UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

METODOLOGIA PARA DESENVOLVIMENTO DE SOFTWARE RECONFIGURÁVEL APOIADA POR FERRAMENTAS DE IMPLEMENTAÇÃO: UMA APLICAÇÃO EM AMBIENTE DE EXECUÇÃO DISTRIBUÍDO E RECONFIGURÁVEL

ALUNO:

FRANK JOSÉ AFFONSO

Tese de doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica

ORIENTADOR:

PROF. DR. EVANDRO LUIS LINHARI RODRIGUES

SÃO CARLOS - SP

2009

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Affonso, Frank José
A257m Metodologia para desenvolvimento de software reconfigurável apoiada por ferramentas de implementação : uma aplicação em ambiente de execução distribuído e reconfigurável / Frank José Affonso ; orientador Evandro Luis Linhari Rodrigues -- São Carlos, 2009.

Tese (Doutorado-Programa de Pós-Graduação em Engenharia Elétrica e Área de Concentração Processamento de Sinais e Instrumentação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2009.

1. Reconfiguração de software. 2. Ambiente de execução reconfigurável. 3. Reúso. 4. Adaptação. I. Título.

Pode ser que um dia deixemos de nos falar...
Mas, enquanto houver amizade,
Faremos as pazes de novo.

Pode ser que um dia o tempo passe...
Mas, se a amizade permanecer,
Um do outro se há-de lembrar.

Pode ser que um dia nos afastemos...
Mas, se formos amigos de verdade,
A amizade nos reaproximará.

Pode ser que um dia não mais existamos...
Mas, se ainda sobrar amizade,
Nasceremos de novo, um para o outro.

Pode ser que um dia tudo acabe...
Mas, com a amizade construiremos tudo novamente,
Cada vez de forma diferente.
Sendo único e inesquecível cada momento
Que juntos viveremos e nos lembraremos para sempre.

Há duas formas para viver a sua vida:
Uma é acreditar que não existe milagre.
A outra é acreditar que todas as coisas são um milagre.

Albert Einstein

Dedico este trabalho

aos meus pais, Antonio e Odila, ao meu irmão Alex e

à minha querida esposa Cátia

Agradecimentos

Agradeço a Deus, por ter me guiado até o caminho do doutorado e por te me dado forças para que eu chegasse até aqui.

Agradeço ao Prof. Dr. Evandro Luis Linhari Rodrigues, pela oportunidade de fazer um doutorado, pela orientação, pela conduta no desenvolvimento deste trabalho, pela amizade conquistada nesses anos, pela compreensão e paciência nos bons e maus momentos. Meu sincero e eterno agradecimento.

Agradeço a meus pais, pelo esforço feito a vida toda para eu chegasse até aqui, pelo incentivo e força nesses anos. Não sei o que seria de mim sem vocês. Ao meu irmão Alex, pelo incentivo, preocupação e palavras de apoio durante esses anos.

Agradeço aos amigos do Laboratório de Visão Computacional II, Clayton, Celso, Aline, Ednilson, Evandra e a todos que por ali passaram esses anos.

Agradeço a todas as amizades formadas nesses anos, não citarei nomes para não esquecer de alguém.

Agradeço ao amigo Ednilson, pelo companheirismo e trabalhos realizados nesses anos, sua participação e contribuição no desenvolvimento deste trabalho.

Agradeço aos funcionários da pós-graduação SEL/EESC/USP.

Agradeço a todos que me ajudaram chegar até aqui.

Agradeço a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Finalmente, agradeço, em especial, à minha esposa Cátia, pelo apoio, compreensão e incentivo nos bons e maus momentos.

Meu muito obrigado.

Resumo

AFFONSO, F. J. **Metodologia para desenvolvimento de software reconfigurável apoiada por ferramentas de implementação: uma aplicação em ambiente de execução distribuído e reconfigurável**, 2009. 190 f. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2009

O desenvolvimento de software reconfigurável é uma abordagem que requer padrões nas atividades e nos artefatos produzidos ao longo da elaboração de um projeto de software. Além disso, também prevê uma conduta sistemática do pessoal envolvido, para que as diretrizes de uma metodologia sejam executadas e os benefícios por ela previstos sejam alcançados. Neste trabalho, uma metodologia para o desenvolvimento de software reconfigurável foi elaborada para apoiar essa abordagem de desenvolvimento. Como forma de auxiliar as atividades existentes nesta metodologia e padronizar as atividades por ela previstas, minimizando a participação de seres humanos (desenvolvedores), foi confeccionada uma ferramenta composta por um conjunto de subsistemas capazes de gerar, de maneira automática, informações necessárias, para que a padronização dos procedimentos possa ser realizada e, consequentemente, que a reconfiguração e reutilização dos artefatos ocorram de maneira natural. Essa ferramenta atua em um ambiente distribuído e organizado pelos domínios de atuação e a reutilização/reconfiguração pode ocorrer em artefatos confeccionados para atuar em domínio específico, mas que podem ser adaptados/reutilizados em outros.

Palavras-chaves: reconfiguração, reúso, componentes de software, metodologia, ferramenta.

Abstract

AFFONSO, F. J. **Methodology to Reconfigurable Software Development supported by implementation tools: an application in distributed and reconfigurable execution environment**, 2009. 190 f. Thesis (Doctoral) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2009

The reconfigurable software development is an approach that requires patterns in the activities and in the artifacts produced during the development of a software project. It also requires a systematic conduct of the staff involved in the methodology guidelines, so that the benefits provided can be achieved. In this work, a methodology for the reconfigurable software development was proposed to support this approach. As a way to assist the activities in this methodology and standardize the required activities, minimizing the involvement of humans (developers), a tool was proposed that consists in a set of subsystems capable of generating, in an automatic manner, information needed so that the standardization of information can be performed, therefore, that the reconfiguration and reuse of artifacts could be occur in a natural way. This tool operates in a distributed environment organized by areas of expertise, and reuse/reconfiguration can occur in artifacts constructed to operate in specific domains, but it can be adapted/reused in others.

Keywords: reconfiguration, reuse, software component, methodology, tool.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AGC	Agente Gerenciador de Configuração
API	*Application Programming Interface*
CASE	*Computer-Aided Software Engineering*
CORBA	*Common Object Request Broker Architecture*
CWM	*Common Warehouse MetaModel*
DC	*Dynamic Classloader*
DSBC	Desenvolvimento de Software Baseado em Componentes
DSR	Desenvolvimento de Software Reconfigurável
DSW	Descrição de Serviço *Web*
EJB	*Enterprise JavaBeans*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IC	Inteligência Computacional
IDE	*Integrated Development Environment*
IDL	*Interface Definition Language*
JAVASSIST	JAVA *Programming Assistant*
JDK	JAVA *Development Kit*
JNDI	JAVA *Name and Directory Interface*
JSE	JAVA *Standard Edition*
JSP	*JavaServer Pages*
JVM	JAVA *Virtual Machine*
MDSR	Metodologia para Desenvolvimento de Software Reconfigurável
MOF	*Meta-Object Facility*
OCL	*Object Constraint Language*

OMG	*Object Management Group*
OO	Orientado a Objetos
OP	Objeto *Proxy*
OR	Objeto Real
ORB	*Object Request Broker*
PMO	Protocolo de Meta Objetos
POA	Programação Orientada a Aspectos
POC	Programação Orientada a Componentes
RC	Reflexão Computacional
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais
RMI	*Remote Method Invocation*
RMI-IIOP	*Remote Method Invocation over Inter-Orb Protocol*
RNA	Redes Neurais Artificiais
RNF	Requisitos Não Funcionais
SGML	*Standard Generalized Markup Language*
SHs	Sistemas Híbridos
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SR	Sistemas Reflexivos
UML	*Unified Modeling Language*
WSD	*Web Service Description*
WSDL	*Web Service Description Language*
XML	*eXtensible Markup Language*
XRMI	*eXtended* JAVA RMI

Sumário

CAPÍTULO 1

Introdução

1.1. CONTEXTUALIZAÇÃO

Atualmente tem-se notado novas tendências no desenvolvimento de software, que deixa de ser direcionado a sistemas restritos, a domínios específicos e baixo número de usuários, passando a atingir diversos domínios e quantidade elevada de usuários. Esse cenário pode ser facilmente encontrado em empresas e instituições governamentais que possuem sistemas antigos, de boa funcionalidade, e que desejam disponibilizá-los para plataformas de softwares mais atuais, ou que desejam atender uma quantidade elevada de usuários como a *Web*.

As situações apresentadas evidenciam as mudanças de requisitos de softwares para os quais foram projetados (MAES, 1987). As novas tendências podem ser consideradas como uma nova abordagem de desenvolvimento, enquanto que a segunda, migração de sistemas, pode ser solucionada por um processo de reengenharia (PRESSMAN, 2006) de sistemas com, ou sem, mudança de plataforma e/ou interface. Para ambos os casos, um novo conceito de Engenharia de Software pode ser aplicado, que se chama integração de serviços via *web services* (ERL, 2004) (ERL, 2005). Pode-se entender *web service* como um *middleware* que permite encapsular

uma ou mais funcionalidades de um sistema e disponibilizá-la(s) na forma de serviços com interfaces de acesso bem definidas.

No que se refere à aplicação de um processo de reengenharia pode-se dizer que toda inovação tecnológica causa impacto entre os desenvolvedores por estarem acostumados a um conjunto delimitado de tecnologias e possuírem conhecimento sobre o sistema que mantinham (PRESSAN, 2006) (SOMMERVILLE, 2008) (PFLEEGER, 2004). Em contrapartida, a integração de sistemas via *web services* tende a preservar esse conhecimento, apenas exigindo que uma camada de software seja construída para realizar a troca de mensagens com os sistemas clientes (ERL, 2004) (ERL, 2005). Além disso, vale ressaltar que o desenvolvimento de novas aplicações distribuídas e voltadas para *Web* também podem utilizar a padronização de troca de mensagens via *web services* (ERL, 2005).

Quanto ao desenvolvimento de software utilizando tecnologias atuais, são abordados vários modelos, técnicas e ferramentas direcionadas ao desenvolvimento de aplicações distribuídas ou para *Web*. No entanto, tem-se notado que tecnologias de desenvolvimento e ferramentas de apoio estão direcionadas ao tipo de sistema que se pretende desenvolver, fazendo com que o desenvolvedor utilize um conjunto delas no desenvolvimento de um sistema. Dessa forma, acaba-se atribuindo uma carga elevada de conhecimento, em diversas ferramentas, aos desenvolvedores de software e analistas (KUCHARMA, 2004) (NAKAGAWA, 2006) (PREVITALI, 2007).

Outro problema emergente neste contexto é a mudança de requisitos de sistemas, pois a mutação com eles ocorrida ao longo do tempo, seja por novas necessidades dos usuários ou por novas tendências tecnológicas a serem incorporadas no desenvolvimento e/ou na execução de software, é consideravelmente significativa (FERNANDES & LISBÔA, 2001) (FORMAN & FORMAN, 2004) (GIARRATANO & RILEY, 1998) (HEINEMAN, 1999). Diante disso, existe um interesse em associar ao software alguma “capacidade de raciocínio”, de modo que ele

possa ser capaz de coletar as informações do meio ao qual está inserido, processá-las e tomar alguma decisão para que sua execução não seja interrompida (GIARRATANO & RILEY, 1998) (WATSON, 1997) (WATSON, 2009).

1.2. Formulação do Problema da Pesquisa

Desde os primórdios da computação, a Engenharia de Software vem enfrentando problemas quanto à existência de métodos e ferramentas que auxiliem os engenheiros de software desde a fase de obtenção de requisitos até a implementação do sistema (NAKAGAWA, 2006) (PRESSMAN, 2006). Atualmente é grande o número de ferramentas de apoio direcionadas à solução destes problemas, no entanto, tem-se observado que poucas atendem plenamente as necessidades dos desenvolvedores, devido às mutações existentes nas tendências de desenvolvimento de sistemas (PRESSMAN, 2006) (SOMMERVILLE, 2008) (PFLEEGER, 2004). Quando atendem, existe o problema relacionado ao custo elevado que inviabiliza sua aquisição por empresas de pequeno e médio porte. Diante do cenário apresentado podem ser citadas *suites* de desenvolvimento como Eclipse (ECLIPSE, 2009), *Netbeans* (NETBEANS, 2009), IBM-*Rational* (IBM-RATIONAL, 2009), entre outras. As duas primeiras são consideradas as mais populares IDEs (*Integrated Development Environment*) na comunidade por serem *freeware* e *Open Source*. A última é uma *suite* completa, porém proprietária.

Além do contexto apresentado, outros fatores são de grande relevância neste trabalho, tais como: a necessidade de muitas aplicações serem direcionadas a *Web* ou distribuídas numa rede (local ou não) e a constante mudança nas necessidades dos clientes dessas aplicações. Este último sugere que as aplicações possam ser modificadas em tempo de execução, sendo assim, este trabalho tem como objeto de estudo os problemas emergentes da reconfiguração de sistemas (MAES, 1987), tanto no que se refere ao código de execução (WEISS, 2001) (BEDER, 2001) quanto ao armazenamento de dados (EGE, 1999a) (EGE, 1999b).

A reconfiguração de sistemas quando realizada de maneira local exige alguns cuidados de seus gerenciadores (MAES, 1987), no entanto, em aplicações distribuídas (CHEN, 2002), estes cuidados aumentam consideravelmente, pois vários objetos podem estar sendo acessados por diferentes clientes ao mesmo tempo. Assim, antes de realizar qualquer modificação, seu estado deve ser salvo para que as execuções anteriores sejam preservadas.

Quanto à conformidade que deve existir entre o código da aplicação e o armazenamento de dados, nota-se a equivalência de operações, pois quando uma característica é modificada, incorporada ou retirada de um objeto, a mesma deve ser realizada na fonte de armazenamento, no banco de dados.

Os aspectos apresentados estão direcionados ao desenvolvimento de aplicações distribuídas reconfiguráveis. Além destes, uma consideração técnica quanto à implementação é factível neste momento: o conhecimento prévio que um desenvolvedor deve possuir para implementar um sistema distribuído. Tecnologias como RMI (*Remote Method Invocation*) (SUN-RMI, 2009) e implementações CORBA (*Common Object Request Broker Architecture*) (OMG, 2009), têm propósitos semelhantes, recursos de distribuição distintos e conhecimento técnico diferenciado. Dessa forma, pode-se dizer que existe uma divergência quanto aos recursos de implementação e conhecimento de implementação, sendo as aplicações com recursos da especificação CORBA mais complexas, porém com mais recursos de distribuição e gerenciamento de serviços.

Após realizar a contextualização sobre reconfiguração e seus aspectos relevantes para o gerenciamento das adaptações em diferentes ambientes (locais, distribuído e *Web*), julga-se necessário apresentar a maneira como é realizado o desenvolvimento deste tipo de aplicação utilizando a metodologia (Capítulo 4) e o ambiente de execução (Capítulo 5). Antes de detalhar o processo de desenvolvimento, vale ressaltar que a metodologia, a ferramenta e o ambiente de execução, desenvolvidos neste trabalho, atuam em paralelo às ferramentas Eclipse e *Netbeans*.

A ferramenta proposta neste trabalho atua de maneira complementar, implementando os mecanismos necessários para oferecer suporte à reconfiguração.

Inicialmente o engenheiro de software inicia o desenho dos modelos de lógica e de dados na IDE de sua escolha (Eclipse ou *Netbeans*). Em seguida, o projeto será importado para a ferramenta desenvolvida neste trabalho. A partir dessa etapa, informações referentes à classificação dos artefatos desenvolvidos são geradas por ela e inseridas nos repositórios de informações, conforme previsto no ambiente de execução reconfigurável (Capítulo 5).

Dessa forma, quando novos artefatos forem sendo desenvolvidos, maiores são as chances de reutilização, pois mais ampla é a base de artefatos. Essa reutilização pode ocorrer de duas formas: (1) total: quando os artefatos existentes (execução ou "adormecidos") atendem plenamente os requisitos de pesquisa; ou (2) parcial: quando pequenas adaptações são realizadas para adequar o artefato encontrado ao solicitado.

Sobre a reutilização total, pode-se dizer que a ferramenta tem a capacidade de localizar o melhor artefato em execução e encaminhá-lo para execução e, ainda, de "resgatar" o artefato no repositório de código, compilá-lo e inseri-lo no ambiente de execução. A reutilização parcial ocorre quando mecanismos de herança são utilizados para inserir novas características aos artefatos existentes, criando um novo artefato, que é inserido no ambiente de execução.

1.3. Objetivos

Este trabalho tem por objetivo propor uma Metodologia para o Desenvolvimento de Software Reconfigurável (MDSR) para sistemas *desktop* (locais), *Web* e distribuídos (Chamada de Métodos Remotos e Serviços *Web*). Além disso, esta metodologia prevê a criação de um Ambiente de Execução Reconfigurável (AER) composto por vários domínios de desenvolvimento. Cada um deles possui um conjunto de repositórios de código para reutilização e reconfiguração/adaptação dos artefatos (classes, componentes e serviços) produzidos ao longo

do tempo.

Outro objetivo deste trabalho é a implementação de uma ferramenta, pois a metodologia proposta prevê que um conjunto de passos que devem ser realizados de maneira automática, tanto na fase de desenvolvimento quanto na recuperação de informações para reconfiguração/adaptação dos artefatos de software.

A implementação desta ferramenta, para gerenciar a MDSR e o AER, não está relacionada ao desenvolvimento de um software tradicional, sua finalidade é atuar de maneira cooperativas às IDEs: Eclipse e *Netbeans*. Sua finalidade está direcionada a um conjunto de subsistemas (métodos) capazes de gerenciar: (1) a reconfiguração/adaptação dos artefatos existentes nos repositórios e no ambiente de execução; e (2) a geração das regras de classificação dos artefatos de software. Além disso, oferece como benefício a automatização e padronização (geração e recuperação) das informações, minimizando a participação humana na manipulação das informações geradas ao longo do processo de desenvolvimento.

Ainda sobre a implementação da ferramenta, destaca-se a implementação de um subsistema, responsável por gerenciar a reconfiguração dos artefatos em tempo de execução. Este subsistema possui "objetos tutores", cuja finalidade é interceptar todas as solicitações vindas das aplicações clientes e verificar se elas podem ser atendidas. Em caso negativo, modificações são inseridas no artefato de sua responsabilidade que, posteriormente, é compilado e inserido no ambiente de execução. Os "objetos tutores" também podem direcionar a solicitação de um cliente para um outro objeto em execução, tornando a execução mais dinâmica.

Finalmente, esta metodologia oferece aos desenvolvedores facilidades no desenvolvimento de aplicações distribuídas utilizando a tecnologia RMI, associada aos aspectos dinâmicos de manipulação de objetos da especificação CORBA. Essas facilidades tendem a minimizar o esforço e tempo dos desenvolvedores com este tipo de tecnologia de distribuição.

1.4. Relevância da Pesquisa

Este trabalho de pesquisa é relevante no contexto do estudo realizado no processo de desenvolvimento de aplicações distribuídas e/ou para *Web*, pois diretrizes para condução deste processo foram criadas. Além disso, destaca-se o apoio de ferramentas que conduzem esse processo e favorecem o aumento na produtividade de desenvolvimento.

Destaca-se também, a padronização no desenvolvimento de aplicações conforme premissas da Engenharia de Software, pois os sistemas passam a ser: documentados, modelados e armazenados em repositórios de código. Além disso, destaca-se a padronização empregada no desenvolvimento, em que abordagens como Desenvolvimento Orientado a Objetos, Desenvolvimento Orientado a Componentes e Desenvolvimento Orientado a Aspectos, são adotadas pelas organizações e, consequentemente, observa-se mudanças na cultura de desenvolvimento e melhorias no gerenciamento de projetos.

O desenvolvimento dos tutores na reconfiguração de software mostra-se como um novo recurso no desenvolvimento de software, pois são responsáveis por interpretar as mudanças ocorridas no ambiente de execução e adaptar o software em "tempo de execução". Para que isso ocorra, o apoio de ferramentas na etapa de desenvolvimento é fundamental para a padronização de código (classes e dados), pois facilitam a identificação das características e funcionalidades que se pretende alterar/adaptar.

As contribuições aqui mencionadas referem-se aos novos sistemas a serem desenvolvidos e àqueles antigos que devem ser integrados; também é realizada a padronização na interface de comunicação, em relação ao meio externo via *web services*. Dessa forma, a proposta de padronização no desenvolvimento de aplicações distribuídas e integração das existentes, mostra-se ser de grande interesse tanto pela comunidade científica quanto pelas corporações públicas e privadas.

1.5. Organização do Trabalho

Este capítulo apresentou o contexto deste trabalho, a formulação do problema a ser abordado e os objetivos pretendidos.

O Capítulo 2, Assuntos Relacionados, apresenta os assuntos referentes a este trabalho. Pela presença de diversas áreas de pesquisa, Engenharia de Software, Reconfiguração de Software, Inteligência Computacional e Programação Distribuída, os tópicos foram organizados em seções, com o intuito de apresentá-los de maneira organizada e objetiva, além de destacar quais são as contribuições que cada uma delas oferece ao trabalho.

O Capítulo 3, Reconfiguração de Software, aborda de maneira objetiva os assuntos relacionados à reconfiguração de software, apresentando conceitos, *frameworks* e técnicas de reconfiguração de software.

O Capítulo 4, Metodologia para Desenvolvimento de Software Reconfigurável, apresenta um conjunto de diretrizes para desenvolvimento de software em um ambiente reconfigurável, assim como a recuperação de objetos, componentes e/ou serviços existentes para a construção de uma nova aplicação ou na modificação da já existente, em execução.

O Capítulo 5, Ambiente de Execução Reconfigurável, mostra o ambiente de execução das aplicações distribuídas, a Ferramenta *ReflectTools*® e seus subsistemas. O ambiente caracteriza como os desenvolvedores e ferramentas estão organizados. A ferramenta e seus subsistemas são responsáveis por suportar as diretrizes apresentadas na metodologia e viabilizar sua aplicação para que artefatos reutilizáveis e reconfiguráveis sejam desenvolvidos.

O Capítulo 6, Avaliação dos Resultados, apresenta os resultados obtidos com o desenvolvimento deste trabalho. Estes estão embasados em estudos de casos, não formais, conforme recomendado nas normas da estatística, como forma de aplicação da metodologia e avaliação do produto de software obtido. Dessa forma, com os resultados obtidos pode-se dimensionar a automatização das tarefas que se pretende implementar.

Finalmente, no Capítulo 7, Conclusão e Trabalhos Futuros, são apresentadas as conclusões obtidas com a execução deste trabalho, ressaltando as principais contribuições e perspectivas para trabalhos futuros.

CAPÍTULO 2

Assuntos Relacionados

2.1. CONSIDERAÇÕES INICIAIS

A Reconfiguração de Software é uma atividade que requer algumas premissas para sua aplicação ao desenvolvimento e alguns cuidados relevantes quanto à sua execução. Essas premissas estão diretamente relacionadas aos recursos tecnológicos exigidos, aliados à carga de conhecimento, necessária pelos engenheiros de software e desenvolvedores, nas fases de análise e implementação de um sistema com essa característica (reconfiguração). Os cuidados podem ser resumidos na capacidade de reconfigurar para atender a uma nova solicitação, sem afetar as execuções vigentes.

Outro fator relevante ao contexto de reconfiguração de software é a padronização nas fases de análise e desenvolvimento. Além disso, destaca-se a automatização na recompilação e implantação do novo sistema (parcial ou integral), pois diminui a participação humana e, consequentemente, a geração de inconsistência nos sistemas na fase de adaptação.

Para viabilizar a execução deste trabalho, proposta de uma Metodologia para Desenvolvimento de Software Reconfigurável (MDSR) apoiada por um ambiente de execução,

houve a necessidade de se estudar um conjunto de assuntos, encontrados na literatura especializada. Pela natureza deste trabalho ser multidisciplinar, envolvendo várias áreas da computação, esses assuntos foram organizados em três grandes áreas de atuação: Engenharia de Software, Inteligência Computacional e Tecnologias de Distribuição.

Este capítulo está organizado da seguinte maneira: na Seção 2.2 são apresentados os assuntos relacionados à Engenharia de Software: técnicas de modelagem, padrões de projeto, técnicas de programação, recurso de integração de sistemas e processo de software; na Seção 2.3 são apresentados os assuntos referentes à Inteligência Computacional: agentes de software e o *Framework* DROOLS; na Seção 2.4 são apresentados os assuntos referentes às Tecnologias de Distribuição: chamadas de métodos remotos, tecnologias e desenvolvimento orientado a serviços; e finalmente, na Seção 2.5 são apresentadas as considerações finais.

2.2. Engenharia de Software

Nesta seção são apresentadas as técnicas e métodos de modelagem de sistemas, as ferramentas CASE (*Computer Aided Software Engineering*) de apoio ao desenvolvimento de software, que oferecem suporte às técnicas e métodos estudados e, finalmente, as soluções de software por padrões de projeto. Esta seção está organizada da seguinte maneira: na Seção 2.2.1 são apresentados os conceitos e técnicas da linguagem UML (*Unified Modeling Language*) e do método *Catalysis*; e na Seção 2.2.2 são apresentados os padrões de projeto que oferecem suporte para a adaptação de software.

2.2.1. Técnicas de Modelagem

A linguagem UML (BOOCH et al., 2005) (UML, 2009) é um conjunto de técnicas que apoiam a elaboração de projetos de sistemas orientados a objetos. Essas técnicas podem ser aplicadas na visualização, na especificação, na construção e na documentação de sistemas. Essa

linguagem é uma evolução e agregação de vários conceitos de outros métodos existentes na literatura, tornando-se a integração de várias técnicas.

A linguagem UML é regulamentada pelo OMG (*Object Management Group*) (OMG, 2009) (MEDEIROS, 2004), que trabalha juntamente com várias empresas com objetivo de fornecer técnicas e recursos para a modelagem de vários tipos de sistemas. Atualmente, encontra-se na versão 2.1.2 e está organizada em um conjunto de pacotes, chamados de *profile*, conforme seu contexto de atuação. A Figura 1 mostra a arquitetura desses pacotes, em que se pode destacar a reutilização de classes e recursos (UML, 2009).

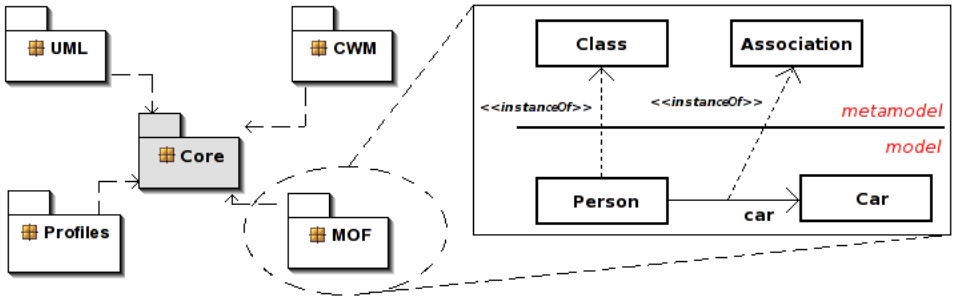

Figura 1: Arquitetura dos pacotes UML (UML, 2009)

O pacote *Core* representa o coração da arquitetura, fornecendo os recursos base, como tipos primitivos e construtores, para os demais pacotes: MOF, *Meta-Object Facility*; CWM, *Common Warehouse MetaModel*; *Profiles* e UML. Nessa arquitetura pode-se destacar o pacote MOF para realização deste trabalho, devido às seguintes características:

- disponibilização de recursos e representação para sistemas reconfiguráveis, organizados em nível-base e meta-nível; e
- notação para interceptação de mensagens e fluxo de informação entre os objetos.

Com base nos conceitos, na organização da UML e na necessidade de um processo de desenvolvimento de software automatizado, a metodologia proposta foi elaborada com base em

duas ferramentas: (1)*Netbeans* (NETBEANS, 2009) com *plugin* UML (NETBEANS-UML, 2009) e (2)Eclipse (ECLIPSE, 2009) com *plugin Omondo* (OMONDO, 2009). Ambas as ferramentas permitem a geração de código JAVA, a partir dos modelos elaborados na fase de análise.

Dentre as metodologias de desenvolvimento de software existentes na literatura, destaca-se a de Desenvolvimento de Software Baseado em Componentes (DSBC) (PRESSMAN, 2005) (PRESSMAN, 2006), que tem sua fundamentação teórica apoiada pelos conceitos da orientação a objetos e pela especificação UML (WERNER & BRAGA, 2000). Neste trabalho, essa metodologia foi utilizada, pois, se tratando de aplicações distribuídas reconfiguráveis, interfaces de acesso devem ser disponibilizadas para os clientes e, quando um objeto/componente servidor for invocado e não possuir a característica desejada, sua adaptação torna-se mais amigável. Outro fator relevante dessa metodologia é a reutilização que pode ser realizada através da adaptação e derivação para a criação de novos componentes (KIM, 2001). No entanto, alguns conceitos sobre componentes são fundamentais para o desenvolvimento deste trabalho.

Segundo Aoyama (1998), o DSBC é um paradigma para o desenvolvimento de software, cuja meta é compor componentes de software "*plug & play*" em um *framework*, que corresponde a conjunto de componentes que podem ser utilizados em vários sistemas, mudando radicalmente a forma de desenvolvimento de software. O conceito apresentado pelo autor pode ser utilizado na MDSR, proposta neste trabalho.

