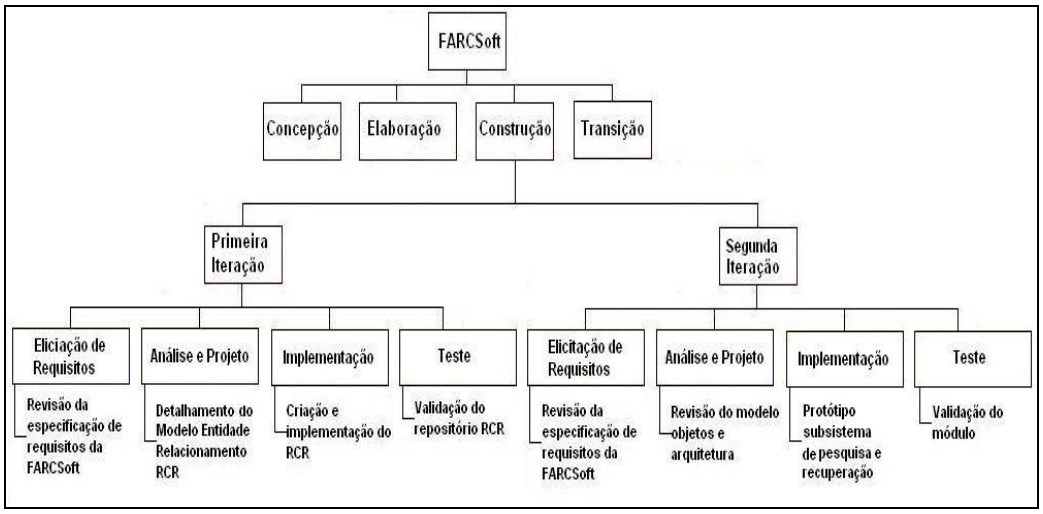

Figura 19: Ciclos e atividades da fase construção da ferramenta.

O processo definido foi adaptado para contemplar as atividades relacionadas com o desenvolvimento da dissertação. A fase inicial do desenvolvimento da FARCSoft tinha a finalidade de validar a capacidade do repositório em representar e documentar diferentes tipos de componentes de software, uma vez que, como foi citado anteriormente, um dos aspectos essenciais na reutilização de um componente é o entendimento de sua funcionalidade e de como integrá-lo em uma nova aplicação.

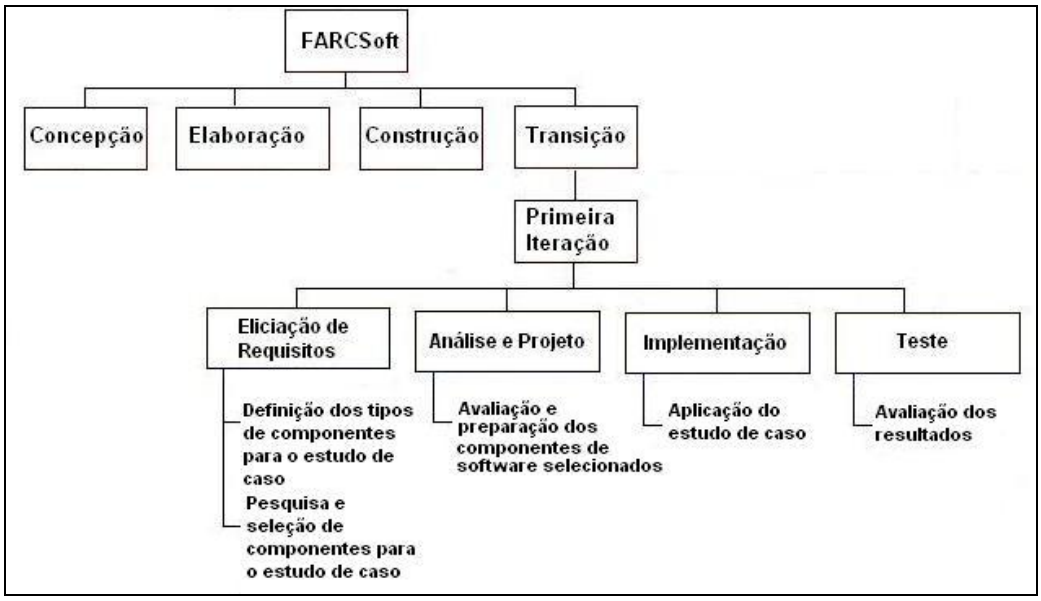

Figura 20: Ciclos e atividades da fase de transição da ferramenta.

A Tabela 12 apresenta a duração de cada fase (em dias) do processo de desenvolvimento que foi aplicado.

Fases do Processo	Tempo gasto
Concepção	5,0
Elaboração	34,5
Construção	20,0
Transição	14,0

Tabela 12: Tempo gastos nas fases (em dias).

Na Figura 21 pode-se observar que um tempo maior foi gasto na fase de elaboração, mas o fato é que neste cronograma não está representado a totalidade e somente a parte que se refere a conclusão do estudo de caso. Desta forma, falta representar a construção do subsistema de gerência do repositório que não foi implementado ainda.

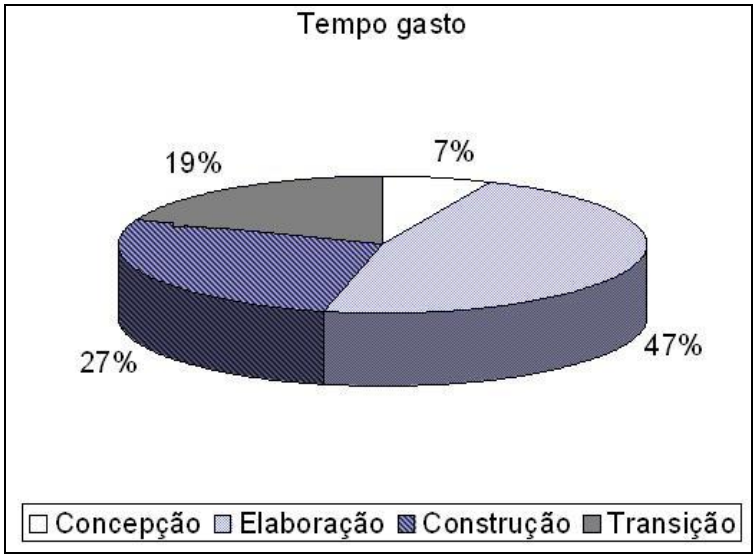

Figura 21: Gráfico comparativo da duração de cada fase.

Durante a fase de concepção do desenvolvimento da ferramenta, foi elaborado o modelo de processo de negócio do processo de reutilização de um componente de software e também os casos de uso principais, com a finalidade de definir os requisitos e a abrangência da FARCSoft.

Na fase de elaboração foram relizados a eliciação dos requisitos, o detalhamento dos modelos e a definição da arquitetura da ferramenta para fornecer suporte ao processo de reutilização de componentes. Neste momento, foi percebido que a necessidade de dividir a ferramenta em dois subsistemas para atender as necessidades do administrador do repositório e dos projetistas separadamente.

O principal fator considerado para adotar esta solução foi quanto à segurança. O papel do administrador do repositório será exercido por uma única pessoa que deve garantir a integridade das informações armazenadas no repositório; desta forma, para evitar possíveis falhas nos mecanismos de controle de acesso, optou-se por deixar o recurso de gerência do repositório fora do ambiente WEB.

Durante a fase de construção, foi implementado um protótipo inicial da ferramenta e implementação da base de dados de componentes, para permitir a aplicação do estudo de caso. Desta forma, foi implementado o protótipo do subsistema de recuperação e aquisição de componentes de software.

A aplicação do estudo de caso e a avaliação dos resultados ocorreram durante a fase de transição em que foi possível avaliar a capacidade da ferramenta em representar e documentar diferentes tipos de componente de software. Nesta fase foram executadas as seguintes atividades:

1. Definição dos tipos de componentes de software para compor o estudo de caso;
2. Pesquisa e seleção dos componentes de software para compor o estudo de caso;
3. Avaliação e preparação dos componentes de software para a catalogação;
4. Aplicação do estudo de caso;

5. Avaliação dos resultados.

A atividade de teste, em todas a fases, teve a finalidade de rever os artefatos gerados nas atividades anteriores e realizar possíveis acertos necessários na especificação da ferramenta e nos modelos gerados.

4.3 Considerações sobre as Soluções Adotadas para a Construção da FARCSoft

Como foi apresentado anteriormente, a FARCSoft foi dividida em dois subsistemas, que são: subsistema de gerência do repositório e o subsistema de recuperação e aquisição de componentes. Estes dois subsistemas acessam o Repositório de Componente Reutilizáveis (RCR) que contém uma base de dados única e compartilhada e que centraliza as informações relativas aos componentes de software.

Conforme apresentado na seção 4.1.1, cada um destes subsistemas agrupa um conjunto de serviços disponibilizados a cada tipo de usuário que utiliza este ambiente: projetista (produtor ou consumidor de bens reutilizáveis) e o administrador do repositório. A visão geral da arquitetura de software da ferramenta é apresentada na Figura 22.

Cada um destes subsistemas é formado por um conjunto de pacotes que auxiliam na execução das tarefas para atender os serviços solicitados, que são:

- Esquema de classificação: disponibiliza recursos, para a manutenção da estrutura do esquema de classificação dos componentes do repositório;
- Catalogação de componentes: disponibiliza recursos, para a manutenção das informações do componente de software e dos artefatos relacionados;
- Controle de Versão: permite o controle das diferentes versões do um mesmo componente de software armazenado no repositório;

- Análise de estatística de reutilização: disponibiliza os recursos para permitir a análise de estatísticas sobre a reutilização dos componentes de software armazenados no repositório;
- Controle de Acesso: administra os usuários e controla o acesso às funções da ferramenta;
- Visualização de Componentes: disponibiliza recursos para a consulta detalhada das informações armazenadas dos componentes de software;
- Pesquisa por componentes: disponibiliza recursos, para realizar pesquisas na base de dados do repositório sobre componentes para reutilização em uma nova aplicação;
- Submissão de componentes: permite aos projetistas submeter propostas de componentes, para catalogação no repositório;
- Extração de componente: permite a extração de um componente e de toda a informação relacionada a ele, para auxiliar no processo de entendimento de como reutilizar o componente em uma nova aplicação.

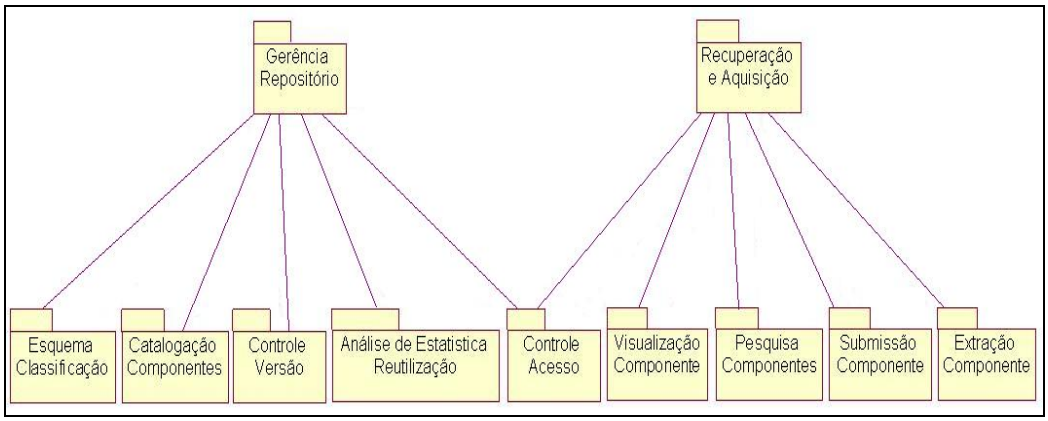

Figura 22: Visão geral da FARCSoft

Nas seções 4.3.1 e 4.3.2 são apresentadas o modelos UML relevantes para a compreensão do trabalho e as soluções adotadas para o desenvolvimento do protótipo da FARCSoft.

4.3.1 Subsistema de Gerência do Repositório

O subsistema de gerência do repositório tem, como principal função, fornecer recursos ao administrador do repositório para realizar as tarefas de manutenção, controle e administração dos componentes de software reutilizáveis armazenados no repositório. Estes recursos estão apresentados no diagrama de casos de uso da Figura 23.

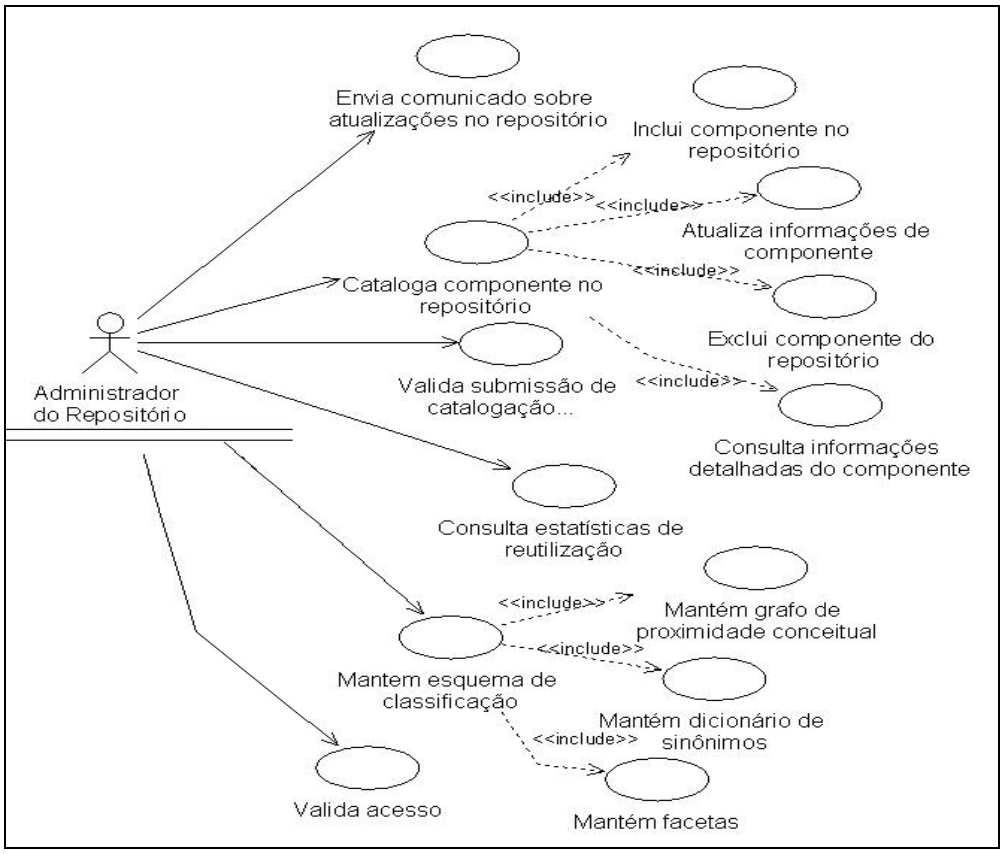

Figura 23: Diagrama de casos de uso do subsistema de gerência do repositório

O caso de uso **Valida acesso** tem a finalidade de garantir o acesso às funções do subsistema de gerência do repositório apenas para o administrador do repositório, este papel deve ser desempenhado por uma única pessoa para garantir uma maior integridade das informações armazenadas no repositório.

A finalidade do caso de uso **Valida submissão de catalogação de componente** é de permitir, ao administrador do repositório, consultar e recuperar os componentes e as informações relacionadas que foram submetidos para inclusão no repositório pelos projetistas. Desta forma, ele pode recuperar as informações relacionadas aos componentes para análise e avaliação.

Os componentes candidatos a fazerem parte do repositório que foram aprovados e certificados são catalogados no repositório através do caso de uso **Cataloga componente no repositório**. Através deste caso de uso, é possível realizar a inclusão de novos componentes, consulta e atualização das informações do componente e exclusão de um componente do repositório.

Um outro recurso disponibilizado para o administrador do repositório é o caso de uso **Consulta estatística de reutilização de componentes** que permite consultar os dados históricos sobre a reutilização do componente para avaliar sua viabilidade em ser mantido no repositório.

O caso de uso **Envia comunicado sobre atualizações no repositório** permite, ao administrador do repositório, enviar mensagens aos projetistas, que são usuários da FARCSoft, comunicando possíveis atualizações realizadas no repositório, tais como: inclusão de novos componentes de software ou novas versões de um componente já catalogado, erros detectados ou correções realizadas em componentes catalogados e outros comunicados.

O caso de uso **Mantém esquema de classificação do repositório** tem a finalidade de auxiliar o administrador de repositório a definir a estrutura de classificação do componente dentro do repositório, para facilitar a busca e recuperação de componentes no repositório. Na seção 4.3.1.1 é discutido este assunto mais detalhadamente.

4.3.1.1 Método de Classificação, Armazenamento, Busca e Recuperação dos Componentes de Software

A classificação é uma atividade necessária para auxiliar no processo de busca e recuperação de componentes de software em uma base de armazenamento. Desta forma, entre os métodos apresentados na seção 2.2.6, o método de classificação por facetas (PRIETO-DÍAZ e FREEMAN, 1987) foi o escolhido para a classificação dos componentes de software armazenados no repositório da FARCSoft.

A escolha do método de classificação por facetas, para a FARCSoft, resultou devido aos seguintes motivos:

- Apresenta maior flexibilidade, permitindo a expansão dos elementos do esquema de classificação;
- Possibilita a criação de relações complexas através da combinação de facetas e termos;
- Permite o tratamento de sinônimos com mais facilidade que os outros métodos apresentados
- Permite estabelecer uma relação de similaridade entre os termos de uma faceta, através do grafo de proximidade conceitual.

Por outro lado, vários pesquisadores da área de reutilização afirmam que o problema de busca e recuperação de informação permanece ainda uma questão em aberto, pois as diversas soluções propostas não oferecem a combinação adequada de eficiência, precisão, interface amigável e generalidade [(MILI et al., 1998) apud (VITHARANA; ZAHEDI; JAIN, 2003)].

Considerando este motivo, e também que, o estudo de métodos de classificação não ser o foco principal deste trabalho, foi escolhido o método de classificação em facetas.

Este método tem sido referenciado e tem sido base de outras propostas de classificação de informação encontradas na literatura sobre reutilização [(GIOVANI;

MELNIKOFF, 1996), (DAMIANI; FUGINI; FUSASCHI, 1997), (MCCLURE, 1998), (CASANOVA; STRAETEN; JONCKES, 2003), (VITHARANA; ZAHEDI; JAIN, 2003)], o que justifica a sua adoção no presente trabalho.

Como foi definido na seção 2.2.6, o método de classificação por facetas é baseado na análise de um assunto em função de seus termos básicos para caracterizá-lo através de facetas ou classes elementares básicas. As facetas são consideradas como perspectivas, visões ou dimensões de um domínio de aplicação em particular (PRIETO-DÍAZ; FREEMAN, 1987). Assim, o assunto em um domínio é descrito pela síntese destas classes ou facetas básicas.

Usualmente as facetas são ordenadas na sua apresentação da esquerda para a direita, baseadas no grau de importância observada. Os componentes das facetas são então classificados, sintetizando quais termos das facetas serão utilizados na classificação (FRAKES; POLE, 1994). A Figura 24 apresenta o modelo de classes do esquema de classificação do subsistema de gerência do repositório.

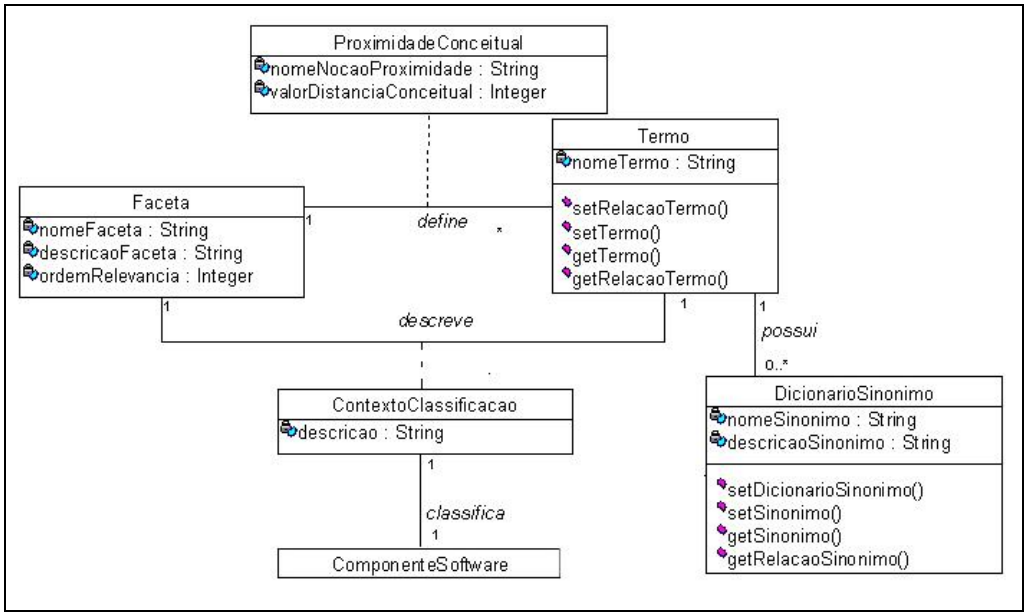

Figura 24: Diagrama de classes esquema de classificação.

No caso da classificação em facetas, o método de busca consiste na seleção de termos das facetas para se chegar aos componentes que satisfaçam esta seleção. Desta forma, as facetas fornecem as listas pré-definidas de termos que são apresentadas aos usuários para continuar a formulação da pesquisa.

O grafo de proximidade conceitual tem a finalidade de medir a proximidade de conceito entre os termos de uma faceta, permitindo estabelecer um grau de similaridade entre os conceitos, ou seja, quanto menor a distância conceitual entre os termos, maior é a similaridade de conceitos. Desta forma, durante a busca por componentes que satisfaçam um determinado critério de pesquisa, o sistema pode recuperar também os componentes descritos através de conceitos similares ao informado na busca.

Para isso, o subsistema de gerência do repositório deve fornecer os recursos necessários para auxiliar o administrador do repositório na tarefa de definir o esquema de classificação em facetas. A Figura 25 retrata, através do diagrama de seqüência, o processo de definição do esquema de classificação por facetas.

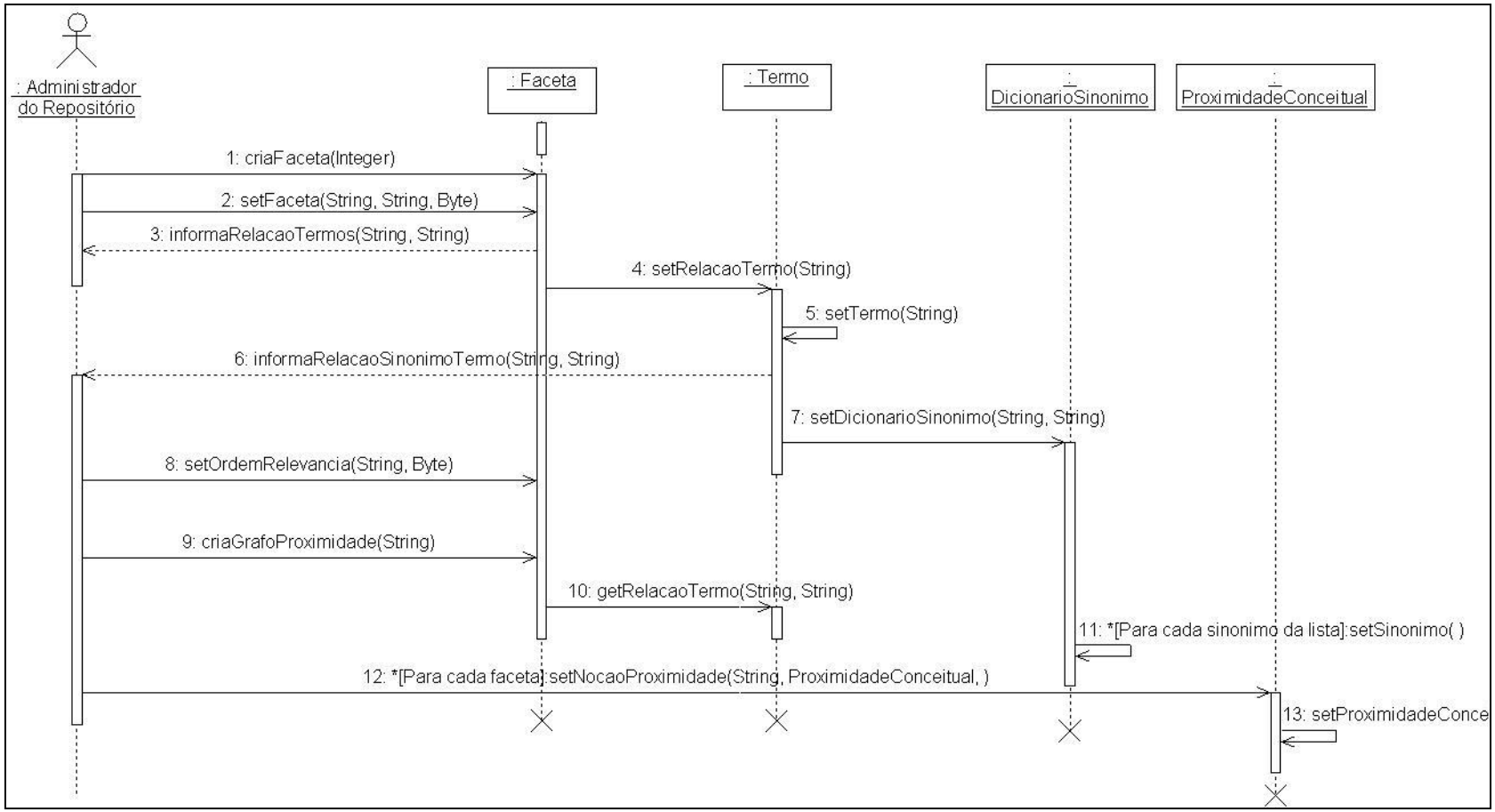

Figura 25: Diagrama de seqüência para criar esquema de classificação.

De forma resumida pode-se descrever o processo de criação do esquema de classificação do repositório através da seguinte seqüência de passos (extraído da descrição do caso de uso **Mantém esquema de classificação**):

Processo de criação do esquema de classificação

1. O caso de uso se inicia quando o Administrador do Repositório acessa página de manutenção do esquema de classificação selecionando a opção ***Definição do Esquema de Classificação***;
    1.1 O administrador do repositório inicia o primeiro passo acionando a opção ***Definição das facetas*** no qual deve informar o número de facetas, informações sobre cada uma e determinar a ordem de relevância para exibição;
2. O administrador do repositório inicia o próximo passo acionando a opção ***Definição dos termos***, devendo informar a relação de termos que descreve cada faceta;
    2.1 O administrador do repositório inicia o próximo passo acionando a opção ***Criação do gráfico de proximidade conceitual***, quando será definido o grafo de proximidade conceitual entre os termos de cada faceta;
3. O sistema apresenta o esquema de classificação que foi definido;
4. O administrador do repositório confirma o esquema de classificação definido;
5. O sistema armazena as informações da estrutura do esquema de classificação no repositório.

A solução adotada é baseada na proposta apresentada por Pietro-Diaz (1991), que é uma solução amplamente citada e adotada em trabalhos de bibliotecas e repositórios de bens reutilizáveis.

4.3.1.2 Solução adotada para a catalogação do componente de software

Para auxiliar o processo de busca e recuperação de componentes de software armazenados no repositório, é necessário o armazenamento de um conjunto de informações relativas aos componentes de software, que descreva as suas características e funcionalidade, promovendo, desta forma, o entendimento de como utilizá-los.

Para isso, durante o processo de catalogação, o componente de software deve ser classificado, de acordo com esquema de classificação estabelecido para o repositório, o que possibilita a busca e a recuperação do componente de software de modo mais eficiente. Depois que o componente foi classificado, ele é catalogado, ou seja, é armazenado o conjunto de informações que tem a finalidade de descrever a funcionalidade do componente, as questões relativas ao ambiente, como usar, e outros aspectos que auxiliam no entendimento do componente. A Figura 26 apresenta o diagrama de classes referente a catalogação de componentes de software.

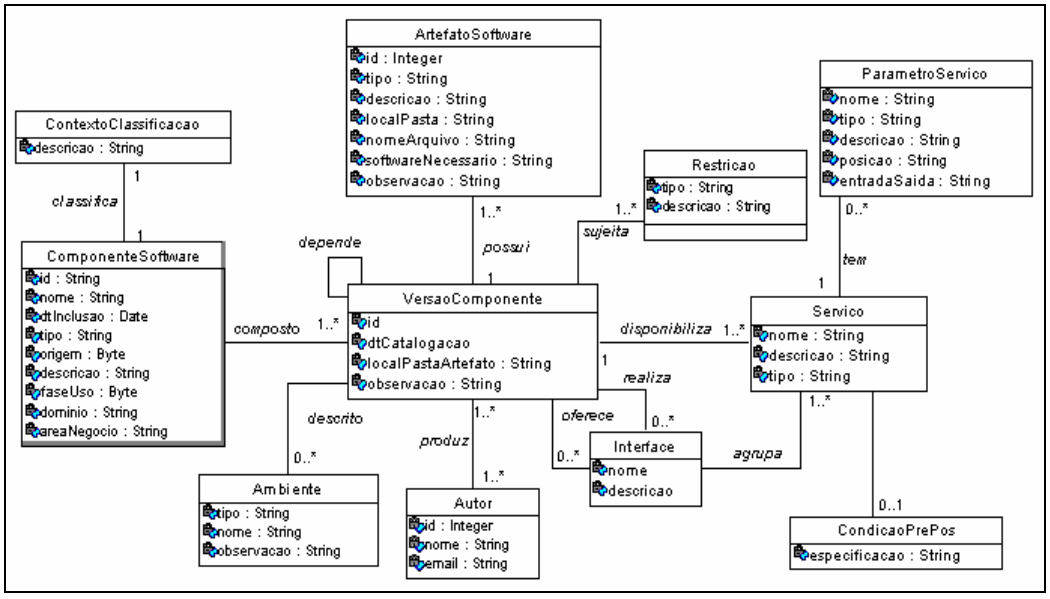

Figura 26: Diagrama de Classes referente à catalogação do componente de software

O modelo para descrição do componente de software proposto neste trabalho, foi baseado nas propostas apresentadas por McClure (1997), por Yacoub, Ammar e Mili (1999) e na proposta da Rational Software (2001) para especificação de bens reutilizáveis.

As características que qualificam o componente de software foram agrupadas em classes que representam o meta-modelo da estrutura necessária para contemplar o conjunto de informações que descrevem o componente de software, ou seja, um modelo que procura ser adequado aos diferentes tipos de componentes que podem armazenados no repositório. As classes deste meta-modelo são as seguintes: **ComponenteSoftware**, **ContextoClassificacao**, **VersaoComponente**, **ArtefatoSoftware**, **Autor**, **Ambiente**, **Restricao**, **Interface**, **Servico**, **ParametroServico** e **CondicaoPrePos**.

A classe **ComponenteSoftware** especifica a estrutura comum do componente de software entre as diferentes versões do mesmo, descrevendo de modo resumido sua funcionalidade e finalidade, ou seja, o que descreve o componente e auxilia no seu entendimento dos problemas que aquele componente resolve, que são:

- id: identifica de modo único do componente de software;
- nome: especifica o nome do componente;
- dtInclusao: especifica a data em que o componente de software foi inserido no repositório, permite calcular a sua idade, refletindo a estabilidade e maturidade;
- tipo: especifica o tipo do componente de software, tais como: subrotina, classe, padrão de projeto, DLL, executável, *framework* ou sistema;
- origem: caracteriza o componente pela sua origem de fornecimento, que pode ser: desenvolvimento interno, fornecedor externo, componente de prateleira ou outros;
- descricao: descreve, em alto nível, o propósito do componente, ou seja, funcionalidade oferecida expressando o que o componente faz;

- faseUso: especifica a fase do desenvolvimento em que o componente de software pode ser reutilizado, tais como, implementação, projeto ou especificação;
- dominio: especifica o domínio que é atendido pelos serviços do componente, como por exemplo, indústria, automação bancária, seguros, contabilidade e outros;
- areaNegocio: especifica a área de negócio que os serviços do componente atendem, tais como controle de reservas, inventário, cadastro cliente, sinistro de seguros e outros.

Um outro aspecto importante no processo de catalogação de um componente de software é a definição do esquema de classificação. A classe **ContextoClassificação** estabelece a ligação com o conjunto de classes que definem o esquema de classificação para o componente de software, permitindo, desta forma, a busca e recuperação de componente de software no repositório.

Um componente de software pode possuir uma ou várias versões, e cada versão pode oferecer funcionalidade diferente, ambientes ou soluções de implementação diferentes. Desta forma, a classe **VersaoComponente** representa este aspecto, como os seguintes atributos:

- id: identifica o número da versão do componente de software, sendo que a versão original do componente tem a numeração menor;
- dataCatalogacao: especifica a data em que a versão foi incluída o repositório, permite calcular a sua idade, refletindo a estabilidade e maturidade;
- localPastaArtefato: especifica a localização da pasta em que serão armazenados os artefatos relacionados à versão do componente de software;
- observacao: especifica um conjunto de instruções que auxiliam na compreensão de como integrar o componente em uma nova aplicação.

Uma vez que, um componente de software pode ter diferentes versões e cada versão pode ser construída por diferentes autores, adotou-se a classe **Autor** para representar este aspecto:

- id: especifica a identificação do autor;
- nome: especifica o nome do projetista responsável pela criação desta versão;
- email: identifica o e-mail do projetista autor desta versão para contato.

A classe **Ambiente** tem a finalidade de descrever as informações e as restrições relacionadas à linguagem de programação, sistema operacional, banco de dados e outros aspectos. Para isso, possui os seguintes atributos:

- tipo: especifica o tipo do ambiente, tais como, linguagem de programação, sistema operacional, banco de dados e outros;
- nome: identifica o nome do ambiente;
- observacao: especifica as características e restrições relativas ao ambiente.

Já a classe **Restricao**, tem a finalidade de especificar tanto as restrições comerciais ou legais sobre a utilização do componente de software, ou seja, compra, licença especial ou permissão requerida.

A associação reflexiva na classe **VersaoComponent**e ,denominada ***depende***, estabelece a relação de componente que um determinado componente de software necessita para realizar a sua funcionalidade.

A descrição mais detalhada e o conjunto de artefatos relacionados ao componente de software são tratados pela classe **ArtefatoSoftware**. Através desta classe são especificados conjunto de informações, documentos e arquivos relacionados ao componente de software. Para isso, possui os seguintes atributos:

- tipo: identifica o tipo de artefato relacionado ao componente de software, que pode ser: especificação requisitos, projeto, modelos, código fonte ou executável e outros tipos de artefatos;
- descricao: especifica mais informações que descrevem os artefatos de software relacionados ao componente, tais como: finalidade, soluções adotadas e outros comentários;

- localPasta: identifica a localização do arquivo no servidor do repositório para auxiliar no processo de extrair as informações do componente de software do repositório;
- nomeArquivo: especifica o nome do arquivo do artefato de software;
- softwareNecessario: descreve o software necessário para manipular as informações contidas no artefato;
- observacao: relaciona qualquer comentário ou conjunto de instruções para utiliza o artefato de software caso seja necessário.

As classes **InterfaceComponente**, **OperacaoInterface**, **ParametroOperacao** e **CondicaoPrePos** descrevem os serviços oferecidos por um componente e suas respectivas versões do mesmo, possibilitando um melhor entendimento de quais serviços oferecidos por um componente e de como acionar um serviço oferecido pelo mesmo.

4.3.2 Subsistema de Recuperação e Aquisição de Componentes de Software

O subsistema de recuperação e aquisição de componentes de software tem a finalidade de realizar a interface entre projetista que desenvolve componente para reutilização com o repositório, e o projetista que os reutiliza com o repositório de componentes reutilizáveis.

Para isso, este subsistema deve fornecer recursos para que os projetistas submetam componentes para catalogação no repositório, realizem pesquisas na base de dados do repositório para recuperação de componentes e a coleta, pelo sistema, de informações referentes à reutilização de um componente contido no repositório.

O ambiente WEB foi escolhido para a construção do subsistema de recuperação e aquisição de componentes, com a finalidade de promover o compartilhamento do RCR (Repositório de Componentes Reutilizáveis) de forma mais heterogênea e

permitindo um compartilhamento mais amplo entre projetistas que podem estar separados fisicamente.

Inicialmente, o acesso será restrito aos pesquisadores e alunos ligados ao LTS do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da USP, mas pretende-se expandir o acesso ao repositório a outros projetistas e pesquisadores no futuro.A Figura 27 apresenta, através do diagrama de casos de uso, os recursos oferecidos por este subsistema.

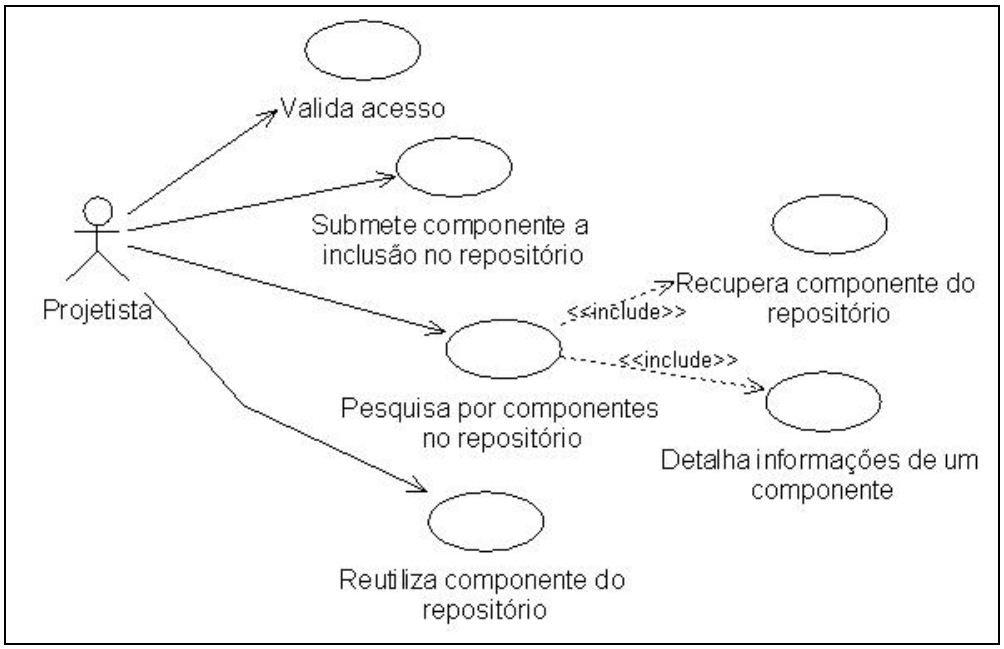

Figura 27: Diagrama de casos de uso do subsistema de recuperação e pesquisa de componentes no repositório.

Por meio deste subsistema, o projetista pode propor a inclusão de novos componentes para o administrador do repositório, pesquisar e recuperar componentes de software e, além disso, possibilitar ao sistema coletar as informações sobre cada reutilização de componente que é realizada. A Figura 28 mostra a esquematização do diagrama de classes referente a este subsistema.

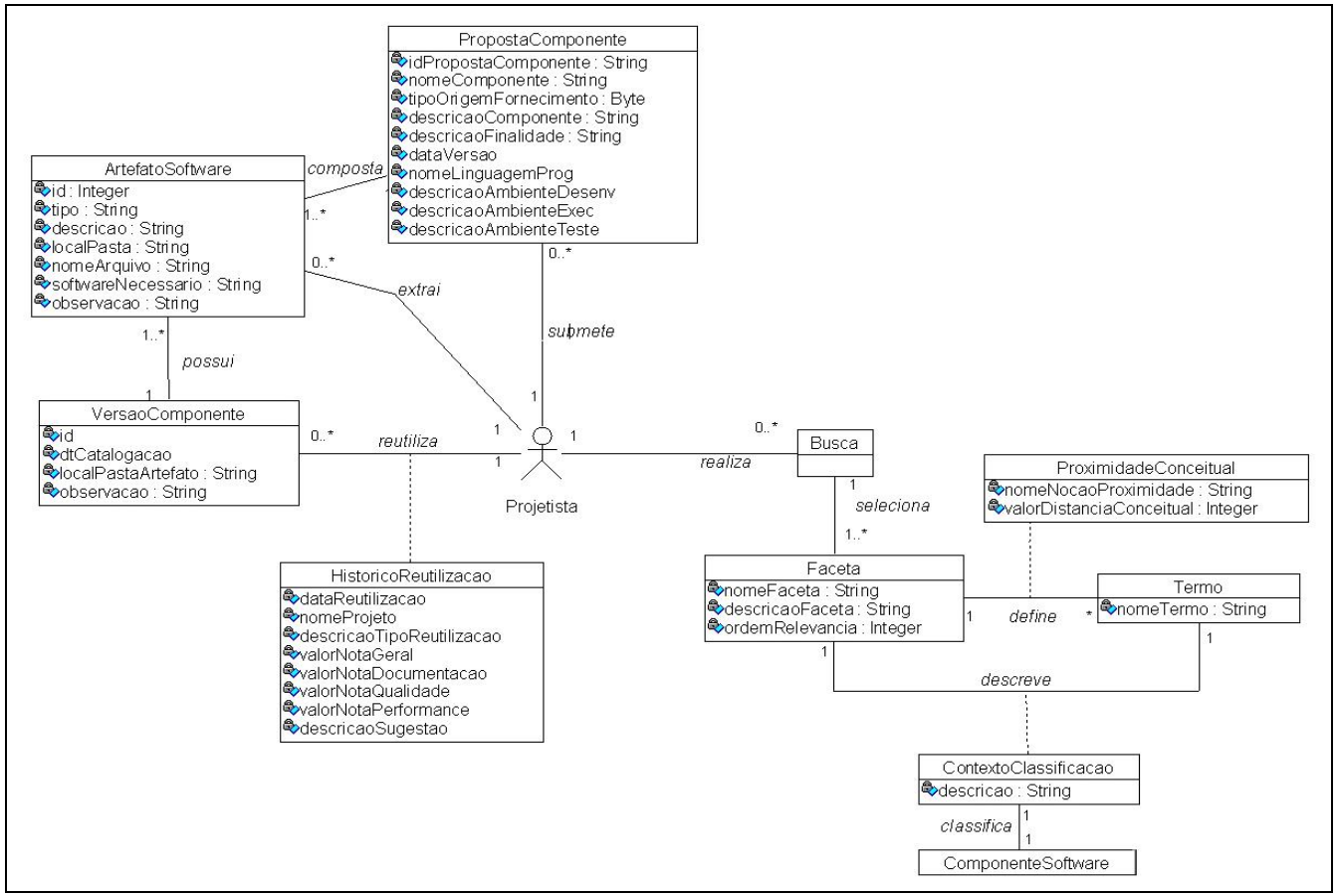

Figura 28: Diagrama de classes do subsistema de recuperação e aquisição de componentes reutilizáveis.

O processo de pesquisa e recuperação de um componente de software se inicia quando o projetista seleciona a partir de cada faceta os termos que deseja pesquisar no repositório.

Para auxiliar a construção de um critério de busca para formular uma pesquisa no repositório, o projetista deve selecionar, em cada faceta, um termo para refinar a busca por um determinado tipo de componente. Desta forma, após formular a busca e submetê-la ao sistema, é apresentada, na página de pesquisa, uma relação de componentes que atendem os critérios especificados na pesquisa.

A relação de componentes apresenta um conjunto de informações, de forma resumida, sobre cada componente com o intuito de auxiliar o projetista a ter um entendimento inicial sobre a funcionalidade de cada componente. Sendo assim, um outro aspecto importante deste subsistema é fornecer recursos para auxiliar no entendimento da funcionalidade de um componente candidato a ser reutilizado. Deste modo, é disponibilizado um recurso que permite o projetista visualizar todas as informações de um determinado componente de modo mais detalhado.

O projetista seleciona os componentes que deseja visualizar as informações de modo mais detalhado e solicita a recuperação destas informações. Como resultado desta requisição, o sistema irá exibir uma página que agrupa toda informação que descreve e auxilia no entendimento das funcionalidades, comportamento, questões de ambiente, especificação de interface e outras informações que auxiliam na compreensão das funcionalidades e de como reutilizar o componente.

De forma resumida pode-se descrever o processo de pesquisa e recuperação por componentes no repositório através da seguinte seqüência de passos (extraída da descrição do caso de uso **Pesquisa por componentes no repositório**):

Processo de criação do esquema de classificação

1. O projetista acessa página de formulação de consulta através do *Browser* selecionando a opção de ***Pesquisa no Repositório;***

2. O sistema apresenta a relação de facetas do esquema de classificação do repositório de componentes reutilizáveis;
3. O projetista elabora a consulta através da seleção dos termos em cada faceta do esquema de classificação de componentes no repositório;
4. O sistema reformula as relações de termos de cada faceta de acordo com os termos selecionados nas facetas anteriores;
5. O projetista repete o passo 2 até finalizar a elaboração da consulta;
6. O projetista, após a seleção da facetas, submete a consulta formulada pressionando a opção ***Busca***;
7. O sistema executa solicitação da busca acionando o caso de uso **Executa pesquisa no repositório**, que realiza a busca por componentes de software armazenados no repositório que atendam os parâmetros solicitados na pesquisa;
8. O sistema apresenta o resultado da consulta solicitada.

A Figura 29 retrata o processo de pesquisa e recuperação através do diagrama de seqüências.

4.4 Considerações sobre a representação do componente de software na FARCSoft

Nesta seção são apresentadas algumas considerações sobre a representação do componente de software no repositório quanto ao aspecto da catalogação dos diferentes tipos de informação que descrevem o componente e a proposta inicial para das facetas para o esquema de classificação do repositório.

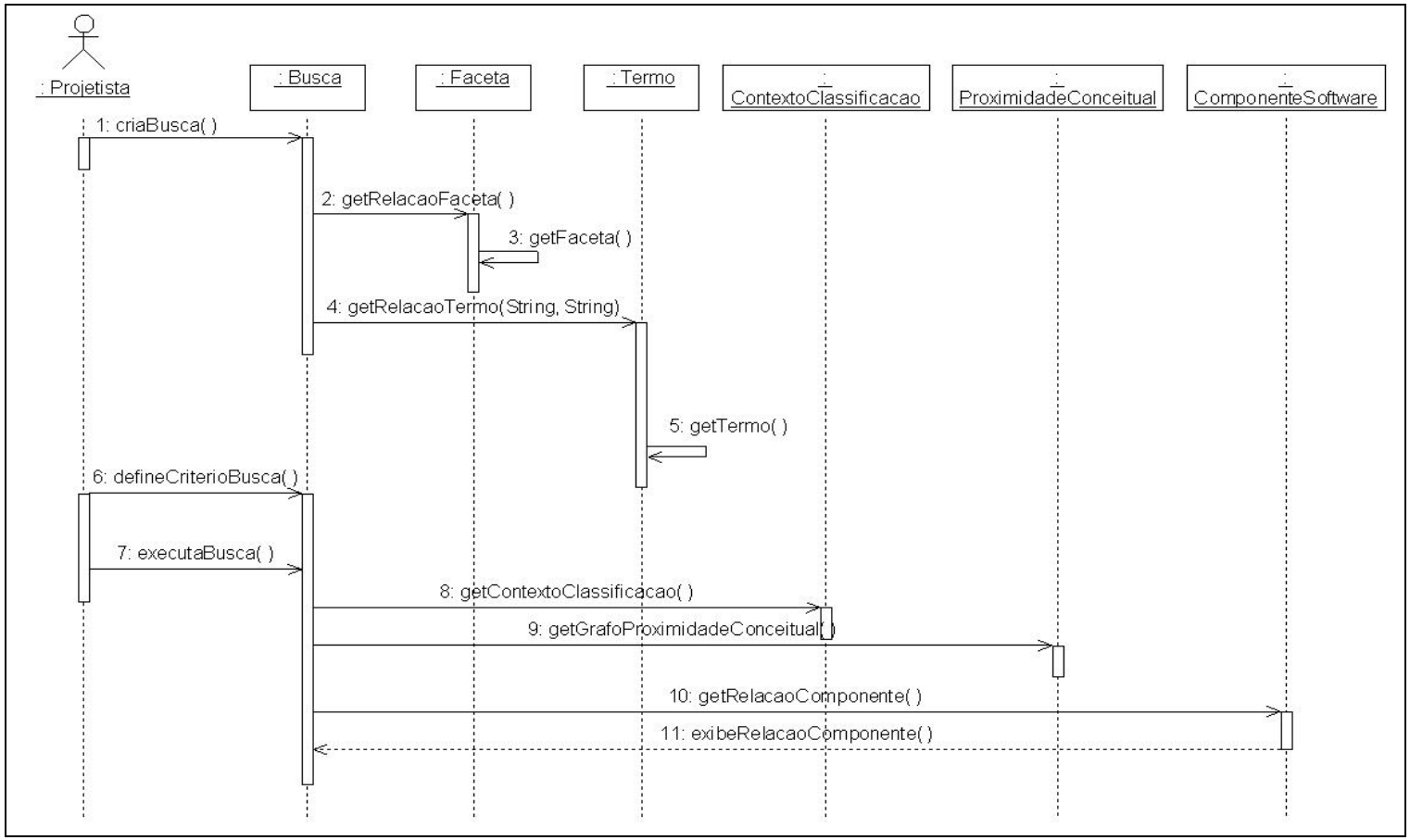

Figura 29: Diagrama de seqüência do processo de pesquisa e recuperação de componentes

4.4.1 Representação de Diferentes Tipos de Informação

É importante analisar algumas considerações a respeito da representação do componente de software com o intuito de validar o meta modelo do RCR, que visa mapear toda a informação relativa a um componente de software em uma estrutura de uma base de dados.

Um componente de software que implementa sua funcionalidade em uma sub-rotina ou uma classe, pode ser considerado a unidade básica de informação a ser representada por meio do meta modelo do RCR. Este tipo de componente demonstrou a necessidade de contemplar os seguintes elementos de informação do RCR, que são:

- Descrição geral da subrotina: as informações que descrevem em termos gerais o objetivo, a finalidade do componente, nome e a idade que permitem identificar e localizar as informações do componente através das classes **ComponenteSoftware**, **VersaoComponente**;
- Propriedade intelectual: informações do projetista que desenvolveu o componente para possibilitar contato através da classe **Autor**;
- Descrição do ambiente: especificação de restrições e considerações com relação ao ambiente do componente, como por exemplo, linguagem de programação, sistema operacional ou banco de dados, que são representados pela classe **Ambiente**;
- Descrição dos serviços: informações da relação de serviços oferecidos pelo componente e as informações que necessita para ser executado, que são representados através das classes **Servico** e **Parametro**;
- Descrição dos artefatos de software: especificação de um ou mais artefatos de software relacionado ao componente, tais como, código fonte, exemplo de código ou especificação de utilização; representados através da classe **ArtefatoSoftware**.

A Figura 30 mostra através do diagrama de classes a unidade básica de informação.

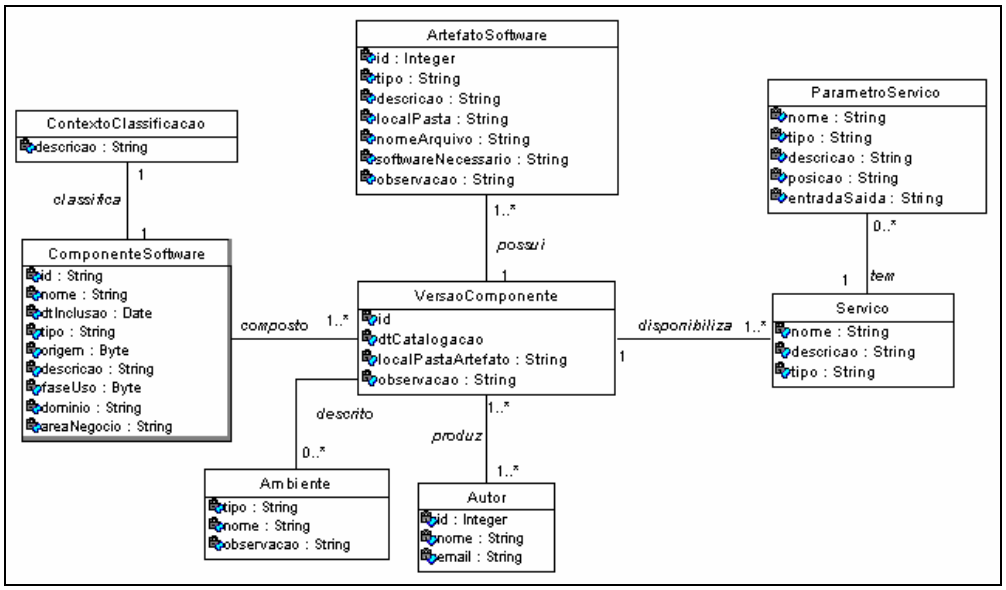

Figura 30: Unidade básica de informação de um componente de software.

Pode-se observar que, uma subrotina ou classe simples, como uma unidade básica de informação do meta modelo deve ter documentado sua funcionalidade e finalidade, existir em pelo menos uma versão, possuir um responsável pela sua construção, descrever o ambiente em que se insere, descrever o serviço que oferece e conter pelo menos um artefato de software que pode ser um algoritmo, código fonte ou até mesmo executável.

Já um componente de software do tipo de um padrão de projeto, tem os mesmos elementos de uma unidade básica de informação, mas necessita representar a relação de dependência que pode possuir com outros padrões de projeto. A representação da dependência de um componente de outros componentes é contemplada por meio do relacionamento "**depende**" que estabelece a dependência entre os componentes de uma versão. A Figura 31 apresenta a representação do relacionamento de dependência.

É importante ressaltar que, a dependência entre componentes não é restrita a apenas componentes do tipo padrão de projeto, ela pode estar presente em pacotes de subrotinas e de classes, componentes UML ou em pacotes de sistemas.

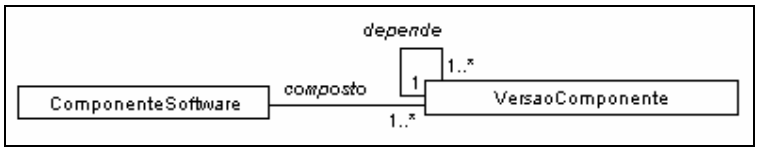

Figura 31: Diagrama de classes do relacionamento de dependência entra componentes.