Jacobson et al (1997) comentam que o grande interesse do DSBC é o reúso, para isso, é necessário que os componentes sejam independentes da aplicação, para a qual foram projetados, e que tenham um grau de qualidade elevado. Sendo assim, um componente deve ter interfaces bem projetadas, que facilitam sua integração; deve ser bem testado, que garante sua eficiência; e deve ser bem documentado, que garante sua utilização.

Dentro de um ambiente distribuído, um sistema de componentes é composto por um repositório, que tem por objetivo armazenar, classificar, recuperar componentes, de forma a aumentar a reutilização de software e organização dos sistemas (PRESSMAN, 2006). Para este tipo de ambiente, o método *Catalysis* (CATALYSIS, 2009), que utiliza os conceitos da orientação a objetos apoiados pela UML e da computação distribuída, é utilizado na modelagem dos componentes distribuídos. Outros métodos de modelagem também podem ser utilizados.

O DSBC requer, por parte do desenvolvedor, o conhecimento e aplicação de técnicas e padrões de projeto, que auxiliam na identificação e criação de componentes de software. Na Seção 2.2.2, são apresentados os padrões de projeto utilizados neste trabalho.

2.2.2. Padrões de Projeto

Esta seção apresenta um conjunto de padrões de projeto que podem ser empregados na adaptação de componentes (GAMMA et al, 1994) (STELTING & MAASSEN, 2002) (KUCHARMA, 2004). Esses são empregados em diferentes etapas do desenvolvimento deste trabalho, seja de maneira isolada, ou associados às técnicas de adaptação, que são apresentadas no Capítulo 3.

Para Gamma et al (1994) e Stelting & Maassen (2002), os padrões de projeto podem ser organizados em três categorias: criação, estrutura e comportamento. Dentre elas, existe uma segmentação pelo escopo de atuação, Classe e Objeto, que cada padrão pode atuar conforme mostra a Figura 2.

Ainda sobre os padrões de projetos vale ressaltar que os mesmos são genéricos e foram propostos pelos autores Gamma et al (1994) e, neste trabalho, são utilizadas as implementações dos mesmos na linguagem JAVA (STELTING & MAASSEN, 2002).

A MDSR proposta neste trabalho está centralizada em aplicações desenvolvidas com a linguagem JAVA (SUN-JAVA, 2009), em diferentes abordagens, orientada a objetos, a

componentes, a aspectos e/ou a serviços. Nestas abordagens, pode-se destacar como principal técnica utilizada na reconfiguração a API (*Application Programming Interface*) *Reflect* (JAVA-REFLECT, 2009), que permite a descoberta de características e funcionalidades em tempo de execução e/ou o carregamento de outras classes/objetos no sistema.

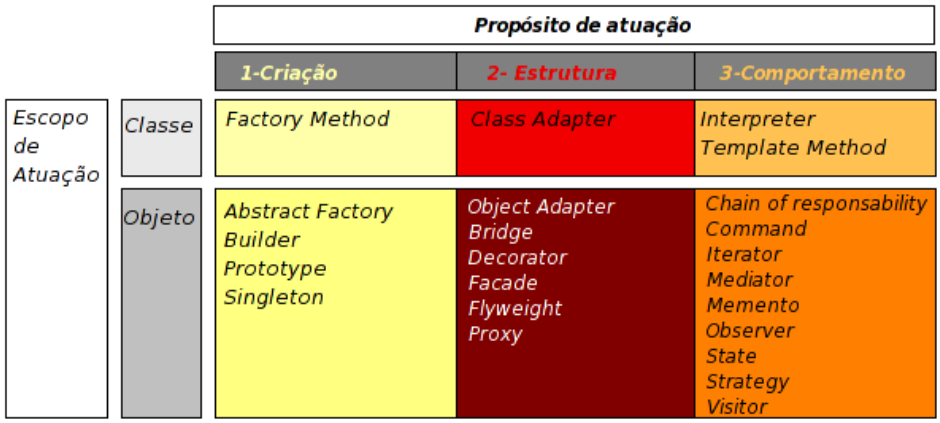

Figura 2: Organização dos padrões de projeto (GAMMA et al, 1994) (STELTING & MAASSEN, 2002)

Segundo Stelting & Maassen (2002) e Kucharma (2004), a API *Reflect* é fundamental para a adaptação de objetos, porém padrões de projeto como *Factory Method*, *Facade* e *Proxy*, podem oferecer soluções personalizadas às limitações desta API. Os padrões citados foram utilizados neste trabalho na adaptação das funcionalidades de software.

O padrão *Factory Method*, também é conhecido como *Virtual Construtor*, é utilizado quando se desconhece os tipos de objetos que serão utilizados por uma aplicação. Normalmente, é possível que se tenha alguma referência sobre a sua funcionalidade, mas não como ela foi implementada. A flexibilidade deste padrão pode ser resumida na extração de uma funcionalidade de um construtor, que é inserida como método fabricado em outra classe (KUCHARMA, 2004).

O padrão *Facade* tem por objetivo fornecer um conjunto de interfaces de um subsistema ou de um sistema complexo, atuando como uma camada de comunicação entre o novo sistema e o sistema que se deseja estabelecer comunicação. Para construir a camada de comunicação é

necessário, em alguns casos, que as funcionalidades desses sistemas sejam particionadas em blocos menores (classe/objetos) com interfaces de comunicação bem definidas, para que sua complexidade seja reduzida. No entanto, cabe ressaltar que esse particionamento deve ser realizado de maneira documentada, pois os sistemas clientes devem conhecer a maneira e parâmetros de comunicação com os novos blocos gerados, caso contrário, a comunicação pode se tornar complexa, devido ao número elevado de classes que foi gerado no particionamento (KUCHARMA, 2004).

O padrão *Proxy* fornece uma representação de um objeto para outro por razões de acesso, velocidade e segurança. Para que um Objeto *Proxy* (OP) represente um Objeto Real (OR) é necessário que ele implemente a mesma interface do OR. O OP necessita manter uma referência direta com o OR para que possa utilizar seus métodos. Dessa forma, o cliente faz a solicitação ao OP, que encaminha para o OR, onde será realizada a execução (STELTING & MAASSEN, 2002) (KUCHARMA, 2004). Um OP pode implementar os métodos e características do OR, além de outras funcionalidades como distribuição, segurança e outros. A Figura 3 mostra o diagrama de classe desse padrão.

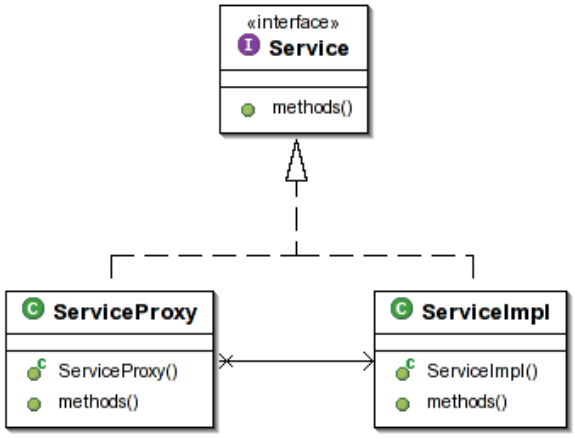

Figura 3: Diagrama de classe do padrão Proxy (STELTING & MAASSEN, 2002)

As funcionalidades dessas classes são:

- `Service`: é a interface que será implementada pelo OR e OP;

- `ServiceProxy`: implementa a interface `Service` e encaminha as chamadas para `ServiceImpl` (OR) quando necessário; e
- `ServiceImpl`: representa o OR e será representado no ambiente pelo OP.

Stelting & Maassen (2002) e Kucharma (2004) estabelecem que este padrão deve ser utilizado quando for necessário criar mais referências para um objeto, tais como:

- *remote proxy*: quando for necessário referenciar um objeto em uma JVM (JAVA *Virtual Machine*) distinta da aplicação como se fosse local;
- *virtual proxy*: atua como um local responsável por manter referência para objetos ao invés de criá-los, que representa um processo de alto custo; e
- *protection proxy*: determina o acesso para objetos.

Esses padrões, associados aos recursos de reconfiguração que são apresentados no Capítulo 3, fornecem mecanismos de modificação de objetos/componentes, por meio de invocação de objetos, interface de comunicação em subsistemas e comunicação remota de objetos de maneira segura e transparente.

2.2.3. Técnicas de Programação

Nesta seção são apresentadas as técnicas de programação utilizadas neste trabalho para o desenvolvimento do ambiente de reconfiguração e das aplicações que são nele inseridas. Por motivos de organização deste trabalho e de relevância de cada uma delas, a Seção 2.2.3.1 apresenta a Programação Orientada a Aspectos (POA) e a Seção 2.2.3.2 apresenta a Programação Orientada a Componentes (POC). Em cada uma delas são destacadas suas principais características e contribuições para a realização deste trabalho.

2.2.3.1. Programação Orientada a Aspectos

Um paradigma de destaque nos últimos anos é o Orientado a Objetos (OO), pois trouxe inúmeras inovações quanto à organização da lógica de código. Além disso, pode-se dizer que seus benefícios permitem maior velocidade de desenvolvimento, devido à alta reusabilidade e modularização de código (KICZALES, 2001). No entanto, problemas de implementação surgiram e se tornaram insolúveis pela OO, que necessita de outros recursos para que sua finalidade seja cumprida (GRADERICKI, & LESIECKI, 2003). Neste contexto, tem o surgimento da Programação Orientada a Aspectos (POA), que combina os conceitos da orientação a objetos associados a novos recursos de programação, entrelaçamento de código entre requisitos funcionais e não-funcionais, capazes de atuar de maneira cooperativa (KICZALES et al, 1997) (REZENDE & SILVA, 2005).

A POA introduz alguns conceitos na maneira de modelar e desenvolver um sistema. Dentre eles, pode-se citar a separação de interesses, que mais especificamente, divide os requisitos de um sistema em dois tipos (GRADERICKI, & LESIECKI, 2003) (KICZALES et al., 2001): **funcionais**: que representa a lógica do sistema, são os requisitos que fazem parte apenas da abstração na definição de classes e objetos; **não-funcionais**: representam algo externo ao sistema, ou seja, atuam de maneira externa à definição do escopo de classe e objeto. Esse tipo de requisito pode estar associado à conexão de banco de dados, distribuição da aplicação, segurança, etc. O conceito existente no Desenvolvimento de Software Orientado a Aspecto é mais amplo do que o conceito apresentado. No entanto, neste trabalho, será utilizado apenas o conceito de separação de interesses, como mecanismo de distinguir os requisitos funcionais e não-funcionais.

Baseado na ideia apresentada, pode-se ressaltar que o engenheiro de software encontra-se centrado em modelar apenas as funcionalidades essenciais do sistema e, qualquer item adicional, pode ser tratado de maneira complementar como um aspecto. A Figura 4 mostra a

construção dessa abordagem de desenvolvimento (GRADERICKI, & LESIECKI, 2003).

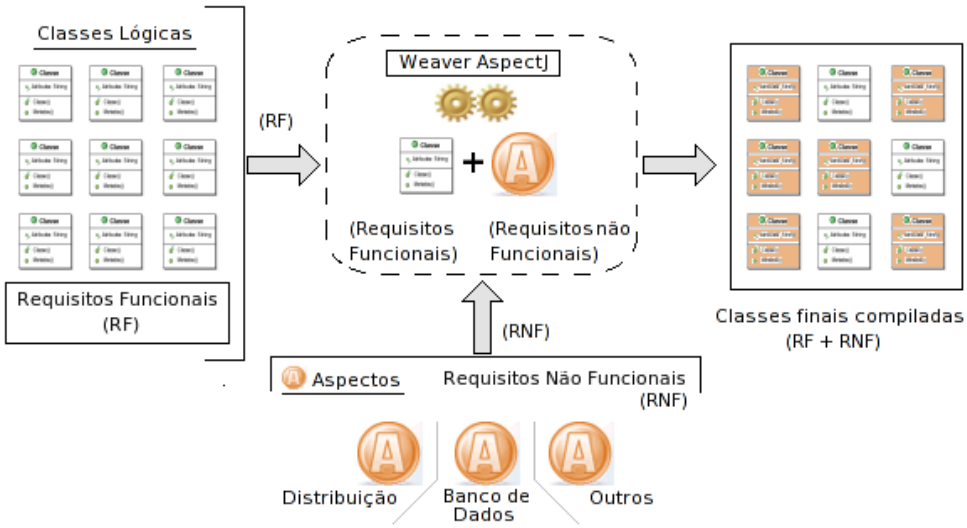

Figura 4: Weaver do AspectJ adaptado de (GRADERICKI, & LESIECKI, 2003)

As especificações de Requisitos Funcionais (RF) e Não-Funcionais (RNF) podem ser desenvolvidas de maneiras independentes, paralelas e/ou separadas. Em seguida, o compilador do aspecto, por exemplo o *AspetJ* (ASPECTJ, 2009) ou *JBoosAOP* (JBOSS-AOP, 2009), faz o *weavering* (entrelaçamento) dos RF e RNF constituindo um novo objeto, que é o objeto lógico dotado de características de infraestrutura (JBOSS-AOP, 2009) (ASPECTJ, 2009).

Outra vantagem que a POA trouxe é a capacidade de adaptação e dinamismo na utilização e carregamento de funcionalidades, isto é, um objeto, em tempo de execução, pode ter seu comportamento ou estrutura modificada por meio de um aspecto. Este conceito é utilizado neste trabalho como capacidade de adaptação de objetos/componentes, além dos conceitos já apresentados e utilizados no desenvolvimento das aplicações distribuídas.

2.2.3.2. Programação Orientada a Componentes

Baseado nos conceitos das Técnicas de Modelagem, apresentadas na Seção 2.2.1, e, principalmente, sobre aqueles relacionados ao DSBC, esta seção apresenta algumas tecnologias que permitem a implementação de componentes, que estão diretamente associadas ao

desenvolvimento deste trabalho, como *JavaBeans* e *Enterprise JavaBeans* (EJB) (SUN-JEE, 2009).

O modelo de componentes *JavaBeans* permite alta reusabilidade no desenvolvimento de software, pois os desenvolvedores estão direcionados ao uso de componentes com interfaces de acesso bem definidas, para que funcionalidades robustas sejam criadas. Dessa forma, quando esses componentes são utilizados, detalhes de implementação não são apresentados para o desenvolvedor, importando apenas os serviços que ele oferece (SUN-JAVABEANS, 2009).

O EJB é um modelo de componentes centralizado no conceito de arquitetura de software em camadas, com responsabilidade bem definidas: Lógica de Apresentação, Lógica de Negócios e Lógica de Acesso a Dados. A implementação desse modelo de componentes requer, basicamente, um ambiente de execução preparado com os seguintes itens (BOND, 2003):

- **servidor de EJB**: que é responsável por hospedar os componentes no ambiente distribuído;
- **serviço de nomes**: utilizado para atribuir nomes aos componentes para posterior recuperação; e
- **serviço de diretórios**: responsável por armazenar os objetos e facilitar a localização, atua como páginas amarelas.

Esses itens são tratados pela JNDI (*JAVA Name and Directory Interface*), que é parte integrante da Especificação dos EJBs (AHMED & UMRYSH, 2002). Sua principal finalidade é a utilização e recuperação de objetos, sem conhecer sua localização. Eles podem estar localizados num ambiente distribuído ou em máquinas locais (BOND, 2003).

2.2.4. Recurso de Integração

Segundo Anderson, R. et al. (2001), a XML (*eXtensible Markup Language*) é uma

linguagem que combina a flexibilidade e capacidade do SGML (*Standard Generalized Markup Language*), com a ampla aceitação do HTML (*HyperText Markup Language*). Trata-se de uma linguagem universal que pode ser escrita e lida por diversas linguagens de programação, tais como: JAVA, C++, *Object Pascal*, entre outras. Dessa forma, pode-se dizer que vários sistemas, desenvolvidos com linguagens de programação distintas, podem ser integrados através de troca de mensagens de arquivos XML (XML, 2009).

Os principais benefícios da utilização de XML podem ser resumidos em: (i) Adaptação, que representa a capacidade de adaptação para representar vários tipos de informações; (ii)Simplicidade, devido a sua semelhança aos arquivos HTML, a representação de informações por TAGs também é utilizada nesta linguagem; e (iii)Portabilidade, que representa a utilização de XML em diversos tipos de sistemas e pela integração com diversas linguagens de programação.

A manipulação de XML pela linguagem JAVA pode ser realizada por várias APIs, porém duas delas, DOM (*Document Object Model*) e SAX (*Simple API for XML*), se destacam pelos recursos básicos que oferecem as demais APIs e as aplicações (ANDERSON, R. et al., 2001).

Um breve comparativo entre as APIs, DOM e SAX, mostra que a DOM oferece maior agilidade de manipulação de informação, pois os dados são tratados em memória. No entanto, alguns cuidados devem ser tomados para que o arquivo não tenha um tamanho elevado, o que pode acarretar problemas de carga de memória. A API SAX permite que arquivos de tamanho elevado possam ser analisados, porém seu processamento é mais lento devido ao acesso que é realizado em disco (ANDERSON, R. et al., 2001) (XML, 2009).

Neste trabalho, a linguagem XML e suas APIs de integração com JAVA são utilizadas: (i)na extração de informação do servidor de banco de dados em arquivos; (ii)na leitura desses arquivos para manipular as informações nele contidas; (iii)pela leitura e escrita de arquivos pelos subsistemas; e (iv)pela apresentação das informações geradas em interfaces *Web*. Além

disso, ela é utilizada juntamente com a API *Reflect*, que é apresentada no Capítulo 3, para documentar as características de objetos e banco de dados, pois mecanismos de reflexão "descobrem" comportamento e características de objetos e entidades, os quais são organizados em arquivos XML, para que facilitem a extração de informação no processo de adaptação.

2.2.5. A Linguagem JAVA

A linguagem de programação JAVA (SUN-JAVA, 2009) é considerada uma das mais poderosas e mais abrangentes das existentes atualmente no mercado (DEITEL & DEITEL, 2005). Ela pode ser caracterizada como uma linguagem orientada a objetos, robusta, interoperável, portável, multiplataforma e utilizada em vários ambientes de implementação computacional. Essas características levaram a Sun a investir num *slogan*: "compile uma vez e execute em qualquer lugar".

A linguagem JAVA é abrangente quanto aos segmentos de desenvolvimento que pode atuar, com destaque, em sistemas *desktop*, voltados para *Web*, distribuídos, móveis, orientados a componentes e orientados a serviços. Além desses, outras APIs oferecem recursos fundamentais de implementação, tais como XML, *JavaCard*, *JavaMail*, *JavaMedia*, *Web Services*, entre outras (SUN-JAVA, 2009) (SUN-JEE, 2009).

Por motivos de organização deste trabalho, não serão apresentados detalhes de todos recursos da linguagem JAVA, devido a extensão de cada um deles. No entanto, são destacados recursos e APIs que contribuem para o desenvolvimento deste trabalho, conforme já mencionados neste capítulo. Outros recursos e APIs são tratados especificamente nos próximos capítulos pela organização lógica dos assuntos deste trabalho e pela relevância de contribuição.

2.2.6. Processo de Software

Segundo Sommerville (2008), as empresas que obtiveram sucesso no desenvolvimento

de software estão utilizando processos de software definidos. Em linhas gerais, um processo de software pode ser definido como um conjunto de etapas que devem ser seguidas para o desenvolvimento de um produto (SOMMERVILLE, 2008).

Sommerville (2008) define: "*O processo é um conjunto de atividades e resultados associados que produzem um produto de software*". O autor comenta sobre a existência de quatro atividades fundamentais que são comuns a todos processos de software: especificação, desenvolvimento, validação e evolução do software. No entanto, essas atividades ocorrem de diferentes maneiras e são detalhadas em diferentes níveis e prazos. Dessa forma, pode-se notar que diferentes organizações podem utilizar diferentes processos e produzirem o mesmo resultado.

Para Pfeeger (2004), o processo de software é um conjunto de passos com uma ordem de execução estabelecida. Esses passos envolvem uma série de atividades, restrições e recursos para alcançar a saída desejada. A autora afirma que, geralmente, um processo de software envolve um conjunto de ferramentas e técnicas que automatizam a execução de suas tarefas/etapas.

Segundo Pressman (2005, 2006), processo de software, é um conjunto de passos constituindo um guia, que ajuda na execução destes com garantia de qualidade e com prazos estabelecidos. Sua importância está centralizada no controle de execução das atividades estabelecidas e, dessa forma, facilitando a execução do trabalho em equipe.

Diante dos conceitos apresentados sobre processo de software é necessário comentar sobre os modelos de processos de software existentes. Na literatura podem ser encontrados diversos modelos: modelo em cascata, desenvolvimento evolucionário, transformação formal, montagem de um sistema a partir de componentes reutilizáveis, entre outros (SOMMERVILLE, 2008) (PFLEEGER, 2004) (PRESSMAN, 2005) (PRESSMAN, 2006). No entanto, detalhes sobre as fases de desenvolvimento não são abordadas neste trabalho por

motivos de escopo. Os autores mencionam também a importância de ferramentas CASE na execução das atividades para automatização das etapas de documentação, modelagem, desenvolvimento e teste.

Neste trabalho, os conceitos de processo de software e a automatização de suas atividades são utilizados na elaboração da metodologia de desenvolvimento de software reconfigurável e na implementação do ambiente de execução.

2.3. Inteligência Computacional

Esta seção apresenta os assuntos encontrados na literatura especializada sobre recursos de Inteligência Computacional (IC) que oferecem suporte para realização deste trabalho. Esses recursos são utilizados para classificar, localizar e reconhecer um objeto num sistema *desktop*, local ou distribuído.

Os agentes inteligentes e agentes móveis são utilizados como mecanismos de coleta de conhecimento e tomada de decisão no ambiente distribuído (BIGUS & BIGUS, 2001) (RUSSEL & NORVIG, 2003) (GIARRATANO & RILEY, 1998). Todos esses assuntos atuam, neste trabalho, de maneira isolada e/ou cooperativa, sendo esta última uma característica importante na IC, pois caracteriza os chamados Sistemas Híbridos (KHOSLA & DILLON, 1997).

Além desses conceitos e recursos de IC, são apresentadas as suas respectivas implementações em JAVA (Watson 1997) (Watson, 2009), pois todo trabalho desenvolvido nesta Tese está implementado nesta linguagem. Sendo assim, destacam-se na avaliação realizada, os conceitos e recursos do *framework* DROOLS (Proctor et al., 2009), juntamente com o apoio computacional na fase de desenvolvimento de software.

Na Seção 2.3.1 são apresentados os Agentes Móveis e Inteligentes, na Seção 2.3.2 os Sistemas Híbridos Inteligentes e, na Seção 2.3.3, a implementação dos Recursos de IC e o

framework DROOLS.

2.3.1. Agentes de Software

A definição de agente de software não está limitada apenas na especificação de um conceito e/ou de um segmento de aplicação, devido a sua aplicabilidade e poder de atuação. No entanto, pela extensão que está entorno do assunto e pela objetividade de aplicação neste trabalho, apenas as definições, conceitos e características pertinentes ao mesmo serão apresentados como meio de mostrar sua contribuição.

Ferber & Gasser (1991) *apud* (REZENDE, 2003) apresentam uma definição abrangente sobre agentes, dentre elas pode-se destacar que

> um agente é uma entidade real ou virtual, capaz de agir num ambiente, de se comunicar com outros agentes, que é movido por um conjunto de inclinações (sejam objetivos individuais a atingir ou uma função de satisfação a otimizar); que possui recursos próprios; que é capaz de perceber seu ambiente (de modo limitado); que dispõe (eventualmente) de uma representação parcial deste ambiente; que possui competência e oferece serviços; que pode eventualmente se reproduzir e cujo comportamento tende a atender seus objetivos utilizando as competências e os recursos que dispõe e levando em conta os resultados de suas funções de percepção e comunicação, bem como suas representações internas.

Em Bernardes (1999), é encontrada uma definição quanto à tipologia dos agentes, sua aplicabilidade e organização. Para isso, ela é disposta em cinco dimensões:

1. **mobilidade**: representa sua habilidade de locomoção por uma rede de computadores. Os agentes podem ser classificados como estáticos ou móveis;
2. **funcionamento deliberativo ou reativo**: representa a maneira como o agente executa suas tarefas. No primeiro, existe um modelo de raciocínio simbólico, responsável pelo planejamento e negociação de tarefas com outros agentes. No segundo, o agente atua dentro do ambiente baseado num mecanismo de estímulo e

resposta, não possuindo "raciocínio";

3. **propriedades ideais**: representa um conjunto de propriedades e funcionalidades que um agente deveria possuir, destacando-se três: autonomia, aprendizagem e cooperação;
4. **papel do agente na aplicação**: representa a finalidade que ele deve cumprir; e
5. **híbrido**: representa a combinação de uma ou mais das dimensões anteriores em um único agente.

Para Bernardes (1999) e Tavares (2002), os agentes podem ser organizados em sete tipos: colaboradores, de interface, móveis, de informação, reativos, inteligentes e híbridos. Detalhes aprofundados sobre definição, características e aplicação de cada um deles não serão apresentados neste trabalho por motivos de objetividade, no entanto, eles podem ser obtidos nos trabalhos dos autores citados. Neste trabalho são utilizados os conceitos e recursos dos agentes móveis e inteligentes.

Os agentes móveis, pela própria definição de seu nome, tem como destaque a mobilidade em uma rede de computadores. Seu modelo de execução difere do modelo tradicional "cliente-servidor", pois a aplicação encaminha sua(s) necessidade(s) aos agentes móveis, encarregados de localizar outro (objeto/componente) servidor, capaz de atendê-la. Esse processo tem como funcionamento uma estrutura em camadas: aplicação, agentes e subsistema de mensagens. O agente pode, por meio de seus atributos e funcionalidades, executar uma solicitação no servidor e devolvê-la ao requisitante ou habilitar (invocar) um servidor para isso (BERNARDES, 1999) (FILHO, 2001) (LANCE & OSHIMA ,1998).

Segundo Lucena (2003) *apud* (RUSSEL, 2005), a introdução de inteligência nos agentes está associada ao mecanismo de "aprendizado" e ao "raciocínio" que pode ser realizado diante de estímulos do ambiente de execução que está inserido. O autor apresenta também um

contexto comparativo entre a inteligência humana e a pretendida com os agentes, definindo um conjunto de características que resultam em comportamentos esperados pelos agentes. São elas: autonomia, capacidade de comunicação, cooperação, raciocínio, planejamento e adaptabilidade. Estas oferecem suporte ao processo de conhecimento simbólico que se assemelha ao ser humano (BIGUS & BIGUS, 2001).

Lucena (2003) também aborda o processo de reutilização de agentes, pois num ambiente computacional baseado em agentes, muitos deles foram definidos e outros devem ser criados ou modificados, por não atender às novas necessidades. Sendo assim, algumas funcionalidades de agentes já desenvolvidos podem ser inseridas (reutilizadas) na construção de novos.

Outro modelo de destaque no desenvolvimento de sistemas utilizando agentes é o *Aglet* (LANCE & OSHIMA, 1998). Nele um agente é um objeto com uma *Thread* de controle próprio, orientado a evento e com comunicação por passagem de mensagem. Seu modelo de comunicação é composto por quatro abstrações:

- ***aglet***: objeto JAVA que possui mobilidade numa rede de computadores;
- ***proxy***: mecanismo contra acesso direto ao *aglet*, sendo transparente quanto sua localização para o cliente;
- **contexto**: representa o local de trabalho do *aglet*, ou seja, seu local de execução; e
- **identificador**: atributo único e imutável de um *aglet* no seu ciclo de vida.

Os conceitos e recursos dos agentes móveis e inteligentes contribuem, respectivamente, com a navegação pelo ambiente distribuído, coletando e transportando informações de/para objetos e, com a capacidade de aprendizado, cooperação e manipulação dessas, para que um processo decisório seja por eles aprendido e executado. Destaca-se também a reutilização de software nos sistemas baseados em agentes, em que características podem ser incorporadas aos agentes já existentes ou aos novos que serão desenvolvidos.

2.3.2. Sistemas Híbridos Inteligentes

Os recursos de IC (Agentes, Algoritmos Genéticos, Redes Neurais, Programação Declarativa) atuam na solução de problemas e contexto específicos, no entanto, eles podem atuar de maneira cooperativa e colaborativa, constituindo os chamados Sistemas Híbridos (SHs). Esse processo pode ser necessário quando, após análise circunstanciada e conclusiva, os recursos atuando de maneira isolada não oferecerem uma solução viável ou insuficiente, e a combinação dos mesmos tendem a resolvê-la ou otimizá-la.

Rezende (2003) comenta que a combinação de recursos de IC não leva necessariamente à melhoria de desempenho de um sistema, podendo até resultar em soluções adversas às esperadas. No entanto, para alcançar os objetivos esperados, os SHs devem ser projetados e sua utilização deve ser plenamente justificada. Os relatos encontrados na literatura mostram que as pesquisas estão direcionadas para a combinação de recursos baseados em dados submetidos a Redes Neurais Artificiais (RNA) (HAYKIN, 2001), com recursos que utilizam conhecimento (Lógica *Fuzzy*) (PIRES, 2004) e (REZENDE, 2003).

O desenvolvimento de SHs e outros tipos de sistemas em IC possuem fundamentação em metodologia na Engenharia de Software, pois requerem os mesmos requisitos que qualquer outro tipo de sistema, como qualidade e alto desempenho. Dessa forma, atividades como Planejamento, Projeto, Documentação, Tempo, Ciclo de vida, Manutenibilidade, entre outras características, que são herdadas da Engenharia de Software e empregadas no desenvolvimento desses sistemas (GIARRATANO & RILEY, 1998).