A representação da interface, que um componente pode oferecer ou que pode realizar, pode ser necessário a representação para um componente construído por meio da UML. Este tipo de componente, além da unidade básica de representação, precisa documentar as interfaces que oferece, representando também os serviços que agrupa nesta interface; e as interfaces que realiza de outro componente. E também, pode precisar documentar as pré e pós condições associadas a determinado serviço. Esta representação está prevista através das classes VersaoComponente, Interface, Servico, ParametroServico e CondicaoPrePos e os relacinamentos entre as mesmas, como é mostrado na Figura 32.

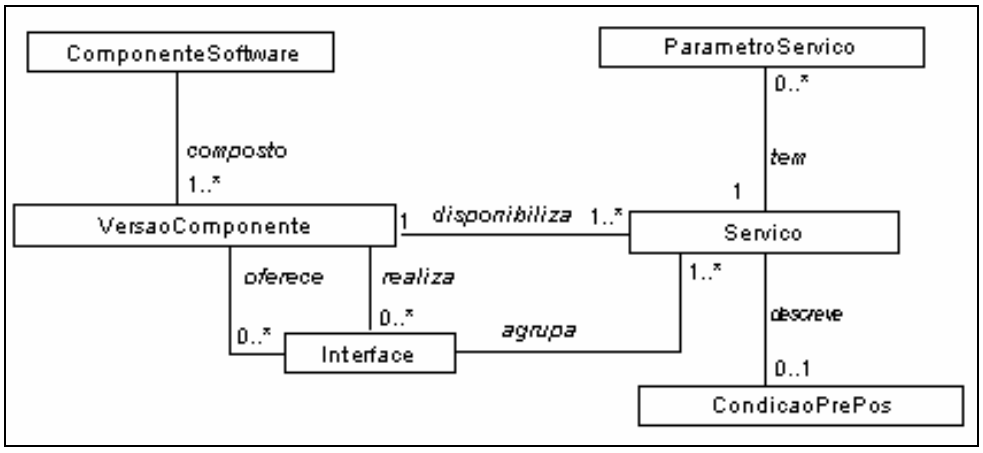

Figura 32: Diagrama de classes para representação do conceito de interface.

Portanto, pode-se notar que o modelo proposto é capaz de representa diferentes formas de componente de software através de um meta modelo que permite representar diferentes tipos de informação.

Conforme foi discutido na seção 4.1.1.3, diante das diferentes necessidades de representação da informação, para cada tipo de componentes de software, a interface gráfica deve ser adequada a cada tipo de necessidade. Portanto, a alternativa mais adequada, é a adoção de uma interface gráfica dinâmica que permita visualização somente da informação é relevante para cada tipo de componente de software e da informação que estiver catalogada no repositório.

Para isso, o módulo de pesquisa e recuperação de componentes de software da FARCSoft está sendo implementado utilizando HTML e a tecnologia de JaveServer Pages (JSP) e Servelts. Esta plataforma permite o desenvolvimento de uma página com o conteúdo mais dinâmico e com mais eficiência para desenvolver soluções baseadas na Web.

4.4.2 Proposta das Facetas para o Esquema de Classificação

Uma vez que, foi validado o modelo de representação do componente de software para auxiliar na tarefa de entendimento de sua funcionalidade e de como reutilizá-lo na integração a um novo sistema, é necessário estabelecer os mecanismos para localizar o componente de software no repositório.

O esquema de classificação permite a definição de um método para a categorização do componente de software reutilizável, de modo que, possibilite a busca e a localização de um componente de um modo mais fácil e rápido. Como foi dito anteriormente, o esquema de classificação adotado nesta proposta do ambiente da FARCSoft, foi o esquema de classificação em facetas, conforme o apresentado por Prieto Diaz (1991).

Considerando a aplicação do estudo de caso e os resultados obtidos referente aos diferentes tipos de componentes que podem ser catalogados no repositório, foi elaborada a seguinte relação de facetas que devem auxiliar na classificação dos componentes no repositório:

- Descrição: conjunto de palavras que descrevam a finalidade do componente, que tipo de problema ele resolve;
- Domínio: especifica a área que é atendida pelos serviços oferecidos pelo componente, como por exemplo, instituições financeiras, bancário, hotelaria, indústria, seguros ou acadêmicos;
- Aplicação: especifica qual a finalidade dos serviços oferecidos dentro do domínio do componente, como por exemplo, dentro do domínio de hotelaria, pode-se considerar, reservas, *check-in, check-out*, ou alocação de quartos;
- Tecnologia: especifica o ambiente tecnológico a que o componente de software pode ser utilizado;
- Tipo: especifica o tipo do componente de software, ou seja, sub-rotina, classe, pacote, *framework* ou sistema;
- Técnica: especifica técnica aplicada para a construção do componente, que pode ser desde um tipo de algoritmo ou método como análise estruturada, orientação objetos, UML, Catalisys ou qualquer outro conceito associado à técnica de construção do componente.

A elaboração do critério de busca no repositório será baseada nas seis facetas apresentadas anteriormente. O projetista poderá formular a busca selecionando os termos de cada faceta, podendo não selecionar critérios para todas as facetas, desta forma, construindo uma pesquisa mais abrangente.

5. ESTUDO DE CASO: AVALIAÇÃO DA REPRESENTAÇÃO DO COMPONENTE DE SOFTWARE NO AMBIENTE DA FARCSOFT

A reutilização de um componente de software na construção de um novo sistema é uma tarefa que exige o conhecimento e compreensão de vários aspectos relativos ao componente candidato à reutilização. Para isso, deve ser disponibilizado ao projetista o conhecimento para auxiliar, tanto no entendimento de sua funcionalidade e em como reutilizar o componente de software, quanto no julgamento se atende ou não os requisitos do sistema que está sendo desenvolvido.

O objetivo deste capítulo é avaliar a capacidade de representação e documentação do conjunto de informação relacionada aos componentes de software, disponibilizada pela FARCSoft, para auxiliar o entendimento e a compreensão da funcionalidade e da forma de reutilização destes componentes para a construção de um novo sistema de software.

Uma vez que, a proposta da FARCSoft é não estar vinculada a uma metodologia de desenvolvimento de software, técnica, método ou ambiente CASE, e que seu objetivo é permitir o compartilhamento e difusão de componentes de software construídos em um ambiente de pesquisa, empregando tecnologias diferentes e por grupos diferentes de pesquisa na área de engenharia de software.

Para isso, foi selecionado um conjunto de diferentes tipos de componentes de software, juntamente com o respectivo conjunto de informações adicionais. Os dados destes componentes foram mapeados conforme a proposta da FARCSoft, e o resultado deste experimento, foi analisado para avaliar a adequação do repositório. A metodologia de avaliação, a sua aplicação e os resultados obtidos estão também apresentados neste capítulo.

5.1 Metodologia de Avaliação

O estudo de caso consiste na avaliação do conteúdo do RCR, que foi especificado na seção 4.3.1 para armazenar o seguinte conjunto de informações de um componente:

- Descrição geral: especifica o propósito, funcionalidade oferecida, tipo, origem de fornecimento, fase do processo de desenvolvimento que se aplica, domínio e área de aplicação;
- Versão do componente: especifica a versão do componente a que se refere as informações específicas de cada versão;
- Autoria: especifica a propriedade intelectual sobre o componente, ou seja, a responsabilidade de sua construção;
- Ambiente: descreve as exigências relativas ao ambiente, como por exemplo, linguagem de programação, sistema operacional, banco de dados, ferramentas Case e outros;
- Restrição: especifica as restrições relativas a aspectos técnicos, comerciais ou legais;
- Artefatos de Software: especifica a relação de documentos associados ao componente, tais como: especificação de requisitos, arquitetura, modelos, planos e casos de teste ou qualquer outro documento associado ao desenvolvimento do componente independentemente do método, técnica ou ferramenta;
- Dependência: especifica a relação de outros componentes que são necessários para o componente ser reutilizado adequadamente;
- Interfaces do componente: descreve as interfaces que um componente pode oferecer e realizar, também quais serviços agrupa;
- Serviços oferecidos: especifica os serviços que um determinado componente oferece;
- Parâmetros do serviço: especifica a relação de parâmetros necessários para execução de um serviço, seu tipo de dado e se é de entrada ou de saída;
- Condição pré/pós do serviço: descreve a especificação descritiva da condição pré e pós de um serviço.

A metodologia aplicada para a realização deste estudo de caso, consistiu do seguinte conjunto de atividades:

1. Definição dos tipos de componentes de software para compor o estudo de caso procurando cobrir uma variedade significativa;
2. Busca e seleção dos componentes de software existentes para descrevê-los de acordo com o conteúdo proposto para o repositório;
3. Avaliação e preparação dos componentes de software para a catalogação, ou seja, verificar se o componente selecionado possui uma documentação adequada para auxiliar no entendimento, e, se necessário, extrair, formatar e reproduzir os artefatos de software relacionados ao componente selecionado;
4. Catalogação dos componentes de software, ou seja, mapear o conjunto de informações associado a cada tipo de componente de software, selecionado para compor o estudo de caso, para a estrutura definida para o RCR;
5. Avaliação dos resultados.

5.1.1 Definição dos Tipos de Componentes de Software de Utilizados

Conforme a definição dada na seção 4.1.2, o componente de software, no contexto do ambiente da FARCSoft, pode ser qualquer elemento de sistema de software que possa ser reutilizado, podendo ser uma classe, um subsistema, um pacote, um padrão de projeto, uma subrotina, uma DLL ou uma API. De uma forma geral, pode ser qualquer elemento de sistema que defina uma seqüência de comandos que descrevem um processamento, seja através de uma linguagem de programação ou não.

Para exercitar a capacidade de representação e documentação da informação relacionada aos diferentes tipos de componente de software no RCR, foram selecionados os componentes de software dos seguintes tipos:

- Subrotina: corresponde à reutilização da unidade de código fonte da programação convencional;

- Classe: corresponde à reutilização da unidade de código da programação orientada a objetos;
- Pacote de Classes: corresponde à reutilização de um conjunto de classes que dependem umas das outras, para executar seus serviços;
- Padrão de projeto: corresponde à reutilização de artefato da fase de projeto;
- Componente UML: corresponde à reutilização de componente de software construído utilizando a UML;
- Sistema ou subsistema: corresponde à reutilização de uma aplicação que empacota um conjunto de serviços.

5.1.2 Busca e Seleção dos Componentes de Software

Uma vez definidos os tipos de componentes a serem utilizados para a avaliação, fez-se a atividade de busca e seleção de componentes de software para catalogá-los no repositório. Optou-se por componentes existentes, em vez de desenvolver os componentes, para verificar a capacidade do repositório de comportar os componentes de diversas fontes. Os componentes foram procurados nos seguintes meios:

- Biblioteca pessoal;
- Livros sobre o assunto;
- Bibliotecas de componentes de código livre na Internet;
- Sites especializados em componentes;
- Sites de Universidades.

5.1.3 Avaliação e Preparação dos Componentes de Software

Os meios citados anteriormente contêm um número relativamente grande de componentes, que foram avaliados em relação à qualidade da documentação

disponibilizada, à funcionalidade e à origem de fornecimento. Muitos componentes foram descartados por falta de documentação ou pela documentação de má qualidade, o que dificultou o entendimento de sua funcionalidade e a forma de reutilização.

Para os componentes considerados aprovados, foi realizada a sua preparação para catalogação no repositório. Para isso, foi necessário estudar e, se necessário adequar a documentação disponível sobre o componente para extrair as informações, formatar e reproduzir os artefatos de software do componente selecionado a inclusão no repositório. Os componentes de software selecionados para compor o estudo de caso são os seguintes:

- Subrotina simples extraída de biblioteca pessoal para validação de código nacional de pessoal jurídica (CNPJ);
- Classe simples para validação de usuário e senha - Login.java (SANTOS, 2003);
- Pacote de classes para cálculo de vencimentos (DEITEL; DEITEL, 2003);
- Padrão de projeto *BusinessDelegate* (ADATIA et al., 2001);
- Componente UML para reserva de quatros de uma rede de hotéis (CHEESMAN; DANIELS, 2001);
- Sistema de simulação de sistema de elevador (DEITEL; DEITEL, 2003);
- Projeto de sistema de elevador em análise estruturada (YOURDON, 1990);

5.1.4 Catalogação dos componentes de software

Para a aplicação do estudo de caso para avaliação da capacidade de representação da informação do componente de software no repositório foi executada a seguinte seqüência de passos:

1. Mapeamento de toda a informação relacionada a cada componente de software para o meta modelo do repositório

2. Validação e ajustes da sua estrutura do RCR para armazenar o conjunto de informações relacionadas a cada tipo de componente de software.

5.1.5 Avaliação dos Resultados

O mapeamento do conjunto de informações para o meta modelo do repositório permitiu fazer pequenos ajustes na sua especificação. Na maior parte dos casos, o meta modelo atendeu as necessidades para a descrição dos componentes selecionados.

5.2 Representação dos Componentes no Repositório da FARCSoft

O Anexo 1 apresenta a documentação da catalogação de toda a informação relacionada aos componentes de software que fizeram parte do estudo de caso, que possibilita visualizar como a informação foi armazenada na base dados do repositório.

Para a representação das informações relativas aos componentes de software selecionados para compor o estudo de caso, foi elaborado o diagrama de instâncias ou objetos para cada elemento do estudo de caso, que corresponde à catalogação do componente de software no repositório. Desta forma, foi possível analisar, para os diferentes tipos de componente de software, como seria a estrutura de representação da informação do componente de software durante o processo de catalogação do mesmo no repositório.

Nesta seção são apresentadas a descrição dos componentes selecionados e a análise da sua representação no repositório, ou seja, os casos de sucesso, as soluções adotadas e as dificuldades encontradas e sua solução.

5.2.1 Subrotina: Valida_CNPJ

A subrotina **ValidaCNPJ** tem a funcionalidade de verificar se um determinado código informado, como parâmetro de entrada, é um código nacional de pessoal jurídica (CNPJ) válido ou não.Esta subrotina foi extraída de uma coleção de programas de uso pessoal da autora deste trabalho.

A linguagem de programação desta subrotina é VBScript e possui apenas um serviço para verificar se um determinado CNPJ é válido ou não, para isso, necessita dos seguintes parâmetros:

- checkStr: cadeia de caracteres a ser validada;
- Nulo: valor lógico que indica se permite valor nulo no campo ou não;
- Branco: valor lógico que indica se permite campo em branco ou não.

O código-fonte da subrotina é o único artefato disponível. A Figura 33 apresenta o diagrama de instâncias para o conjunto de informações que descrevem este componente de software.

Conforme o que foi discutido na seção 4.4.1, a subrotina Valida_CNPJ se caracteriza como uma unidade básica de informação a ser representada por meio do meta modelo do RCR, como mostra a Figura 33.

5.2.2 Classe Simples: Login.java

A classe ***Login*** é um exemplo de componente de software do tipo classe simples e foi extraído do livro de Santos (2003). Está classe encapsula o nome e a senha de um usuário, ambos instâncias da classe String, e contém métodos para verificar se o nome e a senha passados como argumentos são iguais aos encapsulados, tanto de forma rigorosa (considerando maiúsculas e minúsculas como sendo diferentes) como

de forma menos rigorosa (considerando maiúsculas e minúsculas como sendo iguais).

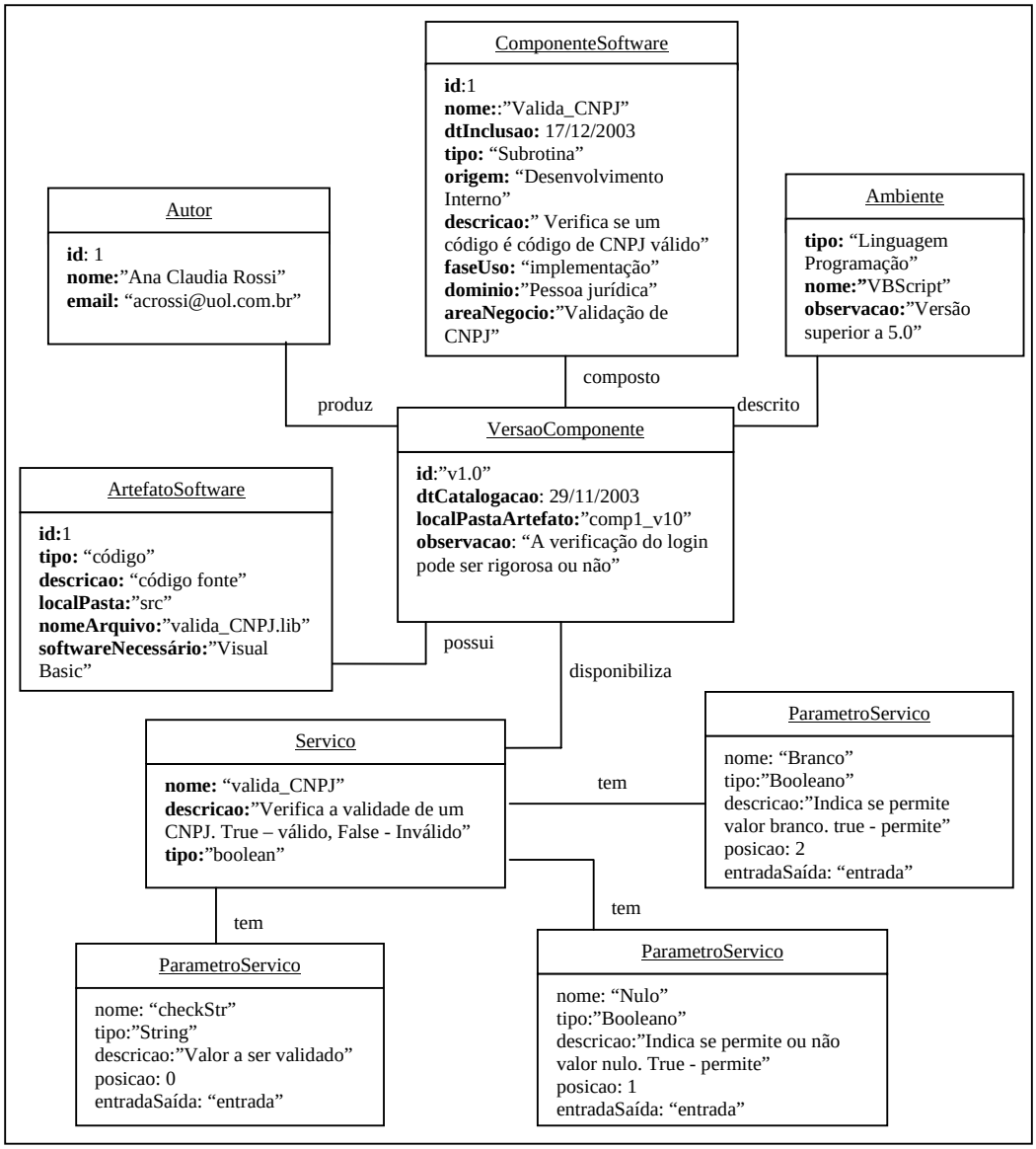

Figura 33: Diagrama de instâncias da subrotina Valida_CNPJ.

A representação deste componente de software é mostrada através da Figura 34 e Figura 35 que exibem os diagrama de instâncias para este exemplo.

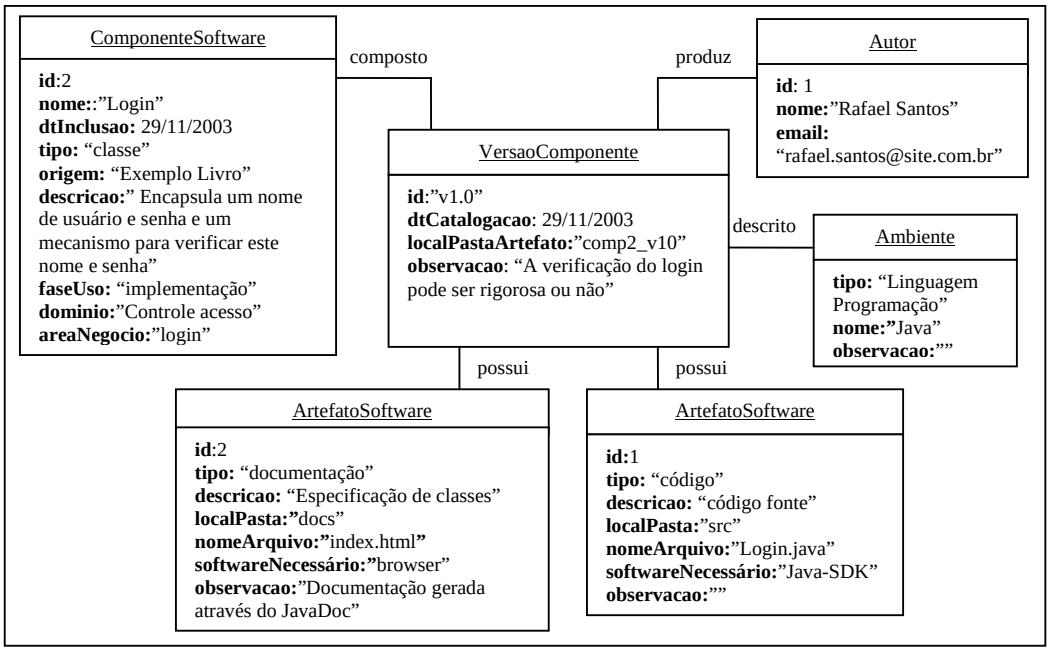

Figura 34: Diagrama de instâncias da classe Login (parte 1).

Para a classe **Login** foi possível representar toda a informação relacionada a sua descrição geral, a versão, autor e ambiente. O código fonte da classe e a documentação de especificação da classe foram representados como artefatos de software (ArtefatoSoftware).

Os serviços oferecidos pela classe **Login** teve todo a sua informação representada através do diagrama de instâncias da Figura 35.

Portanto, o modelo do repositório é adequado para cadastramento de classe simples, qualquer documentação adicional pode ser catalogada como um artefato de software e, desta forma, possibilitar a sua consulta durante o processo de entendimento do componente de software. A classe Loginpode ser considerada, também como uma unidade básica de informação no meta modelo do repositório.

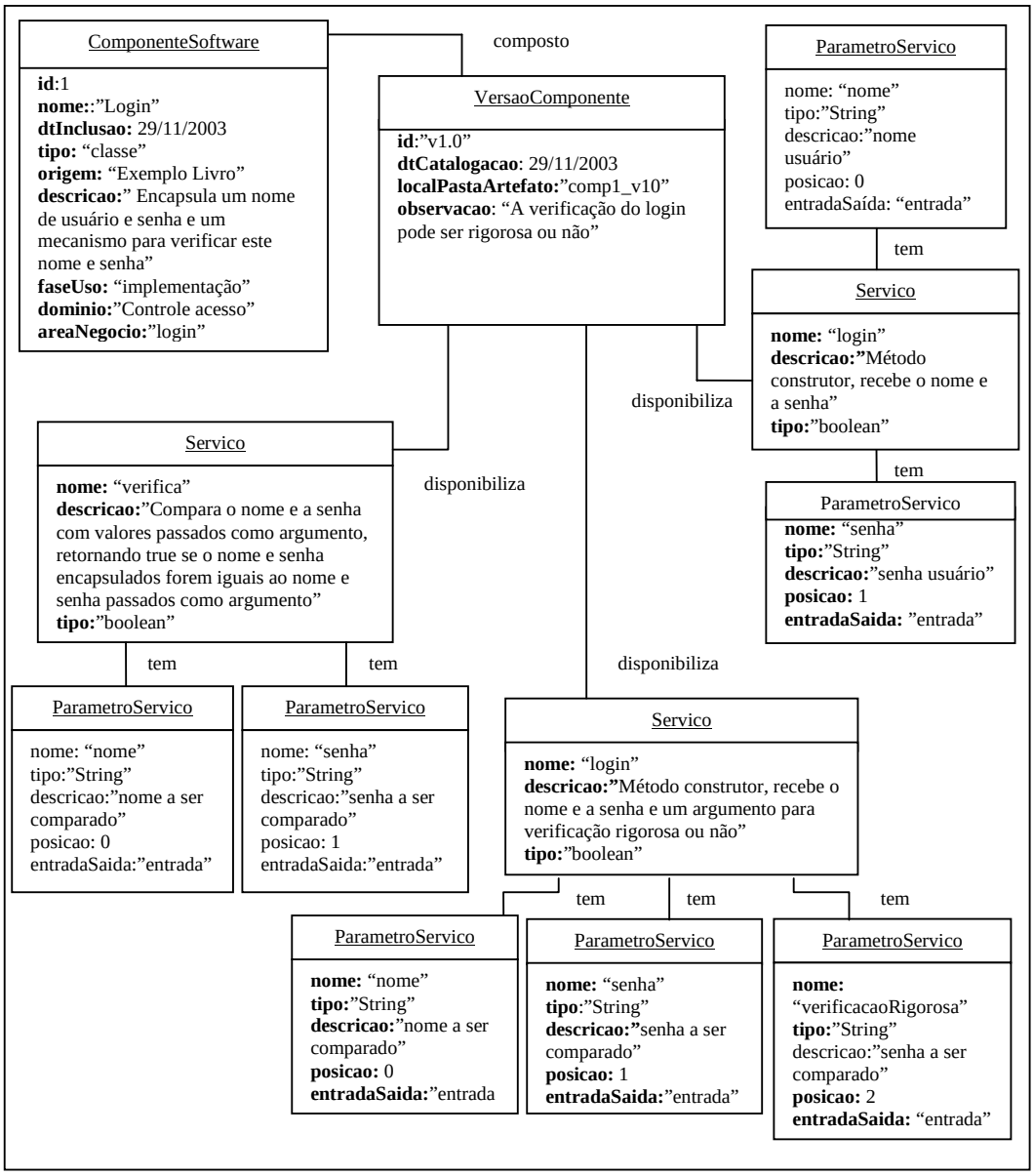

Figura 35: Diagrama de instâncias da classe *Login* (parte 2).

5.2.3 Pacote de Classes para cálculo de vencimentos

O pacote ***Employee*** é um conjunto de classes que disponibilizam funcionalidade para cálculo de vencimentos de funcionários em diferentes categorias, e foi extraído do livro de Deitel e Deitel (2003).

A classe ***Employee*** é a superclasse, composta pelas seguintes subclasses:

- ***Boss***: representa os funcionários pagos com salário fixo semanal independentemente do número de horas trabalhadas;
- ***CommissionWorker***: representa os funcionários pagos com salário básico fixo mais uma porcentagem sobre as vendas;
- ***PieceWorker***: representa os funcionários pagos pelo número de itens produzidos;
- ***HourlyWorker***: representa os funcionários que são pagos por hora e que recebem um adicional por hora extra trabalhada.

A Figura 36 apresenta o diagrama de instâncias correspondente à catalogação da classe **Employee** mostrando a representação das informações gerais, autores, ambiente, os artefatos relacionados e os serviços oferecidos.

A Figura 37 apresenta o diagrama de instâncias para as informações referentes à classe ***Boss*** que deriva da classe ***Employee***.

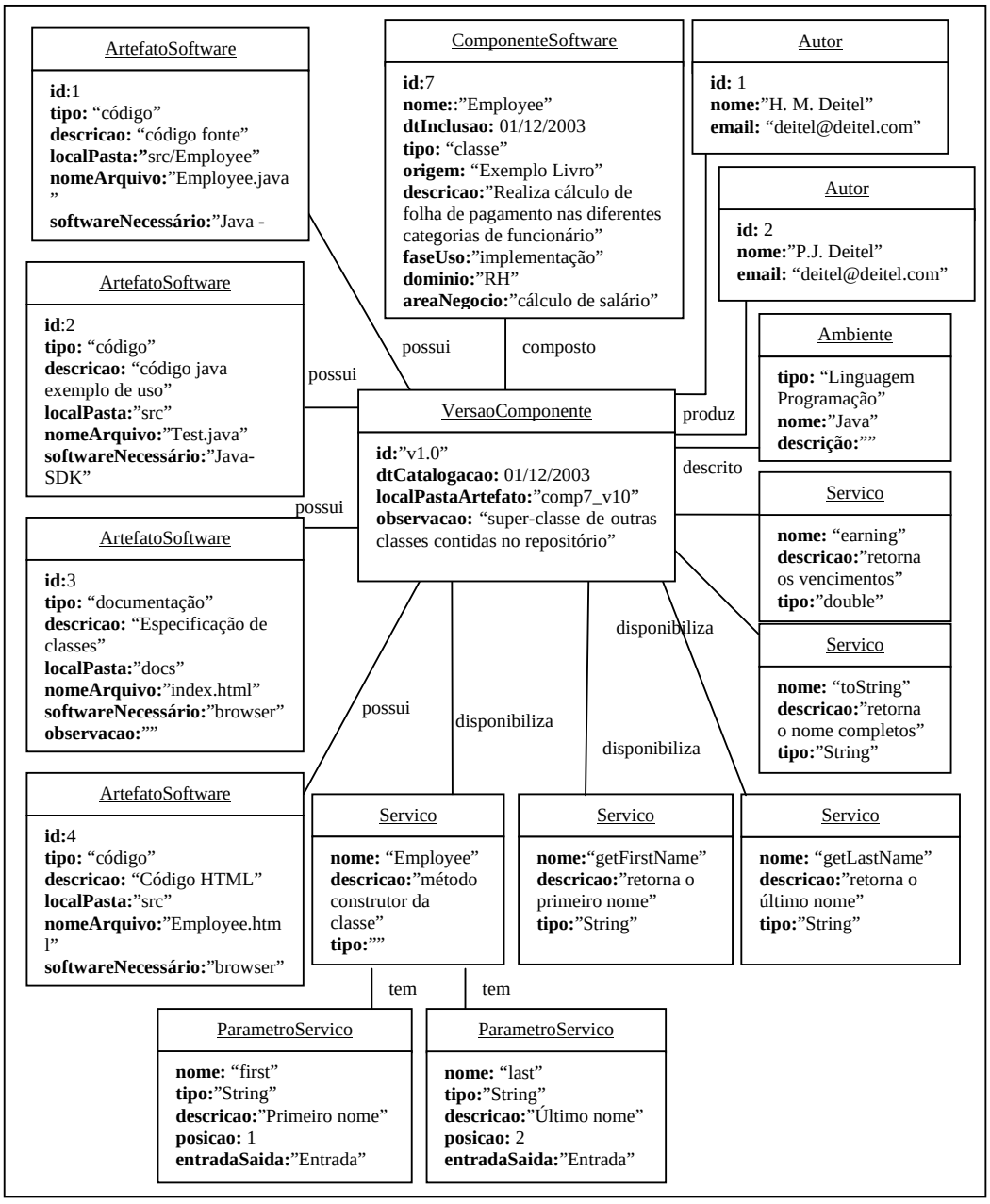

Figura 36: Diagrama de instância da superclasse *Employee*.

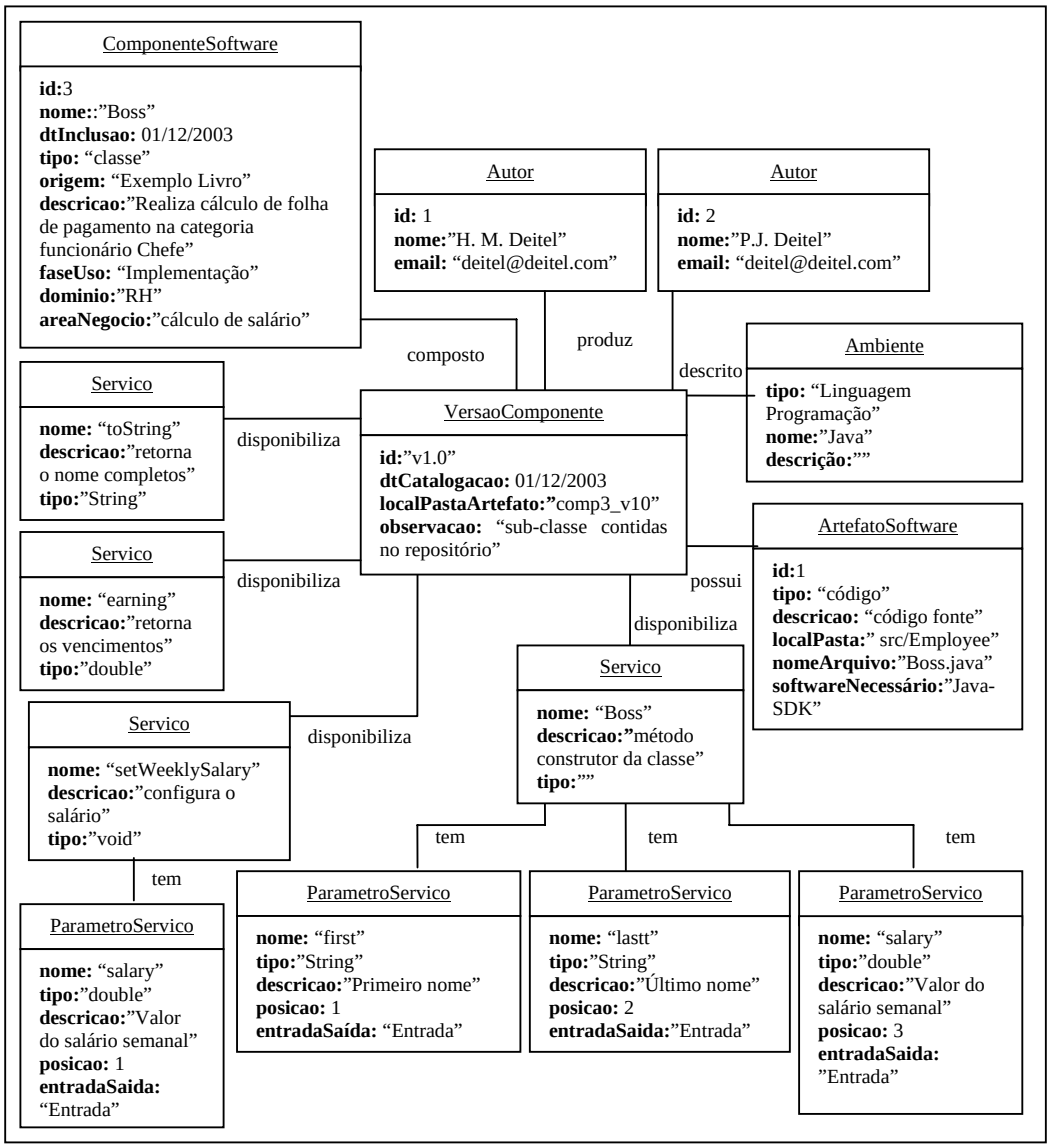

Figura 37: Diagrama de instância da superclasse *Boss.*

As classes ***CommissionWorker***, ***PieceWorker***, e ***HourlyWorker*** que também derivam da classe ***Employee***, realizam os cálculos de vencimentos para os outros três tipos de funcionários. As Figura 38 a Figura 42 apresentam os diagramas de instâncias para a representação da informação para estas três classes.

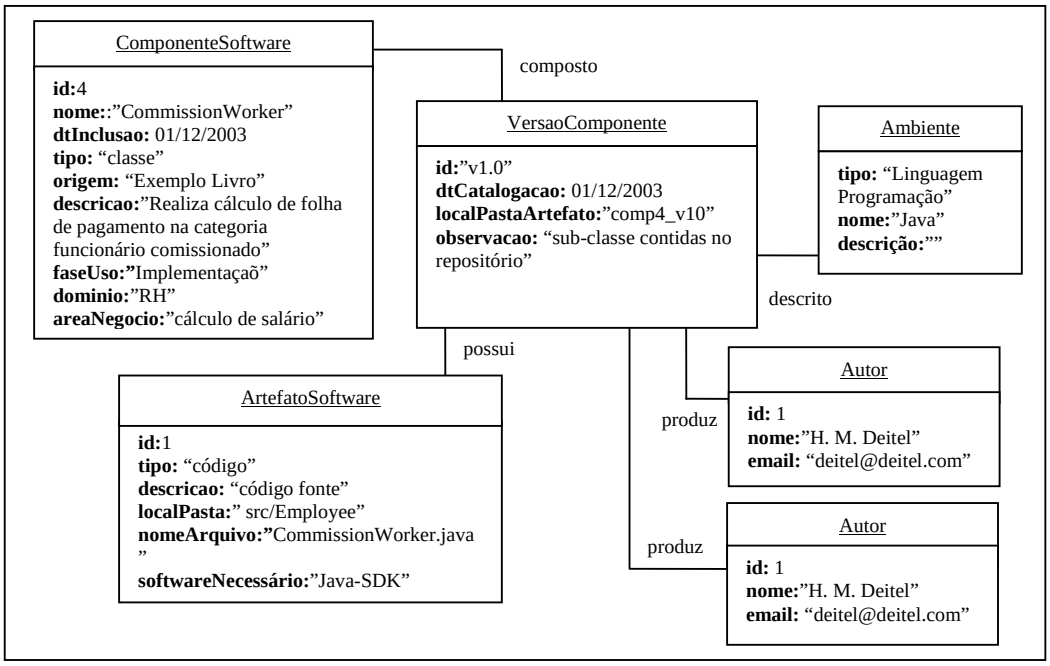

Figura 38: Diagrama de instância da superclasse *CommissionWorker* informações gerais, ambiente, autores e artefatos relacionados.

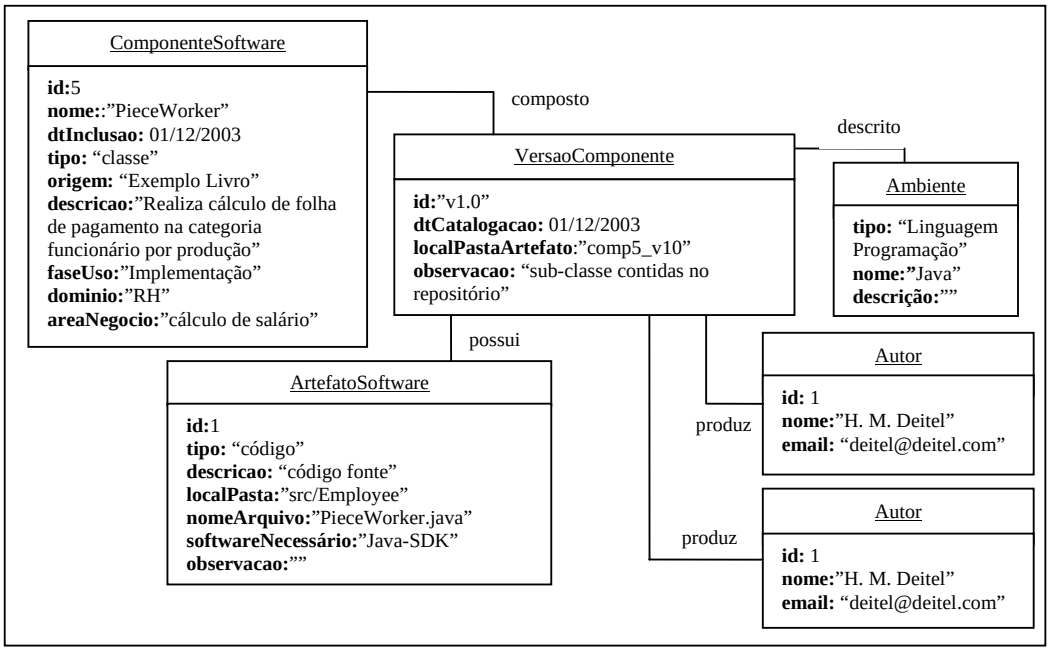

Figura 39: Diagrama de instância da superclasse *PieceWorker* informações gerais, ambiente, autores e artefatos relacionados.

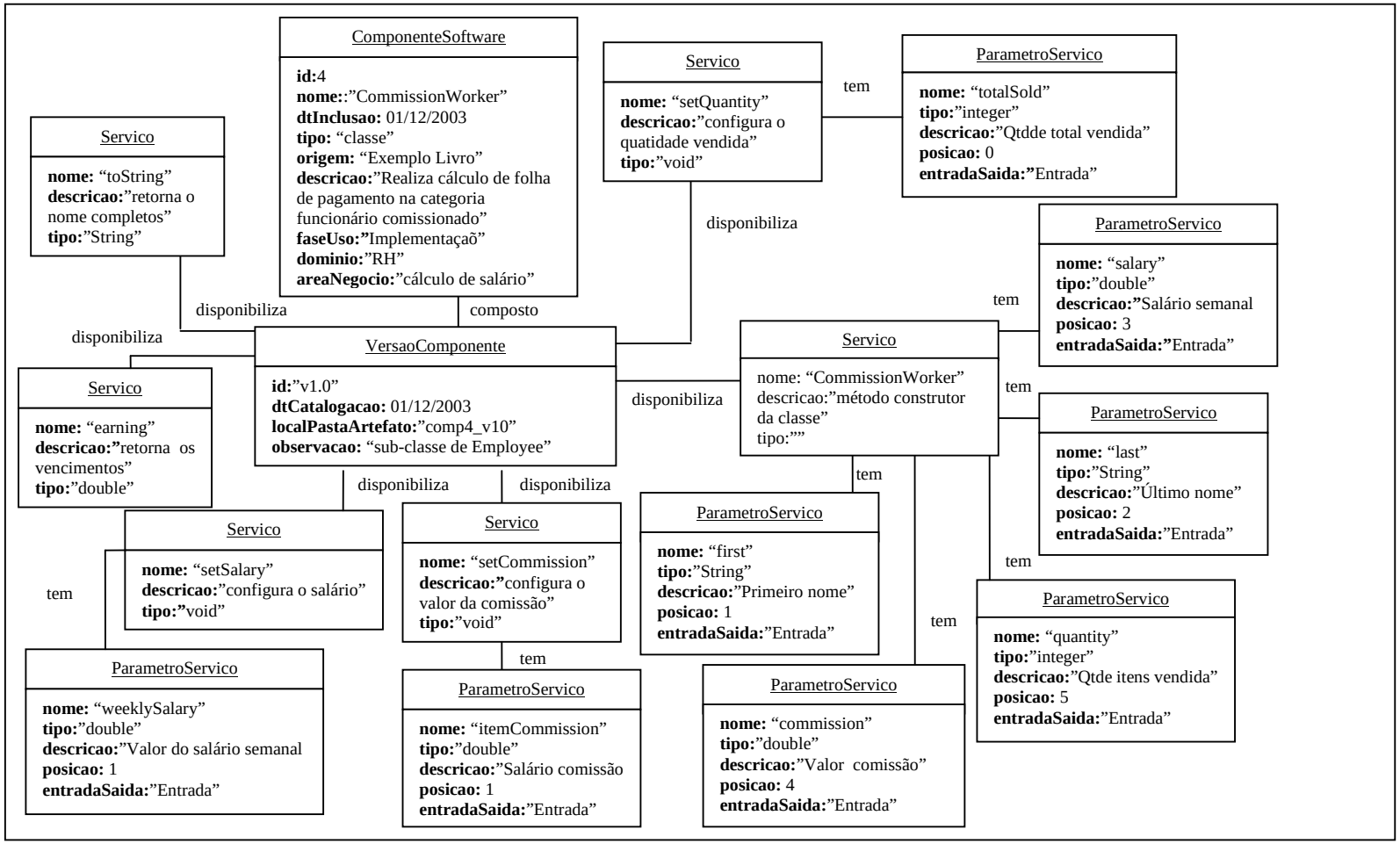

Figura 40: Diagrama de instância da superclasse *CommissionWorker* representando serviços e parâmetros.

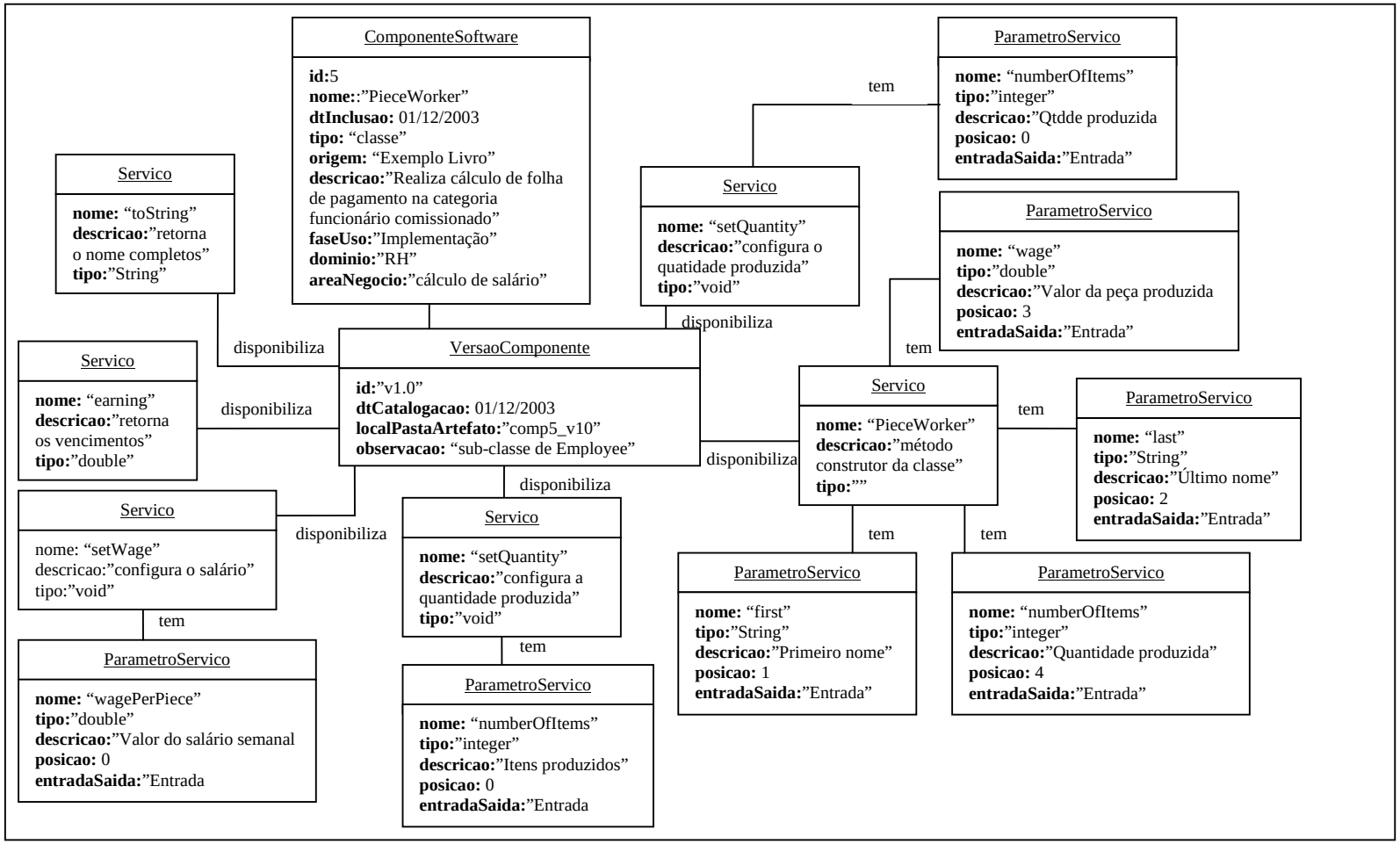

Figura 41: Diagrama de instância da superclasse *PieceWorker*.

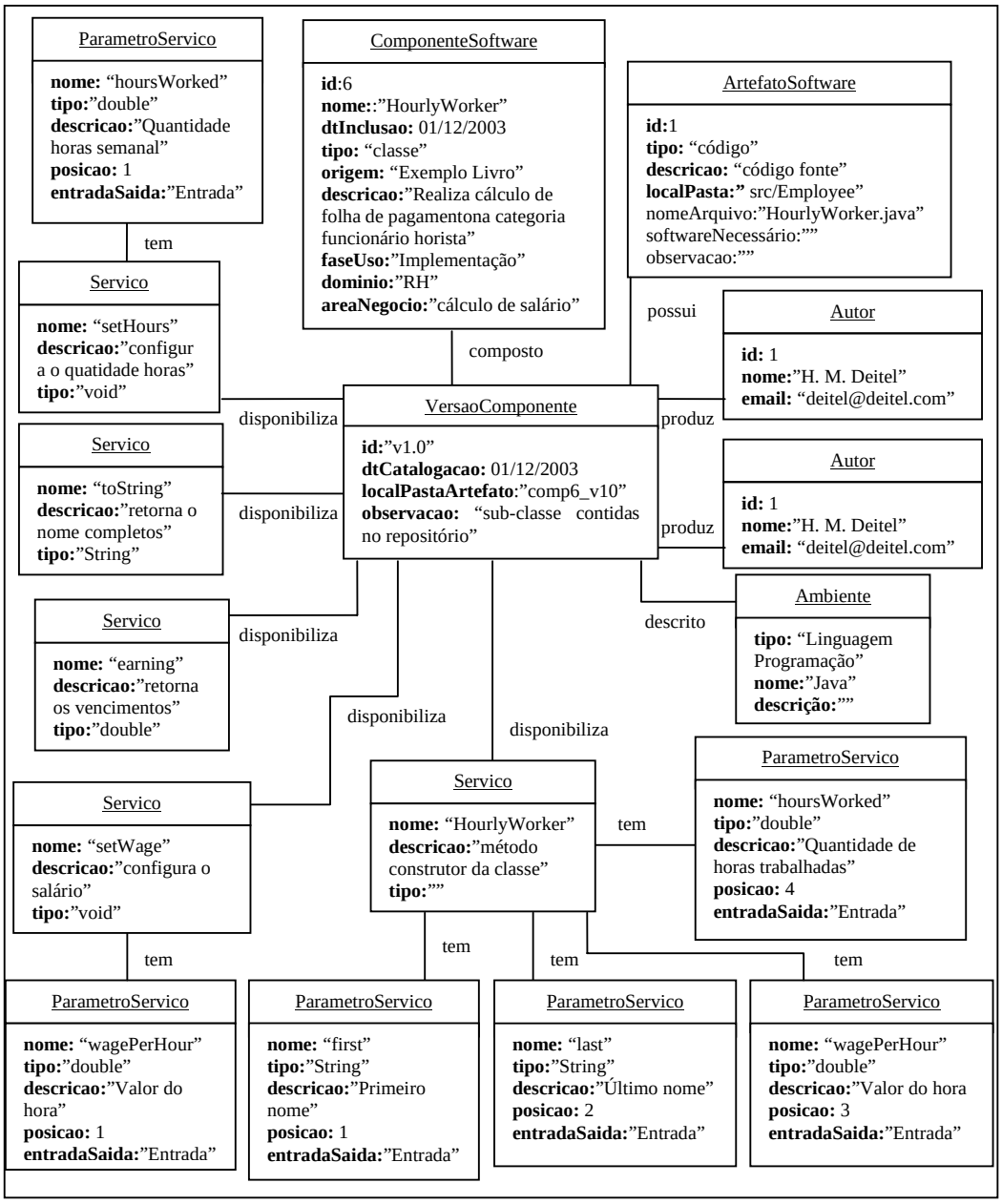

Figura 42: Diagrama de instância da superclasse HourlyWorker

A dependência entre a superclasse e as subclasses, deste exemplo, poderia ser representada de duas formas.

No primeiro caso, apenas a superclasse ***Employee*** seria representada como um componente de software e as subclasses seriam representadas como artefatos da superclasse. Nesta forma de representação, os serviços oferecidos pelas subclasses e

os seus respectivos parâmetros e a relação de dependência entre os pacotes não seriam representados.

Na outra forma de representação, tanto a superclasse como as suas subclasses são representadas como componentes de software e, assim, as informações referentes aos serviços oferecidos pelas subclasses são representadas. A relação entre os componentes (superclasse ***Employee*** e suas subclasses) é representadas através do relacionamento de dependência entre componentes.

A Figura 43 apresenta a representação do relacionamento de dependência entre a superclasse e a subclasses deste exemplo.

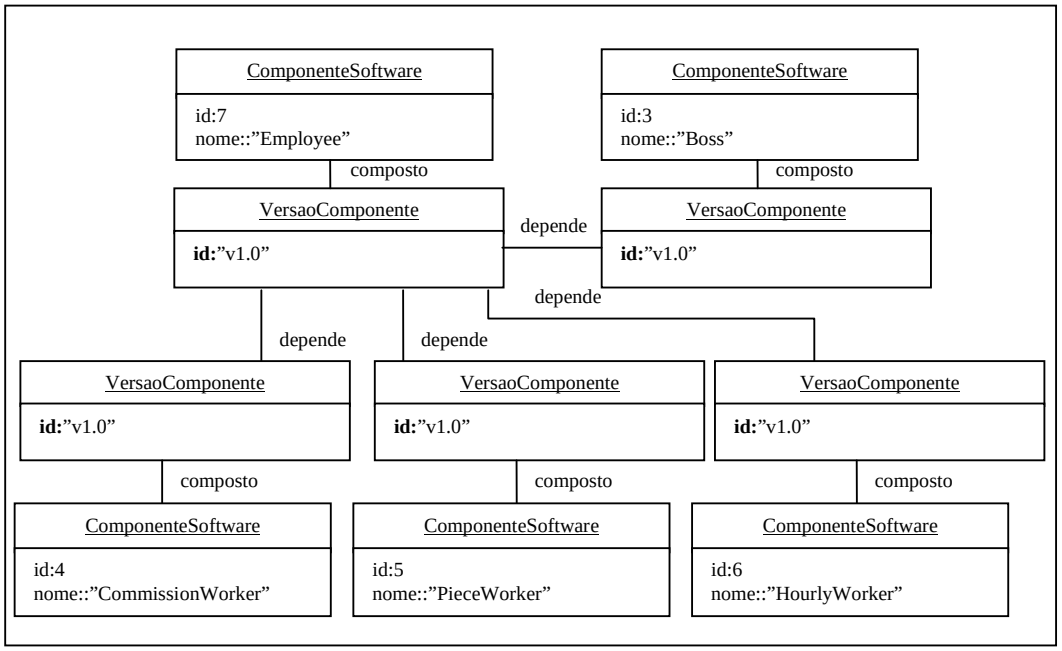

Figura 43: Diagrama de instância para o relacionamento entre a superclasse e subclasses.

A representação de um pacote de classes foi realizada adequadamente, ou seja, nenhuma informação referente ao componente de software deixou de ser representada para este exemplo.