Uma das características mais importantes no desenvolvimento de software é o ciclo de vida, pois determina o início, as fases que devem ser executadas e quando finalizar. Na literatura são encontrados vários modelos de desenvolvimento de software (Cascata, Incremental e Espiral) e cada um deles com características peculiares ao processo de desenvolvimento adotado (PRESSMAN, 2006).

Dentre os ciclos de vida existentes, nenhum deles atende plenamente a necessidade de SHs. Dessa forma, um ciclo de vida linear, com um conjunto de etapas específicas à sua natureza, é utilizado. Apesar do nome, este ciclo de vida atua de maneira cíclica, quando as atividades finais são atingidas. Suas atividades de destaque são: Planejamento das atividades, Definição do Conhecimento, Projeto do Conhecimento, Verificação do Conhecimento e Avaliação do Sistema (GIARRATANO & RILEY, 1998).

Ferber & Gasser (1991) *apud* (REZENDE, 2003) apresentam um esquema de classificação de SHs organizado em três classes, que levam em consideração fatores como funcionalidade, arquitetura de processamento e requerimento de comunicação. São elas:

- **substituição de função**: representa a implementação de uma técnica para executar a função de outra. Seu objetivo é otimizar ou superar alguma limitação das existentes;
- **intercomunicativo**: representa a modularização do SH em módulos independentes com objetivo de reduzir sua complexidade; e
- **polimórficos**: representa a adaptação de uma técnica para a realização de uma tarefa.

Apesar dos melhoramentos eminentes por meio da combinação entre recursos, existe algumas delas que podem ser consideradas clássicas pela literatura. Por exemplo, pode-se apresentar dois casos (REZENDE, 2003):

- extração de conhecimento de RNA: devido ao alto poder de processamento das RNA (HAYKIN, 2001) (REZENDE, 2003) (KHOSLA & DILLON, 1997) nos diversos tipos de aplicações e pela baixa interpretação dos resultados pelos usuários, métodos de extração de informações são associados a elas de maneira que atuem como caixas pretas na execução e exista um mecanismo de explicar suas decisões; e
- otimização evolutiva de sistemas *fuzzy* (sf): refere-se à incapacidade dos sf's

(CASTRO, 2004) & (REZENDE, 2003) em aprender a partir de dados. Algoritmos Genéticos (REZENDE, 2003) são utilizados para automatizar esse processo, constituindo os Sistemas *Fuzzy* Genéticos.

Os conceitos apresentados nesta seção mostram que os recursos de IC podem atuar de maneira complementar, otimizando ou permitindo que uma solução factível seja obtida. Destaca-se a combinação clássica desses para a construção de sistemas otimizados.

2.3.3. Recursos de Implementação para IC

Para viabilizar a implementação deste trabalho, alguns recursos IC foram estudados para avaliar sua incorporação e utilização total e/ou parcial na MDSR desenvolvida. Os recursos avaliados atuam diretamente na documentação e recuperação das informações por sistemas de consultas manuais e automáticas.

Dentre os trabalhos encontrados na literatura e avaliados, destaca-se um conjunto de classes apresentados por Watson (1997, 2009) (JESS, 2009), organizado em um *framework*. Essa implementação oferece recursos de apoio ao desenvolvimento de Agentes, Redes Neurais, Algoritmos Genéticos e Sistemas *Fuzzy*. O interesse por esse *framework* está diretamente relacionada à facilidade de utilização das funcionalidades por ele disponibilizadas e por estar desenvolvido na linguagem JAVA, que é utilizada em todo trabalho.

Outro projeto de destaque encontrado na literatura é o *framework* DROOLS (PROCTOR et al., 2009). Pela importância e recursos deste *framework* seus detalhes são apresentados na Seção 2.3.3.1.

2.3.3.1. O *Framework* DROOLS

Segundo Proctor et al. (2009), um dos objetivos fundamentais da IA é “fazer com que o

computador pense como os seres humanos" e, sendo assim, recursos como Redes Neurais, Algoritmos Genéticos e, principalmente, a representação do conhecimento, serão utilizados para a tomada de decisões em um Sistema Inteligente. Para isso, um sistema deve traduzir o conhecimento cotidiano em representações interpretáveis à sua linguagem de comunicação, surgindo assim as chamadas regras de produção.

Um sistema de regra de produção centraliza a representação do conhecimento de maneira concisa, não ambígua e declarativa. O processador de um sistema baseado em regras é chamado de motor de inferência, que é capaz de dimensionar o número de regras e fatos. Uma regra é formada por duas partes básicas, sendo a primeira chamada de condicional ou representação do conhecimento, e a segunda chamada de ação (PROCTOR et al. 2009).

Proctor et al. (2009) afirma que para executar um conjunto de regras é necessário um padrão de reconhecimento que atua juntamente com o motor de inferência. Além disso, existe um conjunto de algoritmos utilizados como padrão de reconhecimento pelo motor de inferência. São eles: *Linear, Rete, Treat* e *Leaps*. Esses algoritmos não são detalhados aqui por motivos de escopo, maiores detalhes podem ser obtidos na documentação do autor.

Para Browne (2009a, 2009b) e Proctor et al. (2009), a programação declarativa permite os seguintes benefícios:

- **programação declarativa**: representa a capacidade de resolver um problema e ainda explicar como ele foi resolvido;
- **separação da lógica e dos dados**: amplamente utilizada nos sistemas orientados a objetos e aplicável nos sistemas orientados a aspectos pelo princípio da separação de interesses;
- **velocidade e escalabilidade**: fornece mecanismos eficientes para os padrões de reconhecimento;
- **centralização do conhecimento**: representa a criação de um repositório de

informação utilizado como fonte de documentação do sistema;

- **integração de ferramentas**: apoio computacional para o desenvolvimento de sistemas baseados em regras, por exemplo, Eclipse; e
- **facilidade de explicação**: representa a facilidade de apresentação das informações e explicação dos resultados.

Além desses benefícios mencionados, os autores comentam sobre a dificuldade de desenvolvimento de sistemas utilizando este tipo de recurso computacional, pois causa dificuldade de entendimento. Esta situação não está necessariamente associada à complexidade de desenvolvimento, mas sim ao domínio de informações entorno do problema e, consequentemente, à capacidade de absorção e processamento destas.

Este *framework* oferece, para a realização deste trabalho, um conjunto de recursos e conceitos já mencionados e relacionados à programação declarativa. Além disso, destaca-se a utilização da IDE Eclipse e o *plugin* de apoio, como ferramenta de desenvolvimento, pois esta pode ser utilizada em várias etapas de um projeto de software. Ainda sobre o *framework* DROOLS, vale ressaltar que a versão 4.0 foi utilizada quando esta Tese foi elaborada.

2.4. Tecnologias de Distribuição

Nesta seção são apresentados os assuntos encontrados na literatura especializada sobre Tecnologias de Programação Distribuída que oferecem suporte para realização deste trabalho. A RMI (*Remote Method Invocation*) (DEITEL & DEITEL, 2001) (DEITEL et al, 2001) (SUN, 2009a), permite que objetos sejam invocados e acessados em um ambiente distribuído de maneira estática ou dinâmica, em relação ao conhecimento sobre o local e o nome em que o objeto solicitado está hospedado. A RMI-IIOP (*Remote Method Invocation over Inter-Orb Protocol*) (SUN-RMI-IIOP, 2009) fornece maior flexibilidade de desenvolvimento e

comunicação, pois um objeto remoto pode ser invocado de maneira dinâmica e comunicar-se com outro objeto escrito em linguagem diferente. Finalmente é apresentada a Especificação SOA (*Service Oriented Architecture*) (ERL, 2004) (ERL, 2005), que permite uma comunicação transparente entre os serviços disponibilizados pelos servidores para os consumidores (clientes).

Na Seção 2.4.1 são apresentados os conceitos e recursos da RMI; na Seção 2.4.2 são apresentados os conceitos e inovações do RMI-IIOP em relação ao RMI; finalmente, na Seção 2.4.3 conceitos de tecnologias orientadas a serviços são apresentadas.

2.4.1. Invocação Remota de Métodos

Nesta seção será apresentada uma visão geral da tecnologia RMI para implementação da computação distribuída. Esta permite que objetos JAVA (SUN-JAVA, 2009), que estejam sendo executados no mesmo computador ou em um ambiente distribuído, se comuniquem entre si por meio de chamadas remotas aos métodos que implementam (DEITEL & DEITEL, 2001). Essas chamadas são simples e transparentes aos objetos clientes, pois representam um processo de comunicação entre objetos de maneira local.

A RMI possui algumas limitações, dentre elas, destaca-se a impossibilidade de comunicação entre objetos escritos em diferentes linguagens de programação, deixando para o desenvolvedor a agregação de um recurso como IDL (*Interface Definition Language*) para que isso ocorra (DEITEL & DEITEL, 2001) (DEITEL et al, 2001) (SUN-JAVA-IDL, 2009).

Para desenvolver um sistema com objetos remotos baseados na tecnologia RMI, recomenda-se a seguinte sequência de passos (DEITEL & DEITEL, 2001): (1) definição dos objetos do sistema em função dos requisitos funcionais; (2) dos objetos definidos no passo 1, apontar quais serão acessados remotamente; (3) implementação das características de distribuição; (4) compilação dos objetos pelo compilador (`rmic`), para geração de *stubs* e *skeletons*, objetos responsáveis pela comunicação no cliente e servidor, respectivamente; (5)

distribuição dos *stubs* para os objetos clientes; (6) no lado cliente - realizar um *lookup* (busca no registrador de objetos remoto) para acessar os métodos remotos; (7)no lado servidor - aguardar a chamada por métodos.

A implementação de objetos remotos com RMI remete um paradoxo dentro do ambiente de execução, pois objetos clientes e servidores estão sendo executados em máquinas diferentes e, consequentemente, por JVMs distintas. Essa questão pode ser esclarecida entendendo o processo de criação de *stubs* e *skeletons*, citados no item quatro do parágrafo anterior desta seção. Eles são criados quando um objeto remoto é compilado pelo `rmic`, para viabilizar a comunicação entre clientes e servidores. No entanto, a partir da versão 6.0 da linguagem JAVA, a geração de objetos *skeletons* é opcional (DEITEL & DEITEL, 2001) (SUN-RMI, 2009).

A RMI oferece a transferência de objetos complexos, tipos de dados complexos, através de um mecanismo de serialização, cujo propósito principal é viabilizar a comunicação entre os objetos remotos, codificando-os como um fluxo de *bytes*. Para isso, duas classes são fundamentais: a `ObjectOutputStream`, que converte qualquer objeto `Serializable` em fluxo de *bytes* para transmissão na rede; e a `ObjectInputStream`, que reconstitui os *bytes* em objetos (DEITEL & DEITEL, 2001) (DEITEL et al, 2001) (SUN-RMI, 2009).

A seguir, na Seção 2.4.1.1, são apresentados conceitos de engenharia de software no desenvolvimento de sistemas distribuídos.

2.4.1.1. A Engenharia de Software no Desenvolvimento Distribuído

Do ponto de vista do engenheiro de software, o desenvolvimento de um sistema distribuído, utilizando RMI, deve ser detalhado em um conjunto de etapas que se estende desde à especificação de requisitos até os testes de implantação do sistema (PRESSMAN, 2006). A Figura 5 mostra o processo de especificação e implementação de um sistema distribuído associado a recursos de engenharia de software, elaborados neste trabalho.

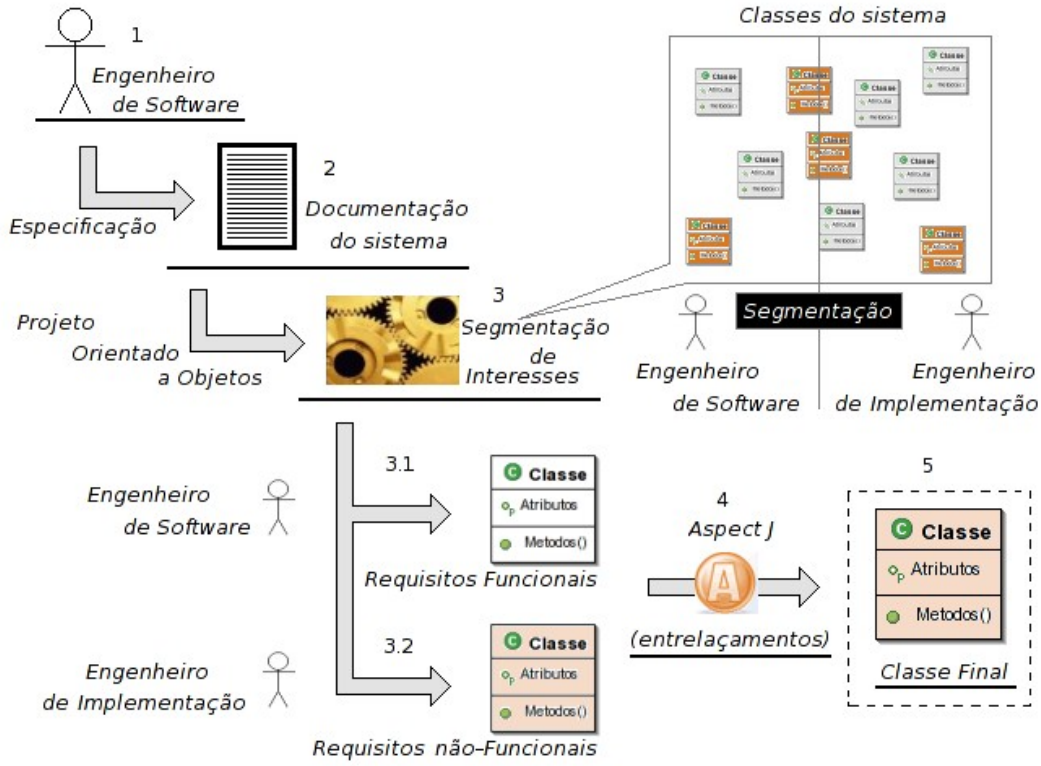

Figura 5: Processo de identificação e fusão de funcionalidades

Esses conceitos são utilizados neste trabalho para facilitar a adaptação dos objetos no ambiente de execução. No entanto, existem alguns conceitos que devem ser apresentados para que objetos hospedados em locais e JVMs distintos sejam adaptados, sem interferir no funcionamento do sistema. Dessa forma, conceitos e funcionamento do carregamento dinâmico de objetos pelos *Classloaders* são necessários e, por questão de organização lógica deste trabalho e da importância deste assunto, eles são apresentados na Seção 2.4.1.2.

2.4.1.2. Carregamento dinâmico de objetos e *Classloader*

Em um sistema distribuído, a atividade de *deploy* (implantação) de suas aplicações não é uma tarefa complicada, pois elas estão subdivididas em máquinas inseridas numa rede local. No entanto, nas aplicações voltadas para *Web*, essa tarefa torna-se complexa devido à localização de servidores não ser limitada a uma rede local. As aplicações desenvolvidas com tecnologia

RMI possuem um recurso chamado *Dynamic Classloader* (DC), que tem por objetivo facilitar a atividade de *deploy* das aplicações (GROSSO, 2001).

Mesmo com um DC, alguns cuidados devem ser tomados para não gerar inconsistência de código na aplicação, tais como: configuração das máquinas servidoras; ajuste das classes *stubs* no CLASSPATH da aplicação; reiniciar o registrador de nomes (`rmiregistry`); e instalar e configurar a aplicação cliente.

Os dois últimos passos apresentam uma interferência na execução das aplicações e fazem referência a outro componente de um sistema desenvolvido com tecnologia RMI, o *Classload*. Em um sistema JAVA (SUN-JAVA, 2009), cada arquivo fonte (`.java`) é compilado e um *bytecode*, arquivo (`.class`), é gerado. Quando o sistema é executado, a JVM cria uma referência para cada objeto (`.class`), assegurando que estão em uso no sistema, dessa forma, cada objeto passa a cumprir seu papel no sistema e, quando não estiver sendo utilizado, é descarregado dinamicamente pelo *Garbage Collection* (GROSSO, 2001) (SUN-JAVA, 2009).

Para permitir que vários objetos sejam carregados dinamicamente, cada JVM possui um conjunto de *Classloaders*, que são encarregados de referenciar um objeto e inseri-lo no sistema em execução. O funcionamento cooperativo entre JVM e *Classloader* pode ser resumido da seguinte maneira (GROSSO, 2001):

- a JVM aciona seu *Classloader*, que invoca o método `loadClass()` para "suspender" o objeto na memória;
- os *Classloaders* trabalham na invocação de objetos com o princípio de herança, ou seja, um *Classload* pai cria um *Classload* filho para carregar um novo objeto; e
- quando uma invocação de um novo objeto é solicitada, os *Classloaders* fazem um processo de busca interna entre eles, para verificar se o objeto está presente na memória, em caso positivo, uma referência a ele será criada ou um novo objeto será invocado.

Os conceitos e recursos da tecnologia RMI apresentados nesta seção são utilizados, neste trabalho, para mostrar o processo de desenvolvimento de sistemas distribuídos, através de conceitos primitivos da tecnologia e com associação de recursos complementares da Engenharia de Software. Com isso, o processo de adaptação e reutilização do sistema torna-se mais natural, conforme já mencionado nesta seção. A seguir, são apresentados os conceitos sobre invocação remota sobre IIOP.

2.4.2. Invocação Remota de Métodos sobre IIOP

Esta seção apresenta uma visão geral da tecnologia RMI-IIOP (SUN-RMI-IIOP, 2009), para implementação da computação distribuída, utilizando conceitos da especificação CORBA (OMG, 2009) (SIEGEL, 2000). Esta tecnologia é uma extensão da RMI e ambas encontram-se inseridas na plataforma JAVA *Standard Edition* (JAVA SE) (SUN-JAVA, 2009). No entanto, a tecnologia RMI-IIOP foi desenvolvida pelo consórcio realizado entre a Sun (SUN, 2009) e a IBM (IBM, 2009), que implementou o padrão de especificação da OMG (OMG, 2009), sobre interoperabilidade entre os sistemas desenvolvidos em linguagens distintas, tais como C++, COBOL, *Smaltalk*, entre outras (JAVAWORLD, 2009).

A RMI sobre IIOP possui os mesmos recursos de sua precursora, RMI, associados aos melhores recursos da especificação CORBA (OMG, 2009). Ao implementar o protocolo IIOP, esta tecnologia garante que objetos podem se comunicar com outros, desenvolvidos em diferentes linguagens. Essa característica é viabilizada pela IDL, que faz o mapeamento das interfaces remotas dos objetos e permite que elas sejam implementadas com diferentes linguagens de programação. O compilador IDL (SUN-JAVA-IDL, 2009) é um componente nativo da especificação CORBA e encontra-se implementado também na plataforma JAVA SE para RMI-IIOP e outros recursos (JAVAWORLD, 2009).

Os detalhes da IDL não serão abordados nesta seção por motivos de escopo do trabalho,

no entanto, é necessário apresentar os passos essenciais para o projeto desse tipo de interface para o desenvolvimento de um sistema distribuído (SUN-RMI-IIOP, 2009):

1. definição das funcionalidades que devem ser disponibilizadas para os clientes através de um arquivo (`.idl`);
2. esse arquivo é compilado para que seja gerada a estrutura de distribuição (*stubs* e *skeletons*);
3. em seguida, o desenvolvedor implementa os métodos definidos na interface (*idl*) gerados no passo 2; e
4. finalmente, os objetos são inseridos no ambiente de execução.

A Figura 6 mostra a estrutura de uma aplicação desenvolvida com JAVA_IDL, executando sob um ORB (*Object Request Broker*) com protocolo IIOP (SUN-RMI-IIOP, 2009).

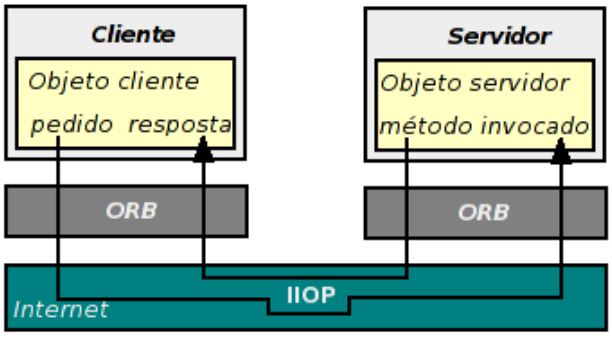

Figura 6: Estrutura de uma aplicação RMI-IIOP (JAVA-RMI-IIOP, 2009)

O objeto cliente envia um pedido ao ORB, que recebe e localiza um objeto servidor para executar o pedido enviado pelo cliente. O objeto servidor processa a solicitação e devolve ao ORB, para encaminhar o resultado ao cliente. Nesta estrutura, os objetos podem ou não ser implementados em linguagens distintas devido ao mapeamento IDL que foi realizado no desenvolvimento (SUN-RMI-IIOP, 2009).

Com a utilização de RMI-IIOP no desenvolvimento de aplicações distribuídas, os

objetos podem ser implementados em JAVA e compilados com o `rmic`, para que os arquivos necessários para comunicação com o ORB (OMG, 2009) sejam gerados e, a comunicação com objetos implementados em diferentes linguagens seja viabilizada.

Os conceitos da estrutura e do mecanismo de comunicação da RMI-IIOP são utilizados, neste trabalho, para realizar a integração entre os sistemas novos e os já existentes, pois o protocolo IIOP permite que objetos escritos em linguagens distintas se comuniquem de forma transparente. Destaca-se também o compilador IDL, que é componente fundamental para a RMI-IIOP, pois permite que objetos tenham suas interfaces remotas mapeadas para diferentes linguagens de programação. Finalmente, é notável a contribuição dos recursos da especificação CORBA, com recursos nativos, tais como objetos de conectores, protocolo IIOP, compilador IDL e interfaces de comunicação *stubs* e *skeletons*.

2.4.3. Desenvolvimento orientado a serviços

Com base nos conceitos de distribuição apresentados nas Seções 2.4.1 e 2.4.2 e, motivado pelas necessidades atuais de empresas e instituições governamentais, que demandam: por novos sistemas voltados para *Web*; por migrar os sistemas existentes (legado) para *Web*, caracterizando o processo de reengenharia; ou por integrar o legado com sistemas modernos através de *Web Services*. Dentre essas necessidades, a terceira caracteriza-se como um novo "ponto de referência" no desenvolvimento de software chamado de Orientação a Serviços. A seguir são apresentados os conceitos e recursos da especificação SOA.

Deitel et al. (2001) e Jini (2009) comentam sobre a existência de várias tecnologias para o Desenvolvimento de Software Orientado a Serviços (DSOS): JINI, *JavaSpace* e *Jiro*. No entanto, elas não são abordadas neste trabalho, por motivos de escopo e por restrições técnicas quanto ao atendimento à distribuição da aplicação ser restrita a redes locais ou a configuração específica de software para execução.

Em CBDI (2009) pode-se encontrar a seguinte definição sobre SOA

> Essencialmente, SOA é uma arquitetura de software que define a topologia das interfaces, implementação das interfaces e as chamadas das interfaces. SOA é um relacionamento de serviços e consumidores dos serviços, ambos módulos de software grandes o bastante para representar uma completa função do negócio.

Para Oasis-soa (2009) e Erl (2004, 2005), a reutilização de software desejada ao longo dos anos pode ser realizada através da integração de dois ou mais sistemas, pois, ambos, implementam funcionalidades distintas que podem atuar em diversos domínios. No entanto, algumas premissas devem ser implementadas para viabilizar essa necessidade, destacando-se a padronização das interfaces dos sistemas, para que ambos se comuniquem de forma transparente como se fosse um único, ou seja, que as partes a serem reutilizadas possam ser utilizadas como se fossem locais.

Esses conceitos deixam claro que SOA não é um produto que se instala ou configura para realizar a distribuição de uma aplicação. Trata-se de uma especificação para o desenvolvimento de aplicações distribuídas orientadas a serviços, podendo partes delas estar distribuídas em outras aplicações, *Web Services*, repositórios de aplicações, ou todo artefato de software que permita acesso de suas funcionalidades através de suas interfaces. Pode-se dizer que o desenvolvimento de software baseado nessa especificação permite:

- a integração de aplicações de maneira transparente, atuando nas aplicações chamadas legadas, desenvolvidas com tecnologias antigas e desprovidas de qualquer documentação, até os sistemas atuais, desenvolvidos com linguagens de programação modernas. No entanto, em ambos os casos, é necessária a construção de uma camada intermediária, encarregada de realizar a integração dos sistemas, que pode ser acessada via *Web Services* ou outro recurso de distribuição (ERL, 2004) (ERL, 2005); e
- a padronização das interfaces dos componentes de software, que representa uma

nova cultura de desenvolvimento para as empresas, pois a comunicação entre eles ocorrerá por meio das funcionalidades que foram disponibilizadas em suas interfaces, enfatizando o DSBC (D'SOUZA & WILLS, 1998).

Esses dois itens mostram a abrangência dessa especificação, pois atua na integração de sistemas legados e novos. No entanto, para viabilizar a implementação com essa especificação é necessário o conhecimento de alguns de seus componentes internos, como mostra a Figura 7.

Nesta arquitetura destaca-se a presença dos *Web Services*, que representam um software encarregado de oferecer a interoperabilidade entre as máquinas numa rede. Eles possuem uma interface, especificamente um WSDL (*Web Service Description Language*) para comunicar-se com os demais serviços. Outros sistemas utilizam mensagens SOAP (*Simple Object Access Protocol*) sob o protocolo HTTP (*HyperText Transfer Protocol*) com serialização via XML (W3C-SOA, 2009) (ERL, 2004).

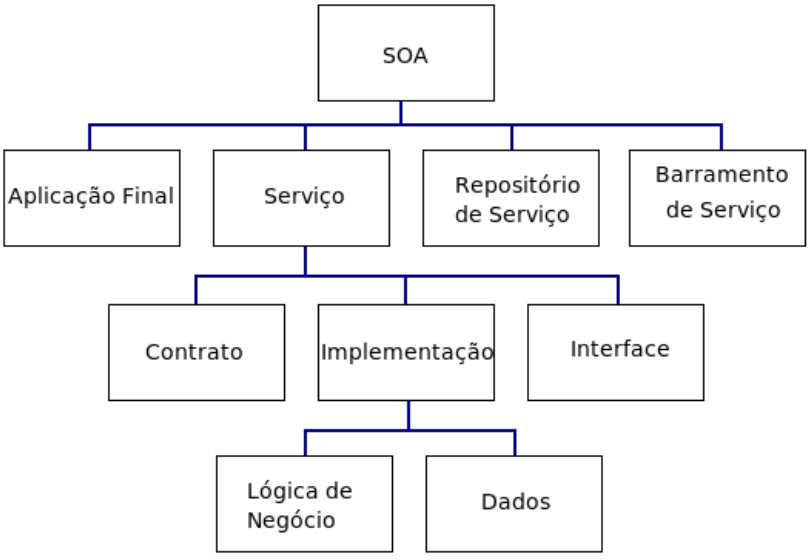

Figura 7: Arquitetura de serviços SOA (KRAFZIG et al, 2005)

Segundo W3C-soa (2009), *Web Service* é um conceito abstrato que pode ser implementado em um agente de software. Este é uma entidade de software encarregada de

enviar e receber mensagens, que se encontra padronizada através de uma WSDL. Cada mensagem possui uma representação semântica para a descrição de cada serviço disponibilizado. A Figura 8 mostra a solicitação de um serviço para um *Web Service*.

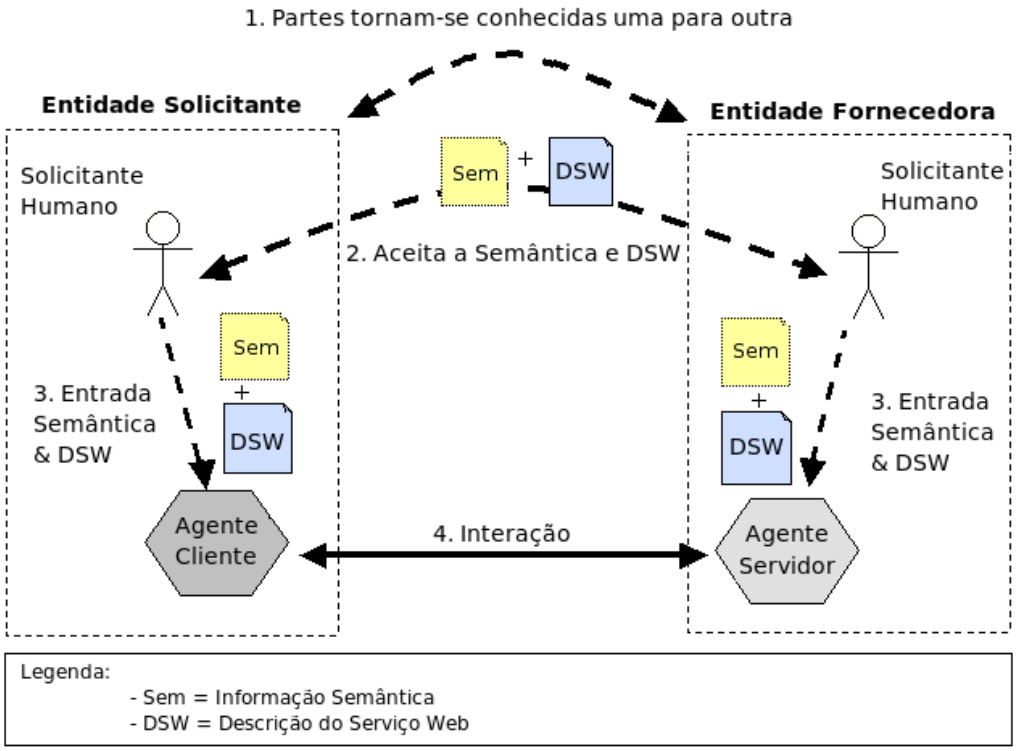

Figura 8: Solicitação de serviços na especificação SOA adaptado de (W3C-SOA, 2009)

Nesta arquitetura, o cliente, requisitante de serviço, envia ao servidor uma solicitação de serviço associada a uma descrição semântica e especificada através de uma DSW (Descrição de Serviço *Web*). O servidor recebe essas informações e localiza o serviço desejado, retornando também uma descrição semântica e uma referência para o serviço (ERL, 2004) (ERL, 2005) (DONG, 2008).

A especificação SOA fornece um conjunto de diretrizes para o desenvolvimento de um ambiente orientado a serviços, independentemente da escolha das ferramentas de implementação. Esses conceitos estão diretamente relacionados às tecnologias de distribuição apresentadas neste capítulo e são utilizados neste trabalho, devido aos seguintes fatores: às mutações constantes que o ambiente de execução pode sofrer; a flexibilidade de integração dos

sistemas existentes com os atuais; a portabilidade da aplicação entre vários ambientes; e a interoperabilidade com outros sistemas/componentes. A seguir, na Seção 2.5, são apresentadas as considerações finais deste capítulo.

2.5. Considerações Finais

Este capítulo apresentou um conjunto de assuntos que oferecem suporte para o desenvolvimento deste trabalho. As técnicas de modelagem, linguagem UML e método *Catalysis*, oferecem recursos para o engenheiro de software modelar sistemas conforme o paradigma escolhido, orientado a objetos e/ou orientado a componentes, no desenvolvimento da aplicação, além dos padrões de projeto como forma de solução para a adaptação de software.

Para oferecer suporte às técnicas de modelagem apresentadas neste capítulo, têm-se as técnicas de implementação, Programação Orientada Aspectos (POA) e Programação Orientada a Componentes (POC). A POA atua de forma cooperativa à programação Orientada a Objetos (OO), permitindo que o desenvolvimento das classes lógicas sejam puras (OO) e os requisitos relacionados à infraestrutura são tratados pela POA. Além disso, ela é utilizada como mecanismo de adaptação de classes adicionando características e/ou funcionalidades. A POC, juntamente com a Especificação EJB, disponibiliza um ambiente distribuído de componentes/objetos de maneira tradicional ou orientado a serviços, sendo este último o mais utilizado neste trabalho, devido à padronização oferecida na comunicação e acesso às informações, pois facilitam o processo de adaptação.

O recurso de integração XML é parte fundamental para o desenvolvimento deste trabalho, pela facilidade de manipulação de informações dos componentes/objetos. Além disso, no modelo orientado a serviços, arquivos XMLs são utilizados para o envio de mensagem aos servidores para execução de uma funcionalidade desejada, assim como o caminho inverso.

A linguagem JAVA é utilizada no desenvolvimento do ambiente distribuído

reconfigurável proposto neste trabalho (Capítulo 5) e na criação das aplicações nele inseridas. A principal justificativa para o uso dessa linguagem é a sua abrangência, pois ela pode atuar em vários segmentos de desenvolvimento, tais como: ambientes distribuídos, sistemas *Web*, aplicativos *desktop*, aplicações móveis, entre outros. Merece destaque também as APIs para XML, Reflexão e outras, pois ofereceram recursos ao processo de reconfiguração de objetos/componentes.

Os agentes móveis e inteligentes atuam na navegação pelo ambiente distribuído (Capítulo 5), coletando e transportando informações de/para objetos e, com a capacidade de aprendizado, cooperação e manipulação dessas para que um processo decisório seja por eles aprendido e executado.

Os Sistemas Híbridos representam a associação dos recursos apresentados, pois, em determinados problemas, atuam de maneira insuficiente ou não são aplicáveis, enquanto que de maneira cooperativa tendem a otimizar ou viabilizar as soluções desejadas.

Os recursos de implementação destacam-se pela atuação harmônica no desenvolvimento deste trabalho, devido à integração imediata com a linguagem JAVA e pelo suporte em ferramentas (IDE) de desenvolvimento, que aceleram esse processo.

A tecnologia RMI é a base de comunicação para outras tecnologias, tais como RMI-IIOP, JINI, *JavaSpace*, entre outras. Seus recursos e conceitos são utilizados de forma nativa ou modificados conforme necessidade de cada uma delas.

Dentre as tecnologias estudadas, destaca-se a RMI-IIOP, pois ela herda os conceitos básicos da RMI e agrega o protocolo IIOP da Especificação CORBA, como mecanismo de integração com outras aplicações desenvolvidas com linguagens de programação distintas e com outras aplicações nativas CORBA. Essa agregação de tecnologias aponta a divergência das tecnologias quanto ao atendimento das necessidades dos desenvolvedores, pois a RMI mostra-se limitada quanto aos recursos e a implementação com um ORB-CORBA requer conhecimento

elevado, quanto aos serviços da especificação, para que suas vantagens sejam exploradas/aplicadas no desenvolvimento das aplicações.

A especificação CORBA, apesar de não abordada em detalhes neste trabalho, mostra-se completa quanto aos recursos que descreve, no entanto, a execução destes está diretamente relacionada ao ORB utilizado no desenvolvimento. Segundo Omg (2009), existe um conjunto de ORBs, proprietários e de domínio público, para o desenvolvimento de aplicações distribuídas. No entanto, nem todos estão no mesmo estágio de desenvolvimento e apenas alguns encontram-se compatíveis com a versão atual (3.0), e outros em versões anteriores. Devido a esse motivo e pelo grau de detalhes da especificação, este trabalho faz uso apenas de serviços e conceitos de algumas interfaces de comunicação dinâmicas, pois a reconfiguração de um sistema prevê o monitoramento e alteração da aplicação em tempo de execução.

A Tabela 1 mostra um conjunto de características da RMI e da Especificação CORBA, apontando quais delas serão incorporadas à RMI para o desenvolvimento de uma aplicação distribuída reconfigurável. Dentre as características apresentadas, destaca-se (*) a incorporação à RMI, a descoberta dinâmica, a independência de linguagem e o protocolo neutro de comunicação, pois essas garantem maior reusabilidade da aplicação e facilidade de integração com outros sistemas.

A especificação SOA apoiada pelas tecnologias orientadas à serviços contribuem com uma nova abordagem de desenvolvimento de software e, consequentemente, com uma nova organização das empresas. Conceitos como repositórios de objetos (componentes), tecnologia de integração/distribuição e *Web Services* estão relacionados diretamente a ela. Porém, tecnologias como RMI, RMI-IIOP e CORBA também podem ser utilizadas dentro desta, pois fornecem infraestrutura de implementação.

Neste trabalho, tanto as premissas básicas das tecnologias (RMI) quanto as especificações de desenvolvimento de software distribuído, CORBA e SOA, são utilizadas. As

básicas são utilizadas para viabilizar a implementação das aplicações, pois nele são realizadas reconfigurações em vários níveis: banco de dados, aplicações *Web*, distribuídas utilizando RMI e utilizando serviços *Web*.

Tabela 1: Características CORBA e RMI

Características	**CORBA**	**RMI**	**Característica CORBA associada à RMI**
Chamada de retorno ao servidor	sim	sim	
Passagem por valor	não	sim	
Descoberta dinâmica	sim	não	X *
Invocação dinâmica	sim	sim	
Suporte a Transporte Múltiplo	sim	sim	
Nomeador de URL	não	sim	X
Firewall Proxy	não	sim	X
Independência de Linguagem	sim	não	X *
Protocolo “Neutro”	sim	não	X *
Nome de Persistência	sim	não	X
Segurança	sim	sim	
Transação	sim	sim	

O próximo capítulo apresenta os conceitos de reflexão computacional e as técnicas de reconfiguração de objetos/componentes utilizadas neste trabalho.

CAPÍTULO 3

Reconfiguração de Software

3.1. CONSIDERAÇÕES INICIAIS

Neste capítulo, são apresentados os assuntos encontrados na literatura especializada sobre Reconfiguração de Software, que oferecem suporte para o desenvolvimento deste trabalho. A seguir é apresentado um conjunto de conceitos e recursos que viabilizam a reconfiguração de um sistema em tempo de execução. Como destaque tem-se a Computação Reflexiva (MAES, 1987), que fornece um conjunto de conceitos para modificação do estado de objetos, capacitando-os a atender novos requisitos. São apresentados ainda os recursos da API JAVA *Reflect* (SUN-REFLECT, 2009) (FORMAN & FORMAN, 2004), que fornecem mecanismos para obtenção de características e comportamentos dos objetos, sem a necessidade da presença do código fonte. Por fim, o *framework* JAVASSIST, JAVA *Programming Assistant*, (JAVASSIST, 2009) também é apresentado neste capítulo como suporte e utilização de recursos de Reflexão já desenvolvidos.

Este capítulo, apresenta na Seção 3.2, a Computação Reflexiva, na Seção 3.3, os recursos da API JAVA *Reflect*, na Seção 3.4, os recursos e funcionalidades do *framework*

JAVASSIST e, finalmente, na Seção 3.5, as considerações finais.

3.2. Computação Reflexiva

Um dos grandes desafios enfrentados pelos engenheiros de software é incorporar aos sistemas atuais novos requisitos. Atualmente, essa necessidade pode ser realizada através de um processo de reengenharia no sistema que se deseja modificar, adicionando a nova funcionalidade. No entanto, esse processo deve ser realizado de maneira cautelosa, para que o engenheiro de software obtenha o conhecimento necessário sobre o sistema que está modificando, que pode demandar altos investimentos com tempo e custo.

Uma alternativa ao processo de adaptação é a Reflexão Computacional (RC), que pode ser definida como qualquer atividade executada por um sistema sobre si mesma, com intuito de obter informações sobre suas próprias atividades, tendo como objetivo melhorar seu desempenho, introduzir novas capacidades (características ou funcionalidades), ou ainda, resolver seus problemas, escolhendo o melhor procedimento (LISBÔA, 1997) (CAZZOLA, 1999) *apud* (FERNANDES, 2009).

A RC permite que um sistema monitore seu contexto atual e, a partir dessa análise, decida qual procedimento deverá ser executado. Essa característica mostra a flexibilidade de modificação de um sistema desenvolvido com reflexão computacional, chamados deste ponto em diante de Sistemas Reflexivos (SR).

Um SR é formado por duas camadas lógicas, como mostra a Figura 9. A camada inferior, nível-base, representa os dados e o código responsável pela execução dos requisitos para os quais a aplicação foi desenvolvida. A segunda, meta-nível, onde se encontram estruturadas ações que devem ser executadas dinamicamente em resposta às ações do nível-base do programa (FERNANDES, 2009).

Os requisitos funcionais e não-funcionais de um SR estão alocados no nível-base, e as

funcionalidades de dinamizar a aplicação por meio da reconfiguração, encontram-se organizadas no meta-nível. Dessa forma, para incorporar um novo requisito a uma aplicação em execução é necessário que uma análise da solicitação seja feita pelo meta-objeto, responsável por monitorar um objeto, para uma posterior adaptação da aplicação em tempo de execução.

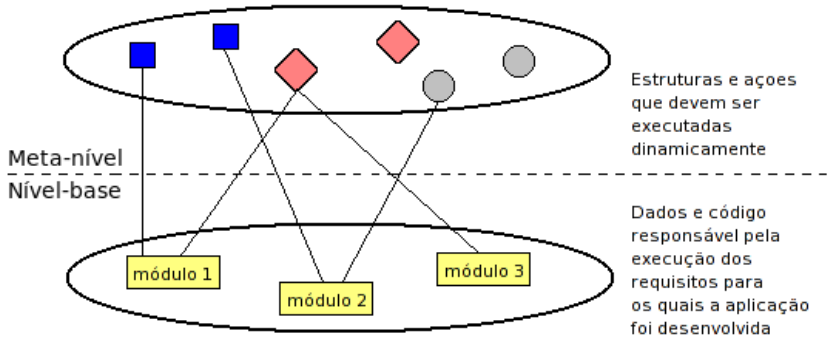

Figura 9: Estrutura de aplicações reflexivas (LISBÔA, 1997)

Segundo Fernandes (2009), Beder (2001) e Weiss (2001), para que um meta-objeto monitore um determinado objeto é necessário criar uma conexão e associação entre eles. A comunicação entre os níveis requer um Protocolo de Meta Objetos (PMO), que permite a troca de mensagens e a passagem da execução do programa. A Figura 10 mostra a associação entre os níveis base e meta ligados pelo PMO.

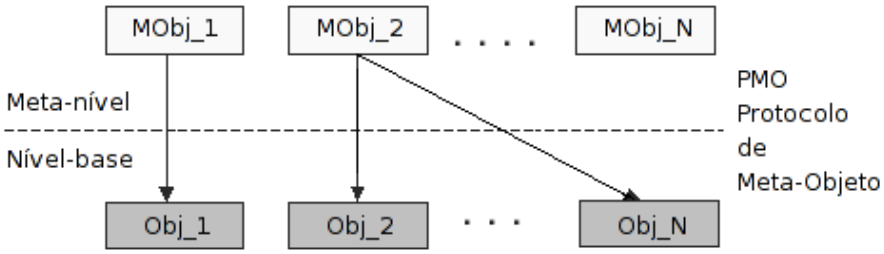

Figura 10: Protocolo de Meta-Objetos (FERNANDES, 2009)

No modelo de computação reflexiva, um meta-objeto do meta-nível é responsável por interceptar a solicitação de um objeto do nível-base e encaminhá-la para outro meta-objeto do meta-nível, que a entregará a um objeto do nível-base, para que a mesma seja atendida. O

caminho inverso também é considerado (FERNANDES, 2009). A Figura 11 mostra o Fluxo de informação entre objetos e meta-objetos.

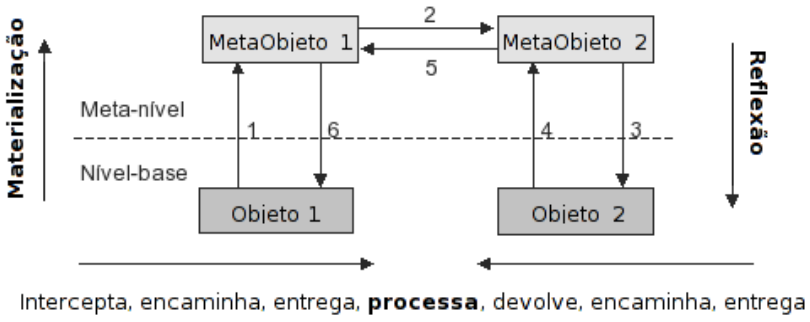

Figura 11: Fluxo de comunicação entre objetos e meta-objetos (FERNANDES, 2009)

Existem dois conceitos importantes presentes nesse processo de comunicação, quando as informações são enviadas do nível-base para o meta-nível e vice-versa, são eles (FERNANDES & LISBÔA, 2006):

- **materialização**, que representa o ato de disponibilizar o estado atual e a estrutura de um objeto, para que um meta-objeto possa atuar sobre ele;
- **reflexão**, que representa o ato de um objeto ter seu estado atual, estrutura ou comportamento, alterado por um meta-objeto.

Lisbôa (1997) *apud* (WEISS, 2001) relatam que a Reflexão Computacional permite alterar os aspectos estruturais e comportamentais de um sistema através dos seguintes mecanismos: **Introspecção:** encarregada das informações da estrutura do sistema (atributos e métodos); **Reflexão estrutural:** responsável por obter e alterar o estado do sistema para o qual ele foi projetado, inserindo/retirando atributos e métodos; e **Reflexão comportamental:** responsável por obter e realizar modificações no nível base (sistema).

Os conceitos da reflexão computacional e mecanismos de adaptação de objetos são utilizados neste trabalho para o processo de reconfiguração e descoberta de características dos mesmos. A seguir são apresentadas as técnicas de reconfiguração de software.

3.3. Técnicas de Reconfiguração

Baseados nos conceitos de RC, várias técnicas de adaptação de software foram encontradas na literatura especializada. Estas atuam em diferentes contextos da computação e trabalham em diferentes níveis de implementação, mas sempre associada a outros recursos de programação, tais como POA e POC, ambos apresentados no Capítulo 2.

Weiss (2001) utiliza a RC para a adaptação de componentes de software centralizada em duas maneiras: **estruturais**, que envolve a estrutura do componente como a alteração na interface ou no nome de uma operação, e **comportamentais**, que afetam as propriedades funcionais e não-funcionais. A adaptação proposta prevê a segmentação dos requisitos, conforme premissas fundamentais da POA e a implementação apoiada por Padrões de Projeto, *Coding Around*, *Open Source*, *Wrapping*, *Reflective Wrappers*, entre outros.

A Figura 12 mostra a adaptação de componentes, que inicialmente foram projetados através de requisitos e contextos distintos, sendo adaptados para atuar em novos contextos e compostos por outras funcionalidades (GOMAA & HUSSEIN, 2004).

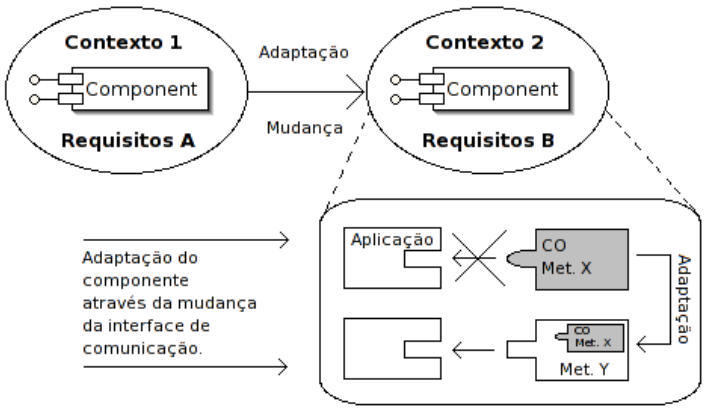

Figura 12: Adaptação de componentes, adaptado de (WEISS, 2001)

A adaptação está centralizada no empacotamento do componente, devido a incompatibilidade de interface de comunicação, onde são preservadas as funcionalidades originais e adicionadas outras referente às novas necessidades/funcionalidades.

Para Silva (2003) o nível de complexidade de reconfiguração/adaptação de uma aplicação distribuída é maior do que uma local, devido a organização desta em uma rede de computadores. O princípio fundamental para realização da adaptação é a capacidade do software possuir mecanismos que permitam coletar dados a respeito de seu estado no ambiente de execução, analisá-los visando identificar mudanças significativas e alterar dinamicamente seu comportamento/estrutura. Com base nesses conceitos, um *framework* de reconfiguração foi desenvolvido e encontra-se fundamentado em quatro características (SILVA et al., 2002) (SILVA et al., 2003) (SILVA, 2003):

- análise de componentes adaptativos e o relacionamento entre eles;
- projeto de componentes, que envolve a monitoração do ambiente de execução, detecção de suas variações significativas no mesmo, reconfiguração da aplicação, além da definição das interfaces de comunicação dos componentes;
- implementação de componentes que podem ser "unidos" aos demais existentes no *framework*; e
- execução dos componentes, corresponde ao início, término, suspensão e reinício. Alteração das propriedades e incorporação ou remoção, em tempo de execução, de novas funcionalidades também tratadas.

Segundo Kim (2001), o DSBC ou POC tem sido empregado pelos melhores métodos de engenharia de software na atividade de desenvolvimento. Sendo assim, recursos tecnológicos modernos têm sido utilizados no desenvolvimento de componentes e *deploy* (registro) por terceiros ou adaptados, para que atenda aos requisitos de seus clientes. Dessa forma, um método de adaptação de componentes binários é proposto pelo autor, cujo objetivo é controlar o tamanho dos componentes a cada adaptação que é realizada, pois a técnica empregada para a adaptação é o empacotamento de novas funcionalidades e, assim os componentes aumentam de

tamanho muito rapidamente. Nesse método, os componentes são utilizados como *plugins* por uma ou mais aplicações.

Outra consideração importante destacada por Kim (2001), é a difícil aplicação do conceito "componente caixa-preta" no desenvolvimento de software, pois qualquer adaptação necessita do código fonte, tanto para empacotar quanto para aplicar mecanismos de herança. A reutilização de componentes depende das funcionalidades que disponibilizam em suas interfaces, pois estas serão "plugadas" em outros componentes.

Heineman (1999, 2009) define 11 requisitos para que um componente possa ser adaptado. Os mais importantes são:

- **homogeneidade de código:** o componente original deve ser utilizado num componente adaptado;
- **conservativo:** um componente original pode ser acessado por um adaptado sem esforço de acesso;
- **composição:** um componente adaptado deve estar apto para novas mudanças que sejam nele realizadas;
- **caixa-preta:** as técnicas de adaptação não devem ter conhecimento sobre seu estado interno; e
- **foco arquitetural:** deve haver uma descrição global da arquitetura de uma aplicação destino associada ao componente modificado, ou seja, a descrição das alterações.

Baseado nesses conceitos, Kim (2001) propõe a adaptação de componentes por meio da técnica de empacotamento. Quando novos requisitos são incorporados ao original, é criada uma camada que o envolve gerando um novo componente. Caso outros requisitos necessitem de implementação, novas camadas são criadas, caracterizando o aumento do componente. Toda modificação de componente requer que informações, atributos e/ou métodos, sejam adicionados

ou removidos. A Figura 13 mostra a adaptação do componente utilizando uma escala (+) para representar as novas funcionalidades.

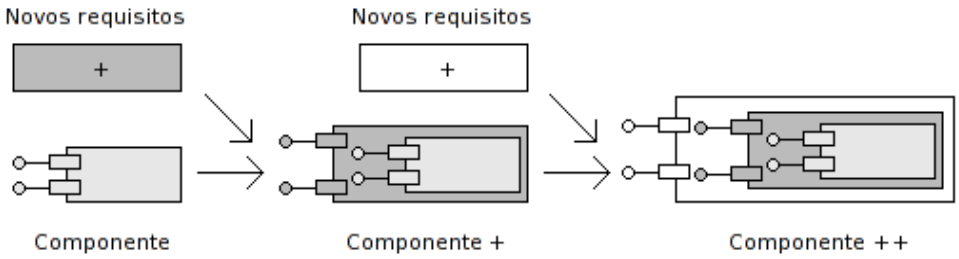

Figura 13: Adaptação de componentes por empacotamento (KIM, 2001)

Dantas & Borba (2003) propõem um padrão para adaptação de software baseado na POA. O ponto central desse padrão é a segmentação da aplicação em requisitos funcionais e não-funcionais, modularizando-o em blocos menores e aumentando sua reusabilidade. Dessa forma, pode-se dizer que o sistema se torna composto em aspectos. Segundo Dantas et al. (2003), a aplicação desse padrão pode gerar os seguintes benefícios: (i) modularidade entre a aplicação e os aspectos de adaptação; (ii) reúso entre as classes auxiliares, devido a segmentação entre as classes auxiliares, classes adaptativas e núcleo de aplicação, com isso, facilitando futuras manutenções; (iii) independência de plataforma: por ser aplicável a uma grande quantidade de sistemas e; (iv) por permitir mudanças dinâmicas: que corresponde a alteração do comportamento da aplicação.

Kasten et al. (2002) definem a adaptação centralizada em componentes de software num modelo dividido em dois enfoques: observação do comportamento (*introspection*) e observação da mudança do comportamento (*intercession*). Nesse modelo, destacam-se as operações realizadas no meta-nível:

- ***refractions*:** fornecendo uma visão limitada sobre o nível base do componente; e
- ***transmutations*:** representa a capacidade de modificar a funcionalidade de um componente no meta-nível.

Essa última tem uma característica especial, pois mostra a capacidade de adaptação do meta-nível, que, num sistema reconfigurável, tem a capacidade de modificar outro objeto/componente no nível base. Portanto, pode-se dizer que existe a reconfiguração de meta-objetos responsáveis pela reconfiguração objetos-base (KASTEN et al., 2002).

Para Kleinoder & Golm (1996), a reconfiguração está centralizada em duas abordagens: reflexão e metaprogramação. A primeira já foi abordada neste capítulo em recursos e características, dispensando maiores detalhes neste momento. A segunda representa a segmentação do sistema em: (i) funcional, que representa a funcionalidade do sistema, e (ii) não-funcional, que corresponde aos meta-objetos supervisores da aplicação capazes de introduzir novos comportamentos a eles (aplicação).

Previtali (2007) define a reconfiguração de sistemas utilizando interceptação das chamadas por meio da POA (ASPECTJ, 2009) associada à programação reflexiva JAVA *Reflect* (SUN-REFLECT, 2009), para modificação dos objetos em seus atributos, métodos, construtores. Sua técnica de adaptação consiste em criar um novo objeto a cada solicitação de modificação enviada por seus clientes. Dessa forma, dois aspectos podem ser facilmente identificados: 1) os estados dos objetos são facilmente preservados e a adaptação não afeta as execuções vigentes; 2) com essa estratégia de adaptação, muitos objetos são criados em memória, fazendo com que aplicação principal tenha problemas relacionados ao seu desempenho e sobrecarga de memória. No entanto, em (PREVITALI & GROSS, 2006) a técnica empregada na reconfiguração foi o uso de objetos *proxies*, apoiada pela substituição do código em execução. Dessa forma, muitos esforços tinham que ser designados ao desenvolvimento do monitoramento e manutenção do estado de execução dos objetos.

Segundo Oreizy & Taylor (1998), a reconfiguração de sistemas pode ser centralizada em termos de uma arquitetura automatizada e os artefatos de software (objetos, componentes e serviço) aptos a mudarem de comportamento e aceitar a reconfiguração por meio de métodos

previamente implementados na etapa de automatização e disponibilizados como conectores.

Miladi et al. (2008) apresentam um trabalho sobre arquitetura de softwares reconfiguráveis por meio de OCL, *Object Constraint Language*. Basicamente, um conjunto de regras (restrições) é formalizado para demonstrar as funcionalidades de uma operação. Transformadores de regras traduzem essas regras em código fonte e, posteriormente, utilizam essas regras para que possam realizar alguma reconfiguração.

Em Mckinley et al. (2004) é apresentado um modelo para adaptação de software em ambiente distribuído. Como em outros trabalhos apresentados neste Capítulo, este também utiliza um conjunto de recursos computacionais para alcançar esse objetivo. Basicamente, eles são organizados em quatro grupos: separação de interesses, reflexão computacional, projeto baseado em componentes e *middleware*.

O primeiro, separação de interesses, fornece recursos para a segmentação do sistema em requisitos funcionais e não-funcionais, já citados no Capítulo 2 - Seção 2.2.3.1. O mesmo ocorre com o segundo recurso, reflexão computacional, e sua habilidade de descobrir suas funcionalidades e decidir como alterar seu comportamento. O terceiro, projeto baseado em componentes, aborda as tecnologias capazes de suportar essa maneira de desenvolvimento. Finalmente, o *middleware*, representa as tecnologias encarregadas de fazer a distribuição dos componentes (MCKINLEY et al., 2004) (ARSHAD et. al., 2003)

Pode-se observar que os recursos apresentados por Mckinley et al. (2004) para realizar a adaptação de software não são inovadores, no entanto, sua contribuição está na maneira como são utilizados para alcançar este objetivo. Os autores sugerem dois tipos de adaptação (estática e dinâmica) por meio de composição, sendo a primeira delegada ao desenvolvedor da aplicação e a segunda em tempo de execução, adicionando, removendo ou reconfigurando componentes de uma aplicação. Dentre as tecnologias de reconfiguração utilizadas, destacam-se a *AspectJ* (ASPECTJ, 2009), que atua na modificação dos objetos em tempo de execução adicionando

características e funcionalidades, e a OpenORB (OPENORB, 2009), com a distribuição de objetos num *middleware* CORBA (OMG, 2009) (SIEGEL, 1996) com interfaces reflexivas.

Kim & Bohner (2008) utilizam a reconfiguração de componentes de software por meio da POA. Os autores comentam sobre a facilidade de armazenamento e restauração dos estados dos objetos com a adoção desta técnica de programação (POA). Além disso, destacam os benefícios da reconfiguração dinâmica pela adição e/ou remoção de características/ funcionalidades. No entanto, não trabalham com nenhum sistema supervisor para monitoramento das mudanças dos requisitos em tempo de execução.

Em Yang et al. (2008) pode-se encontrar a POA como mecanismo principal de adaptação de software. No entanto, sua proposta está centralizada em uma arquitetura que avalia as condições para realização da adaptação.

Outro conceito interessante a ser abordado neste trabalho é a utilização de objeto *proxy* combinado com a POA, para interceptação e modificação dos objetos em tempo de execução (AKKAWI et al., 2002). Para viabilizar essa proposta, os autores implementaram um *framework* orientado a aspectos para modificação do software em tempo de execução.

Gustavsson et al. (2004) também comentam sobre a utilização da POA para reconfiguração de software. Sua proposta está centralizada nos conceitos da segmentação dos sistemas em requisitos funcionais (lógica do sistema) e não-funcionais (infraestrutura), destacando as limitações da API JAVA *Reflect* para modificação da lógica do sistema.

Segundo Richmond & Noble (2001), a reconfiguração de aplicações distribuídas desenvolvidas com a RMI (SUN-RMI, 2009) é limitada, devido à falta de mecanismos de reflexão em objetos remotos. Sua proposta é a utilização do padrão *proxy* (STELTING & MAASSEN, 2002) para que os objetos remotos se comportem de maneira transparente à aplicação como se fossem locais.