5.2.4 Padrões de Projeto

Os padrões de projeto disponibilizam soluções para problemas comumente encontrados em projeto de vários de sistemas; em outras palavras, são arquiteturas ou partes de arquitetura comprovadas para construir software orientado a objetos flexível e de fácil manutenção (Deitel e Deitel (2003)).

Os benefícios gerados aos projetistas são os seguintes:

- Auxilia na construção de software confiável com arquiteturas comprovadas e a experiência acumulada das organizações;
- Promove a reutilização de projeto em sistemas futuros;
- Ajuda a identificação de erros e armadilhas comuns que ocorrem na construção de sistemas;
- Estabelece um vocabulário de projeto comum entre os projetistas;
- Reduz a duração da fase de projeto em um processo de desenvolvimento de software.

Um exemplo da aplicação de padrões de projeto é o catálogo de padrões de projeto para EJB (*Enterprise Java Beans*), criado pelo *Sun Java Center*, que é uma organização com o foco na produção de uma arquitetura para soluções baseadas na tecnologia Java. Este catálogo descreve um conjunto de 15 (quinze) padrões classificados nas seguintes categorias (ADATIA et al., 2001):

- Padrões da camada de apresentação: contém padrões relacionados à tecnologia de *servlets* e JSP;
- Padrões da camada de negócio: contém padrões relacionados à tecnologia de EJB;
- Padrões da camada de integração: contém padrões relacionados a JMS (*Java Message Service*) e JDBC(*Java DataBase Connectivity*).

Os padrões de projeto possuem um modelo de documentação para auxiliar no entendimento de como utilizá-los, composta dos seguintes tópicos (ADATIA et al., 2001):

- Nome: define a especificação do nome e da classificação do padrão;
- Intenção: exprime um aspecto de um problema particular ou o problema que se refere;
- Motivação: fornece motivos e explicações que afetam o problema e a solução, incluindo explicações do porquê um usuário deve escolher um deles;
- Aplicabilidade: especifica as situações nas quais o padrão de projeto pode ser aplicado;
- Participantes: apresenta a lista de padrões participantes e seus papéis no padrão;
- Estrutura: utiliza diagramas de classes UML para mostrar a estrutura básica dos padrões participantes;
- Colaborações: apresenta o comportamento dinâmico do padrão de projeto, através dos diagramas de colaboração UML;
- Conseqüências: descreve as conseqüências da aplicação da estrutura abstrata para um sistema de arquitetura;
- Implementação: ilustra, com um exemplo de código, a implementação do padrão;
- Exemplo de Código: disponibiliza um exemplo de implementação do padrão;
- Padrões Relacionados: relaciona os padrões relacionados ao padrão que está sendo especificado.

Para a representação de um padrão de projeto no repositório, foi escolhido o padrão **Business Delegate**, que é para auxiliar na separação da camada de apresentação da camada de serviço em uma aplicação. Este exemplo foi extraído do livro de Adatia e outros (2001) e os artefatos de software associados ao padrão são: o *template* que descreve o padrão e o código fonte que mostra a implementação.

A Figura 44 apresenta o diagrama de instâncias que mostra a representação do padrão **Business Delegate**.

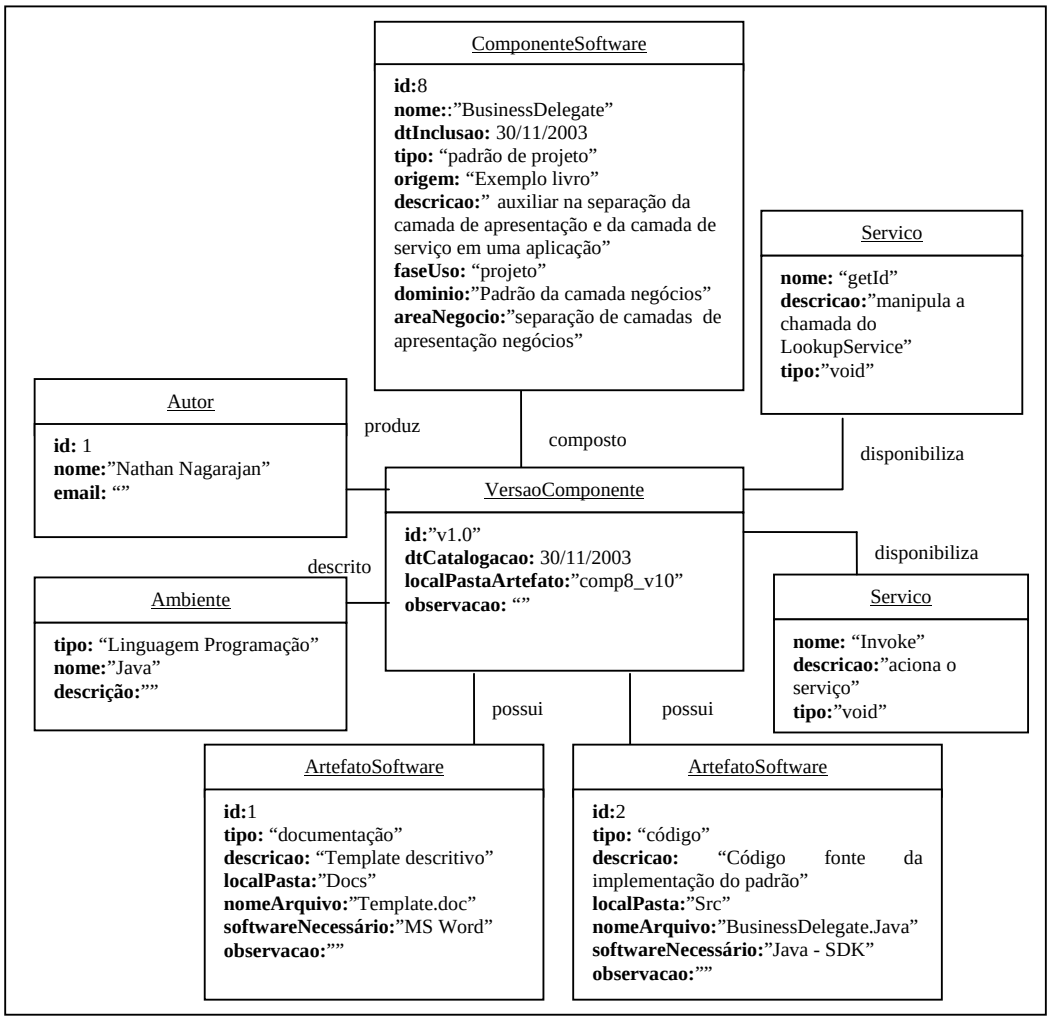

Figura 44: Diagrama de instâncias para o padrão *Business Delegate*.

Como pode ser observado, toda a informação relacionada ao componente de software e aos artefatos relacionados foi representada. Um outro aspecto que pode ser representado no repositório é a relação com os padrões relacionados, como pode ser visto na Figura 45.

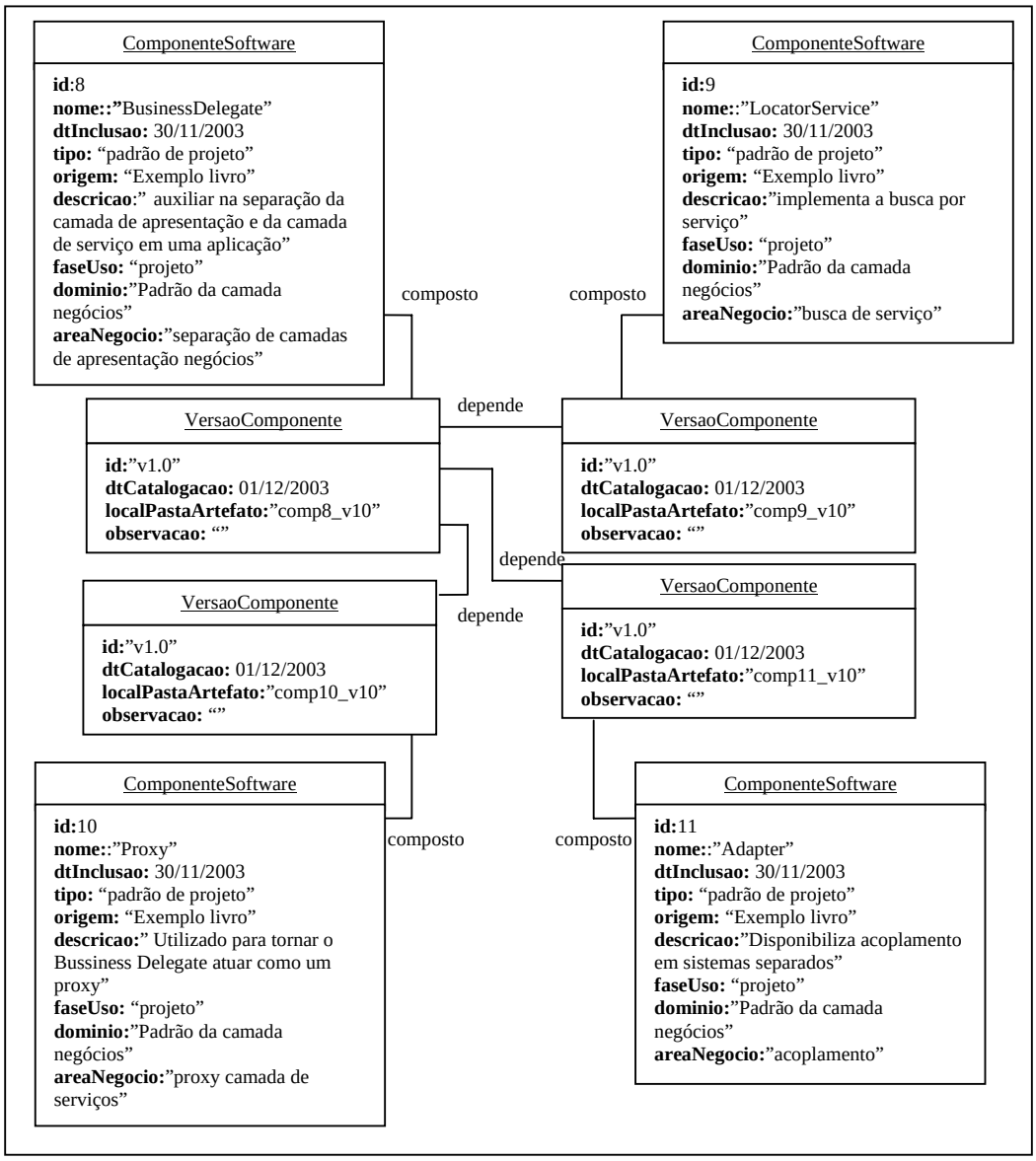

Figura 45: Diagrama de instâncias do padrões relacionados ao *Business Delegate*.

5.2.5 Componente UML: Controle de Reserva de Quartos

O componente **ReservationSystem** contém a especificação em UML de um componente para controle de reservas de quartos de uma rede de hotéis, extraído do

livro de Cheesman e Daniels (2001). O componente deve permitir o controle das reservas solicitadas pelos clientes até o *check-in* ou cancelamento da mesma; quando o cliente realiza o *check-in*, o componente de faturamento (**BillingSystem**) é avisado do início da estadia de um cliente. A Figura 46 apresenta o diagrama de casos de uso do controle de reservas de quartos.

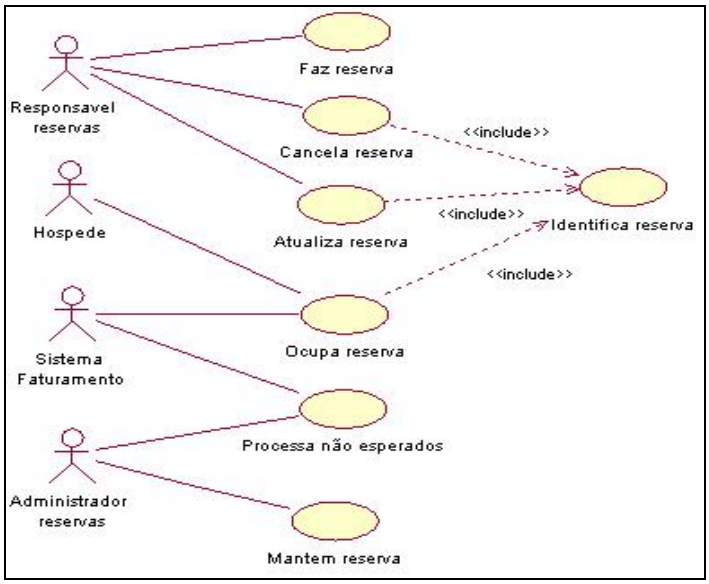

Figura 46: Diagrama de casos de uso do controle de reserva de quartos.

A representação das informações relacionadas à descrição da funcionalidade, finalidade, versão, propriedade intelectual, descrição do ambiente e os artefatos que documentam o componentes são mostrados por meio da Figura 47.

O componente de software **ReservationSystem** para realizar o processo de controle de reserva de quartos necessita de utilizar os serviços de outros três componentes de software:

- **BillingSystem**: sistema de faturamento que deve ser informado do início da estadia de um cliente que ocupa um quarto;
- **CustomerMgr**: componente de software que deve controlar as informações do cliente;

- **HotelMgr**: componente de software que deve controlar as informações referentes aos hotéis.

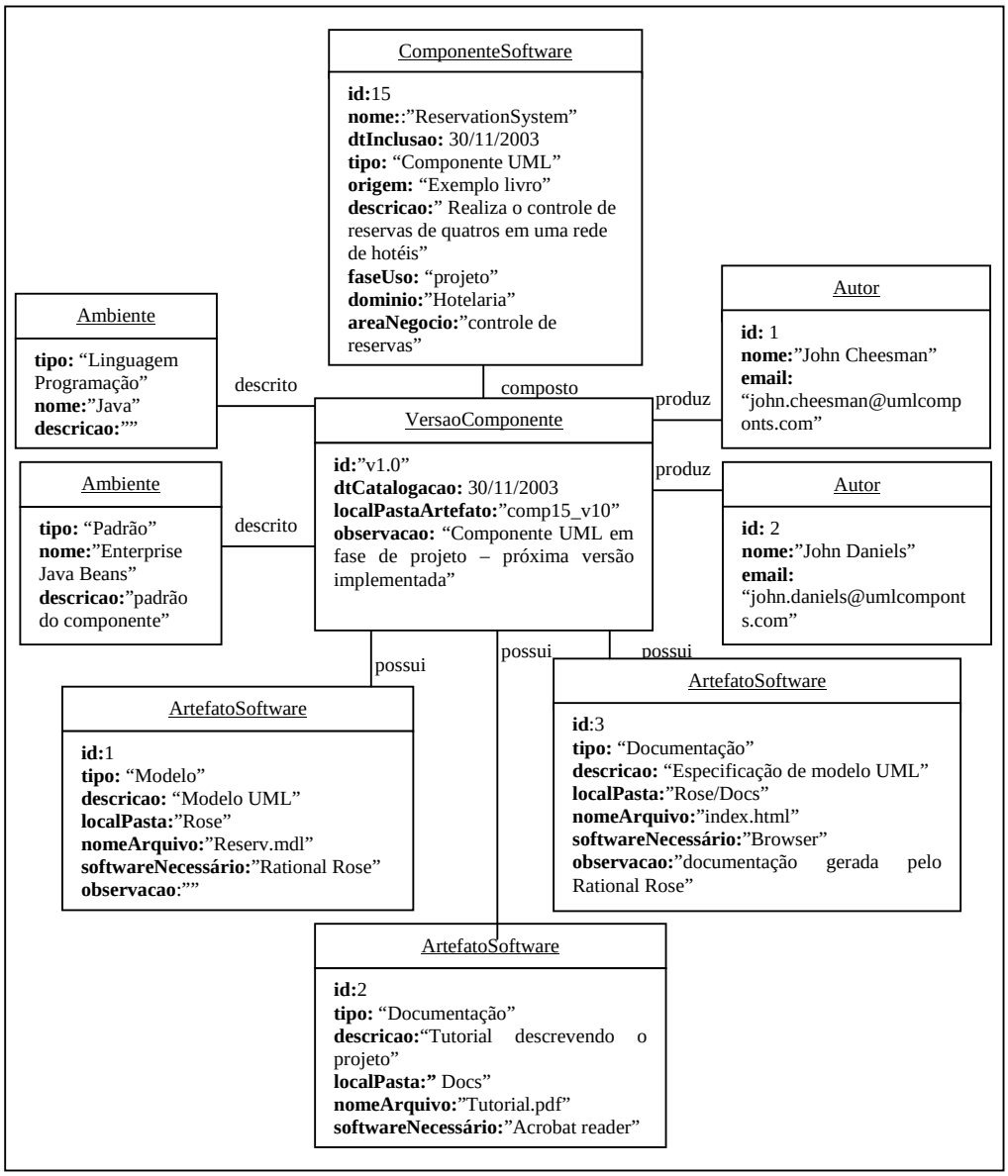

Figura 47: Diagrama de instância do componente *ReservationSystem*.

A relação de dependência entre os componentes ocorre por meio das interfaces, que cada componente oferece e que o outro realiza, deste modo, a Figura 48 mostra a representação das dependências e das interfaces oferecidas e realizadas pelo componente de software **ReservationSystem** através do diagrama de instâncias para este caso.

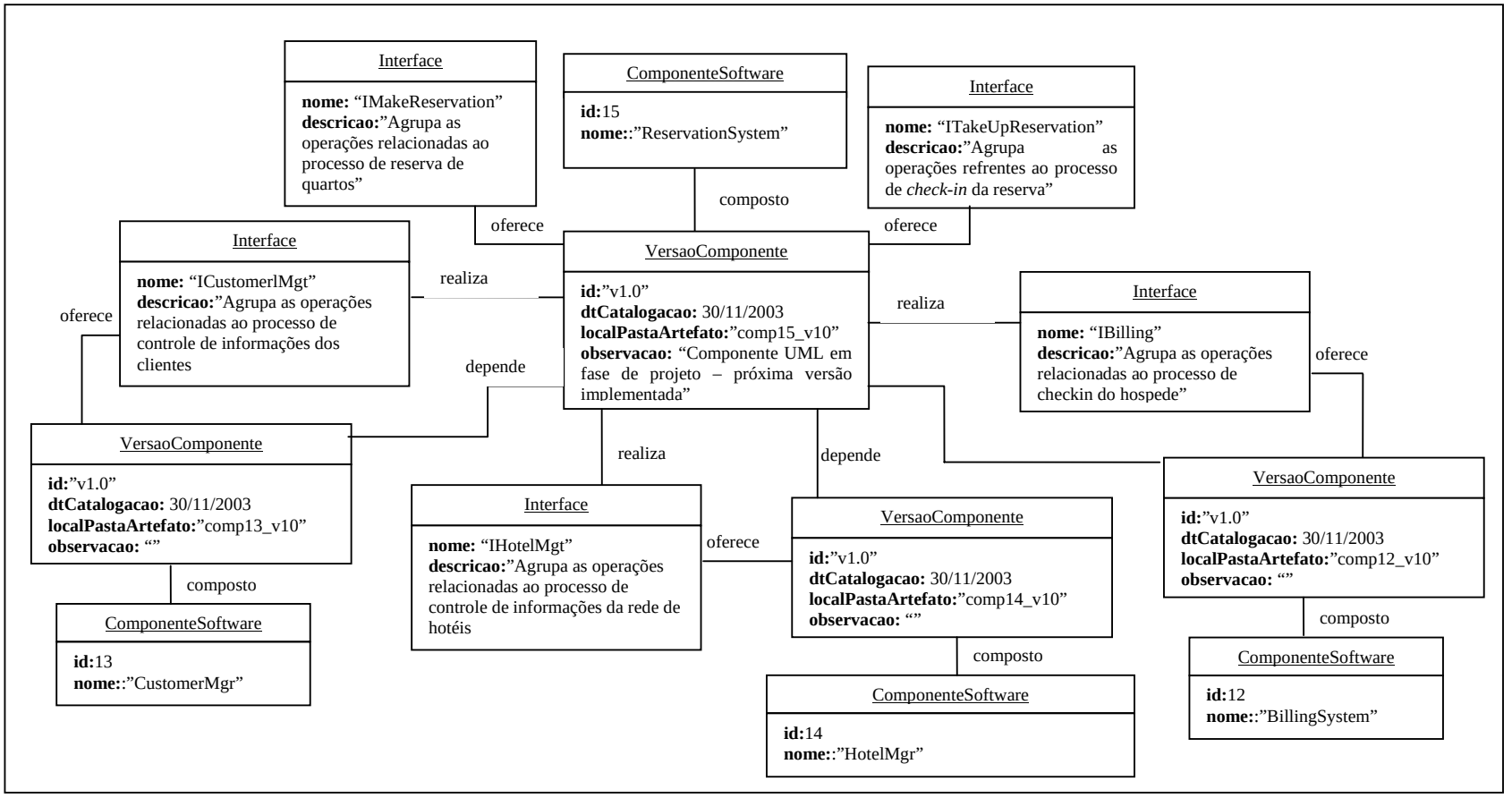

Figura 48: Diagrama de instância que representa a dependência entre componentes e as interfaces oferecidas e realizadas.

Por meio do diagrama de instâncias pode-se observar que o componente de software **ReservationSystem** disponibiliza duas interfaces, que são:

- IMakeReservation: oferece os serviços para execução do caso de uso **Faz reserva**;
- ITakeReservation: oferece os serviços para a execução do caso de uso **Ocupa reserva**.

Além disso, este mesmo componente, realiza as seguintes interfaces oferecidas pelos componentes que possui uma relação de dependência, que são:

- **IBilling**: interface oferecida pelo componente **BillingSystem** que agrupa o conjunto de serviços para efetuar o *check-in* de um hóspede;
- **ICustomerMgt**: interface oferecida pelo componente **CustomerMgr** que agrupa o conjunto de serviços oferecidos para controlar as informações de um cliente;
- **IHotelMgt**: interface oferecida pelo componente **HotelMgr** que agrupa o conjunto de serviços oferecidos para controlar as informações da rede de hotéis.

A representação dos serviços agrupados por meio de uma interface é uma informação importante a ser representada para um componente UML. Também, pode ser necessário descrever os serviços, seus parâmetros necessários e as suas pré e pós condições. As Figura 49 e Figura 50 mostram a representação deste aspecto através do diagrama de instâncias para a interface **ITakeUpReservation**.

Através dos exemplos dos diagramas de instância apresentados anteriormente, pode-se observar que toda a informação relativa a este componente de software modelado utilizando UML foi representada adequadamente.

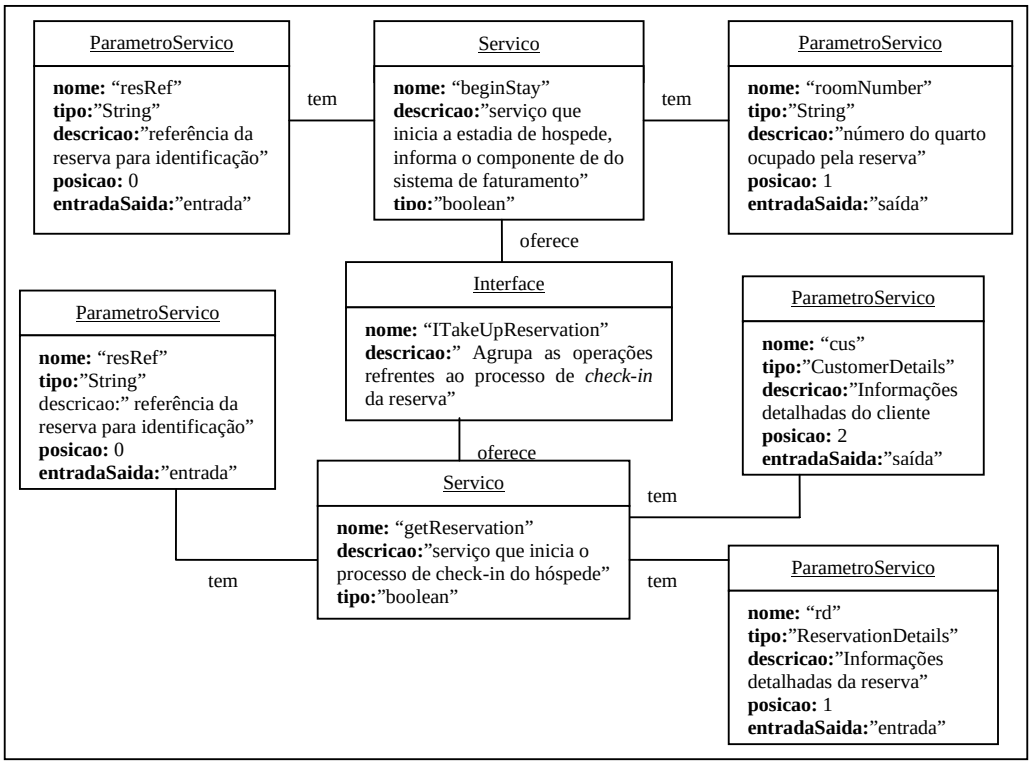

Figura 49: Diagrama de instâncias representação das interfaces de um componente UML- Parte 1.

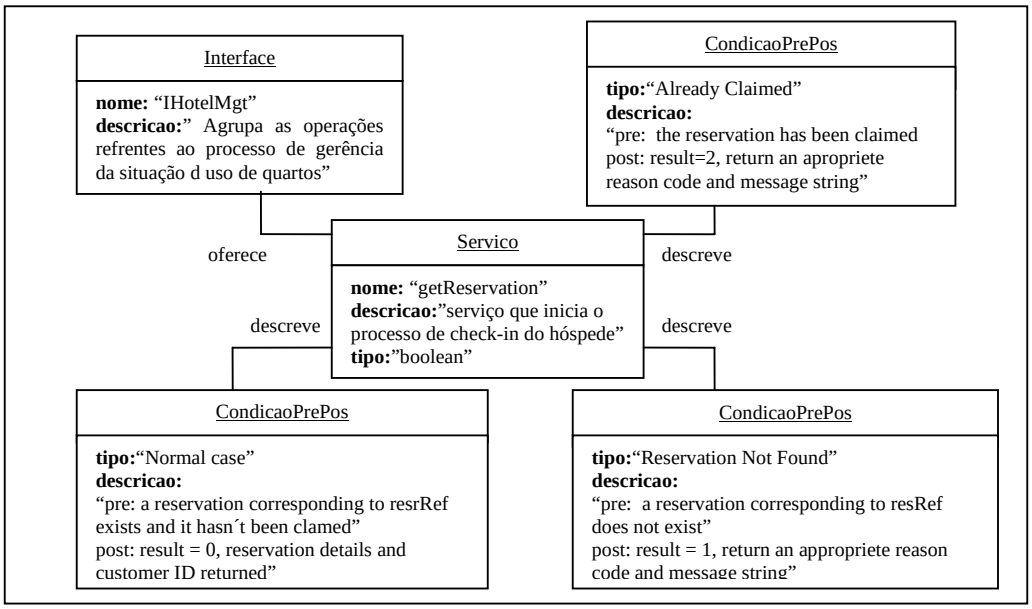

Figura 50: Diagrama de instâncias representação da condição pré/pós da operação de uma interface de um componente UML - Parte 1.

5.2.6 Sistema de Simulação de Sistema de Elevador

O sistema de simulação de elevador é o componente de software de mais alta granularidade no conjunto de componentes selecionados para a avaliação do repositório. Este sistema é um aplicativo de simulação de software orientado a objetos que modela a operação de um elevador. Este exemplo foi extraído livro de Deitel e Deitel (2003) e também se baseia em alguns documentos e informações extraídos do *site* dos autores do livro disponível em www.deitel.com.br. Este sistema está implementado em duas versões em linguagens de programação diferentes, que são: C++ e Java.