Kon & Campell (1999) comentam a falta de um modelo que represente a dependência

entre os sistemas/componentes e o gerenciamento de suas configurações. Para eles, num ambiente distribuído, o sistema operacional e o *middleware* devem fornecer suporte para representação dos componentes de uma aplicação, pois, dessa forma, ela pode reconhecer a necessidade de reconfiguração. A proposta desses autores é a dependência dinâmica de componentes, sendo cada um gerenciado por um "*component configurator*". Para garantir a portabilidade dessa proposta, um *middleware* CORBA é associado aos componentes, para que sejam desenvolvidos em linguagens de programação diferentes em um ambiente heterogêneo.

A necessidade de um ORB reconfigurável pode ser implementada pelo TAO (ORB), *Dynamic* TAO, que tem como características a portabilidade, flexibilidade, extensibilidade e a configurabilidade. Este é baseado no padrão orientado a objetos, suas principais características são (KON & CAMPELL, 1999):

- possui interface para "carga" e "descarga" dos módulos em tempo de execução; e
- possui uma arquitetura para reconfiguração dos componentes internos (módulos CORBA) e para aplicação.

Uma limitação encontrada nessa proposta, não implementada pela implementação CORBA, é reconfigurar um ORB em execução. Com base na implementação *Dynamic* TAO, um pedido para substituir uma estratégia A por uma B é enviado a ele e, dois problemas são identificados (KON & CAMPELL, 1999):

1. antes de descarregar a estratégia A, o sistema deve estar certo que ela não está sendo utilizada ou solicitada por nenhum objeto; e
2. algumas estratégias precisam manter seu estado de informação. Quando a estratégia A é substituída pela B, o estado interno de A é transferido para B.

Esses problemas devem ser considerados pelo sistema gerenciador de configuração para

que inconsistências não sejam geradas, pois um objeto pode estar sendo executado e necessita preservar sua execução para quem o solicitou antes de ser reconfigurado.

A reconfiguração de componentes num ambiente distribuído também é abordada por Chen (2002). Sua proposta é uma extensão na RMI permitindo a uma aplicação monitorar e manipular invocações entre componentes, durante uma reconfiguração dinâmica. Nesse processo, um componente pode ser carregado dinamicamente no sistema, "migrado" de um local para outro, descarregado do sistema em tempo de execução ou atualizado dinamicamente.

Uma das premissas propostas por Chen (2002) é a manutenção da integridade de um componente. Quando ele é "migrado" de um local para outro, a integridade deve ser mantida. Antes que isso ocorra, seu estado pendente (operação em execução) deve ser finalizado. Para isso, monitores de estado dos componentes devem ser implementados para controlar referência a eles. Em um ambiente distribuído, os componentes podem estar espalhados por diversos computadores numa rede e, em cada um deles, existe um agente responsável por monitorar o estado dos artefatos (componentes, objetos e serviços) reduzindo inconsistências de versões e utilizações indevidas sejam inibidas.

Para que a reconfiguração de componentes distribuídos seja realizada são necessárias as seguintes condições (CHEN, 2002):

- não existir nenhuma solicitação por parte do cliente para aquele componente;
- as invocações realizadas sobre determinado componente por um cliente foram completadas (ausência de estado pendente);
- nenhuma nova invocação em qualquer outro componente foi realizada; e
- as invocações inicializadas pelo servidor de componentes foram completadas;

Para Chen (2002), a reconfiguração dinâmica pode causar inconsistência no sistema se essas condições não forem implementadas. Em um sistema distribuído tradicional, quando um

componente é reconfigurado, as referências a ele são perdidas. Os *middlewares*, em geral, não oferecem suporte de reflexão, sendo assim, um ambiente distribuído que ofereça suporte à reflexão deve fornecer as seguintes condições:

- se um componente inicia o processo de reconfiguração, o *middleware* deve ser o primeiro a bloquear novas invocações a ele, salvar seu estado e, após a reconfiguração, reconstituir as referências com os componentes anteriormente conectados;
- mecanismos de análise para interação entre componentes, pois necessitam reconstituir os estados dos componentes quando reconfigurados;
- comunicação transparente entre os componentes, locais ou não, pois se um componente A mudar de local, as referências a ele devem ser atualizadas.
- mecanismo de medição para as taxas de invocação de um objeto, pois estas devem ser utilizadas na "migração" de reconfiguração.

Para viabilizar todas as necessidades apresentadas tem-se a XRMI, *eXtended* JAVA RMI (CHEN, 2002), que é uma extensão de JAVA RMI com suporte à reconfiguração dinâmica. Ela é composta por um agente de configuração e um virtual *Stub* (mecanismo de comunicação cliente-servidor) no topo do JAVA RMI.

A comunicação entre cliente e servidor ocorre por meio de um *Stub* virtual e não pelo tradicional encontrado no RMI. Este é um objeto local utilizado como um componente cliente. Quando um componente faz a busca (*lookup*) por um servidor, o *Stub* virtual do servidor de componentes é dinamicamente carregado em tempo de execução e conectado ao cliente. Ele mantém referência entre o servidor e o componente cliente, caso eles sejam reconfigurados (CHEN, 2002). A Figura 18 mostra a estrutura de comunicação do XRMI e seus componentes que viabilizam a reconfiguração.

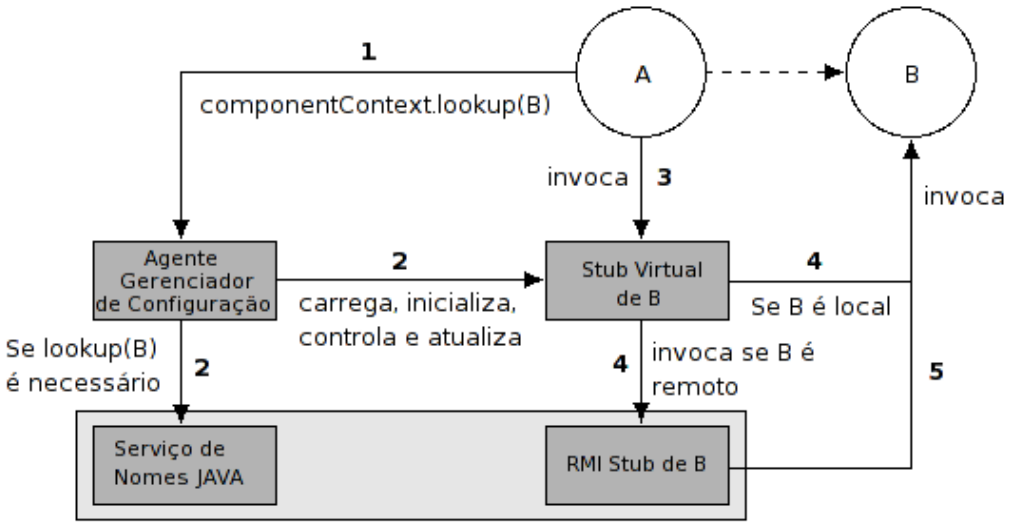

Figura 14: Estrutura do XRMI (CHEN, 2002)

Inicialmente, o cliente procura por um servidor de componentes (`B = serverName`) através do comando "`componentContext.lookup(serverName)`". Em seguida, o Agente Gerenciador de Configuração (AGC) verifica se o servidor de componentes é local, caso contrário, tenta localizar uma referência (endereço) para ele. Então o carregamento do *Stub* virtual é inicializado pelo método `initialization(clientName, serverName, target_ref)`. Para controlar as invocações entre os componentes, foi criado um contador de referência no *Stub* virtual, que armazena o número de invocações entre o cliente e o servidor de componentes (CHEN, 2002).

Finalmente, para finalizar o levantamento de técnicas de reconfiguração aborda-se sua aplicação em banco de dados. Ege (1999a, 1999b) emprega os mecanismos de reflexão para explorar as características e funcionalidades do esquema de banco de dados, avaliando, principalmente, a capacidade de persistência. Sua proposta está centralizada no conflito de paradigma existente entre a linguagem de programação orientada a objetos, utilizada no desenvolvimento da aplicação, e o banco de dados relacional utilizado para armazenamento e recuperação de objetos.

Segundo Ege (1999a, 1999b), duas maneiras de realizar a persistência de objetos em banco de dados relacionais:

- **explícita:** quando é delegada ao desenvolvedor a responsabilidade de programar classes capazes de realizar a persistência de objetos; e
- **implícita:** quando é realizada de maneira automática por um sub-sistema ou objeto monitor do objeto a ser persistido no banco de dados.

A proposta do autor é a criação de um conjunto de classes capazes de inserir, recuperar, atualizar e navegar pelos objetos. Dessa forma, um objeto *proxy* deve ser criado para trabalhar como interpretador de requisição da aplicação para o banco de dados, fazendo com que a solicitação/resposta seja realizada de maneira transparente ao paradigma da aplicação.

Os conceitos, técnicas e recursos apresentados nesta seção são amplamente utilizados no desenvolvimento deste trabalho, pois fornecem um conjunto de diretrizes para adaptação/reconfiguração de sistemas isolados, orientados a componentes, distribuídos e banco de dados. Além disso, são apresentadas outras técnicas de programação (POA) que atuam de maneira associativa à reconfiguração, pois auxiliam na segmentação de requisitos de uma aplicação, reduzindo sua complexidade e facilitando a incorporação de novas necessidades.

Outro destaque desta seção está nas técnicas aplicadas aos sistemas distribuídos, em que é apresentado um ORB com características reflexivas e uma extensão da RMI (XRMI), como forma de facilitar o processo de invocação de objetos distribuídos reconfiguráveis.

3.4. A API JAVA REFLECT

Para implementar os recursos de Reflexão Computacional (RC) deve-se ter um recurso computacional que seja capaz de reconhecer as características e funcionalidades de classes/objetos. Segundo Sun-reflect (2009), desde a versão 1.1 do JDK (JAVA *Development*

Kit) foi implementado esse recurso, com algumas limitações, como já citado pelos autores e seus trabalhos apresentados neste capítulo, mas com recursos que viabilizam as necessidades pela RC. Essas limitações podem ser compensadas através da associação de outros recursos de programação, que trabalham de maneira cooperativa, minimizando os problemas das limitações existentes.

Sun-reflect (2009) mostra que a Reflexão em JAVA pode ser utilizada para a descoberta de informações como campos, métodos, construtores e classes carregadas pela JVM. Dessa forma, uma API específica para reflexão é implementada e suas principais características são (SUN-REFLECT, 2009) (TATSUBORI et al, 2001):

- determinar a classe de objetos;
- obter informações de objetos, tais como: modificadores, métodos, construtores, entre outras;
- criar a instância de uma classe (objeto) em tempo de execução;
- ajustar ou obter valores em/de objetos em tempo de execução;
- invocar métodos de objetos em tempo de execução;
- manipular *array* e seus tipos em tempo de execução; e
- substituir um objeto por outro em tempo de execução.

Por motivos de escopo, apenas as principais características dessa API são apresentadas, pois existem várias funcionalidades que oferecem suporte à "descoberta" de informação. No entanto, outro fator de destaque dessa API é sua adequação aos novos recursos da linguagem JAVA, versão 1.5 ou superior, tais como *Generics*, *Annotation*, *Enums*, entre outros (SUN-REFLECT, 2009). Isso é importante para a evolução dos recursos de reflexão e para execução deste trabalho, pois os recursos de distribuição e outros complementares estão submissos a essa versão da linguagem JAVA.

Além dos recursos de aquisição de conhecimento sobre classes e objetos já mencionados, existe uma implementação de uma classe *proxy* dinâmica, utilizada como mecanismo de reconfiguração. Essa classe implementa um conjunto de interfaces que atende às solicitações em tempo de execução, ou seja, dependendo da informação enviada, um objeto será retornado. Dessa forma, pode-se afirmar que a classe *proxy* dinâmica pode ser utilizada como um tipo seguro para geração de objetos (SUN-REFLECT, 2009).

Sun-reflect (2009) comentam que esse tipo de classe é muito útil em aplicações ou bibliotecas que disponibilizam vários serviços, pois os objetos podem implementar os métodos que foram solicitados pela sua interface e atender a várias solicitações.

Forman & Forman (2004) apresentam o conceito de carregamento dinâmico de objetos pelo *ClassLoader*. Este é um componente nativo de um ambiente de programação JAVA, pois quando uma aplicação é executada, ele é o responsável por carregar todas as classes e referenciá-las, quando necessário. Sendo assim, quando um objeto deixa de ser utilizado pela aplicação, ele é descarregado pelo *Garbage Collection*. Dessa forma, quando pretende-se reconfigurar uma aplicação, refere-se diretamente à manipulação de *ClassLoader* e, consequentemente, em alguns cuidados que devem ser levados em consideração, tais como: preservação do estado de execução de objetos, substituição de objetos, recompilação de objetos, entre outros. Esses problemas foram apresentados na Seção 3.3 juntamente com suas soluções, as quais servem de referência para a execução deste trabalho.

Outro ponto de destaque citado pelos autores é o mecanismo de herança que pode ser aplicado aos *ClassLoaders*, ou seja, as aplicações podem criar seus *ClassLoaders* para referenciar seus objetos. Dessa forma, eles podem assumir papéis de gerenciamento personalizado aos objetos, carregando e descarregando conforme regra de reconfiguração (FORMAN & FORMAN, 2004).

Para Sun-developers (2009), a distribuição de uma aplicação utilizando RMI pode ser

realizada através do carregamento dinâmico de classes. Em versões tradicionais, os clientes necessitam conhecer os *stubs* disponíveis, para garantir a comunicação entre clientes e servidores. Com a utilização dessa característica, carregamento dinâmico de classes, os objetos clientes são capazes de construir *stubs* dinâmicos em tempo de execução, através de *proxies* dinâmicos. Isso permite que JVMs clientes encontrem *stubs* no ambiente distribuído, sem informar características especiais e, mais importante, tudo realizado em tempo de execução.

Os recursos e conceitos dessa API são utilizados, neste trabalho, na descoberta de características e funcionalidades de classes/objetos, tanto para o processo de documentação automática quanto para a reconfiguração, como mostra a Figura 15.

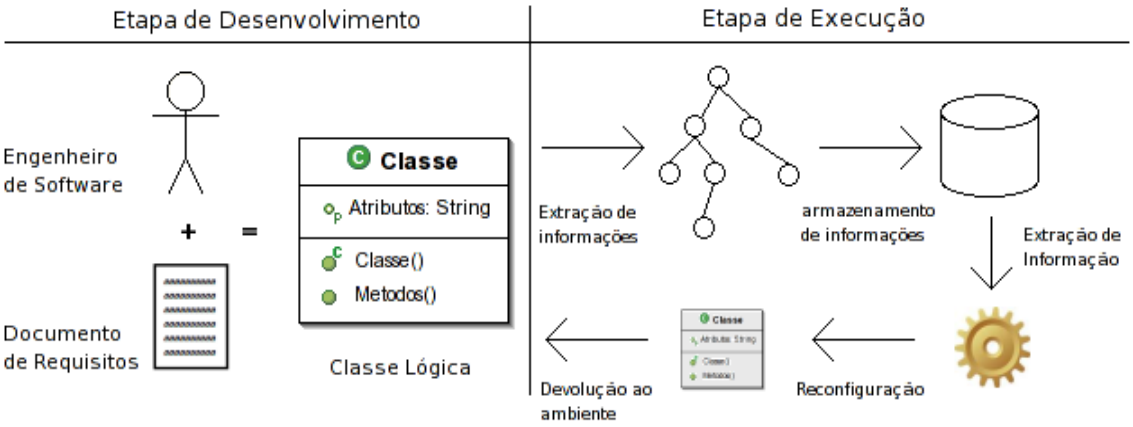

Figura 15: Processo de recuperação e armazenamento de objetos

Na concepção inicial de um sistema, o engenheiro de software, baseado no documento de requisitos, faz a confecção das classes/objetos que serão inseridas(os) no ambiente de execução. Neste, cada objeto (nível base) é monitorado por um objeto tutor (meta-nível), que aciona um subsistema responsável por extrair todas as características e funcionalidades. Essas informações são organizadas em um arquivo XML, que representa uma árvore de informação. Em seguida, essas informações são inseridas num repositório para que sejam utilizadas nas seguintes situações:

- invocação de objetos por clientes com base nas suas características e

funcionalidades documentadas; e

- reutilização ou incorporação de alguma dessas características e/ou funcionalidades documentadas em outros objetos no processo de reconfiguração.

3.5. O FRAMEWORK JAVASSIST

Em Mckinley (2004), pode ser encontrado um estudo relevante sobre a adaptação de componentes de maneira isolada e distribuída. Esse estudo apresenta uma comparação técnica dos recursos de reconfiguração existentes, tais como: *Open JAVA*, *Adaptative JAVA*, *Kava*, *Dynamic* TAO, R-JAVA, Guaraná, entre outros.

Weiss (2001), no processo de reconfiguração de componentes, também apresenta um comparativo entre recursos de reconfiguração, em que são utilizados *OpenJIT*, JAVASSIST, *Dalang*, entre outros.

Nos trabalhos desses autores, outros recursos são utilizados para realizar a reconfiguração. Destaca-se a Programação Orientada a Aspectos (*AspectJ*) e os conceitos de separação de interesses. Baseado nessas informações e, após uma análise circunstanciada nos recursos de reconfiguração, interessou-se pelo *framework* JAVASSIT, pois possui as seguintes características: trabalha na reconfiguração do *bytecode*, realiza a introspecção e permite reflexão estrutural e comportamental.

O *framework* JAVASSIT (CHIBA, 1998) (CHIBA, 2009) é uma biblioteca de classes que permite a manipulação (reconfiguração) de *bytecode* JAVA, que é um arquivo binário, contendo o código fonte compilado. Para isso, não modifica a JVM ou introduz novos compiladores. Uma característica importante desse *framework* é a capacidade de modificar uma classe sem o código fonte e tendo apenas sua interface como fonte de conhecimento. As principais características do *framework* JAVASSIT são (CHIBA, 1998) (CHIBA, 2000) (CHIBA, 2003) (CHIBA, 2009):

- **reificação:** tem a responsabilidade de fazer uma classe/objeto acessível aos outros de sistema. Para isso, um objeto padrão é criado e referenciado como `CtClass`;
- **reflexão:** permite que todas as características e funcionalidades sejam identificadas e atribuídas a um objeto de sistema;
- **introspecção:** fornece alguns métodos para realizar a modificação da classe representada pelo objeto `CtClass`;
- **alteração:** representa os métodos que o *framework* JAVASSIST possui para realizar a alteração da definição das classes. Esses métodos são utilizados pelo objeto `CtClass`, para alterar os modificadores de classe, métodos, a hierarquia de classe e métodos para adicionar novos membros;
- **adição de novos membros:** é a capacidade de modificação de uma classe através da adição de novos métodos;
- **alteração do corpo de métodos:** representa a capacidade de alteração de um corpo de método. Essa característica é essencial devido a não permissão de exclusão de métodos. Sendo assim, cada modificação de comportamento pode ser realizada por meio da substituição do corpo de um método;
- **carregador de classes reflexivo:** é a capacidade que o *framework* JAVASSIST oferece aos sistemas em controlar o carregamento das classes que utilizam; e
- **JAVASSIST sem carregador de classes:** representa as maneiras de funcionamento do carregamento de classes, que podem ser realizadas por meio do *ClassLoader* do usuário, por meio de *Web Services* e/ou *off-line*.

3.6. Considerações Finais

Este capítulo apresentou um conjunto de assuntos que oferecem suporte para o desenvolvimento deste trabalho. Os conceitos da Computação Reflexiva oferecem diretrizes e

recursos para criar os mecanismos de adaptação dos objetos em tempo de execução. Os objetos tutores (RC) são utilizados para monitorar os objetos no ambiente distribuído.

Os mecanismos de adaptação podem ser guiados pelas técnicas de adaptação apresentadas, que podem ocorrer em vários níveis e estágios da aplicação. Nesses trabalhos, podem-se destacar a adaptação de componentes de software e os *frameworks* de suporte à reconfiguração, que oferecem um conjunto de serviços estabelecidos em contextos específicos. Cada um deles trata de uma abordagem para desenvolvimento de software: local, distribuída ou orientada a componentes. Neste trabalho, seus conceitos são unidos com intuito de produzir um *framework* genérico capaz de atuar em vários contextos.

Dentre as técnicas de reconfiguração apresentadas, destacam-se aquelas aplicadas a reconfiguração de ambientes distribuídos utilizando RMI, pois nessa tecnologia existem algumas limitações implícitas já mencionadas e, para resolvê-las, outros recursos computacionais são incorporados para atuar de maneira cooperativa à reconfiguração. Dentre eles, destacam-se a Programação Orientada a Aspectos com a separação de interesses, o padrão *proxy* com o *Stub* virtual e, a reflexão aplicadas aos esquemas de banco dados, que atua num novo nível de reconfiguração.

Ainda dentro do contexto reconfiguração, destacam-se os ORBs reflexivos, que utilizam objetos gerenciadores de configuração para controlar a reconfiguração e referência de objetos no ambiente distribuído.

A API JAVA *Reflect* oferece todo recurso de descoberta de características e funcionalidades de objetos em tempo de execução. Além disso, pode atuar como mecanismo de modificação estrutural e comportamental de uma classe, através da modificação do código fonte associada a nova compilação. Isso pode ocorrer de duas maneiras: retirando o objeto atual da memória, compilando e inserindo novamente ou, criando um novo objeto para atender apenas a necessidade solicitada.

O *framework* JAVASSIST fornece conceitos e recursos de implementação para o processo de reconfiguração. Destaque para os mecanismos de reconfiguração baseado no *bytecode* da classe, que dispensam a existência do código fonte, deixando esse processo mais flexível. Outros recursos como a API *Reflect*, apresentados neste capítulo, assumem o papel de recuperar o código fonte e atuar de maneira cooperativa neste trabalho.

O próximo capítulo apresenta as diretrizes da metodologia de desenvolvimento de software reconfigurável.

CAPÍTULO 4

Metodologia para Desenvolvimento de Software Reconfigurável

4.1. CONSIDERAÇÕES INICIAIS

Este capítulo tem por objetivo apresentar a Metodologia de Desenvolvimento de Software Reconfigurável (MDSR) para aplicações locais e/ou distribuídas. Essas aplicações podem ser desenvolvidas em vários paradigmas: orientado a objetos, orientado a componentes, orientado a aspectos e orientado a serviços. Pela sua natureza multidisciplinar, em relação às áreas do conhecimento envolvidas para elaboração de suas diretrizes, foi necessário organizar os assuntos da seguinte maneira: recursos de Engenharia de Software, recursos de Reconfiguração de Software, recursos de Inteligência Computacional e, finalmente, recursos de Distribuição de Objetos.

Os recursos de engenharia de software auxiliam os engenheiros de software na documentação (JAVADOC), modelagem lógica (Diagramas UML), modelagem de dados

(Diagrama Entidade Relacionamento - Visão Física), desenvolvimento (Classes, Componentes, Serviços e Objetos Remotos) e testes (Testes Unitários). Além disso, esses recursos auxiliam na padronização de comunicação entre objetos/componentes e subsistemas (antigos ou serviços) e na sua integração para a composição de um novo sistema.

Os recursos de reconfiguração oferecem suporte para o monitoramento dos objetos num ambiente local e/ou distribuído. Além disso, possuem mecanismos que permitem, uma aplicação em execução, refletir sobre seu estado atual e avaliar as novas solicitações do ambiente de execução, para que sua adaptação seja realizada em conformidade a elas. Essa adaptação pode ser realizada em vários níveis: código fonte do sistema, banco de dados e sistema compilado em execução.

Os recursos de inteligência computacional atuam na documentação dos objetos/componentes/serviços/objetos remotos, chamados deste ponto em diante de artefatos de software, nas fases de desenvolvimento de novos ou na recuperação de existentes em repositórios. Sobre este último, são utilizados agentes móveis e inteligentes para que as solicitações dos clientes sejam atendidas e, posteriormente, na escolha da melhor solução, para a solicitação enviada.

Os recursos de distribuição oferecem mecanismos que viabilizam a distribuição da aplicação, objetos, componentes e serviços, no ambiente de execução reconfigurável. Além disso, os recursos estudados possuem várias limitações quanto à reconfiguração e distribuição, conforme apresentado no Capítulo 3, e a combinação dos recursos existentes minimiza as limitações apresentadas e viabilizam a reconfiguração de software num ambiente distribuído.

A metodologia proposta neste trabalho está associada a um ambiente de execução reconfigurável, que é apresentado no Capítulo 5, para viabilizar o processo de reconfiguração. Esta visa atender o desenvolvimento de sistemas sob diferentes naturezas: *desktop*, distribuídos e voltados para *Web*. Os engenheiros de software e desenvolvedores poderão consultar no

ambiente de execução reconfigurável por qualquer artefato de software, que possam reutilizar para a construção de um novo sistema. Além disso, os objetos tutores podem fazer o mesmo procedimento para adaptação da aplicação em execução, tanto de maneira temporária, quando apenas uma funcionalidade é invocada e posteriormente descartada, quanto permanente, quando ela passa a ser incorporada ao sistema.

Este capítulo apresenta, na Seção 4.2, a Metodologia para Desenvolvimento de Software Reconfigurável e, na Seção 4.3, as considerações finais.

4.2. Metodologia para Desenvolvimento de Software Reconfigurável

Este trabalho tem como interesse principal o Desenvolvimento de Software Reconfigurável (DSR) em vários ambientes de execução: local, distribuído e *Web*. Além disso, esta abordagem deve atender a diversas abordagens de desenvolvimento de software, tais como: Orientação a Objetos (OO), Orientação a Componentes (OC), Orientação a Aspectos (OA) e Orientação a Serviços (OS) (PRESSMAN, 2006) (KICZALES et al., 1997) (ERL, 2004). Como se pode observar, em cada uma delas, diferentes tipos de softwares (classe, componente, aspecto e serviço) serão desenvolvidos de diferentes maneiras: **1)isolada:** quando a metodologia é capaz de desenvolver o software por completo; **2)cooperativa:** quando se tem uma situação oposta da apresentada em (1) e necessita da combinação de novos recursos para atender às necessidades do sistema em desenvolvimento.

Além do suporte a vários tipos de metodologias de desenvolvimento de software, é necessário que os softwares em desenvolvimento possam se comunicar com outros tipos de sistemas: modernos, antigos (legados) e a combinação deles. Dessa forma, um dos recursos aplicados para viabilizar a comunicação entre sistemas é a linguagem XML e suas APIs de integração com JAVA. Elas são utilizadas, por exemplo, na extração de informação do servidor de banco de dados em arquivos, na leitura desses arquivos pelos sistemas modernos para

manipular as informações nele contidas, na leitura e escrita de arquivos pelos subsistemas e na apresentação das informações geradas em interfaces *Web*, como mostra a Figura 16.

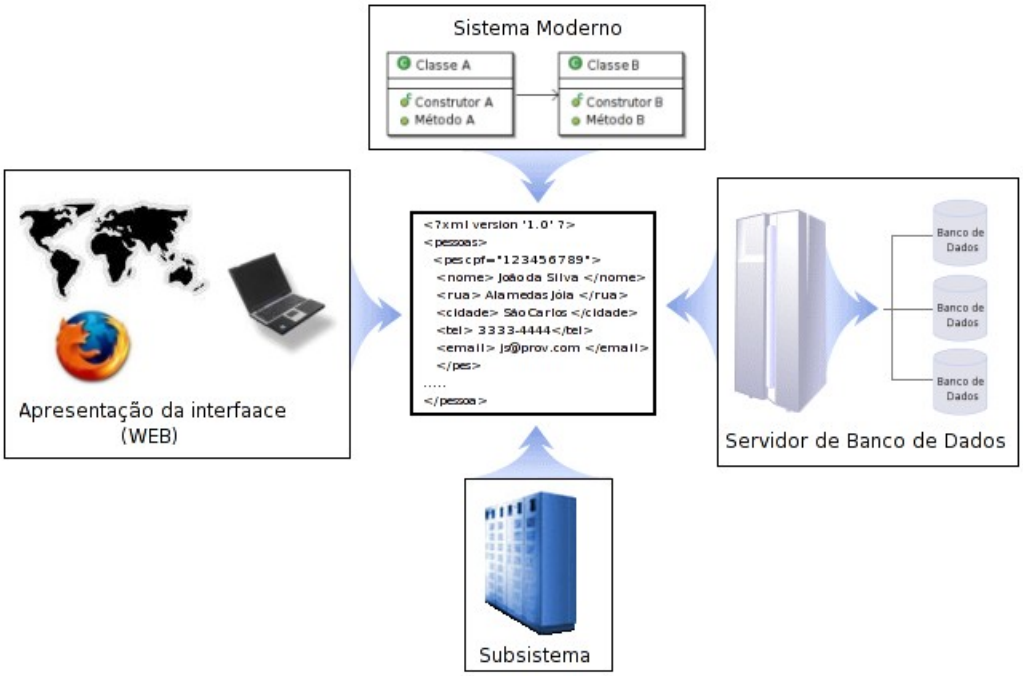

Figura 16: Processo de comunicação entre sistemas via XML

Antes de iniciar os passos da MDSR é necessário apresentar os conceitos sobre arquitetura de software presente em seus passos. Destacam-se a Arquitetura de Software Baseada em Domínio e a Arquitetura de Software em Camadas, cujos conceitos e detalhes são apresentados na Seção 4.2.1 e, na Seção 4.2.2, os detalhes da MDSR.

4.2.1. Arquitetura de Software e Ambientes Computacionais

A Metodologia proposta neste capítulo também tem por objetivo a reutilização de software em tempo de execução ou em procedimentos supervisionados manualmente. No entanto, para viabilizar essa necessidade é necessário, conforme estudos realizados em trabalhos relacionados de diversos autores apresentados no Capítulo 3, um modelo de arquitetura de software genérico para o desenvolvimento dos artefatos de software e, consequentemente, sua

reutilização no ambiente de execução reconfigurável.

Conforme estudos preliminares realizados em várias técnicas de reconfiguração, Capítulo 3, para sistemas locais, distribuídos e *Web*, constatou-se que a reconfiguração ocorre de maneira facilitada em artefatos livres de requisitos não-funcionais. Portanto, diante disso, será adotada uma arquitetura que permite a criação de artefatos "puros", independente de requisitos de infraestrutura, como mostra a Figura 17.

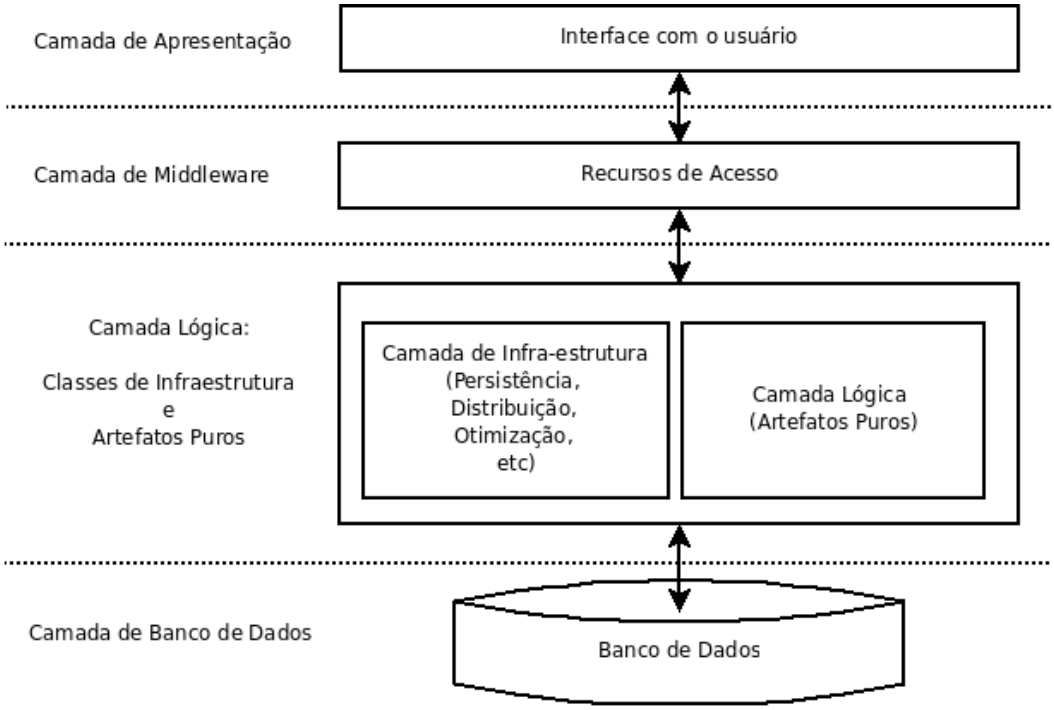

Figura 17: Arquitetura de Software em Camadas

Neste modelo de arquitetura, além da organização lógica em camadas, padrões de projeto como *Data Access Object* (DAO), *Singleton*, *Facade*, *Proxy* e *Model View Controller* (MVC), também estão relacionados no desenvolvimento de software reconfigurável. Basicamente, a arquitetura está organizada em quatro camadas e podem ser resumidas da seguinte maneira:

- banco de dados, que representam os dados a serem armazenados ou recuperados no sistema;

- camada lógica, que está organizada em duas sub-camadas. Pode-se observar que os artefatos de softwares "puros" e os requisitos referentes à infraestrutura estão representados em camadas distintas;
- camada de *middleware*, representam os recursos de acesso aos objetos da camada lógica; e
- camada de apresentação, representa a interface de comunicação entre o usuário e o sistema.

O principal benefício desta arquitetura para a MDSR é a reutilização dos artefatos em diversas abordagens de desenvolvimento (Orientado a Objeto, Orientado a Componentes, Orientado a Aspectos e Orientado a Serviços). Essa facilidade é alcançada devido ao isolamento dos requisitos funcionais nas classes lógicas e dos não-funcionais em classes de infraestrutura. Um exemplo mais evidente deste benefício é uso de anotações JAVA para a persistência dos dados, como por exemplo o *framework Hibernate* (HIBERNATE, 2009). Detalhes sobre o seu uso na MSDR ainda serão abordados no decorrer deste capítulo. Outros *frameworks* de persistência que possuem suporte ao recurso de anotações também podem ser utilizados na MDSR.

Arquiteturas de software, metodologias de desenvolvimento de software e ambientes computacionais são assuntos diretamente relacionados ao trabalho proposto. No entanto, sobre este último, foram encontrados alguns trabalhos na literatura que expressam uma problemática existente entre os ambientes computacionais e o apoio ao desenvolvimento de software.

Segundo Nakagawa (2006), existem algumas dificuldades em encontrar ambientes computacionais capazes de suportar as diversas fases de desenvolvimento de um artefato de software (classe, componente, aspecto ou serviço), desde o levantamento de requisito até sua implantação. No entanto, a autora comenta, em seu trabalho, esforços de vários pesquisadores

na elaboração/apoio de ambientes computacionais e processos de software capazes de suprir as necessidades atuais.

Dentre os trabalhos e esforços encontrados na literatura, destaca-se a Arquitetura de Software Baseada em Domínio (ASBD), devido às fases existentes em sua metodologia para a elaboração de um artefato de software, que podem ser resumidas em: (i) geração do modelo de domínio do artefato; (ii) geração do requisito de referência para o artefato; (iii) geração da arquitetura de referência para o artefato; (iv) existência de um ambiente computacional capaz de automatizar a geração e recuperação dos artefatos; e (v) processo de software definido que viabiliza a execução das atividades para o desenvolvimento do artefato desejado.

A justificativa desse assunto na MDSR é a necessidade da melhor especificação dos artefatos de software e, consequentemente, o melhor nível de reutilização deles no processo de reconfiguração (manual ou automático). A seguir, são apresentadas as diretrizes da MDSR.

4.2.2. Diretrizes da MDSR

Após a apresentação dos conceitos e mecanismos de padronização para a comunicação entre objetos locais e distribuídos, da arquitetura de software e do ambiente de desenvolvimento e execução de software, esta seção apresenta as diretrizes para o Desenvolvimento de Software Reconfigurável, por meio de diversas metodologias de desenvolvimento de software: Orientação a Objetos, Orientação a Componentes, Orientação a Aspectos e Orientação a Serviços.

Antes de iniciar o detalhamento dos passos de execução desta metodologia, julga-se necessário o esclarecimento de seu interesse principal e dos papéis de alguns de seus "personagens" para o desenvolvimento de artefatos de software reutilizáveis, tanto de maneira automática, por meio de subsistemas supervisores, quanto manual, com a supervisão de seres humanos.

Como interesse principal, esta metodologia visa o desenvolvimento de software

reconfigurável e que os artefatos produzidos sejam independentes de recursos de infraestrutura e dotados de uma documentação, que permita sua recuperação pelos mecanismos de pesquisa manuais, com analisadores humanos, ou automáticos, com sistemas supervisórios.

Nesta metodologia, destacam-se três papéis (figuras) importantes, em função de suas responsabilidades nas etapas de desenvolvimento de novos artefatos, documentação e recuperação de artefatos no ambiente de execução. São eles:

- **engenheiro/especialista de domínio:** é o responsável por auxiliar na especificação dos artefatos de software, no respectivo domínio de atuação. Além disso, é o responsável por gerar a documentação semântica utilizada no sistema de busca, quando os artefatos estão inseridos nos repositórios de informação;
- **engenheiro de implementação (desenvolvedor):** é o responsável pelo desenvolvimento dos artefatos de software, testes e inserção no repositório de informações. Além disso, pode atuar na integração de artefatos existentes com os novos que estão sendo desenvolvidos; e
- **engenheiro de software:** é o responsável pela realização da formalização do problema em requisito até a modelagem para o desenvolvimento (implementação/reconfiguração) dos artefatos.

A Figura 18 apresenta os passos da MDSR. Inicialmente, passo 1, o engenheiro de software e o especialista de domínio fazem um estudo sobre a viabilidade do problema e domínio de atuação, para que o documento de requisitos do sistema ou de um artefato seja confeccionado. Contudo, nota-se que os requisitos sofrem mutações de um sistema para outro e ao longo do tempo, sendo assim, esse estudo é plenamente justificado pelo grau de compreensão que se pretende obter em relação às funcionalidades do software que está sendo desenvolvido.

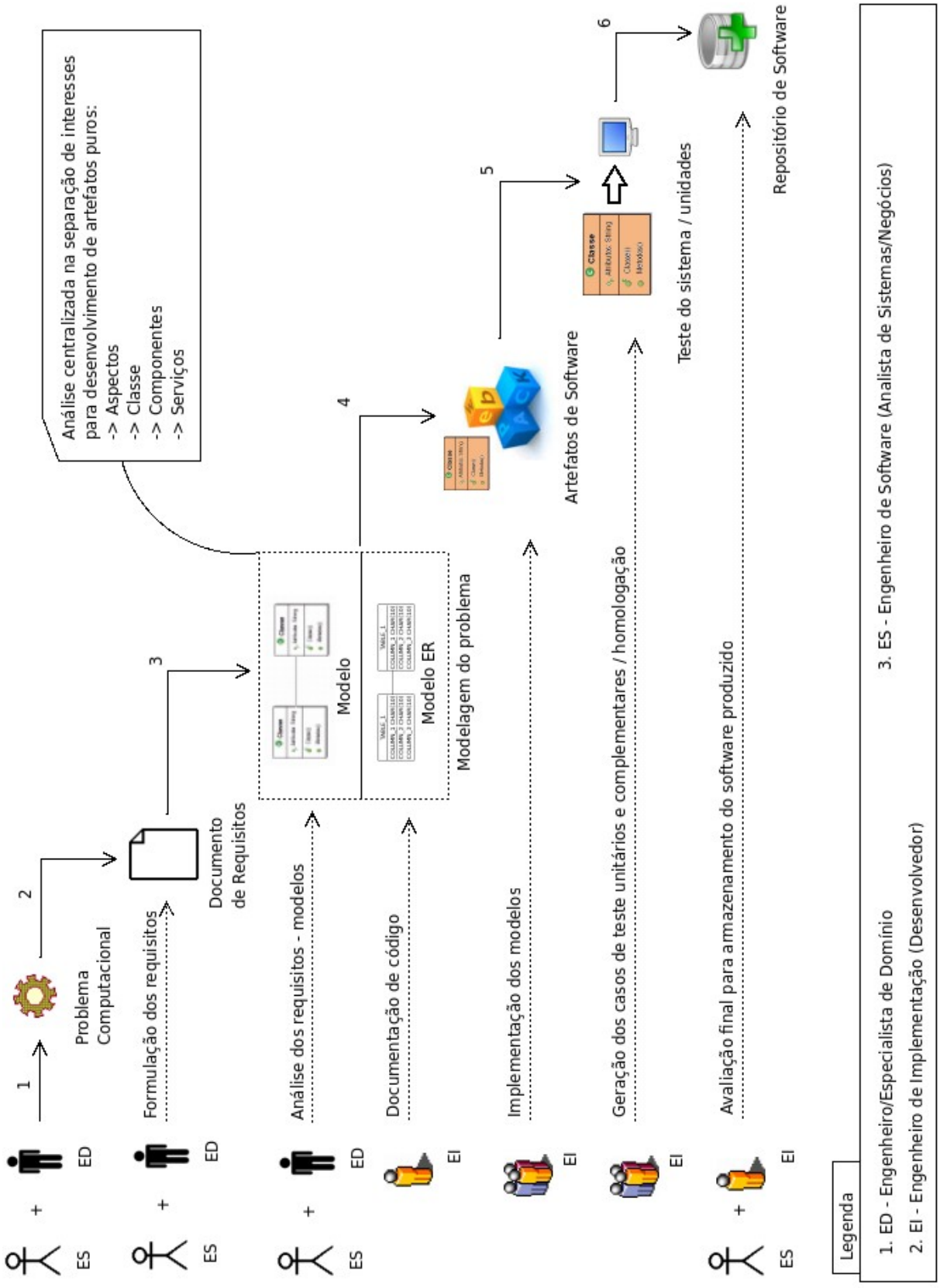

Figura 18: Metodologia de Desenvolvimento de Software Reconfigurável

Destaca-se ainda no passo 1, que o projeto do artefato de software, deve ser o mais especificado possível e isento de funcionalidades de infraestrutura. Isso para que o mesmo tenha um grau de reusabilidade elevado e possa ser utilizado em sistemas de diferentes domínios de atuação.

Após analisada a viabilidade do problema, passo 1, o engenheiro de software e o especialista de domínio iniciam a formalização das ideias, por meio de um documento de requisitos e de informações semânticas que descrevem as funcionalidades do sistema, passo 2. Nesse documento, o engenheiro de software deve expressar as funcionalidades do sistema, apontando quais são relacionadas aos requisitos funcionais (lógica do sistema) e não-funcionais (recursos de infraestrutura). Em um documento similar, o especialista de domínio descreve o significado semântico das funcionalidades e características dos artefatos desenvolvidos. Outro fator relevante é a definição de acoplamento e coesão entre as funcionalidades dos sistemas, pois, neste momento, as funcionalidades de mais baixa coesão e fraco acoplamento podem ser identificadas e isoladas, para serem disponibilizadas como serviços para os demais sistemas.

Com a formalização do documento de requisitos, passo 2, o engenheiro de software e o especialista de domínio podem iniciar a modelagem do sistema (passo 3). Basicamente, são utilizadas as técnicas da linguagem UML, tais como diagramas de classe, de sequência, de caso de uso e de atividade, para os sistemas Orientados a Objetos (OO). No entanto, essa não é a única metodologia de desenvolvimento encontrada na literatura. Outras como o Desenvolvimento de Software Baseado em Componentes (DSBC), o Desenvolvimento Orientado a Aspectos (DOA) e o Desenvolvimento Orientado a Serviços (DOS), também são abordadas neste trabalho. Além dessas, também é interesse deste trabalho a combinação dessas metodologias para o projeto de artefatos de software puros e isentos de requisitos de infraestrutura, que tende a facilitar a integração entre os artefatos desenvolvidos com os existentes, melhorando o poder da reconfiguração em tempo de execução ou manual.

O DSBC é apoiado pelas técnicas UML (BOOCH et al, 2005) presentes nos sistemas OO e fundamentado pelo método *Catalysis* (CATALYSIS, 2009). Nesse método, o engenheiro de software, partindo da especificação dos requisitos funcionais, necessita de três passos: 1)fazer a identificação do domínio de atuação do componente que se pretende modelar, em seguida, 2)fazer o levantamento funcional, definindo seu comportamento e interfaces de acesso e, finalmente, 3) fazer o refinamento do componente, que é realizado com base nos passos anteriores, ajustando os mecanismos de acesso e funcionalidades. Após esses passos, o componente está pronto para ser inserido no *framework* de componentes, como mostra a Figura 5.

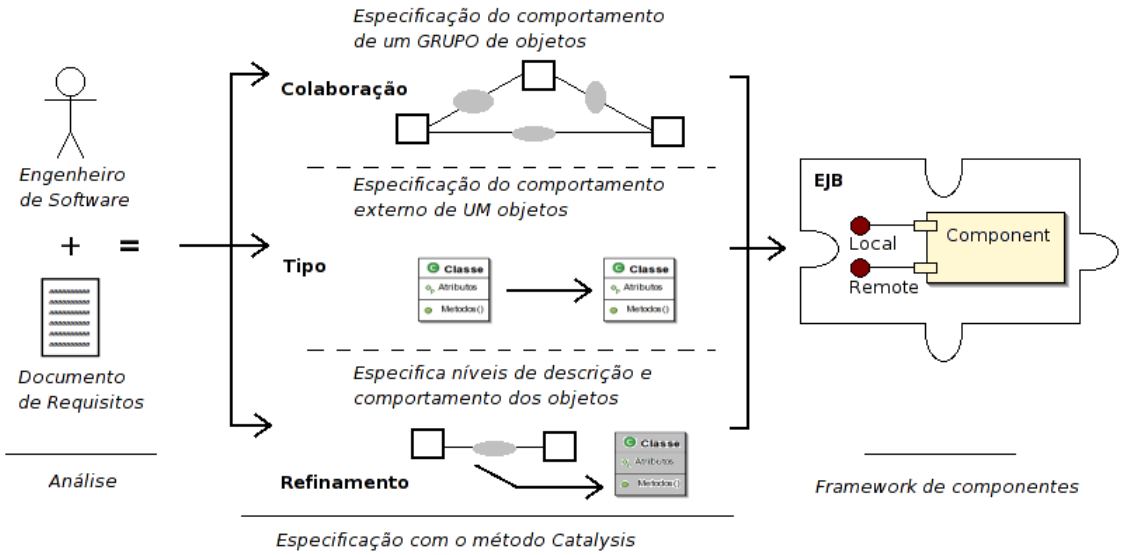

Figura 19: Modelo de desenvolvimento de componentes adaptado de (CATALYSIS, 2009)

Para os sistemas orientados a aspectos, o engenheiro de software, especialista de domínio e os engenheiros de implementação, podem atuar de maneira complementar, sendo os dois primeiros responsáveis apenas pelos requisitos funcionais do sistema e, o terceiro, pelos não-funcionais. A Figura 20 mostra o processo de separação de interesses adotado nesta metodologia para desenvolvimento dos artefatos de software. Nesse processo, passo 3.1, o engenheiro de software de posse do documento de requisitos identifica os possíveis interesses do sistema, para que, em seguida, sejam segmentados, passo 3.2. A segmentação pode ser

resumida em dois tipos de requisitos: **1)funcionais:** que representa a lógica do sistema **2)não-funcionais:** representam algo externo ao sistema, ou seja, infraestrutura para classes e objetos. Após a identificação dos interesses, cabe ao engenheiro de software e ao engenheiro de implementação avaliar a forma de armazenamento dos interesses, passo 3.3, que pode ser no repositório de software ou no *framework* de requisitos não-funcionais, caso o interesse detectado possa ser utilizado em outros domínios de sistemas. Ambos são detalhados no passo 6, que é mostrado na Figura 18.

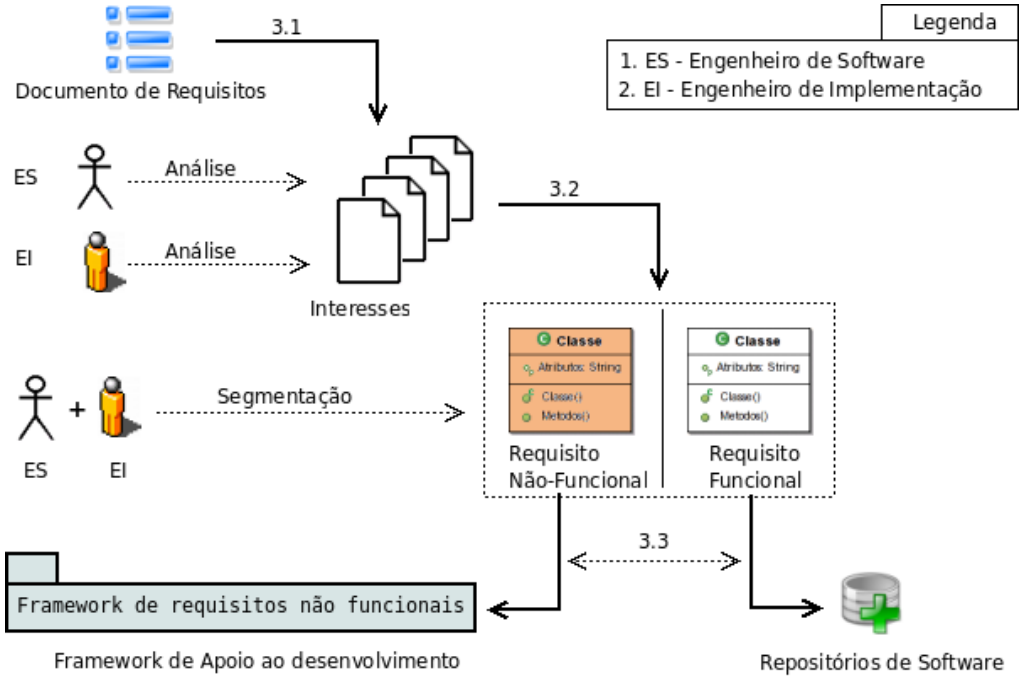

Figura 20: Modelo de desenvolvimento com separação de interesses

Para finalizar o passo 3 da Figura 18, tem-se o desenvolvimento orientado a serviços, que possui os mesmos procedimentos aos orientados a aspectos quanto à identificação das funcionalidades. Quando o engenheiro de software identificar um "requisito funcional puro ou um requisito não-funcional" e avaliar que os mesmos podem se tornar um serviço, segundo os critérios de coesão e acoplamento, para vários sistemas, estes passarão por um processo de registro, utilizando um servidor (provedor de serviço), como por exemplo o *Apache Axis* (APACHE-AXIS, 2009) ou *Sun Application Server* 9 (*Glassfish* v2) (SUN-APPLICATION-

SERVER, 2009). Basicamente, o registro das funcionalidades consiste em disponibilizar os serviços identificados para vários tipos de sistemas. A principal vantagem dessa técnica é a interoperabilidade, pois as funcionalidades registradas, como serviços *Web*, fornecem um mecanismo de comunicação padrão via XML, que descrevem os parâmetros de entrada e saída, e os dados semânticos que representam a descrição de sua funcionalidade. Os passos de registro de funcionalidade nestes servidores *Web* não serão mencionados neste momento, por motivos de escopo do trabalho.

Conforme apresentado neste passo, diversas abordagens de desenvolvimento de software podem ser utilizadas no desenvolvimento de um artefato. No entanto, vale mencionar, que apesar de terem sido apresentadas de maneiras isoladas, elas podem atuar de maneira complementar, mesclando suas características e recursos na confecção ou reconfiguração de um artefato de software.

A realização da modelagem dos artefatos requer por parte dos envolvidos, engenheiro/especialista de domínio e engenheiro de software, alguns cuidados quanto à especificação dos documentos:

1. o nome do artefato de software deve ser um substantivo com a inicial em letra maiúscula e no singular. Exemplos: `Cliente, Carro, Sensor`;
2. o nome dos atributos que qualificam um artefato deve ser um substantivo com inicial em letra minúscula e no singular. Exemplo: `Cliente(nome, email, cpf)`. Quando os nomes forem compostos por duas ou mais palavras `Cliente (dataNascimento)` deve-se concatenar as palavras deixando a palavra de junção com letra em maiúsculo; e
3. o nome dos métodos que definem o comportamento dos artefatos deve ser um verbo com inicial em letra minúscula e no singular. Exemplo: `Cliente(cadastrar, consultar, excluir)`. Quando os nomes forem compostos por duas ou mais

palavras `Cliente(validarCpf)`, deve-se concatenar as palavras deixando a palavra de junção com letra em maiúsculo.

Essas diretrizes auxiliam na formação e padronização dos nomes dos artefatos, seus atributos e métodos. Esses nomes são utilizados por um subsistema reflexivo capaz de gerar, de maneira automática, as regras de classificação dos artefatos desenvolvidos para serem utilizadas num processo de recuperação desses artefatos para reutilização. De maneira resumida, as regras que classificam os objetos são geradas com os mesmos nomes daqueles utilizados na fase de análise (modelagem dos artefatos). O subsistema que faz a geração automática das regras de classificação é apresentado no Capítulo 5. A justificativa para a aplicação dessas diretrizes é a padronização dos nomes e seu significado semântico, no respectivo domínio de atuação que se pretende obter no desenvolvimento de artefatos, que podem ser reutilizados em diferentes domínios. Dessa forma, justifica-se a presença do engenheiro de software e de um especialista de domínio na confecção dos modelos de documentação dos artefatos.

Ainda referente aos passos apresentados na Figura 20, cabe ressaltar que os desenhos dos modelos UML dos sistemas que se pretende desenvolver são implementados por ferramentas capazes de gerar código JAVA a partir dos modelos (diagramas de classes) desenvolvidos. Dentre as ferramentas existentes, destacam-se a *Netbeans* e seu *plugin* UML (NETBEANS, 2009) e a Eclipse com *plugin Omondo* (OMONDO, 2009), por ambas oferecerem suporte à especificação UML e pela praticidade e confiabilidade quanto à geração de código dos artefatos modelados em JAVA.

A partir do código fonte gerado, o desenvolvedor pode optar por fazer o Modelo de Entidade Relacionamento (MER), seguindo os princípios de mapeamento objeto-relacional (ELMASRI, R. E. & NAVATHE, 2005), em dois procedimentos: (1) apoiado por ferramentas de modelagem, desenha-se o MER em nível lógico, realiza-se o mapeamento para o modelo físico

e gera-se o *script* SQL da base de dados. Esse passo pode ser realizado manualmente pelos desenvolvedores ou automaticamente pelas ferramentas de automação do ambiente de execução reconfigurável, que são detalhadas no Capítulo 5; (2) apoiado pelas IDEs de desenvolvimento e pelas API de Anotações de mapeamento objeto relacional (HIBERNATE, 2009), os desenvolvedores anotam em cada relacionamento existente entre as classes como seria o mapeamento no modelo relacional. O recurso de mapeamento com anotações se mostrou mais rápido e de melhor entendimento, além de ser suportado pela API de Reflexão na descoberta de informações dos objetos. As regras de mapeamento utilizando anotações não serão detalhadas neste trabalho, maiores detalhes podem ser obtidos na documentação oficial do projeto (HIBERNATE, 2009).

Em seguida, passo 4 - Figura 18, após realizada a modelagem dos requisitos, conforme uma das metodologias mencionadas, OO, DSBC, DOA e DOS, é iniciada a implementação dos artefatos (classes/componentes/aspectos/serviços) pelos desenvolvedores, utilizando a linguagem JAVA, *AspectJ* e as IDEs *Netbeans* ou Eclipse, conforme experiência dos engenheiros de implementação (desenvolvedores). Basicamente, a implementação dos sistemas segue as diretrizes apresentadas no padrão de projeto DAO, como apresentado na arquitetura da Figura 17. Conforme mencionado na Seção 4.2.1, outros padrões também são utilizados na implementação como melhoria da organização do código fonte e na otimização do sistema.

No passo 5, é iniciada a atividade de teste das unidades (artefatos), em que são avaliadas apenas as funcionalidades de integração com outros sistemas/funcionalidades, sendo testado o conjunto de entradas e saídas. Diretrizes específicas da área de Testes de software podem ser adotadas neste passo, no entanto elas não são abordadas por motivos do contexto. Ainda sobre os testes unitários, vale ressaltar, que *frameworks* de testes como, por exemplo, JUNit (JUNIT, 2009), podem ser utilizados na realização dos testes funcionais das operações (serviços, classes lógicas e componentes), porém, o mesmo não é abordado em detalhes neste trabalho, por

motivos de escopo. Além disso, vale mencionar que esse *framework* pode ser utilizado em várias IDEs de desenvolvimento, destacando-se a Eclipse e a *Netbeans*, que oferecem suporte para a geração de unidades de teste automáticas, para validação das funcionalidades.

Com a finalização dos testes funcionais e de integração dos artefatos, é realizada a inserção dos mesmos nos repositórios de software, conforme mostra o passo 6 da Figura 18. No entanto, por se tratar de uma atividade de maior complexidade e nível de detalhamento, quanto ao número de repositórios existentes e seus respectivos papéis no ambiente de execução, julga-se necessário apresentá-los em maiores detalhes, como mostra a Figura 21.

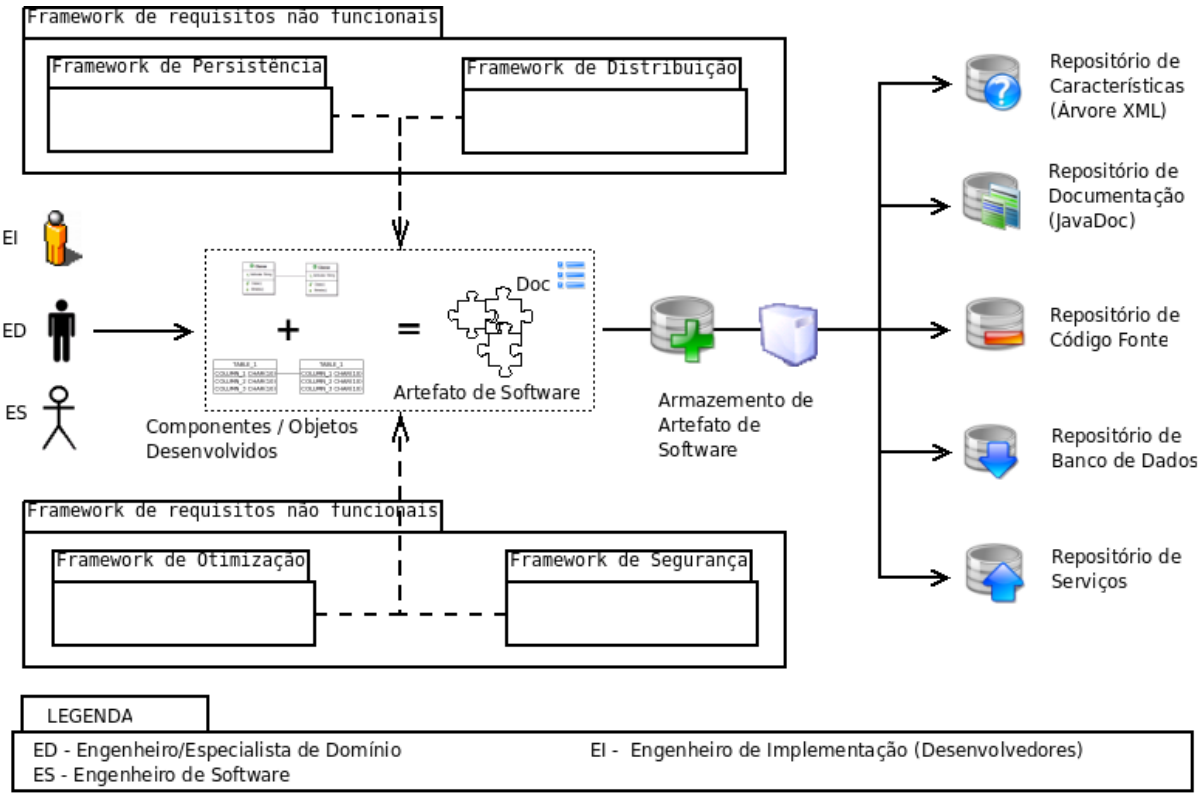

Figura 21: Detalhando os repositórios de armazenamento de informações

Conforme já mencionado neste capítulo, o desenvolvimento dos artefatos de software é orientado por uma metodologia de desenvolvimento, OO, DSBC, DOA e DOS, que tem por objetivo aumentar a reusabilidade de software e, consequentemente, auxiliar na reconfiguração da aplicação no ambiente de execução e na etapa de desenvolvimento. Ainda nessa etapa,

enfatiza-se a importância da documentação gerada nas fases de análise e implementação, pois servirão de fonte de informação para os repositórios de características, de documentação, de código fonte, de banco de dados e de serviços.