Para versão em Java, a arquitetura adotada para o desenvolvimento foi a Model-View-Controler (MVC), que utiliza vários padrões de projeto, conforme o apresentado na Figura 51.

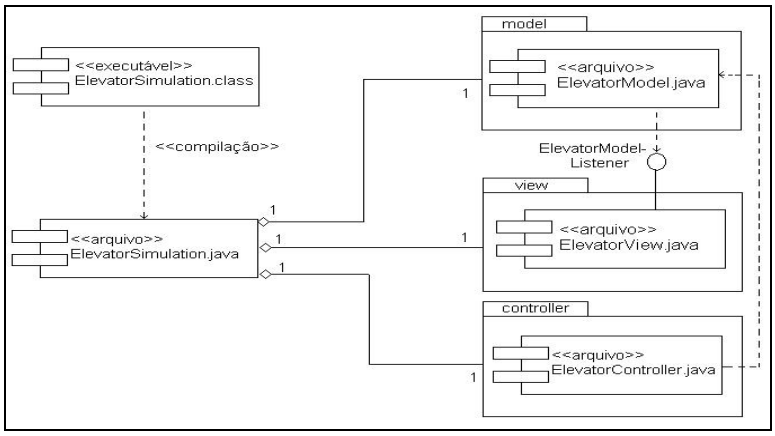

Figura 51: Diagrama de componentes para simulação de elevador (DEITEL e DEITEL, 2003) pp. 724

A arquitetura MVC dividiu as responsabilidades do sistema em três componentes, que são (DEITEL e DEITEL, 2003):

- **ElevatorModel**: contém **ElevatorModel.java**, que representa a operação do sistema do elevador;

- **ElevatorView**: contém **ElevatorView.java**, que representa a exibição deste modelo na tela, permitindo que o usuário visualize a simulação de forma gráfica;
- **ElevatorController**: contém **ElevatorController.java**, que representa a interface gráfica do controle da operação do elevador com o usuário permitindo o usuário simular o acionamento dos botões do elevador.

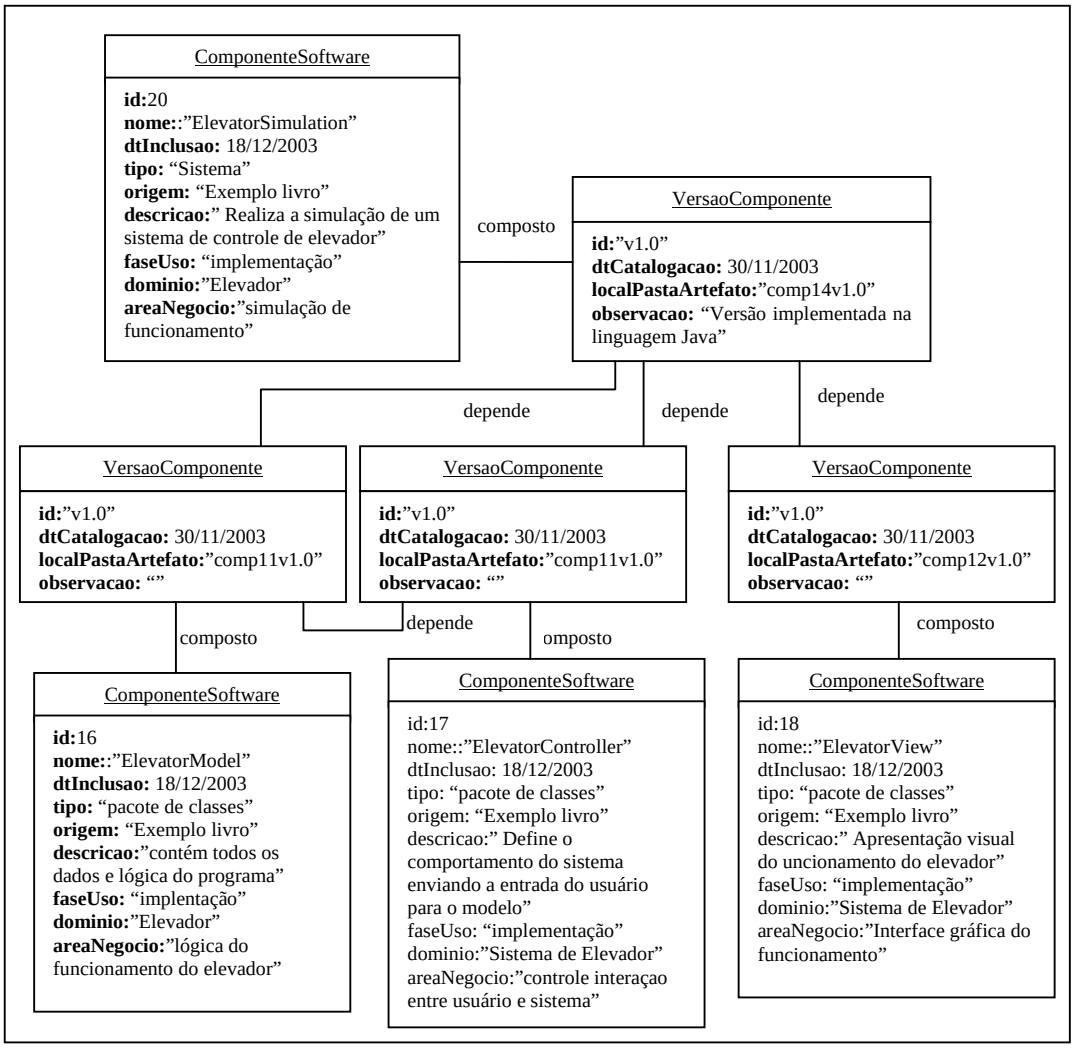

Figura 52: Diagrama de instância do relacionamento entre os componentes.

A Figura 52 mostrou o diagrama de instâncias para a representação da arquitetura MVC, na qual o sistema foi representado com a dependência de outros três componentes, que são: ElevatorModel, ElevatorView e o ElevatorController.

Para a versão em C++, o sistema é formado por um pacote de classes, cujo diagrama de classes se encontra na Figura 53. Através do diagrama de classes da figura, observa-se que o sistema é formado por um conjunto de classes que se relacionam entre si para executar a funcionalidade do sistema e que não estão agrupadas em sub componentes.

Desta forma, nesta versão, para o componente em C++ sua representação no repositório como um pacote de classes que o sistema depende, onde cada classe foi cataloga como o que foi apresentado na seção 5.2.3, ou seja, as classes que compõe o sistema são representadas como componentes de software que o sistema depende.

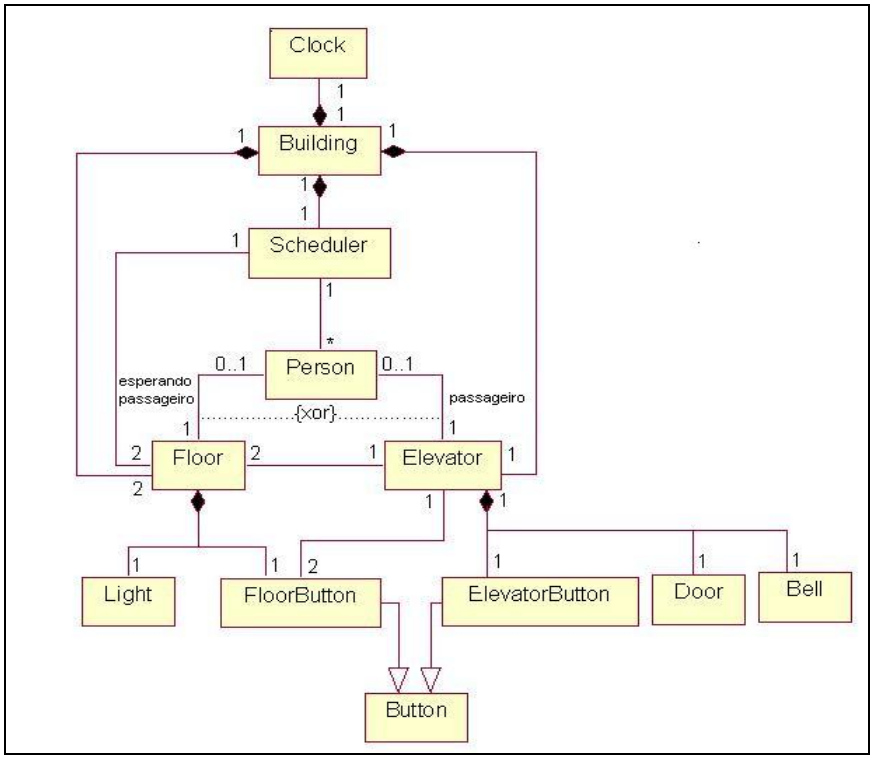

Figura 53: Diagrama de classes da versão do sistema de simulação de elevador em C++.

Este exemplo ilustra o caso em que o componente tem mais de uma versão, pois a documentação do sistema para Java e C++ apresentam algumas diferenças. Por exemplo, a versão na linguagem Java implementa a arquitetura Model-View-Controler, o que não acontece na versão na linguagem C++.

O ambiente da FARCSoft contempla esta visão, uma vez que os artefatos do componente de software devem estar relacionados com a versão do mesmo.

Os artefatos associados a cada versão são compostos por: especificação de requisitos, modelos UML desenvolvidos no Rational Rose, a especificação de cada elemento do modelo em UML gerados através de um recurso do Rational Rose e os códigos fontes das classes. Além disso, para a versão na linguagem Java, foi gerada a especificação em JavaDoc. A Figura 54 e Figura 55 apresentam os diagramas de instâncias para as representações destas duas versões.

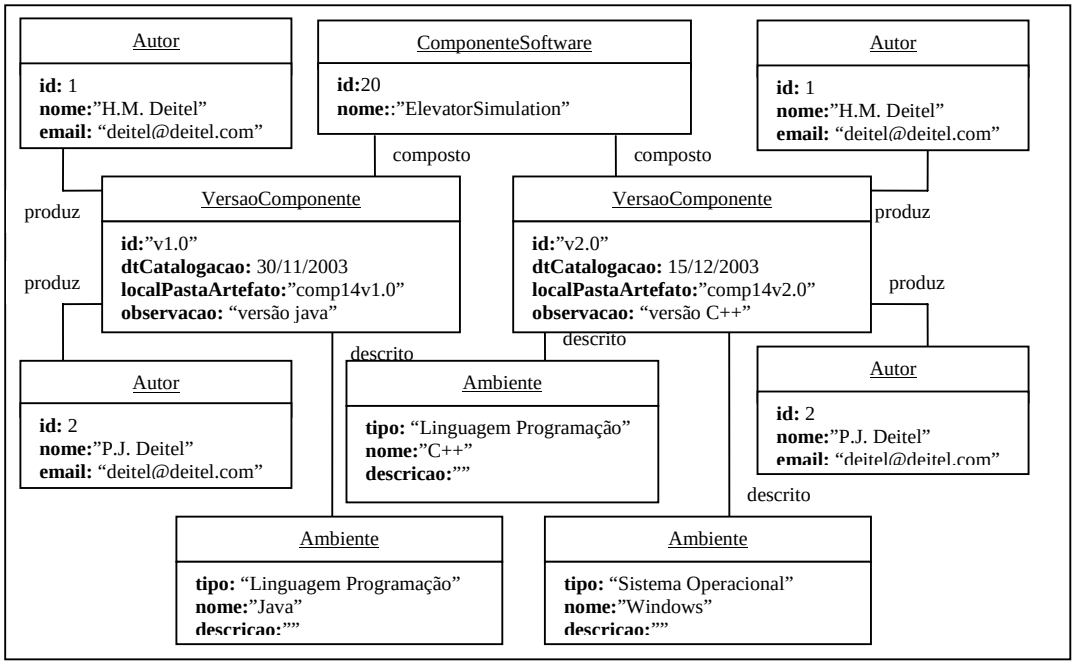

Figura 54: Diagrama de instâncias das versões do sistema de elevador (parte 1).

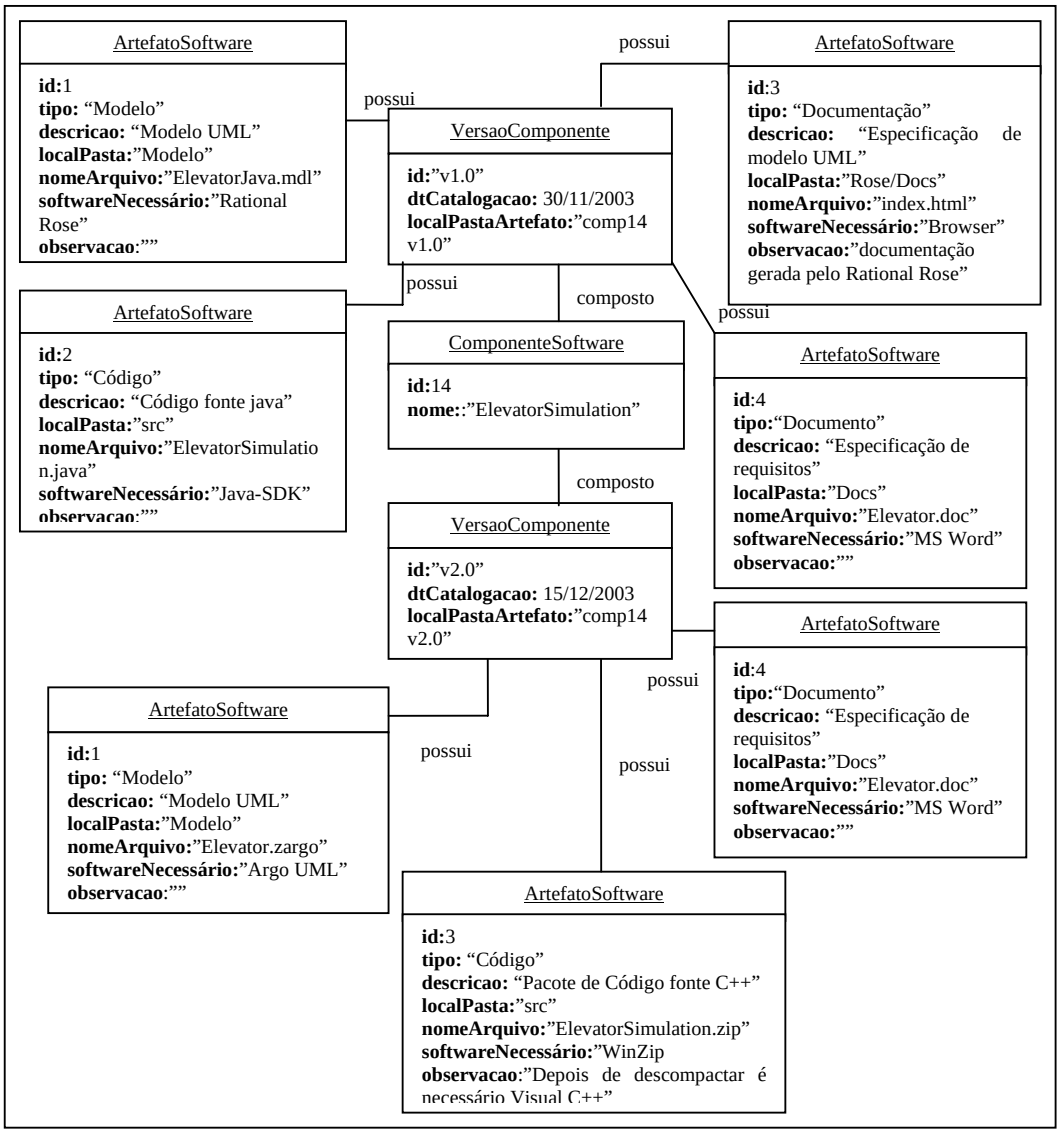

Figura 55: Diagrama de instâncias das versões do sistema de elevador (parte 2).

5.2.7 Projeto de Sistema de Elevador – Análise Estruturada

O projeto de um sistema escalonador e controlador de elevador apresenta um conjunto de documentos de um sistema de tempo real, descrito através da técnica estruturada. Este exemplo foi extraído do livro de Yourdon (1990), e tem a finalidade

de mostrar a capacidade de representação do meta modelo em mapear artefatos de software provinientes de um método de construção de sistemas tradicional.

É importante ressaltar que, este componente não foi considerado uma versão do componente apresentado na seção 5.2.6, pois trata-se de um sistema de tempo real para escalonar e controlar um conjunto de elevadores de um prédio e não um simulador, apresentando, portanto, características diferentes.

O requisito geral deste sistema é escalonar e controlar quatro elevadores de um edifício de quarenta andares, onde os elevadores serão utilizados para transportar pessoas entre os andares na forma convencional.

Os serviços oferecidos pelo sistema não estão representados, pois, como se trata de um projeto de sistema, as subrotinas ainda não estão implementas, desta forma, não existem ainda serviços a serem mapeados. Entretanto, a seçao 5.2.1 apresentou o mapeamento do serviço de um subrotina construído a partir do método tradicional.

Os artefatos de software que documentam este do projeto, elaborados através do método estruturado, consistem dos seguintes documentos:

- Decrição narrativa dos sistemas desenvolvida através do editor Microsoft Word;
- Modelos (diagrama de fluxo de dados e diagrama de transição de estados) elaborados através do software DOME, dsponível em www.htc.honeywell.com/dome;
- Especificação de processos e o dicionário de dados elaborado através do editor Microsoft Word

A representação deste componente de software é apresentada na Figura 56.

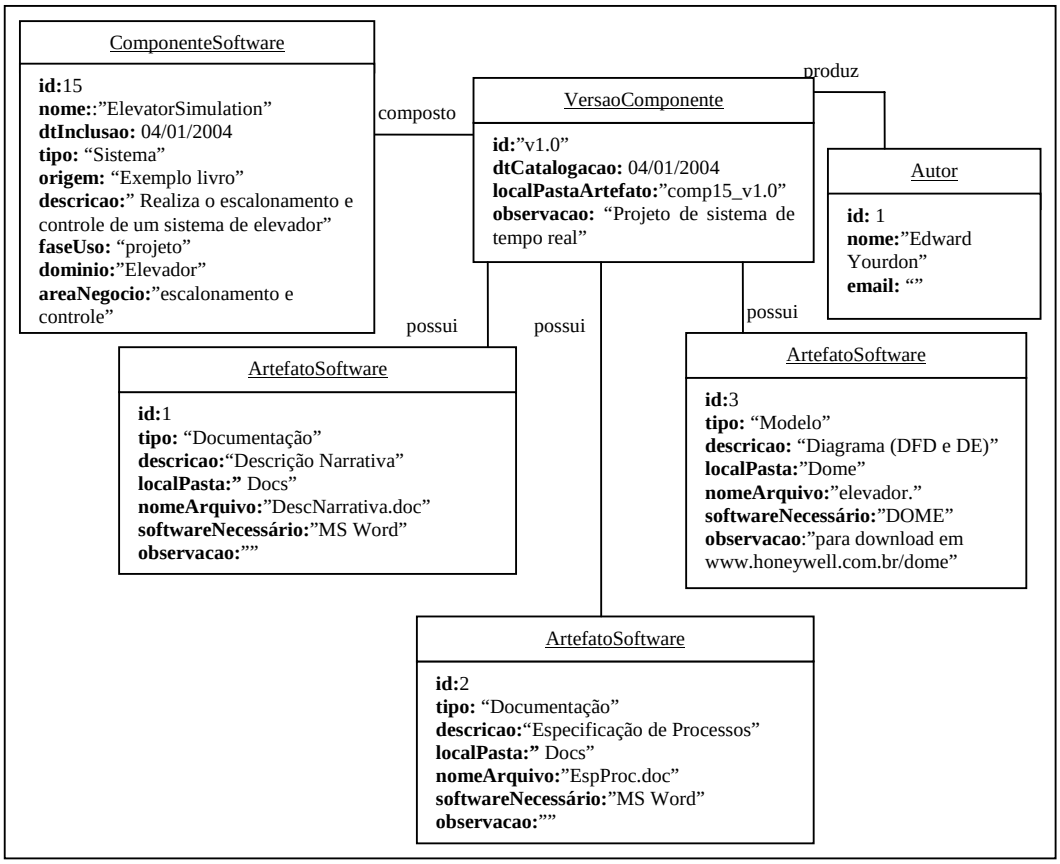

Figura 56: Diagrama de instâncias para projeto do escalonador e controlador de elevadores.

5.3 Análise dos Resultados Obtidos

O objetivo deste estudo de caso foi verificar a capacidade do repositório de componentes de software em representar e documentar o conjunto de informação relacionado aos diferentes tipos de componentes de software. Foram considerados desde fragmentos de código até elementos de software mais complexos, como um *framework* ou um sistema de software.

A catalogação destes componentes mostrou que a proposta do repositório de componentes da FARCSoft permite a representação de componentes com diferentes

níveis de porte e complexidade, e elementos variados para a descrição destes componentes.

A Tabela 13 apresenta, para cada componente do estudo de caso, o conjunto de informações que foram armazenadas para a sua representação. As linhas da tabela especificam o elemento de informação do componente que deve ser representada no repositório, já as colunas, mostram a relação de componentes catalogados no estudo de caso. As células tabela que estão vazias, não possuíam elemento de informação a ser representada para aquele componente de software no repositório.

Do preenchimento da Tabela 13, pode-se observar que os tipos de informação previstos no repositório foram adequados, pois permitiu a representação dos componentes selecionados para a sua avaliação. Pode-se observar que, de acordo com nível de granularidade, fase de uso do componente e complexidade dos serviços oferecidos, o volume de informação a ser armazenado foi maior.

Além disso, vale comentar que nem todos os elementos de informação foram preenchidos para todos os componentes. O não preenchimento ocorre devido à não necessidade ou à não existência de certos tipos de informação. Por exemplo, não é todo componente que necessita de descrição de interface ou possui restrições com relação ao ambiente de sistema operacional ou banco de dados. Ainda, a não existência pode ser devido à baixa qualidade dos artefatos de um dado componente.

Perante os resultados obtidos, contatou-se a representação proposta no repositório é satisfatória para o armazenamento de componentes de porte e complexidade variadas, desenvolvidos com tecnologias diversas e com quantidade diversificada de artefatos para a sua descrição. Como foram utilizados, na sua quase totalidade, os componentes descritos na literatura ou obtidos em *sites* públicos, pode-se dizer que a amostragem foi neutra, não tendo favorecido a avaliação da representação.

Os resultados obtidos vão dar maior segurança no desenvolvimento da FARCSoft

	Componentes do Estudo de Caso						
Elemento de Informação	**Valida_CNPJ**	**Login**	**Employee**	**Padrão de Projeto**	**Componente UML**	**Projeto de Sistema de Elevador**	**Sistema de Simulação de Elevador**
Descrição Geral	sim	sim	sim	sim	sim	sim	sim
Versão do componente	sim	única	única	única	única	única	duas
Autoria	sim	sim	sim	sim	sim	sim	sim
Ambiente	sim	sim	sim	sim	sim		sim
Restrição							Copyright
Artefato de Software	sim	sim	sim	sim	sim	sim	sim
Dependência			sim	sim	sim		sim
Interfaces do componente					sim		
Serviços oferecidos	sim	sim	sim	sim	sim		sim
Parâmetros do serviço	sim	sim	sim		sim		sim
Condicão pré/pós					sim		

Tabela 13: Resultado da representação da informação relacionada ao componente de software.

6. CONSIDERAÇÕES FINAIS

6.1 Conclusão

Este trabalho apresentou a proposta de representação e documentação de componentes de software na Ferramenta de Apoio à Reutilização de Componentes de Software, denominada FARCSoft, que tem a finalidade de promover o compartilhamento de componentes de software para reutilização no processo de desenvolvimento de sistemas por meio de um repositório de componentes reutilizáveis (RCR).

O objetivo deste trabalho foi de validar a capacidade de representação do meta modelo do repositório RCR, que faz parte da FARCsoft, em representar diferentes tipos de componentes de software, que podem variar com relação ao porte, complexidade, tecnologia para a sua construção e com quantidade diversificada de artefatos para a sua descrição.

Para isso, foram avaliados os principais processos existentes na literatura sobre a reutilização de componentes de software e tecnologia de componentes, com o intuito de definir o mecanismo de suporte necessário e investigar as características necessárias e desejáveis de um repositório para fornecer suporte às fases do processo de desenvolvimento de sistemas que emprega reutilização de software.

Além disso, foram estudadas as propostas de documentação de componentes de software para definir o conjunto de informação necessário para auxiliar no entendimento da funcionalidade e da forma de integrar o componente de software reutilizável para a construção de um novo sistema.

O trabalho de pesquisa realizado comprovou a necessidade de ferramentas que forneçam suporte às atividades de produção, consumo e gerência de componentes de software reutilizáveis.

Desta forma, para viabilizar o desenvolvimento baseado na reutilização de componentes, foi elaborada a proposta da FARCSoft, com as características exigidas pelo levantamento feito.

Do ponto de vista do conteúdo do repositótio, a prática e as pesquisas realizadas mostram que o componente de software pode existir de diferentes formas durante desenvolvimento, pode ser construído através de diferentes técnicas, pode ser implementado para ambientes diferentes e pode oferecer serviços adicionais não oferecidos por uma outra versão. Para contemplar estas características, a definição de componente de software apresentada neste trabalho pode ser dos seguintes tipos:

- **elemento de sistema reutilizável**, ou seja, pode ser encontrado em diferentes formas, especificações, códigos-fonte, bibliotecas, classes, padrões de projeto, *framework* ou subsistemas;
- **artefato que executa e disponibiliza a realização de um conjunto de serviços definidos e conhecidos**, ou seja, oferece serviços que tem um propósito estabelecido e público;
- **conjunto de artefatos que disponibiliza a informação necessária para integrar o artefato**, ou seja, possui um conjunto de informação que auxilia no seu entendimento de sua funcionalidade e de como integrá-lo;
- **executável ou não**, ou seja, pode ser apresentado em qualquer forma, como por exemplo, código fonte, executável, DLL (Dinamic Library Link) ou até mesmo em nível de projeto ou pseudocódigo;
- **para um sistema de aplicação, e fornecer suporte durante o ciclo de vida do sistema,** ou seja, o componente de software e toda a informação relacionada a ele será incorporado para a construção de um novo sistema de aplicação.

Desta forma, foi possível definir a estrutura do repositório de componentes reutilizáveis, estabelecendo-se um conjunto de informação para representar e documentar estes diferentes tipos de componente de software.

Conforme o apresentado na seção 5.2, a capacidade do meta modelo do repositório foi avaliada através do estudo de caso, onde foi selecionado um conjunto de componentes com diferentes características. Estes componentes são códigos, padrões de projetos, um conjunto de classes e sistemas, com complexidade variada, e foram obtidos de diversas fontes. As busca pelos componentes para o estudo de caso foi realizada em bibliotecas de código livre, bibliotecas de componentes relacionados à tecnologia de componentes, sites de universidades, livros e revistas. Os dados destes componentes foram mapeados para o meta modelo, o que mostrou a pertinência da sua definição, pois representou todos os tipos de informação dos componentes.