O repositório de característica é formado a partir do código fonte do artefato, onde o mesmo (código fonte) é passado por um processo automatizado, capaz de identificar a classe, seus "atributos-e-tipos"; seus "métodos-tipos" de "retorno-e-parâmetros". Essas informações são organizadas em um arquivo XML que, em seguida, é transformado em um conjunto de regras classificatórias, para serem utilizadas como fonte de recuperação da informação no processo de reconfiguração, desenvolvimento ou em tempo de execução.

O repositório de documentação é gerado desde a fase de análise, em que o especialista de domínio e o engenheiro de software, apoiados pela ferramenta de modelagem, fazem a inserção das TAGs de documentação, conforme estabelecidas no padrão JAVADOC (SUN-JAVA, 2009). Nessa etapa, são definidas as descrições para o artefato de software, para seus atributos e métodos. O mesmo procedimento é adotado pelos engenheiros de implementação na etapa de desenvolvimento para documentação do código fonte. Ao final da implementação, um processo automatizado é acionado para gerar a documentação completa do artefato e inserí-la no repositório de documentação. Isto é uma fonte de referência para engenheiros de software/desenvolvedores e mecanismos automáticos de reconfiguração sobre a funcionalidade e modo de utilização. No entanto, cabe ressaltar que a inserção de documentação deve seguir um padrão de regras, que, consequentemente, auxiliarão o processo de extração automática. Esse padrão de regras é detalhado no Capítulo 5, na etapa de mecanismos automáticos do ambiente de execução.

O repositório de código fonte tem por finalidade o armazenamento dos códigos fontes dos artefatos, tanto da fase de desenvolvimento quanto na de execução no ambiente. Nesse momento, é necessário esclarecer que esse repositório é acionado somente quando uma decisão

a respeito de qual artefato é selecionado pelos engenheiros de software e desenvolvedores e/ou invocado pelo ambiente em tempo de execução. Em ambos os casos, uma referência dos repositórios anteriores, característica e/ou documentação, é enviada aos mecanismos de invocação, ferramentas de desenvolvimento, objetos tutores ou agentes móveis, para sua criação e inserção no ambiente de execução.

O repositório de banco de dados é composto por um conjunto de *scripts* SQLs (*Structured Query Language*), capazes de gerar a estrutura de armazenamento de objetos. Esses são gerados de duas maneiras: **automática**, quando um processo automatizado recebe um artefato lógico e o *script* SQL de armazenamento é gerado, ou quando a base já existe e deseja-se recuperar o *script* equivalente por meio de reflexão computacional nos atributos e anotações dos artefatos; **manual**: quando o desenvolvedor é o responsável pela confecção do MER e equivalente estrutura de armazenamento. O uso desse repositório é opcional, pois mecanismos de persistência encontram-se disponíveis no *framework* de infraestrutura, por meio do pacote Persistência.

Finalmente, o repositório de serviços é formado pelas funcionalidades mais genéricas, que podem ser utilizadas dos diversos sistemas em domínios distintos, ou seja, não estão diretamente relacionadas com a lógica de objetos. No entanto, o mesmo procedimento quanto à utilização e reconfiguração do código fonte é adotado para os serviços.

Com a finalização do detalhamento dos repositórios, encerra-se o ciclo inicial da MSDR apresentada na Figura 18, que representa o desenvolvimento de artefatos de software, sua documentação e armazenamento das informações nos respectivos repositórios, Figura 21. Sendo assim, pode-se iniciar a etapa complementar da MDSR, que representa a busca por informações nos repositórios de informações, para escolha do melhor artefato de software que possa ser utilizado no desenvolvimento de software manual ou num processo de reconfiguração. A Figura 22 mostra os passos para essa abordagem de desenvolvimento.

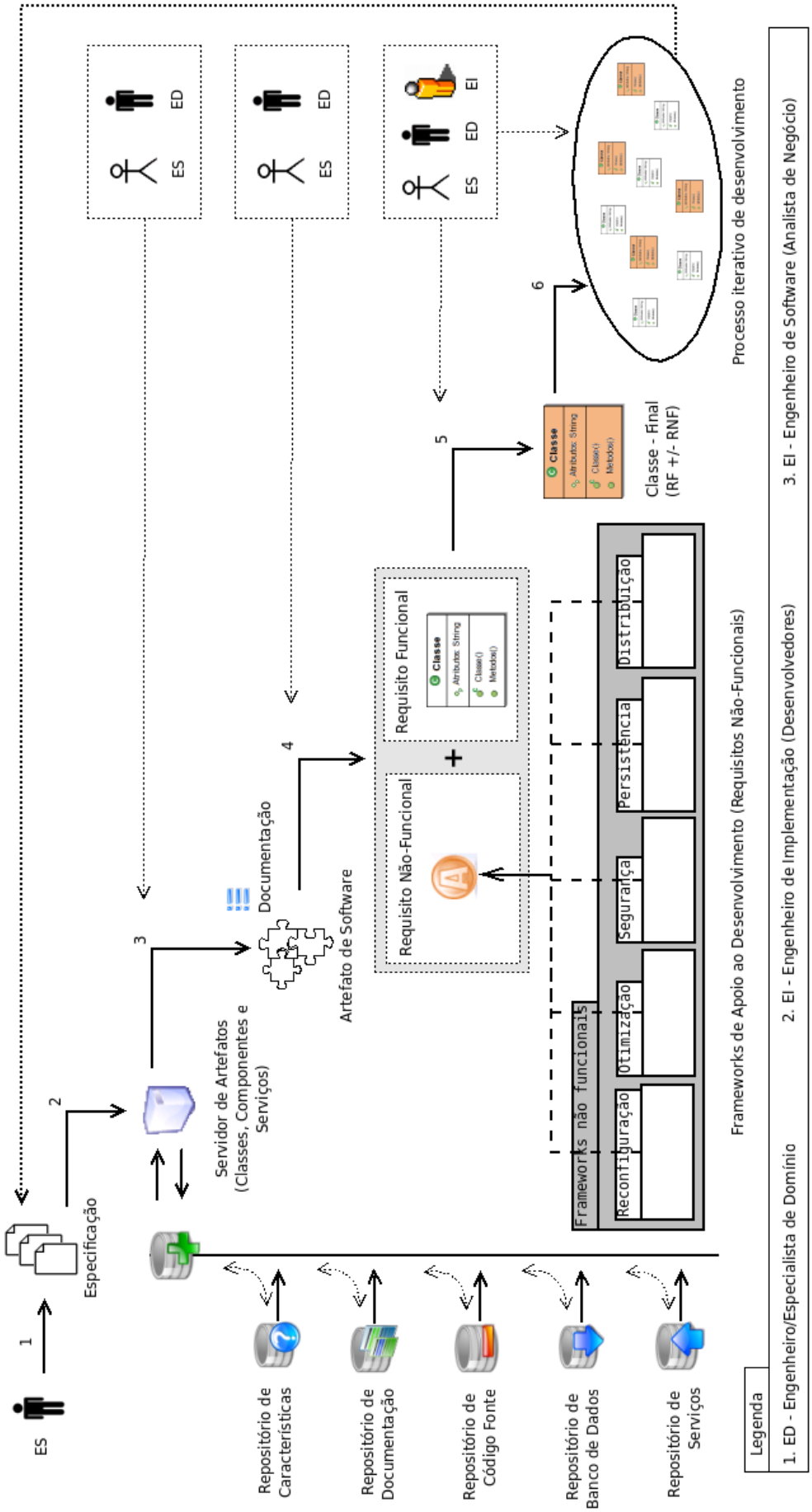

Figura 22: Metodologia de Desenvolvimento Reconfigurável (continuação)

O desenvolvimento de software no ambiente de execução reconfigurável, que é apresentado em maiores detalhes no Capítulo 5, pode ser realizado de duas maneiras:

1. **manual:** quando especialistas de domínio e engenheiros de software realizam pesquisas por meio de ferramentas de busca, para a escolha de um artefato existente no ambiente que possa ser utilizado no desenvolvimento de uma nova aplicação, com ou sem, recursos de reconfiguração; e
2. **automática:** quando mecanismos automáticos, agentes de software e ferramentas, realizam pesquisas e tomam decisões sobre quais artefatos pretendem reutilizar.

Inicialmente, passo 1, o engenheiro de software, de posse da especificação do sistema, faz a consulta nos repositórios de documentação para verificar a existência de artefatos de software que poderão ser reutilizados nos projetos atuais por ele desenvolvidos, passo 2. A busca por artefatos ocorre de maneira transparente, pois, no ambiente de execução reconfigurável, podem existir vários repositórios distribuídos em locais distintos, no entanto, as consultas realizadas oferecem a percepção como se eles fossem locais. Essa relação de transparência é viabilizada pelos servidores de artefatos, que estão presentes em vários domínios e interligados dinamicamente, e quando uma pesquisa realizada não for bem sucedida no servidor local, esta, automaticamente, é encaminhada aos demais. O servidor, passo 3, retorna os possíveis artefatos para o especialista de domínio/engenheiro de software, classificados conforme ordem de adequação à solicitação encaminhada, escala de satisfação, ou finalmente, uma exceção, caso nenhuma solução seja encontrada.

A consulta automática ocorre no ambiente de execução, envolvendo os objetos em execução e que não podem executar as solicitações vindas dos clientes. Dessa forma, os objetos-tutores, monitores destes, encaminham uma requisição, busca conforme descrito no passo 1, aos repositórios de domínio local, que podem ou não atendê-la. Em caso negativo, os

agentes móveis são disparados em busca de soluções pelo ambiente de execução e, quando encontradas, encaminham aos objetos tutores para decidir qual delas deve ser processada. Caso contrário, uma exceção é retornada para que uma resposta ao cliente seja devidamente formalizada.

Com o artefato selecionado no passo 3, independente do método executado, manual ou automático, inicia-se a associação de requisitos de infraestrutura por meio do *framework* de requisitos não-funcionais, passo 4. Esta, pode ser realizada utilizando técnicas de programação que visam associar e/ou encapsular uma funcionalidade lógica a um requisito de outra natureza (não-funcional). As principais técnicas utilizadas neste caso são a programação orientada a aspectos, a programação orientada a serviço (*Web Service*) e a injeção de dependência, ambas apresentadas no Capítulo 2.

No passo 4, o especialista de domínio, engenheiro de software e engenheiro de implementação atuam na confecção do artefato final, pois recursos de infraestrutura como reconfiguração, otimização, segurança, persistência e distribuição podem ser adicionados, conforme a especificação inicial, passo 1. Dessa forma, pode-se dizer que o artefato está pronto para ser reutilizado ou inserido no ambiente de execução, passo 5.

Finalmente, no passo 6, é iniciada a confecção da aplicação segundo um processo iterativo, ou seja, os artefatos são incorporados um a um à aplicação que se pretende desenvolver. Dessa forma, os passos 1 a 6 são repetidos a cada inserção de artefato no novo software, que se pretende desenvolver/reconfigurar, e consequentemente, permite aos engenheiros de software e desenvolvedores uma constante avaliação se aquilo que está sendo desenvolvido está em conformidade com a especificação inicial.

Com a finalização da apresentação da MDSR, pode-se dizer que ela atua no desenvolvimento de artefatos de software preparados para o desenvolvimento reconfigurável. Além disso, atua na recuperação de artefatos existentes no ambiente de execução ou

adormecidos nos repositórios, para que seja realizada a adaptação necessária em conformidade ao atendimento às novas necessidades.

Outro fator relevante dessa metodologia é o processo de desenvolvimento em linha de montagem, onde os artefatos são desenvolvidos e desprovidos de qualquer requisito de infraestrutura, devendo estes ser incorporados na fase de *deploy*, no ambiente de execução. Esse fator é representado diretamente nos passos 4 e 5 da Figura 22, nos quais os requisitos não-funcionais são incorporados ao artefato "puro", até atender os requisitos estabelecidos no passo 1. A Figura 23 mostra essa característica de desenvolvimento.

Figura 23: Desenvolvimento de artefatos em linha de montagem

Nessa linha de raciocínio, Figura 23, o objeto central é constituído apenas por requisitos funcionais, sendo que, requisitos de infraestrutura, como persistência, distribuição, segurança, otimização e reflexão, podem ser a ele agregados formando uma única unidade de software. O objeto central deve ser constituído de atributos, construtores e métodos *getter/setters*, sendo as características responsáveis por funcionalidades estruturais inseridas de maneira gradativa, conforme os requisitos não-funcionais inicialmente estabelecidos (passo 1). Esse processo pode ser conduzido de maneira automática, pelo sistema de reconfiguração no ambiente de execução instantaneamente, ou manual, pelo engenheiro de software na etapa de desenvolvimento do software.

4.3. Considerações Finais

A MDSR fornece um conjunto de diretrizes para especialistas de domínio, engenheiros de software e desenvolvedores que atuam na confecção de software reconfigurável de maneira automática e manual. Além disso, destaca-se um conjunto de ferramentas que auxiliam nas atividades por ela exigidas, acelerando o desenvolvimento/reconfiguração de software. Essas ferramentas são citadas nesta seção, pois existe uma relação de dependência na automatização das atividades apresentadas neste capítulo. Os detalhes de funcionamento e as principais etapas de padronização das informações, geradas em todo processo de desenvolvimento, são apresentados no Capítulo 5.

A principal vantagem da utilização dessa metodologia é a padronização no desenvolvimento dos artefatos de software, que somente é alcançada pela combinação de um especialista de domínio e um engenheiro de software (analista de negócio) na fase inicial de concepção. Dessa forma, pode-se dizer que essa padronização oferece vários benefícios, tais como: (i) facilidade de reconfiguração, tanto em tempo de execução quanto manual, e (ii) melhor execução das consultas por artefatos de software nos repositórios de informações entre os diferentes domínios.

Outra vantagem, em relação à padronização de desenvolvimento, é automatização que se pode empregar nas fases de desenvolvimento. Quando um artefato é desenvolvido, suas informações, nome, características (atributos) e funcionalidades (métodos), são obtidas por meio de reflexão computacional, para que, em seguida, possam ser geradas as regras de classificação, que serão utilizadas nas etapas de pesquisas de artefatos no ambiente (execução ou repositórios).

Outro fator de destaque nessa metodologia é a automatização das atividades por ela requerida, pois a intervenção humana na geração de informações e código fonte é minimizada e, sendo assim, pode-se dizer que menos inconsistências são geradas ao longo do processo de

desenvolvimento e reconfiguração dos artefatos.

Finalmente, destaca-se a técnica de reconfiguração desenvolvida, pois atua como um mecanismo comum e independente da tecnologia de distribuição escolhida pelos desenvolvedores. Dessa forma, é possível reafirmar o conceito central da metodologia, que consiste em desenvolver objetos distribuídos reconfiguráveis em linha de montagem.

A seguir, no Capítulo 5, é apresentada a estrutura do ambiente de execução reconfigurável e os subsistemas responsáveis pela automatização de algumas atividades requeridas pela MDSR.

CAPÍTULO 5

Ambiente de Execução Reconfigurável

5.1. CONSIDERAÇÕES INICIAIS

Este capítulo apresenta o Ambiente de Execução Reconfigurável (AER) e os subsistemas desenvolvidos para automatização das tarefas por ele requeridas. Este ambiente tem por objetivo atender às necessidades da Metodologia para Desenvolvimento de Software Reconfigurável (MDSR), apresentada no Capítulo 4.

A principal justificativa, para que as informações (modelos, código fonte, documentação) sejam manipuladas dessa maneira (automatizada), é a minimização de inconsistências geradas pelos seres humanos. Em algumas etapas, sua participação foi reduzida significativamente, sendo os subsistemas encarregados de gerar os artefatos de software, seguindo os critérios de padronização estabelecidos. Dessa forma, pode-se dizer que inconsistências foram minimizadas no processo de desenvolvimento e, consequentemente, na reutilização e reconfiguração dos artefatos de software, tanto de maneira automática quanto manual.

Outro benefício da padronização das informações é a potencialização quanto à

reutilização e reconfiguração no AER, pois os artefatos desenvolvidos encontram-se puros, ou seja, segmentados em requisitos funcionais e não-funcionais. Dessa forma, pode-se dizer que o processo de adaptação ocorre de maneira mais natural que os desenvolvidos sem essa segmentação.

Este capítulo está organizado da seguinte maneira: na Seção 5.2, é apresentado o ambiente reconfigurável e a Ferramenta *ReflectTools*® e, na Seção 5.3, as considerações finais.

5.2. Ambiente de Execução Reconfigurável

Nesta seção é apresentado o AER e seus subsistemas, responsáveis pela automatização das tarefas. Basicamente, ele pode ser constituído por vários núcleos de desenvolvimento, que representam as equipes de desenvolvimento de software, organizadas conforme o domínio de atuação, como mostra a Figura 24.

Cada núcleo de desenvolvimento é formado por uma equipe composta por um Especialista de domínio, um Engenheiro de Software e um Engenheiro de Implementação, que atuam no desenvolvimento de novos artefatos de software e na reconfiguração daqueles existentes para que possam ser reutilizados em outros sistemas. Esses personagens assumem um papel de grande responsabilidade no AER, pois na fase inicial da metodologia são os responsáveis por adequação e padronização dos nomes que qualificam os artefatos de software.

Sobre o aspecto técnico, pode-se observar em cada domínio a presença de um servidor de objetos. Este tem por objetivo armazenar os projetos que foram desenvolvidos no domínio local para que possam ser reutilizados por outros projetos de domínios distintos. Dessa forma, o servidor de objetos também tem a função do gerenciamento de versões dos artefatos, pois a reconfiguração dos artefatos pode ocorrer tanto de maneira manual quanto automática e, em ambos os casos, as execuções vigentes devem ser preservadas e os estados atuais de execução devem ser mantidos.

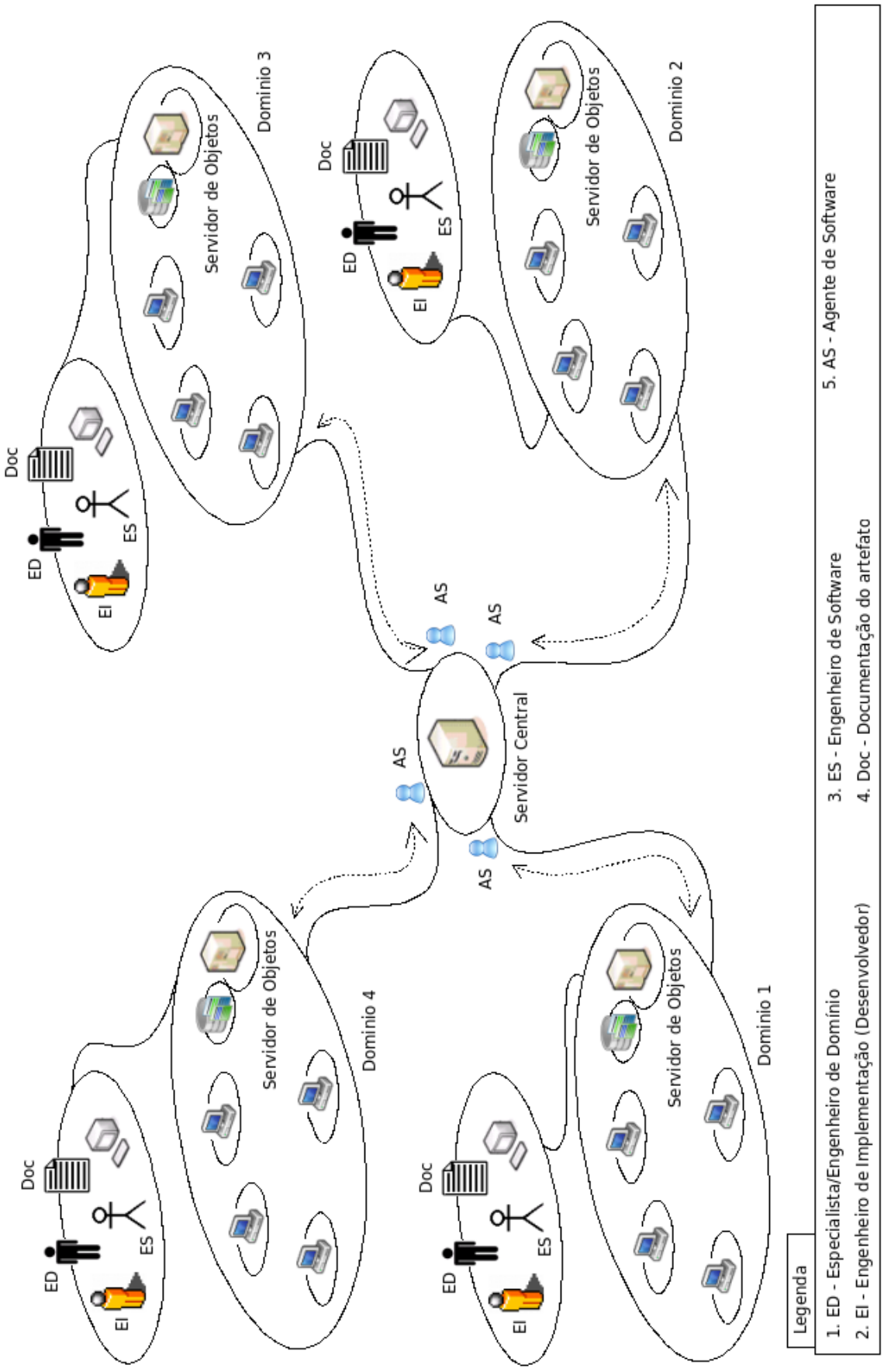

Figura 24: Ambiente de Execução Reconfigurável

Sobre a reutilização do artefato de software pode-se dizer que a mesma pode ocorrer de duas maneiras: **integral**, quando o artefato de software não sofre nenhuma alteração; **parcial**, quando alguns mecanismos de modificação de estruturas ou comportamento são aplicados para que um novo artefato seja produzido.

Ainda sobre o funcionamento do AER, pode-se observar que os núcleos estão interligados por um servidor central, que possui os agentes de software. Estes agentes são utilizados somente quando os engenheiros de software/especialistas de domínio necessitarem de artefatos de software e que não podem ser encontrados no núcleo local de desenvolvimento. Os servidores de objetos encaminham as requisições para o servidor central, que invocam os agentes para percorrerem os repositórios de informações em cada núcleo existente. Os agentes de software podem devolver uma referência, quando uma solução é encontrada ou uma exceção em caso contrário.

Quando o processo de consulta for realizado de maneira automática pelos subsistemas, o procedimento de atuação dos agentes é similar ao manual, no entanto, as soluções e exceções são lançadas também de maneira automática pelos agentes que realizaram as operações de busca no AER.

Outro fator relevante para o AER é a Ferramenta *ReflectTools*®, que permite automatizar as tarefas por ele requeridas e presentes na metodologia apresentada no Capítulo 4. A seguir, na Seção 5.2.1, é apresentada a ferramenta *ReflectTools*® e seus subsistemas.

5.2.1. A Ferramenta *ReflectTools*® e seus subsistemas

Antes de iniciar qualquer detalhamento sobre a ferramenta *ReflectTools*® e suas funcionalidades, é necessário apresentar a especificação técnica e requisitos computacionais utilizados e necessários:

- Linguagem de Programação: desenvolvida 100% com a linguagem JAVA *update* 4

ou superior;

- Sistema Operacional: Linux – distribuição *Slackware*. Porém foi projetada para executar tanto em família Linux como em *Windows* XP.
- Servidores Web: *Glassfish* v2 para hospedagem e execução dos serviços *Web* por ela requeridos;
- Servidor de Banco de Dados: *MySQL* versão 5 ou superior com versão de *driver* JDBC compatível; e
- *Frameworks* adicionais: são fornecidos juntos com a ferramenta.

A ferramenta *ReflectTools*® foi desenvolvida, neste trabalho, com principal objetivo de viabilizar as necessidades da MDSR. Dentre suas principais funcionalidades, destaca-se a padronização na geração e recuperação de informações no processo de desenvolvimento e reconfiguração de software. Para demonstrar o seu funcionamento será considerado como exemplo o desenvolvimento de um sistema para gerenciamento de uma loja virtual, que foi desenvolvido utilizando a MDSR e implementado 100% na linguagem JAVA. A ferramenta *ReflectTools*® foi projetada inicialmente para importar projetos desenvolvidos na ferramenta Eclipse com *plugin Omondo*-UML (OMONDO, 2009) e na ferramenta *Netbeans*-UML (NETBEANS-UML, 2009). Essas ferramentas foram selecionadas pela facilidade de geração automática de código, a partir dos modelos de classe (código JAVA) e MERs (código SQL).

Para melhor apresentar seu funcionamento, sua apresentação é segmentada em duas etapas: na Seção 5.2.1.1 são apresentados os procedimentos e uma visão da ferramenta *ReflectTools*® para armazenamento de informações nos repositórios de informação; na Seção 5.2.1.2 são apresentados os procedimentos para consulta de artefatos nesses repositórios e a geração automática de informações para criação dinâmica e, finalmente, na Seção 5.2.1.3 são apresentados os subsistemas responsáveis pela geração automática de informações.

5.2.1.1. *ReflectTools*® - Instanciando os repositórios de informação

Nesta etapa, o engenheiro de software e o especialista de domínio elaboram o diagrama de classes UML, conforme especificação estabelecida na MDSR, para constituição dos nomes das classes, atributos e métodos. Para cada funcionalidade existente é inserido um comentário utilizando a documentação em JAVADOC. Os demais diagramas para modelagem das funcionalidades, diagramas de caso de uso, diagramas de sequência e outros, também são confeccionados nesta etapa.

Após modelada a lógica do sistema inicia-se a fase de modelagem dos dados. Com base no diagrama de classes do passo anterior, inicia-se a modelagem de dados, seguindo os princípios das regras de mapeamento objeto-relacional (HIBERNATE, 2009). Esta etapa também foi desenvolvida na ferramenta Eclipse com *plugin Azzurri Clay* (AZZURRI, 2009). Os modelos de classe e MER-físico serão omitidos nessa fase de descrição, por motivos de escopo.

Com a definição dos modelos de classes e de dados é iniciada a fase de desenvolvimento e teste das funcionalidades. Esta etapa também é realizada em uma das ferramentas, Eclipse ou *Netbeans*, pelas facilidades e recursos de seus editores de programação, além da integração com *frameworks* de teste como JUNIT (JUNIT, 2009), por exemplo.

Em seguida, inicia-se a fase de importação do projeto desenvolvido para a ferramenta *ReflectTools*®. Para acessar as funcionalidades dessa ferramenta, o usuário pode acessar por meio do ícone, presente na barra de relógio - canto inferior direito, para que o menu de opções seja visualizado. Esse tipo de aplicação é chamada de *SystemTray* e permite que os desenvolvedores (engenheiro de software e especialista de domínio) interajam instantaneamente com suas funcionalidades, por meio de um menu de acesso. Sua principal vantagem é que se trata de uma aplicação que permanece em execução constantemente na barra de tarefa do Sistema Operacional, sem a percepção direta de seu usuário da ferramenta. A Figura 25 mostra a Ferramenta *ReflectTools*® executando em modo *SystemTray.*

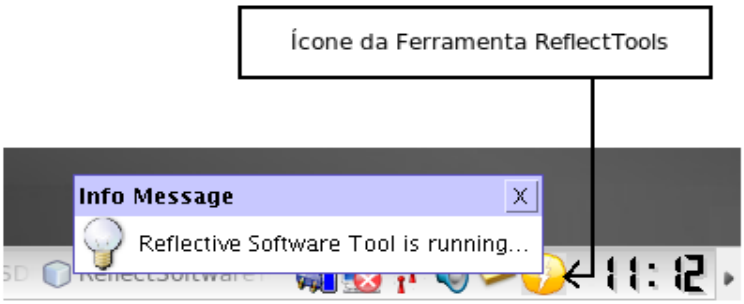

Figura 25: ReflectTools® em execução

Ao clicar sobre o ícone com o botão direito, um menu de opções é exibido. Em seguida, deve-se selecionar a opção (*Tools*) e (*Import project from Eclipse-Omondo/Netbeans-UML*) para que a interface principal seja visualizada, como mostra a Figura 26.