É importante ressaltar que a maior dificuldade foi encontrar componentes de software para serem usados neste estudo de caso. As principais dificuldades encontradas foram referentes à falta de qualidade e de documentação para auxiliar no entendimento da funcionalidade dos componentes selecionados.

Vale ressaltar que o resultado foi satisfatório, uma vez que a maior parte do material de avaliação não foi preparado pela autora do trabalho.

6.2 Contribuição

Conforme foi discutido neste trabalho, para a reutilização de componentes de software acontecer é necessário que existam componentes de software reutilizáveis e que os projetistas tenham acesso a eles. Além disso, uma vez que o componente de software foi localizado é necessário que sejam fornecidas informações que auxiliem

no entendimento dos requisitos do componente e da forma como integrá-lo na construção de um novo sistema.

Portanto, uma ferramenta que oferece recursos para manipular um repositório de componentes reutilizáveis é muito importante para fornecer suporte a um processo de desenvolvimento que emprega reutilização de componentes de software.
O modelo proposto para a representação do componente de software permitiu armazenar e representar os diferentes tipos de componentes de software e a sua informação relacionada, o que demonstrou ser um modelo válido. Esta proposta de representação do componente de software se baseou em trabalhos de pesquisa que buscavam definir um conjunto de informações necessárias para auxiliar na documentação de um componente de software.

Também foi necessário avaliar os principais processos existentes na literatura sobre reutilização de componentes de software, para estabelecer os recursos necessários e pesquisar os principais aspectos para o estabelecimento de mecanismos de classificação, busca e recuperação de componentes de software.

Desta forma, foi possível especificar a FARCSoft, que é uma ferramenta para fornecer suporte ao processo de reutilização de componentes de software, pois oferece recursos para armazenamento, busca e recuperação de componentes de software catalogados em uma base de dados única e compartilhada pelos projetistas.

A definição da estrutura de representação do componente de software no repositório é etapa inicial para a construção do ambiente da ferramenta que deverá difundir e compartilhar o conhecimento armazenado no repositório. Além de oferecer recursos para auxiliar nas atividades de gerência do repositório.

6.3 Trabalhos Futuros

Este trabalho apresentou a proposta inicial para a construção de uma **F**erramenta de **A**poio a **R**eutilização de **C**omponente de **Soft**ware (FARCSoft). Muitos aspectos relacionados a FARCSoft ainda devem ser explorados e validados para que ela se torne operacional. Para isso, é necessário especificar, projetar, implementar e avaliar os outros recursos propostos para ferramenta que respondam as seguintes questões:

- Qual seria uma relação de componentes reutilizáveis para compor o repositório para atender as necessidades do LTS;
- O esquema de classificação é eficiente? Quais seriam as alternativas?
- Quanto à representação do componente do ponto de vista do projetista? A interface gráfica da ferramenta auxilia no entendimento?
- Quais as alternativas em relação à pesquisa e à recuperação de componentes de software?
- Como definir e implementar o controle de versões no repositório?
- Como definir a coleta de métricas sobre a reutilização dos componentes de software do repositório?
- Como consolidar o processo de reutilização de componentes em relação à Certificação, à Submissão e à Avaliação?

Os componentes selecionados para compor o estudo de caso foram catalogados no repositório apenas para avaliar a capacidade de representação do repositório para diferentes tipos de compoentes de componentes de software. É necessário definir um conjunto de componentes que sejam relevantes para as ativiades do LTS. Para isso, é necessário definir as aplicações da ferramenta e realizar um trabalho de análise de domínio para identificar um conjunto e depois, implementa-los.

Quanto ao esquema de classificação e recuperação do componente de software no repositório, é necessário desenvolver pesquisas para avaliar a proposta apresentada neste trabalho e outras propostas da literatura para definir o esquema da ferramenta.

Outra questão importante é relação ao entedimento do componente de software por meio da ferramenta, é necessário aprofundar e avaliar os aspectos relativos a interface homem-máquina, ou seja, como e qual tipo de informação deve ser apresentada ao projetista para auxiliar no processo de compreensão da funcionalidade e de como reutilizar o componente.

Também deve ser definida a questão relativa ao controle de versões dos componentes de software armazenados no repositório, possibilitanto a implementação da reutilização do tipo caixa-branca de um modo mais eficiente.

Uma outra atividade que é interessante a ser desenvolvida é em relação às métricas relacionadas aos componentes. É necessário definir, coletar e disponibilizar a informação sobre a reutilização do componente de software para avaliar a reusabilidade, confiabilidade, desempenho e outras questões relativas à qualidade.

Vale ainda ressaltar que uma questão fundamental, na reutilização de componentes de software, é a definição e implantação de um processo de desenvolvimento baseado em componente que emprega reutilização, para que se tenha sucesso neste tipo de desenvolvimento.

Analisando as questões apresentadas anteriormente, muitos aspectos podem e devem ser explorados para que o ambiente da FARCSoft se torne mais completo e operacional para fornecer suporte à reutilização de componentes de software, o que motiva a continuidade do desenvolvimento desta proposta.

7. ANEXO 1 – RESULTADO DA CATALOGAÇÃO DOS COMPONENTES DE SOFTWARE NO REPOSITÓRIO

Neste anexo é apresentado o resultado da catalogação das informações mais relevantes de cada componente de software que fez parte do estudo de caso deste trabalho.

7.1 Componente de Software: Valida_CNPJ

A classe Valida_CNPJ**,** foi extraída de uma coleção de programas de uso pessoal do autro deste trabalho. Esta subrotina tem a funcionalidade de verficar se um código passado como parâmetro é um código de nacional de pessoa jurídica válido ou não.

Resumo

Nome	Valida_CNPJ
Inclusão	17/12/2003
Tipo	Subrotina
Origem	Desenvolvimento Interno
Descrição	Verifica se um código é um código de CNPJ válido
Fase de Uso	Implementação
Domínio	Pessoa Jurídica
Área de Negócio	Validação de CNPJ

Versão do Componente

Versão	Cataloga	Pasta	Observação
v1.0	17/12/2003	comp1_v10	

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Código fonte em VBScript	Código fonte	src	valida_CNPJ.lib	Visual Basic	

Dependência de outros componentes: Não tem.

Autor

Nome	E-mail
Ana Claudia Rossi	ana.rossi@poli.usp.br

Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Visual Basic	versão superior 5.0

Interface: Não tem.

Serviços: valida_CNPJ

Serviço	Descrição	Tipo Retorno
valida_CNPJ	Verifica a validade de um código de CNPJ (True – válido, False – inválido)	boolean

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	checkStr	Cadeia de caracteres a ser validada	String	Entrada
2	Nulo	Indica se permite valor nulo ou não (true - permite)	Booleano	Entrada
3	Branco	Indica se permite valor branco ou não (true - permite)	Booleano	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

7.2 Componente de Software: Login

A classe ***Login.java,*** componente de software do tipo classe simples,que encapsula o nome e a senha de um usuário, e contém métodos para verificar se o nome e a senha passados como argumentos são iguais aos encapsulados, tanto de forma rigorosa, como de forma menos rigorosa (SANTOS, 2003).

Resumo

Nome	Login
Inclusão	29/11/2003
Tipo	Classe
Origem	Exemplo de Livro

Descrição	Encapsula um nome de usuário e senha e um mecanismo para verificar a validade deste nome e senha
Fase de Uso	Implementação
Domínio	Controle de Acesso
Área de Negócio	Validação de senha

Versão do Componente

Versão	Catalogaçao	Pasta	Observação
v1.0	29/11/2003	comp2_v10	A verificação do login pode ser rigorosa ou não

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Especificação de classes	Documentação	docs	index.html	browser	Documentaçao gerada através do JavaDoc
Código fonte	Código fonte	src	Login.java	Java-SDK	

Dependência de outros componentes: Não tem.

Autor

Nome	E-mail
Rafael Santos	rafael.santos@site.com.br

Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	classe simples para reutilização

Interface: Não tem.

Serviços:Login

Serviço	Descrição	Tipo Retorno
Login	Método construtor, recebe o nome e a senha	boolean

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	nome	Nome do usuário	String	Entrada
2	senha	Descrição da senha	String	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços:Login

Serviço	Descrição	Tipo Retorno
Login	Método construtor, recebe o nome e a senha e um boolean argumento para verificação rigorosa ou não	

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	nome	Nome do usuário	String	Entrada
2	senha	Descrição da senha	String	Entrada
3	verificaoRigorosa	Indicador se a verificação	boolean	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços:Verifica

Serviço	Descrição	Tipo Retorno
verifica	Compara o nome e a senha com valores passados booleancomo argumento, retornando true se o nome e senha encapsulados forem iguais ao nome e senha passados como argumento	

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	nome	Nome do usuário	String	Entrada
2	senha	Senha do usuário	String	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

7.3 Componente de Software: Boss

A subclasse **Boss.java** pertence ao pacote ***Employee,*** que é um conjunto de classes que disponibiliza funcionalidade para cálculo de vencimentos de funcionários em diferentes categorias (DEITEL; DEITEL, 2003). A subclasse ***Boss*** representa os funcionários do nível gerencial pagos com salário fixo semanal independentemente do número de horas trabalhadas.

Resumo

Nome	Boss
Inclusão	1/12/2003
Tipo	Classe
Origem	Exemplo de Livro
Descrição	Sub classe da superclasse Employee - Realiza cálculo de folha de pagamento na

	categoria funcionário chefe
Fase de Uso	Implementação
Domínio	Recursos Humanos
Área de Negócio	Cálculo de salário

Versão do Componente

Versão	Catalogação	Pasta	Observação
v1.0	1/12/2003	comp3_v10	

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Código fonte	Código fonte	src/Employee	Boss.java	Java-SDK	

Dependência de outros componentes: Não tem.

Autor

Nome	E-mail
P.J. Deitel	deitel@deitel.com
H. M. Deitel	deitel@deitel.com

Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	

Interface: Não tem

Serviços: Boss

Serviço	Descrição	Tipo Retorno
Boss	Método construtor da classe	

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	first	Primeiro nome	String	Entrada
2	last	Último nome	String	Entrada
3	salary	Valor do salário semanal	double	Entrada

Descrição Pré-condição e Pós-condição: Não tem

Serviços: Earning

Serviço	Descrição	Tipo Retorno
earning	Retorna os vencimentos	double

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem

Serviços: setWeeklySalary

Serviço	Descrição	Tipo Retorno
setWeeklySalary	Configura o salário	void

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	salary	Valor do salário semanal	double	Entrada

Descrição Pré-condição e Pós-condição: Não tem

Serviços: toString

Serviço	Descrição	Tipo Retorno
toString	Retorna o nome completos	string

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem

7.4 Componente de Software: CommissionWorker

A subclasse CommissionWorker.java pertence ao pacote ***Employee,*** que é um conjunto de classes que disponibiliza funcionalidade para cálculo de vencimentos de funcionários em diferentes categorias (DEITEL; DEITEL, 2003). A subclasse ***CommissionWorker*** representa os funcionários pagos com salário básico fixo mais uma porcentagem sobre as vendas.

Resumo

Nome	CommissionWorker
Inclusão	1/12/2003
Tipo	Classe
Origem	Exemplo de Livro
Descrição	Sub classe da superclasse Employee - Realiza cálculo de folha de pagamento na categoria funcionário comissionado
Fase de Uso	Implementação
Domínio	Recursos Humanos

Área de Negócio	Cálculo de salário

Versão do Componente

Versão	**Cataloga**	**Pasta**	**Observação**
v1.0	1/12/2003	comp4_v10	

Artefato de Software

Descrição	**Tipo**	**Localização Pasta**	**Nome Arquivo**	**Software**	**Observação**
Código fonte	Código fonte	src/Employee	CommissionWorker.java	Java-SDK	

Dependência de outros componentes: Não tem.

Autor

Nome	**E-mail**
P.J. Deitel	deitel@deitel.com
H. M. Deitel	deitel@deitel.com

Ambiente

Tipo Ambiente	**Nome**	**Observação**
Linguagem de Programação	Java	

Interface: Não tem

Serviços: CommissionWorker

Serviço	**Descrição**	**Tipo Retorno**
CommissionWorker	Método construtor da classe	

Parâmetros dos Serviços

Id.	**Nome Parâmetro**	**Descrição**	**Tipo Parâmetro**	**Entrada/Saída**
5	quantity	Qtde itens vendida	Integer	Entrada
1	first	Primeiro nome	String	Entrada
2	last	Último nome	String	Entrada
3	salary	Valor do salário semanal	double	Entrada
4	commission	Valor comissão	double	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: Earning

Serviço	Descrição	Tipo Retorno
earning	Retorna os vencimentos	double

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: setSalary

Serviço	Descrição	Tipo Retorno
setSalary	configura o salário	void

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	weeklySalary	Valor do salário semanal	double	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: setCommission

Serviço	Descrição	Tipo Retorno
setCommission	configura o valor da comissão	void

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	itemCommission	Salário comissão	double	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: toString

Serviço	Descrição	Tipo Retorno
toString	Retorna o nome completos	string

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

7.5 Componente de Software: PieceWorker

A subclasse **PieceWorker.java** pertence ao pacote ***Employee,*** que é um conjunto de classes que disponibiliza funcionalidade para cálculo de vencimentos de funcionários em diferentes categorias (DEITEL; DEITEL, 2003). A subclasse **PieceWorker** os funcionários pagos pelo número de itens produzidos.

Resumo

Nome	PieceWorker
Inclusão	1/12/2003
Tipo	Classe
Origem	Exemplo de Livro
Descrição	Sub classe da superclasse Employee - Realiza cálculo de folha de pagamento na categoria funcionário por produção
Fase de Uso	Implementação
Domínio	Recursos Humanos
Área de Negócio	Cálculo de salário

Versão do Componente

Versão	Cataloga	Pasta	Observação
v1.0	1/12/2003	comp5_v10	

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Código fonte	Código fonte	src/Employee	PieceWorker.java	Java-SDK	

Dependência de outros componentes: Não tem.

Autor

Nome	E-mail
P.J. Deitel	deitel@deitel.com
H. M. Deitel	deitel@deitel.com

Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	

Interface: Não tem.

Serviços: PieceWorker

Serviço	Descrição	Tipo Retorno
PieceWorker	Método construtor da classe	

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída

1	first	Primeiro nome	String	Entrada
2	last	Último nome	String	Entrada
3	wage	Valor da peça produzida	double	Entrada
4	numberOfItems	Quantidade produzida	Integer	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: earning

Serviço	Descrição	Tipo Retorno
earning	Retorna os vencimentos	double

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: toString

Serviço	Descrição	Tipo Retorno
toString	Retorna o nome completos	string

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: setWage

Serviço	Descrição	Tipo Retorno
setWage	configura o salário	void

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	wagePerPiece	Valor do salário semanal	double	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: setQuantity

Serviço	Descrição	Tipo Retorno
setQuantity	configura a quantidade produzida	void

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	numberOfItems	Itens produzidos	Integer	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

7.6 Componente de Software: HourlyWorker

A subclasse **HourlyWorker.java** pertence ao pacote ***Employee,*** que é um conjunto de classes que disponibiliza funcionalidade para cálculo de vencimentos de funcionários em diferentes categorias (DEITEL; DEITEL, 2003). A subclasse ***HourlyWorker*** representa os funcionários que são pagos por hora e que recebem um adicional por hora extra trabalhada.

Resumo

Nome	HourlyWorker
Inclusão	1/12/2003
Tipo	Classe
Origem	Exemplo de Livro
Descrição	Sub classe da superclasse Employee - Realiza cálculo de folha de pagamento na categoria funcionário horista
Fase de Uso	Implementação
Domínio	Recursos Humanos
Área de Negócio	Cálculo de salário

Versão do Componente

Versão	Cataloga	Pasta	Observação
v1.0	1/12/2003	comp6_v10	

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Código fonte	Código fonte	src/Employee	HourlyWorker.java	Java-SDK	

Dependência de outros componentes: Não tem.

Autor

Nome	E-mail
P.J. Deitel	deitel@deitel.com
H. M. Deitel	deitel@deitel.com

Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	

Interface: Não tem.

Serviços: HourlyWorker

Serviço	Descrição	Tipo Retorno
HourlyWorker	Método construtor da classe	

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	first	Primeiro nome	String	Entrada
2	last	Último nome	String	Entrada
3	wagePerHour	Valor do hora	double	Entrada
4	hoursWorked	Quantidade de horas	double	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: toString

Serviço	Descrição	Tipo Retorno
toString	Retorna o nome completos	string

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: earning

Serviço	Descrição	Tipo Retorno
earning	Retorna os vencimentos	double

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: setWage

Serviço	Descrição	Tipo Retorno
setWage	configura o salário	void

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	wagePerHour	Valor da hora	double	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: setHours

Serviço	Descrição	Tipo Retorno
setHours	configura o quatidade horas	void

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	hoursWorked	Quantidade horas semana	double	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

7.7 Componente de Software: Employee

A superclasse **Employee.java** pertence ao pacote ***Employee,*** que é um conjunto de classes que disponibiliza funcionalidade para cálculo de vencimentos de funcionários em diferentes categorias (DEITEL; DEITEL, 2003).

A classe ***Employee*** é a superclasse, composta pelas seguintes subclasses:

- ***Boss***: representa os funcionários pagos com salário fixo semanal independentemente do número de horas trabalhadas;
- ***CommissionWorker***: representa os funcionários pagos com salário básico fixo mais uma porcentagem sobre as vendas;
- ***PieceWorker***: representa os funcionários pagos pelo número de itens produzidos;
- ***HourlyWorker***: representa os funcionários que são pagos por hora e que recebem um adicional por hora extra trabalhada.

Resumo

Nome	Employee
Inclusão	1/12/2003
Tipo	Classe
Origem	Exemplo de Livro
Descrição	Superclasse Employee - Realiza cálculo de folha de pagamento nas diferentes categorias de funcionário
Fase de Uso	Implementação
Domínio	Recursos Humanos
Área de Negócio	Cálculo de salário

Versão do Componente

Versão	Cataloga	Pasta	Observação
v1.0	1/12/2003	comp7_v10	super-classe Employee

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Código fonte	Código fonte	src/Employee	CommissionWorker .java	Java-SDK	
Código html	Código fonte	src	Employee.html	browser	
Especificação de classe gerada com JavaDoc	Documentação	docs	index.html	browser	
código java	Código fonte	src	Test.java	Java-SDK	
código fonte	Código fonte	src/Employee	Employee.java	Java-SDK	

Dependência de outros componentes

Componente	Versão
Boss	v1.0
PieceWorker	v1.0
CommissionWorker	v1.0
HourlyWorker	v1.0

Autor

Nome	E-mail
P.J. Deitel	deitel@deitel.com
H. M. Deitel	deitel@deitel.com

Descrição Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	

Interface: Não tem.

Serviços: Employee

Serviço	Descrição	Tipo Retorno
Employee	Método construtor da classe	

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	first	Primeiro nome	String	Entrada
2	last	Último nome	String	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: toString

Serviço	Descrição	Tipo Retorno
toString	Retorna o nome completos	string

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: getLastName

Serviço	Descrição	Tipo Retorno
getLastName	Retorna o último nome	string

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: getFirstName

Serviço	Descrição	Tipo Retorno
getFirstName	Retorna o primeiro nome	string

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: earning

Serviço	Descrição	Tipo Retorno
earning	Retorna os vencimentos	double

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

7.8 Componente de Software: Business Delagate

O padrão **Business Delegate** auxilia na separação da camada de apresentação da camada de serviço em uma aplicação. Este exemplo foi extraído do livro de Adatia e outros (2001) e os artefatos de software associados ao padrão são: o *template* que descreve o padrão e o código fonte que mostra a implementação do mesmo.

Resumo

Nome	BusinessDelagate

Inclusão	30/11/2003
Tipo	Padrão de projeto
Origem	Exemplo de Livro
Descrição	Auxilia na separação da camada de apresentação e da camada de serviços em uma aplicação
Fase de Uso	Projeto
Domínio	Padrão da camada de negócios
Área de Negócio	Separação das camadas de apresentação e negócios

Versão do Componente

Versão	Cataloga	Pasta	Observação
v1.0	30/11/2003	comp8_v10	

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Código fonte da implementação do padrão	Código	src	BusinessDelegate.java	Java-SDK	
Template descritivo	Documentação	docs	Template.doc	MS Word	

Dependência de outros componentes

Componente	Versão
Adapter	v1.0
LocatorService	v1.0
Proxy	v1.0

Autor

Nome	E-mail
Nathan Nagarajan	

Descrição Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	

Interface: Não tem.

Serviços: getId

Serviço	Descrição	Tipo Retorno
getId	Manipula a chamada do LookupService	void

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: Invoke

Serviço	Descrição	Tipo Retorno
Invoke	Aciona o serviço	void

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

7.9 Componente de Software: BillingSystem

O componente **BillingSystem** implementa a funcionalidade de um sistema de faturamento para uma rede de hotéis, foi extraído do livro de Cheesman e Daniels (2001). O componente **ReservationSystem** necessita deste componente para executar sua funcionalidade.

Resumo

Nome	BillingSystem
Inclusão	30/11/2003
Tipo	Componente UML
Origem	Exemplo de Livro
Descrição	Sistema de faturamento de uma rede de hotéis
Fase de Uso	Projeto
Domínio	Hotelaria
Área de Negócio	Controle de Contas

Versão do Componente

Versão	Cataloga	Pasta	Observação
v1.0	30/11/2003	comp12_v10	

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Modelo UML	Modelo	rose	reserv.mdl	Rational	

				Rose	
Especificação do modelo UML	Documentação	docs	index.html	browser	documentação gerada através do rational rose

Dependência de outros componentes: Não tem.

Autor

Nome	E-mail
John Cheesman	john.cheesman@umlcomponents.com
John Daniels	john.daniels@umlcomponents.com

Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	

Interface Disponibiliza: IBilling

Serviço
openAccount

Serviços: openAccount

Serviço	Descrição	Tipo Retorno
openAccount	Abre uma conta para o hóspede, iniciando sua estadia	

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	res	Detalhes da reserva	ReservationDetails	Entrada
2	cus	Detalhes do cliente	CustomerDetails	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

7.10 Componente de Software: CustomerMgr

O componente **CustomerMgr** implementa a funcionalidade de controlar as informações dos clientes da rede de hotéis, foi extraído do livro de Cheesman e Daniels (2001). O componente **ReservationSystem** necessita deste componente para executar sua funcionalidade.

Resumo

Nome	CustomerMgr
Inclusão	30/11/2003
Tipo	Componente UML
Origem	Exemplo de Livro
Descrição	Realiza o controle de informações de cliented de uma rede de hotéis
Fase de Uso	Projeto
Domínio	Hotelaria
Área de Negócio	Controle de Clientes

Versão do Componente

Versão	Cataloga	Pasta	Observação
v1.0	30/11/2003	comp13_v10	

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Modelo UML	Modelo	rose	reserv.mdl	Rational Rose	
Especificação do modelo UML	Documentação	docs	index.html	browser	documentação gerada através do rational rose

Dependência de outros componentes: Não tem.

Autor

Nome	E-mail
John Cheesman	john.cheesman@umlcomponents.com
John Daniels	john.daniels@umlcomponents.com

Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	

Interface Oferecida: ICustomerMgt

Serviço
getCustomerMatching
createCustomer
getCustomerDetails
notifyCustomer

Serviços: getCustomerMatching

Serviço	Descrição	Tipo Retorno
getCustomerMatching	Retorna a verificação da identificação do cliente	integer

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	cusD	Detalhes do cliente	CustomerDetails	Entrada
2	cusId	Identifcação do cliente	cusId	Saída

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: createCustomer

Serviço	Descrição	Tipo Retorno
createCustomer	Cria um cliente	booleanr

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	cusD	Detalhes do cliente	CustomerDetails	Entrada
2	cusId	Identifcação do cliente	cusId	Saída

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: getCustomerDetails

Serviço	Descrição	Tipo Retorno
getCustomerDetails	Retorna as informações em detalhe do cliente	CustomerDetails

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	cus	Identifcação do cliente	cusId	Entrada

Descrição Pré-condição e Pós-condição:

```
pre:
    --cus is a valid customer id
    customer->exists(c|c.id=cus)
post:
   --the details returned match the detais of the customer
   --whose id is cus
   --find the customer
   Let theCust=customer->select(c.id=cus) in
            -- specify the result
           result.name=theCust.name and
           result.postCode=theCust.postCode and
           result.email=theCust.email
```

Serviços: notifyCustomer

Serviço	Descrição	Tipo Retorno
notifytCustomer	Envia mensagem de notificação para cliente	

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	cus	Identifcação do cliente	cusId	Entrada
2	msg	Mensagem de notificação	String	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

7.11 Componente de Software: HotelMgr

O componente **HotelMgr** implementa a funcionalidade de controlar as informações dos hotéis da rede de hotéis, foi extraído do livro de Cheesman e Daniels (2001). O componente **ReservationSystem** necessita deste componente para executar sua funcionalidade.

Resumo

Nome	HotelMgr
Inclusão	30/11/2003
Tipo	Componente UML
Origem	Exemplo de Livro
Descrição	Realiza o controle de informações dos hotéis de uma rede de hotéis
Fase de Uso	Projeto
Domínio	Hotelaria
Área de Negócio	Controle de hotéis

Versão do Componente

Versão	Cataloga	Pasta	Observação
v1.0	30/11/2003	comp14_v10	

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Modelo UML	Modelo	rose	reserv.mdl	Rational Rose	
Especificação do modelo UML	Documentação	docs	index.html	browser	documentação gerada através do rational rose

Dependência de outros componentes: Não tem.

Autor

Nome	E-mail
John Cheesman	john.cheesman@umlcomponents.com
John Daniels	john.daniels@umlcomponents.com

Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	

Interface Oferecida: IHotelMgt

Serviço
getHotelDetails
getRoomInfo
makeReservation
getReservation
beginStay

Serviços: getHotelDetails

Serviço	Descrição	Tipo Retorno
getHotelDetails	Retorna as informações em detalhe de um hotel da rede	HotelDetails

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	match	Identifcação do hotel	String	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: getRoomInfo

Serviço	Descrição	Tipo Retorno
getRoomInfo	Retorna informações sobre um quarto	

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	res	Detalhes da reserva	ReservationDetails	entrada
2	availability	Indicador de disponibilidade	Boolean	saida
3	price	Valor do preço	Currency	saída

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: makeReservation

Serviço	Descrição	Tipo Retorno
makeReservation	Realiza a reserva de quartos	Boolean

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	res	Detalhes da reserva solicitada	ReservationDetails	entrada
2	cus	Identificação do cliente solicitante	CusId	entrada
3	resRef	Referência da reserva	String	saída

Descrição Pré-condição e Pós-condição:

```
pre:
   -- the hotel id and room tyoe are valid
post:
   -- return value of true implies sucess
   result implies
     -- a reservation was create
     -- identify the hotel
     Let h=hotel->select(x|x.id=res.hotel)->asSequence->first in
           -- only one more reservation now than before
          (h.reservation-h.reservation@pre)->size=1 and
           --identify the reservation
           Let r=(h.reservation-h.reservation@pre)->
           asSequence->first in
             --returned ref is ref of the new reservation
             r.resRef=resRef and
             --other attributes match
             r.dates=res.dateRange and
             r.roomType,name=res.roomType and not r.claimed and
             r.customer.id=cus
```

Serviços: getReservation

Serviço	Descrição	Tipo Retorno
getReservation	Retorna as informações sobre a reserva do hotel e o cliente associado	Boolean

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	resRef	Referência da reserva	String	Entrada/saida
2	rd	Detalhes da reserva	ReservationDetails	saída
3	cusId	Identificação do cliente	CusId	saída

Descrição Pré-condição e Pós-condição:.

```
---"Normal" Case
pre:
   -- a reservation corresponding to resRef exists and it
   -- hasn't been claimed
post:
  -- result=0, reservation detais and customer ID returned
```

```
---"Reservation not found" case
 pre
    -- a reservation corresponding to resRef does not exist"
post:
   --result=1, return an appropriete reason code and message string"
```

```
---"Already Claimed" case
pre:
   -- the reservation has been claimed
```

post: -- result=2, return an apropriete reason code and message string”

Serviços: beginStay

Serviço	Descrição	Tipo Retorno
beginStay	Inicia a estadia de um hospede	Boolean

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	resRef	Número de referência da reserva	String	entrada/saída
2	roomNumber	Número do quarto	String	saída

Descrição Pré-condição e Pós-condição: Não tem.

7.12 Componente de Software: ReservationSystem

O componente **ReservationSystem** contém a especificação em UML de um componente para controle de reservas de quartos de uma rede de hotéis, extraído do livro de Cheesman e Daniels (2001). O componente deve permitir o controle das reservas solicitadas pelos clientes até o *check-in* ou cancelamento da mesma; quando o cliente realiza o *check-in*, o componente de faturamento (**BillingSystem**) é avisado do início da estadia de um cliente.

Para isso, necessita de utilizar os serviços de outros três componentes:

- **BillingSystem**: sistema de faturamento que deve ser informado do início da estadia de um cliente que ocupa um quarto;
- **CustomerMgr**: componente de software que deve controlar as informações do cliente;
- **HotelMgr**: componente de software que deve controlar as informações referentes aos hotéis.

Resumo

Nome	ReservationSystem
Inclusão	30/11/2003
Tipo	Componente UML
Origem	Exemplo de Livro
Descrição	Realiza o controle de reservas de quartos em uma rede de hotéis
Fase de Uso	Projeto

Domínio	Hotelaria
Área de Negócio	Controle de reservas

Versão do Componente

Versão	Cataloga	Pasta	Observação
v1.0	30/11/2003	comp15_v10	

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Modelo UML	Modelo	rose	reserv.mdl	Rational Rose	
Especificação do modelo UML	Documentação	docs	index.html	browser	documentação gerada através do rational rose
Tutorial descrevendo o projeto	Documentação	dosc	Tutorial.pdf	Acrobat Reader	

Dependência de outros componentes:

Componente	Versão
BillingSystem	v1.0
CustomerMgr	v1.0
HotelMgr	v1.0

Autor

Nome	E-mail
John Cheesman	john.cheesman@umlcomponents.com
John Daniels	john.daniels@umlcomponents.com

Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	
Padrão	Enterprise JavaBeans	padrão de componente

Interface Oferecida: IMakeReservation

Serviço
getHotelDetails
getRoomInfo

makeReservation

Serviços: getHotelDetails

Serviço	Descrição	Tipo Retorno
getHotelDetails	Retorna as informações em detalhe de um hotel da rede	HotelDetails

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	match	Identifcação do hotel	String	Entrada

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: getRoomInfo

Serviço	Descrição	Tipo Retorno
getRoomInfo	Retorna informações sobre um quarto	

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	res	Detalhes da reserva	ReservationDetails	entrada
2	availability	Indicador de disponibilidade	Boolean	saida
3	price	Valor do preço	Currency	saída

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: makeReservation

Serviço	Descrição	Tipo Retorno
makeReservation	Realiza a reserva de quartos	Boolean

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	res	Detalhes da reserva solicitada	ReservationDetails	entrada
2	cus	Identificação do cliente solicitante	CusId	entrada
3	resRef	Referência da reserva	String	saída

Descrição Pré-condição e Pós-condição:

```
pre:
   -- the hotel id and room tyoe are valid
post:
   -- return value of true implies sucess
   result implies
     -- a reservation was create
     -- identify the hotel
     Let h=hotel->select(x|x.id=res.hotel)->asSequence->first in
           -- only one more reservation now than before
          (h.reservation-h.reservation@pre)->size=1 and
           --identify the reservation
           Let r=(h.reservation-h.reservation@pre)->
```

```
asSequence->first in
  --returned ref is ref of the new reservation
  r.resRef=resRef and
  --other attributes match
  r.dates=res.dateRange and
  r.roomType,name=res.roomType and not r.claimed and
  r.customer.id=cus
```

Interface Oferecida: ITakeUpReservation

Serviço
getReservations
beginStay

Serviços: getReservation

Serviço	Descrição	Tipo Retorno
getReservation	Retorna as informações sobre a reserva do hotel e o cliente associado	Boolean

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	resRef	Referência da reserva	String	Entrada/saida
2	rd	Detalhes da reserva	ReservationDetails	saída
3	cusId	Identificação do cliente	CusId	saída

Descrição Pré-condição e Pós-condição:.

```
---"Normal" Case
pre:
    -- a reservation corresponding to resRef exists and it
    -- hasn't been claimed
post:
   -- result=0, reservation detais and customer ID returned
```

```
---"Reservation not found" case
 pre
     -- a reservation corresponding to resRef does not exist"
post:
    --result=1, return an appropriete reason code and message string"
```

```
---"Already Claimed" case
pre:
    -- the reservation has been claimed
post:
    -- result=2, return an apropriete reason code and message string"
```

Serviços: beginStay

Serviço	Descrição	Tipo Retorno
beginStay	Inicia a estadia de um hospede	Boolean

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	resRef	Número de referência da	String	entrada/saída

		reserva		
2	roomNumber	Número do quarto	String	saída

Descrição Pré-condição e Pós-condição: Não tem.

Interface Realiza:

Componente	**Versão**	**Interface**
BillingSystem	v 1.0	IBilling
CustomerMgr	v 1.0	ICustomerMgr
HotelMgr	v 1.0	IHotelMgt

7.13 Componente de Software: Sistema de Simulação de Sistema de Elevador

O sistema de simulação de elevador é o componente de software de mais alta granularidade no conjunto de componentes selecionados para a avaliação do repositório, foi extraído livro de Deitel e Deitel (2003) e também se baseia em alguns documentos e informações extraídos do *site* dos autores do livro disponível em www.deitel.com.br.

Este sistema é um aplicativo de simulação de software orientado a objetos que modela a operação de um elevador, a finalidade deste componente é exercitar a representação de um componente de software com mais de uma versão, uma vez que, existe implementação do componente na linguagem C++ e Java.

Os modelo UML da versão para a linguagem Java foi adotada a arquitetura Model-View-Controler (MVC), desta forma, a versão na liguagem Java é composta por três componentes: ElevatorModel, ElevatorView e ElevatorController. Já a versão implementada na linguagem C++ é formada pro um pacote de classes.

Resumo

Nome	ElevatorSimulation
Inclusão	18/12/2003
Tipo	Sistema
Origem	Exemplo de Livro

Descrição	Realiza a simulação de um sistema de controle de levador
Fase de Uso	Implementação
Domínio	Elevador
Área de Negócio	Simulação de funcionamento

Versão do Componente: Linguagem Java

Versão	Cataloga	Pasta	Observação
v1.0	30/11/2003	comp20_v10	Versão Implementada na linguagem Java

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Especificação de Requisitos	Documentação	Docs	Elevator.doc	Ms. Word	
Modelo UML	Modelo	Modelo	ElevatorJava.mdl	Rational Rose	
Especificação do modelo UML	Documentação	Modelo/Docs	Index.Html	Browser	Documentação gerada pelo Rational Rose
Código fonte java	Código	src	ElevatorSimulation. java	Java - SDK	

Dependência de outros componentes:

Componente	Versão
ElevatorModel	v1.0
ElevatorControler	v1.0
ElevatorView	v1.0

Autor

Nome	E-mail
H. M. Deitel	deitel@deitel.com.br
P. J. Deitel	deitel@deitel.com.br

Ambiente

Tipo Ambiente	Nome	Observação
Linguagem de Programação	Java	

Restrição:

Tipo	Descrição
Copyright	(C) Copyright 2003 by Deitel & Associates, Inc. and Prentice Hall. All Rights Reserved DISCLAIMER: The authors and publisher of this book have used best efforts in preparing the book. These efforts include the development, research, and testing of the theories and to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Serviços: ElevatorSimulation

Serviço	Descrição	Tipo Retorno
ElevatorSimulation	Método construtor que instancia os componentes Model, View e controller	

Parâmetros dos Serviços: Não tem

Descrição Pré-condição e Pós-condição: Não tem.

Serviços: main

Serviço	Descrição	Tipo Retorno
main	Método main que inicia a execução do programa	void

Parâmetros dos Serviços

Id.	Nome Parâmetro	Descrição	Tipo Parâmetro	Entrada/Saída
1	args[]	Argumento	string	

Descrição Pré-condição e Pós-condição: Não tem.

Interface Oferecida: Não tem

Interface Realiza: não tem

Versão do Componente: Linguagem C++

Versão	Cataloga	Pasta	Observação
v2.0	30/11/2003	comp20_v20	Versão Implementada na linguagem C++

Artefato de Software

Descrição	Tipo	Localização	Nome Arquivo	Software	Observação

		Pasta			
Especificação de Requisitos	Documentação	docs	Elevator.doc	MS Word	
Modelo UML	Modelo	Modelo	Elevator.zargo	Argo UML	
Relação de Código fonte C++ compactado	Código	src	ElevatorSimulation.zip	WinZip	

Dependência de outros componentes: Não tem

Autor

Nome	**E-mail**
H. M. Deitel	deitel@deitel.com.br
P. J. Deitel	deitel@deitel.com.br

Ambiente

Tipo Ambiente	**Nome**	**Observação**
Linguagem de programação	C++	
Sistema Operacional	Windows	

Restrição:

Tipo	**Descrição**
Copyright	(C) Copyright 2000 by Deitel & Associates, Inc. and Prentice Hall. All Rights Reserved DISCLAIMER: The authors and publisher of this book have used best efforts in preparing the book. These efforts include the development, research, and testing of the theories and to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in these books. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Serviços: main

Serviço	**Descrição**	**Tipo Retorno**
main	Inicia a executção do programa	

Parâmetros dos Serviços: Não tem

Descrição Pré-condição e Pós-condição: Não tem.

Interface Oferecida: Não tem

Interface Realiza: Não tem

7.14 Componente de Software: Projeto de Sistema de Elevador – Análise Estruturada

O projeto de um sistema escalonador e controlador de elevador, extraído do livro de Yourdon (1990), apresenta um conjunto de documentos de um sistema de tempo real, descrito através da técnica estruturada.

Resumo

Nome	ControleElevador
Inclusão	04/01/2004
Tipo	Sistema
Origem	Exemplo de Livro
Descrição	Realiza o o escalonamento e controle de um sistema de elevadores
Fase de Uso	Projeto
Domínio	Elevador
Área de Negócio	Controle de operação

Versão do Componente

Versão	Cataloga	Pasta	Observação
v1.0	04/01/2004	comp20_v10	Projeto de sistema de tempo real

Artefato de Software

Descrição	Tipo	Localização Pasta	Nome Arquivo	Software	Observação
Descrição narrativa	Documentação	docs	DescNarrativa.doc	MS. Word	
Especificação de Processos	Documentação	docs	EspProc.doc	MS. Word	
Diagrama (DFD e DE)	Modelo	docs	elevador.	DOME	disponível para download em www.honeywell.com.br/dome

Dependência de outros componentes: Não tem.

Autor

Nome	E-mail
Edward Yourdon	

Ambiente: Não tem.

Interface Oferecida: Não tem.

Serviços: Não tem.

Parâmetros dos Serviços: Não tem.

Descrição Pré-condição e Pós-condição: Não tem.

Interface Realiza: Não tem.

8. LISTA DE REFERÊNCIAS

ADATIA. R.; et al. **Professional EJB**.Birmingham: Wrox Press, 2001. 1257p. (Wrox Series in Programmer to Programmer)

ANDREWS, A.; GHOSH, S.; CHOI, E. M. A model for Understanding Software Components**.** In**: Internetional Conference on Software Maintenance,** 2002 Proceedings, February 2002.

APPERLY, H. Configuration Management and Component Libraries. In: HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001.

BACHMAN, F. et al. **Volume II: Technical Concepts of Component-Based Software Engineering**. Pittsburgh: Software Engineering Institute, May 2000. (Technical Report CMU/SEI-2000-TR-008)

BARNES, B; et al. A Framework and Economics Foundation for Software Reuse. **Tutorial: Software Reuse – Emerging Technology**. p 77-88, Computer Society Press, 1988.

BASILI, V.R.; CALDIERA, G.; CANTONE, G. A Reference Architecture for the Component Factory. **ACM Transactions on Software Engineering and Methodology**, v.1, n.1, p. 53-80, January 1992.

BASILI, V.R.; BRIAND, C.; MELO, W.L. How Reuse Influences Produtivity in Object-Oriented Systems. **Communications of the ACM**, v.39, n.10, p.104-126, Octuber 1996.

BASS, L. et al. ***Volume I: Market Assessment of Component-Based Software Engineering***. Pittsburgh: Software Engineering Institute, May 2001. (Technical Note CMU/SEI-2001-TN-007)

BATORY, D.; O'MALLEY, S. The Design and Implementation of Hierarchical Software Systems with Reusable Components. **ACM Transactions on Software Engineering and Methodology**, vol. 1, no. 4, p. 355-398, October 1992.

BIGGERSTAFF, T; RICHTER, C. Reusability Framework, Assessment, and Directions. **IEEE Software**, v. 4, n.2, p. 41-49, March 1987.

BOEHM, B. Managing Software Productivity and Reuse. **Computer**, p. 111-113, September 1999.

BOOCH, G., **Software Components with Ada**, Benjamin-Cummings, 1987.

BOOCH, G; RUMBAUGH, J.; JACOBSON, I. ***The Unified Modeling Language: User Guide.*** Addison-Wesley, Reading, MA, 1998.

BROWN, A. W.; SHORT, K. On components and objects: the foundations of component-based development**.** Assessment of Software Tools and Technologies, 1997. In: **Proceedings Fifth International Symposium**, p. 112-121, June 1997.

BROWN, A. W.; WALNAU, K.C. The Current State of CBSE. **IEEE Software**, v.13, n. 9, p. 37-46, September/Octuber 1998.

BROWN, A. W.; WALLNAU, K.C. Engineering of Component Based Systems. **Component-Based Software Engineering**, IEEE Computer Society Press, p. 7-15, 1996.

BROWN, A.W. **Large-Scale Component-Based Development**. 1. ed., New Jersey: Prentice Hall PTR, Object and Component Technology Series, 2000.

BURTON, B.A.; et al. The Reusable Software Library. **IEEE Software**, p.25-33, July 1987.

CARD, D.; COMER, E. Why do so many Reuse Programs fail? **IEEE Software**, v.11, n.5, p. 114-115, September 1994.

CASANOVA, M.; STRAETEN, R. V.; JONCKERS, V. Supporting Evolution in Component-Based Development using Component Libraries. **Proceedings of the Seventh European Conference on Software Maintenance and Reengineering** (CSMR'03), IEEE Computer Society, 2003.

CHENG, J. Improving the Software Reusability in object-oriented Programming. **ACM SIGSOFT Software Engineering Notes**, v.18, n.4, p.70-74, Octuber 1993.

CHEESMAN, J.; DANIELS, J. **UML Components: A Simples Process for SpecifyingComponent-Based Software**. Addison-Wesley, 2001.

CLEMENTS, P.C. From Subrotines to Subsystems: Component-Based Software Development. **The American Programmer**, v. 8, n. 11, November 1995.

COX, B.J. Planning the Software Revolution. **IEEE Software**, v.7, n. 6, p.25-35, November 1990.

CREECH, M. L.; FREEZE, D. F.; GRISSM. L. Using Hypertext in Selecting Reusable Software Components. **Hypertext'91 Proceedings**, San Antonio, Texas, USA, p. 25-38, December 1991.

DAMIANI, E.; FUGINI, M.G.; FUSASCHI, E. A Descriptor-Based Aproach to OO Code Reuse. **Computer**, p. 73-80, October 1997.

DEITEL, H. M.; DEITEL, P. J. **Java, Como Programar**. Porto Alegre: Bookman, 2003. 1386p.

DEVANBU, P.; et al. LaSSIE: A Knowledge-BAsed Software Information System. **Communications of the ACM**, v.34, n. 5, p.34-39, May 1991.

DIGRES, T., Business Object Component Architecture. **IEEE Software**, p. 60-69, September/Octuber 1998.

D'SOUZA, D.; WILLS, A. C. **Catalysis: A Next Generation UML-Based Method**. Addison-Wesley, 1999.

EDDON, G. COM+: The Evolution of Component Services. **Computer**, v. 32, n. 7, p. 54-61, July 1999.

IEEE std 1517-1999, IEEE Standard for Information Technology – Software Life Cycle Processes – Reuse Processes, **Software Engineering Standards Committee**, July 1999.

FAFCHAMPS, D. Organizational Factors and Reuse. **IEEE Software**, v.11, n.5, p.31-41, September 1994.

FLYNT, J.; DESAI, M. The Future of Software Components: Standards and Certification. In: HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001.

FRAKES, W.B.; GANDEL, P.B. Representing Reusable Software. **Information and Software Technology**, v. 35, n.10, p.653-664, 1990.

FRAKES, W.B.; ISODA, S. Success Factors of Systematic Reuse. **IEEE Software**, v.11, n.5, pp.15-19, September 1994.

FRAKES, W.; TERRY, C. Software Reuse: Metrics and Models. **ACM Computing Surveys**, v. 28, n. 2, p. 415-435, June 1996.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, Reading, MA, 1995.

GARLAN, D.; ALLEN, R.; OCKERBLOOM, J. Architectural Mismatch: Why Reuse is so Hard. **IEEE Software**, v. 12, n. 6, p. 17-26, November 1995.

GIOVANI, P.; MELNIKOFF, S. S.S. **Reutilização em Software Orientado a Objetos: Um Estudo Empírico para Analisar a Dificuldade de Localização e Entendimento de Classes**. Boletim Técnico da Escola politécnica da USP, Departamento de Engenharia de Computação e Sistemas Digitais, BT/PCS/9612, 1996.

GRISS, M. CBSE Success Factors: Integrating Architecture, Process, and Organization. In: HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001.

GUO, J.; LUQI A Survey of Software Reuse Repositories. In: Proceedings, The Seventh IEEE International Conference and Workshop, **Engineering of Computer Based Systems**, p. 92-100, 2000.

HALL, P. Architecture-driven Component Reuse. **Information and Software Technology**, v. 41, p. 963-968, 1999.

HALL, M. **More Servlets and Java Server Pag**es. Prentice Hall PTR, Sun Microsystems Press Publisher, 2002.

HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001

HENNINGER, S. An Evolutionary Approach to Constructing Effective Software Reuse Repositories. **ACM Transactions on Software Engineering and Methodologies**, v.16, n.2, p. 111-140, April 1997.

HENRY, E.; FALLER, B. Large Scale Industrial Reuse to Reduce Cost and Cycle Time. **IEEE Software**, v. 12, n. 5, p. 47-53, September 1995.

HERZUM, P.; SIMS, O. **Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise**, New York: John Wiley & Sons, 2000.

HISSAN, S. A.; SEACORD, R. C.; LEWIS, G. A. **Bulding System from Commercial Components** (tutorial). 24th International Conference on Software Engineering Orlando, FL, May, 2002.

HOPKINS, J. Component Primer: Laying the Foundation. **Communications of the ACM**, v. 43, n.10, p. 27-30, October 2000.

HOUSTON, K.; NORRIS, D. Software Components and the UML. In: HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001.

IEEE STD 1517-1999, **IEEE Standard for Information Tecnology – Software Life Cycle Process – Reuse Process**. Software Engineering Standards Committee of the IEEE Computer Society, July 1999.

ISODA, S. Progress in Reuse in Japan. **IEEE Software**, v.11, n.5, p. 18, September 1994.

JACOBSON, I.; GRISS, M.; JONSSON, P. ***Software Reuse: Architecture, Process and Organization for Business Success***. 1ª. ed, ACM Press, 1997.

JACOBSON, I.; BOOCH, G; RUMBAUGH, J. ***The Unified Software Development Process***. New Jersey: Addison-Wesley, 1998. 463p.

JONES, C. Economics of Software Reuse. **Computer**, p. 106-107, July 1994.

KALAKOTA, R.; ROBINSON, M. **e-Bussines Estratégias para Alcançar Sucesso no Mundo Digital**. São Paulo: Editora Bookman, 2º. Edição, 2002.

KOZACZYNSKI, W. **Composite Nature of Component**, International Worshop on Component-Based Software Engineering, Los Angeles, USA, May, 1999.

KRUCHTEN, P. Modeling **Component Systems with the Unified Modeling Language**, International Workshop on Component-Based Software Engineering (Position Paper), 1998.

KRUCHTEN, P. **The Rational Unified Process:** An Introduction.Upper Saddle River: Addison-Wesley, 2000. 298 p.

KRUEGER, C.W. Software Reuse. **ACM Computing Surveys**, v. 24, n. 2, p. 131-183, 1992.

LAUDON, K.C.; LAUDON, J.P. **Gerenciamento de Sistemas de Informação**. Rio de Janeiro: LTC, 2001.

LENZ, M.; SCHMID, H. A.; WOLF, P. F. Software Reuse through Building Blocks. **IEEE Software**, p. 34- 42, July 1987.

LIM, W.C. **Managing Software Reuse**, 1. ed., New Jersey: Prentice Hall PTR, 1998.

LÜER, C.; ROSENBLUM, D. S, **WREN - An Environment for Component-Based Development**. ESEC/FSE, Vienna, Austria, p. 207- 217, 2001.

MAAREK, Y.S.; SMADJA, F.A. **Full text indexing based on lexcal relations, an application: Software libraries**, In: PROCEEDINGS OF SIGIR'89. New York: ACM, p. 198-206, 1989.

MCCLURE, C. **Software Reuse Techniques:** Adding Reuse to the Ssytem Development Process. 1ª ed., New Jersey: Prentice Hall PTR, 1997.

MCKINSEY, B. Status of CBSE in Europe. In: HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001.

MEYER, B. Reusability: The case for object-oriented design. **IEEE Software**, v. 4, n. 2, March 1987.

MEYER, B. **Reusable Software: The Base Object-Oriented Component Libraries**, Prentice-Hall International, 1994.

MEYER, B.; MINGINS, C.; SCHMIDT, H. Providing Trusted Components to the Industry. **Computer**, p. 104-105, May 1998.

MEYER, B. On To Components. **Computer**, p. 139-140, January 1999.

MILI, H.; MILI, F.; MILI, A. Reusing Software: Issues and Research Directions. **IEEE Transactions on Software Engineering**, v. 21, n. 6, June 1995.

MORISIO, M.; TULLY, C.; EZRAN, M. Diversity in Reuse Process. **IEEE Software**, p. 56- 63, July/August 2000.

POULIN, J.S. Reuse: Been There, Done That. **Communications of the ACM**, v. 42, n.5, p. 98-100, May 1999.

POULIN, J. S. Measurement and Metrics for Software Components. In: HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001.

POUR, G. Component-based software development approach: new opportunities and challenges**.** In**: Technology of Object-Oriented Languages, 1998.** Proceedings, 3-7 August 1998, p. 376 - 383

PRESSMAN, R. **Software Engineering:** A Practioner's Approach. 5ª. ed., United Kingdom:McGraw-Hill, 2000.

PRIETO-DÍAZ, R.; FREEMAN, P. Classifying Software for Reusability**. IEEE Software**, v. 4, n. 1, p. 6-16, January 1987.

PRIETO-DÍAZ, R.; JONES, G. Breathing New Life Into Old Software. **GTE Journal of Science and Technology**, 1987, Spring. **1**(1): p. 22-31.

PRIETO-DÍAZ, R. Status Report: Software Reusability. **IEEE Software**, v.10, n. 3, p. 61-66, May 1993.

RATIONAL SOFTWARE, Reusable Asset Specification, 2001. Disponível em: http://www.rational.com/rda/ras/preview/index.htm . Acesso em: Janeiro/2004.

REIFER, D. J. Implementing a Practical Reuse Program for Software Components. In: HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001.

RIBOT, D; BONGARD, B.; VILLERMAIN, C. Development Life-Cycle with Reuse. In: ACM Symposium on Applied computing, March 1994. **Proceedings**, ACM Press, 1994, p.70-76.

RINE, D. C. Supporting Reuse with Object Technology. **Computer**, p. 43-45, October 1997.

SAMETINGER, J. Reuse **Documentation and Documentation Reuse**. Proceedings of TOOLS Europe '96, Paris, February 1996.

SANTOS, R. **Introdução à Programação Orientada a Objetos Usando Java**. Rio de Janeiro: Campus, 2003. 319p.

SIEGEL, J. A Preview of CORBA 3. **Computer**, v. 32, n. 5, p. 114-116, May 1999.

STAFFORD, J. A.; WOLF, A. L. Software Architecture. In: HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001.

STALLINGER, F.; et al. **Software Process Improvement for Component-Based Software Engineering: Na Introduction to the OOSPICE Project**, In: PROCEEDINGS OF THE 28th EUROMICRO CONFERENCE, EUROMICRO'02: IEEE Computer Society, 2002.

STETS, R.J.; HUNT, G.C.; SCOTT, M.L. Component-Based APIs for Versioning and Distributed Applications. **Computer**, v. 32, n. 7, p. 54-61, July 1999.

TAKATA, K. **Metodologia para utilização e reutilização de componentes de software**. 1999. 160p. Dissertação (Mestrado) – Escola Politécnica, Universidade de São Paulo. São Paulo.

TRACZ, W. ***Software Reuse: motivators and inhibitors***, Workshop on Future Directions in Computer Architecture and Software, Stanford University, p.5-8, May 1986.

UML Revision Task Force, OMG Unified Modeling Language Specification, v. 1.3, document ad/99-06-08. Object Management Group, June 1999.

VITHARANA, P.; ZAHEDI, F. M.; JAIN, H. Knowledge-Based Repository Scheme for Storing and Retrieving Business Components: A Theoretical Design and Empirical Analysis. **IEEE Transactions on Software Engineering**, v. 29, n.7, July 2003.

VOAS, J. Certifying Off-the-Shelf Software Components. **Computer**, v.31, n. 6, p. 53-59, June 1998.

VOAS, J. Developing a Usage-Based Software Certification Process**. Computer**, v.33 , n. 8, p. 32-37, August 2000.

WEINREICH, R.; SAMETINGER, J. Component Models and Component Service: Concepts and Principles. In: HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001

WILLIAMS, J. The Business Case for Components. In: HEINEMAN, G. T.; COUNCIL, W.T. **Component-Based Software Engineering: Putting the Pieces Together**. 1ª.ed, Addison-Wesley, 2001

WOHLIN, C., RUNESON, P. Certification of Software Components. **IEEE Transactions on Software Engineering**, v. 20, n. 6, p. 494-499, June 1994.

YACOUB, S.; AMMAR, H.; MILI, A. **Characterizing a Software Component**, International Workshop on Component-Based Software Engineering, Los Angeles, Estados Unidos, May 1999.

YOURDON, E. **Análise Estruturada Moderna**. Rio de Janeiro: Campus, 3ª. ed., 1990. 836p.