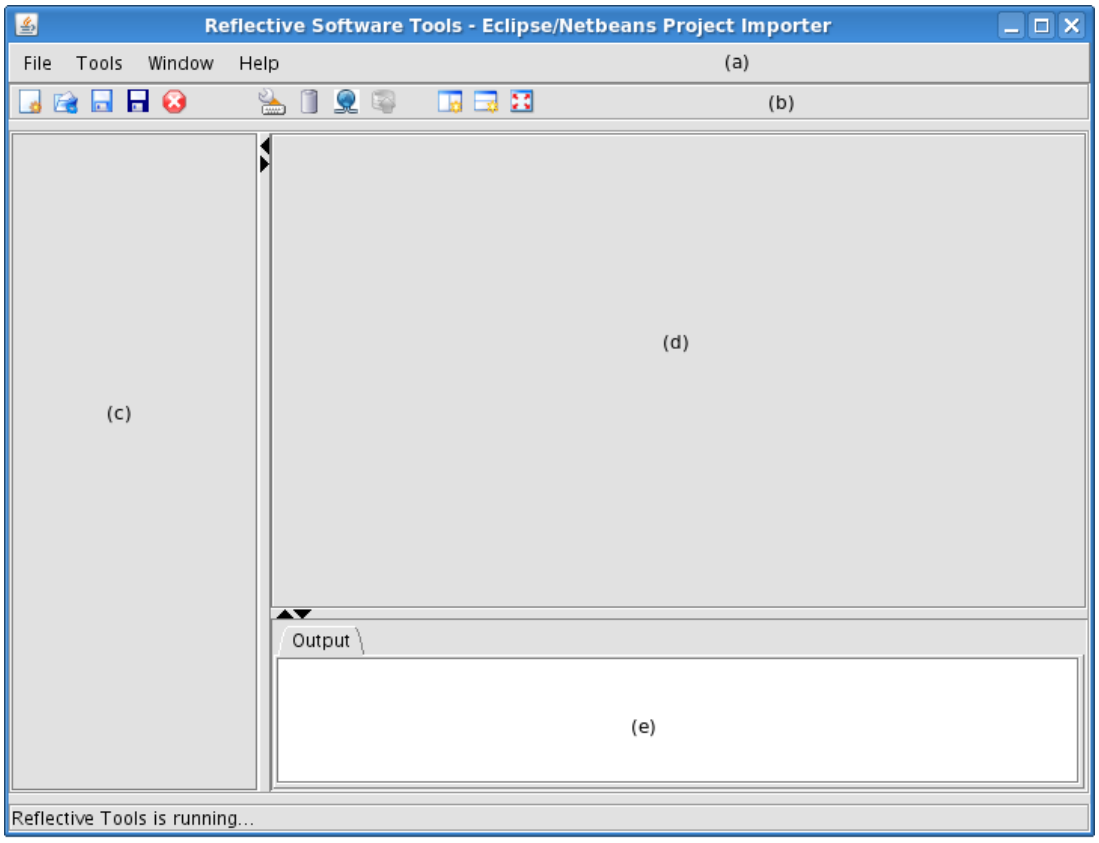

Figura 26: Tela para importar projetos Eclipse/Netbeans

Como pode-se observar na Figura 26, em (a) está presente o menu de operações e, por motivos de facilitar a usabilidade, as principais funcionalidades estão organizadas em uma barra

de atalhos em (b). A área em (c) corresponde a uma árvore lógica, onde serão mapeados os arquivos dos projetos, pastas e subpastas. O local, onde serão inseridos os editores (JAVA, SQL, HTML, outros) da ferramenta, para edição pelos desenvolvedores, está representado em (d). Finalmente, em (e) representa a área de saída, ou seja, todas as operações executadas pelo usuário na ferramenta serão apresentados nesta área. Tanto em (d) como em (e) podem ser criadas abas para que várias operações possam ser analisadas pelos desenvolvedores. Além das operações presentes no menu e barra de atalho, outras podem ser acessadas por meio de um menu flutuante, quando um projeto é criado.

Para a criação de um novo projeto, o usuário da ferramenta deve clicar no botão novo ( ) para que o *wizard* de importação projetos, Figura 27, seja exibido.

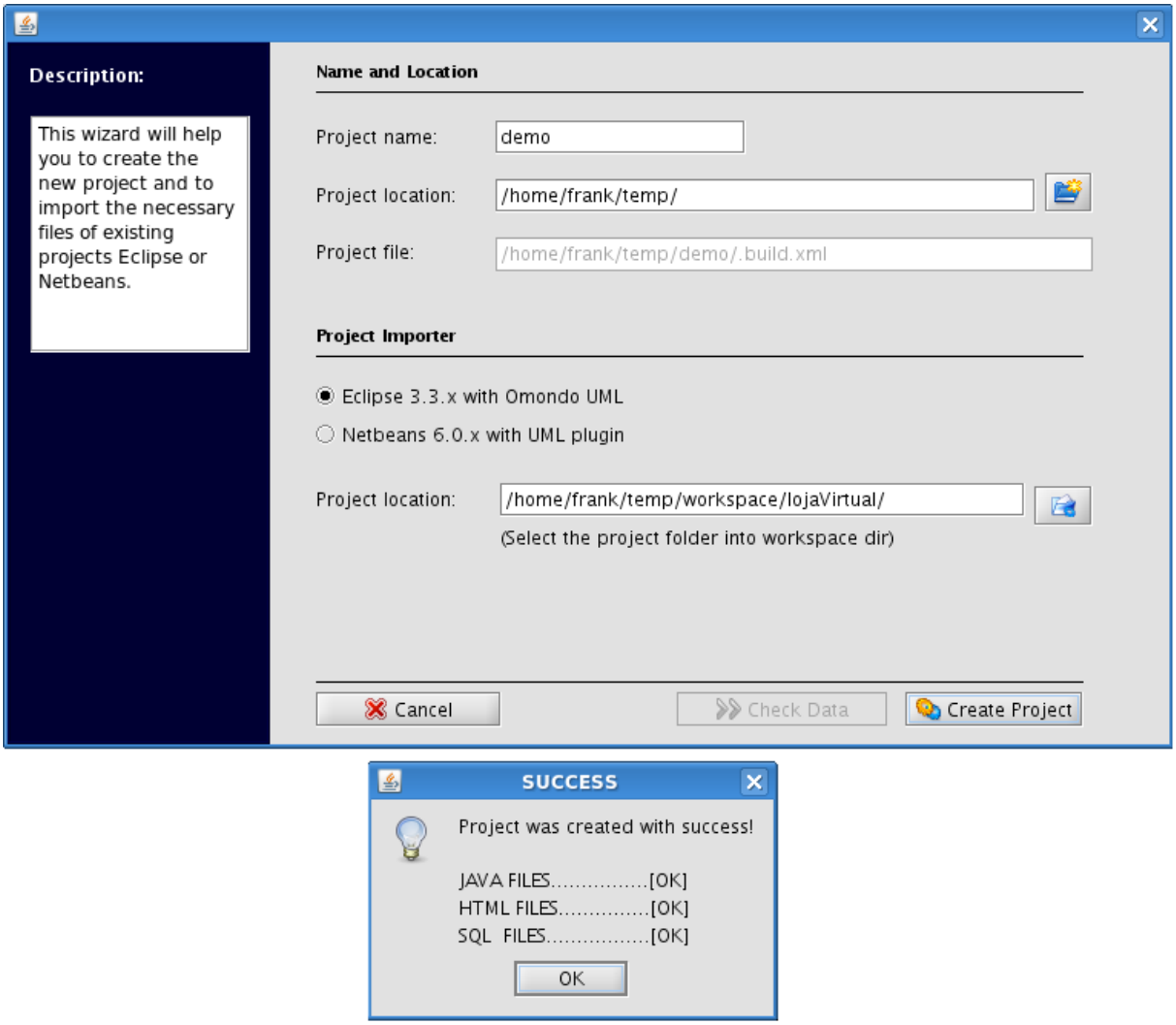

Figura 27: Wizard para importar projetos

Como pode-se observar, o usuário da ferramenta *ReflectTools*® define o nome do projeto e o local onde ele será salvo. Automaticamente, a ferramenta gerará neste local um arquivo chamado (`.build.xml`), que contém todas as informações do projeto que serão utilizadas pela ferramenta. Em seguida, o usuário deve informar também o tipo de projeto que irá importar (Eclipse ou *Netbeans*) e sua a localização. Para verificar se as informações foram inseridas corretamente, o usuário deve clicar no botão `Check data`. Em caso afirmativo, o botão `Create project` é habilitado, caso contrário, uma mensagem de erro será exibida. Para finalizar, o usuário clica no botão `Create project` e, se todas operações para criação do projeto foram executadas corretamente, a tela SUCCESS é exibida (parte inferior da Figura 27). Ao finalizar o *wizard*, tem-se a ferramenta *ReflectTools*® instanciada com um projeto para ser encaminhado aos repositórios de informação, conforme previsto na MDSR, Figura 28.

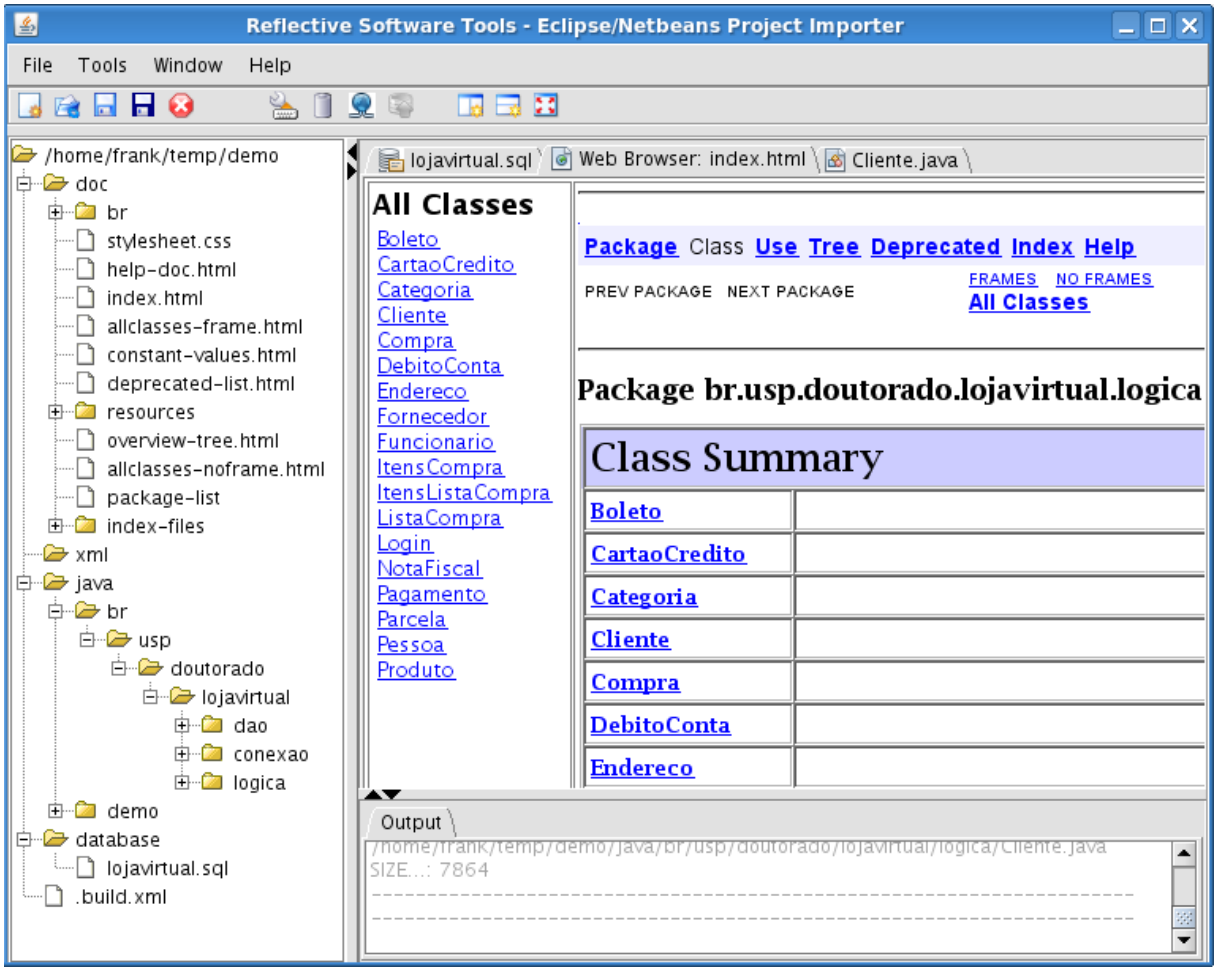

Figura 28: Ferramenta ReflectTools® com projeto em execução.

Pode-se observar que os projetos foram importados corretamente pela ferramenta *ReflectTools*. Para o projeto inicial (`demo`), foram criadas quatro pastas que podem ser observadas na árvore do projeto:

- `doc`: que contém toda documentação do projeto no padrão JAVADOC, gerada no projeto inicial com a ferramenta Eclipse. A ferramenta também permite que seja visualizada no formato HTML, como mostrado na área de edição;
- `java`: que contém todo código fonte do projeto organizado em pacotes. A Ferramenta permite que o usuário edite e compile este tipo de arquivo antes de enviá-lo para o repositório de informações;
- `xml`: representa uma pasta vazia que ainda deve ser processada antes de enviar o projeto para os repositórios. O procedimento de geração de XML e mapeamento para regras merece uma apresentação mais específica, Seção 5.2.1.3, devido à complexidade e importância para a MDSR;
- `database`: contém o *script* SQL para geração de base de dados. Caso o projeto original tenha sido desenvolvido com o recurso de anotações esta pasta permanecerá vazia.

Para gerar os arquivos XMLs, é necessário que o usuário selecione um pacote dentro da pasta (`java`), para que menu flutuante contendo as funcionalidades da ferramenta seja exibido. Ao selecionar a opção (`Map Java Files to XML Files`), um *wizard* de apoio é visualizado, como mostra a Figura 29.

Inicialmente, o usuário clica no botão (`Load Java File`) para que os arquivos fontes do pacote selecionado sejam carregados na lista de arquivos (`Source Java File`). Em seguida, o usuário pode selecionar o número de arquivos que deseja mapear ou selecionar todos na opção (`select all`), na caixa de seleção abaixo. Ao clicar em (`next`), os nomes dos arquivos

com extensão (`.xsd`) são gerados e o usuário também deve selecionar quantos arquivos deseja mapear ou selecionar todos na opção (`select all`). Após concluída a seleção deve-se clicar no botão (`Generate XML`), para que o processo de mapeamento `java` para `xsd` seja acionado. Esta etapa é realizada utilizando o compilador JAXB (JAXB, 2009), que faz o *binding* (equivalência) de cada arquivo `java` para `xsd`. Este procedimento é similar ao realizado pelos *web services* na formatação mensagens e passagem de objetos entre clientes e servidores.

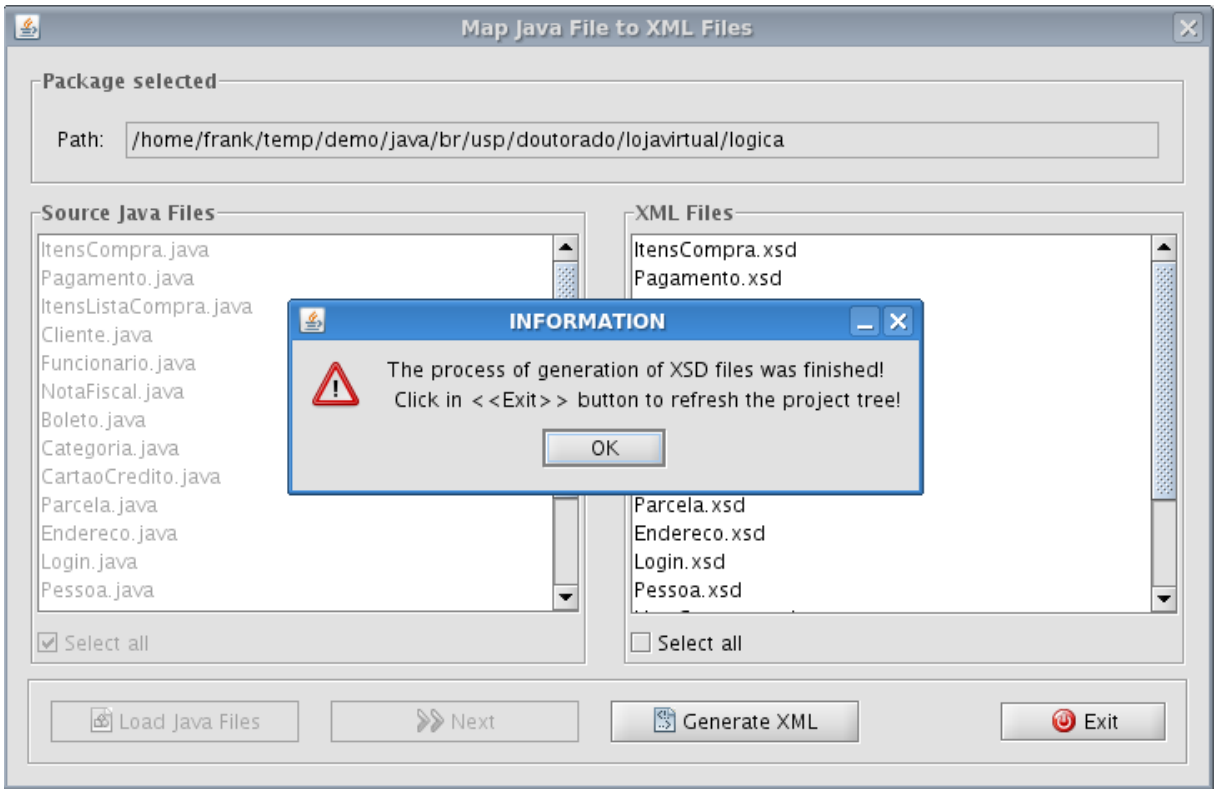

Figura 29: Processo de mapeamento JAVA para xsd

Por se tratar de um processo compilado, o resultado da operação pode ser visualizado na mensagem (INFORMATION). Caso não haja nenhum erro no código fonte em JAVA os arquivos (`.xsd`) são gerados, caso contrário, os erros devem ser corrigidos na ferramenta *ReflectTools*® e o procedimento deve ser realizado novamente. Ao clicar no botão (`Exit`), o *wizard* é fechado e a árvore de projeto é atualizada automaticamente, como mostra a Figura 30. Além disso, é possível observar o arquivo (`Cliente.xsd`) aberto na ferramenta. Este arquivo descreve, de maneira estruturada, conforme padrão JAXB, a classe JAVA mapeada, os atributos,

tipos e métodos.

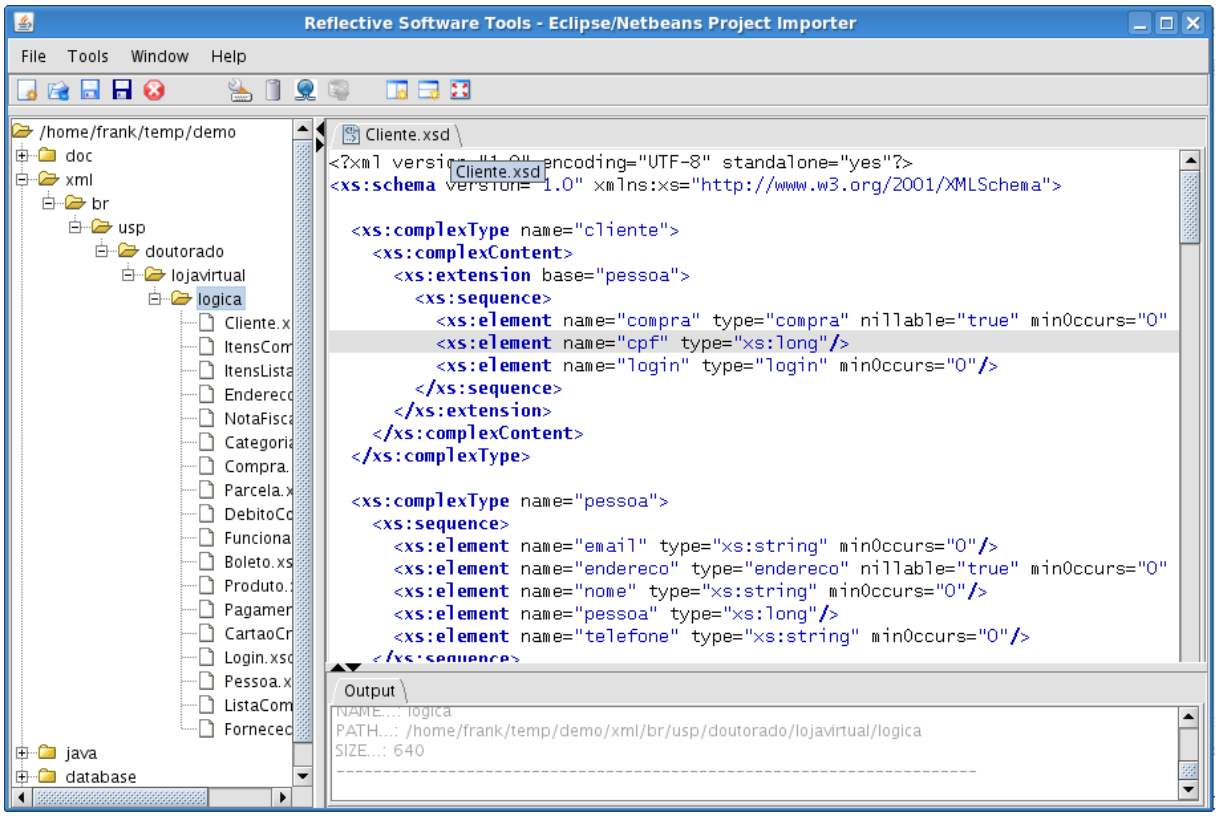

Figura 30: Projeto após realização do mapeamento JAVA para xsd

Com a finalização desta etapa, o projeto (`demo`) pode ser enviado para os repositórios de informação conforme previsto na MDSR. Para realizar esta operação, o usuário deve selecionar a pasta do projeto (`/home/frank/temp/demo`) e selecionar a opção ( ) na barra de atalhos (`Send the current project to remote software repository`), que significa enviar o projeto atual para o repositório remoto local referente ao domínio de atuação.

Para enviar o projeto para os repositórios remotos, foi necessário elaborar um procedimento que minimiza o tempo de transferência de arquivos pela rede. A solução adotada consiste em empacotar o projeto atual em um arquivo (`.zip`) e enviar para o servidor remoto em formato de serialização binária. Ao concluir esta etapa, o ícone ( ) na barra de atalho é habilitado para que o processo de extração seja iniciado. Em seguida, o usuário ao clicar sobre este atalho e inicia o procedimento de extração dos arquivos no servidor remoto. Os arquivos

são organizados em diretórios e banco de dados de maneira lógica para facilitar a recuperação. Outra questão importante referente à organização é a eventual duplicidade de nomes em projetos e, consequentemente, perda de informações dos projetos quando eles são enviados para os repositórios remotos. Dessa forma, a solução adotada foi nomear os projetos utilizando a seguinte regra:

```
Nome_do_projeto[data-hora].zip
```

Para evitar problemas com os nomes dos arquivos, os separadores de data (/) e hora(:) foram substituídos por ponto (.).

Além de enviar os projetos para os repositórios, a ferramenta permite que os bancos de dados sejam administrados remotamente. Ela permite que os usuários saibam quais bases existem nos servidores de banco de dados antes de realizar a criação de uma nova, como mostra a Figura 31.

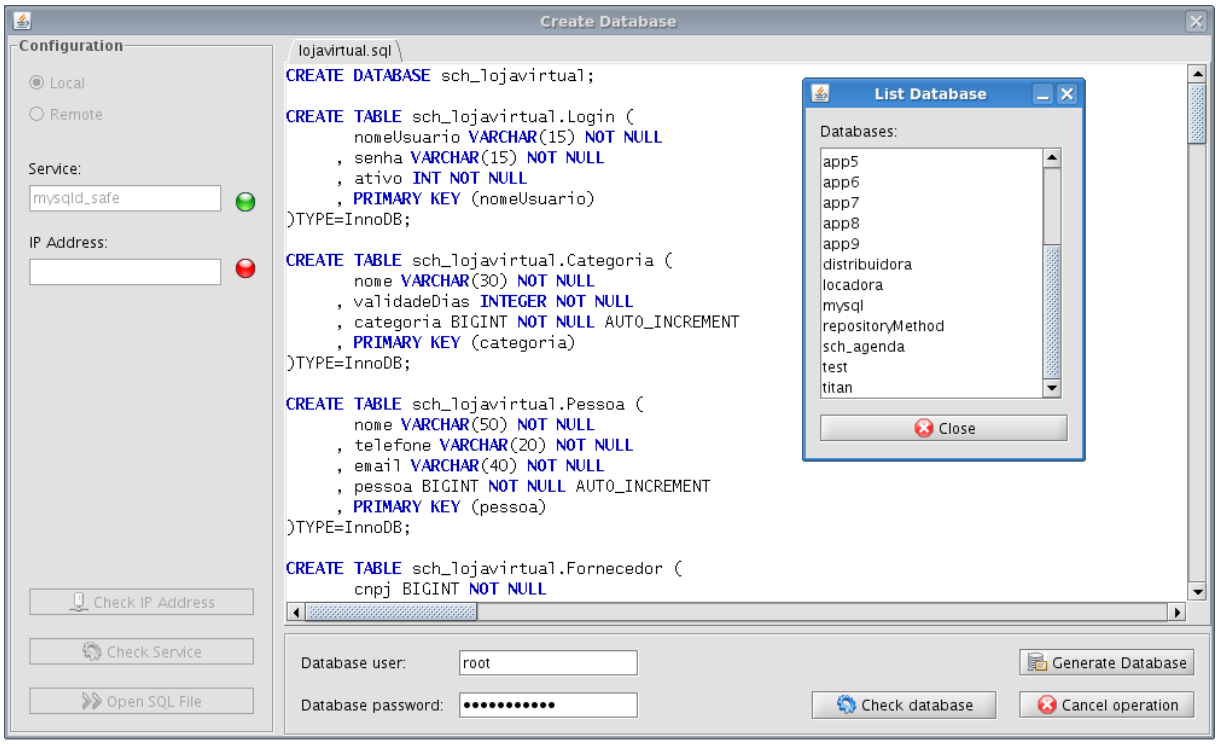

Figura 31: Ferramenta para manipulação do Banco de dados

Como pode-se observar na Figura 31, é possível verificar se o serviço do banco de dados

está ativo no Sistema Operacional. Para isso, o usuário deve clicar no botão (`Check Service`) e, se o *led* de serviço for alterado para a cor verde, o mesmo está ativo, caso contrário, uma mensagem de erro é exibida para o usuário. Essa funcionalidade pode ser executada tanto no Sistema Operacional Linux quanto *Windows*. Além disso, vale ressaltar que a operação foi realizada com o servidor de banco de dados MySQL, mas poderia ser utilizado outro servidor, devendo o usuário alterar apenas o nome do serviço e usuário do banco de dados.

Ao validar a existência do serviço, o usuário pode clicar no botão (`Open SQL File`), para que o arquivo de *script* do banco de dados seja aberto na ferramenta. Em seguida, o usuário pode listar todas as bases de dados existentes no servidor de banco de dados. O usuário deve preencher o campo senha e clicar no botão (`Check Database`), para que a janela (`List Database`) seja exibida.

Finalmente, para criar a base de dados no servidor remoto, o usuário deve clicar no botão (`Create Database`) e uma mensagem indicando o resultado da operação (sucesso ou erro) é exibida.

Este procedimento torna-se desnecessário em sistemas desenvolvidos com *frameworks* de persistência, como por exemplo, *Hibernate* (HIBERNATE, 2009) ou JPA (JPA, 2009), ambos com recurso de anotações. Com o uso dessas tecnologias, o desenvolvedor fica isento de criar manualmente a base de dados no servidor de banco de dados, pois ao executar a aplicação pela primeira vez, caso a base não exista, ela será criada automaticamente. No entanto, para seu funcionamento é necessário que seu mapeamento seja realizado corretamente na fase de desenvolvimento, indicando a relação entre os objetos e o modelo equivalente na abordagem relacional.

Para finalizar o processo de envio de informações aos repositórios de informação, tem-se a identificação de funcionalidades como serviços. Estas podem ser operações implementadas como métodos nas classes que possuem alto grau de reusabilidade, ou seja, que podem ser

reutilizadas em outros sistemas de mesmo domínio de atuação ou não. Esta atividade envolve o especialista de domínio, o engenheiro de software e os desenvolvedores, pois ambos possuem conhecimento sobre o sistema e as funcionalidades, que podem ser enquadradas como requisito não-funcional do sistema ou funcionalidade complementar.

Para demonstrar a aplicação dessa funcionalidade na ferramenta *ReflectTools®* será utilizado o projeto `demo`, que é formado por 18 classes lógicas. Analisando o projeto pode-se tomar como exemplo a classe `Cliente` e o método `validarCpf`, que pode se tornar uma funcionalidade genérica e que pode ser reutilizada em vários sistemas.

Para acessar essa operação o usuário deve selecionar o arquivo (`Cliente.java`) na árvore do projeto e clicar com o botão direito para acessar o menu flutuante. Em seguida, selecione a opção (`Send Method to Repository`), para que um *wizard* seja carregado, conforme mostra a Figura 32.

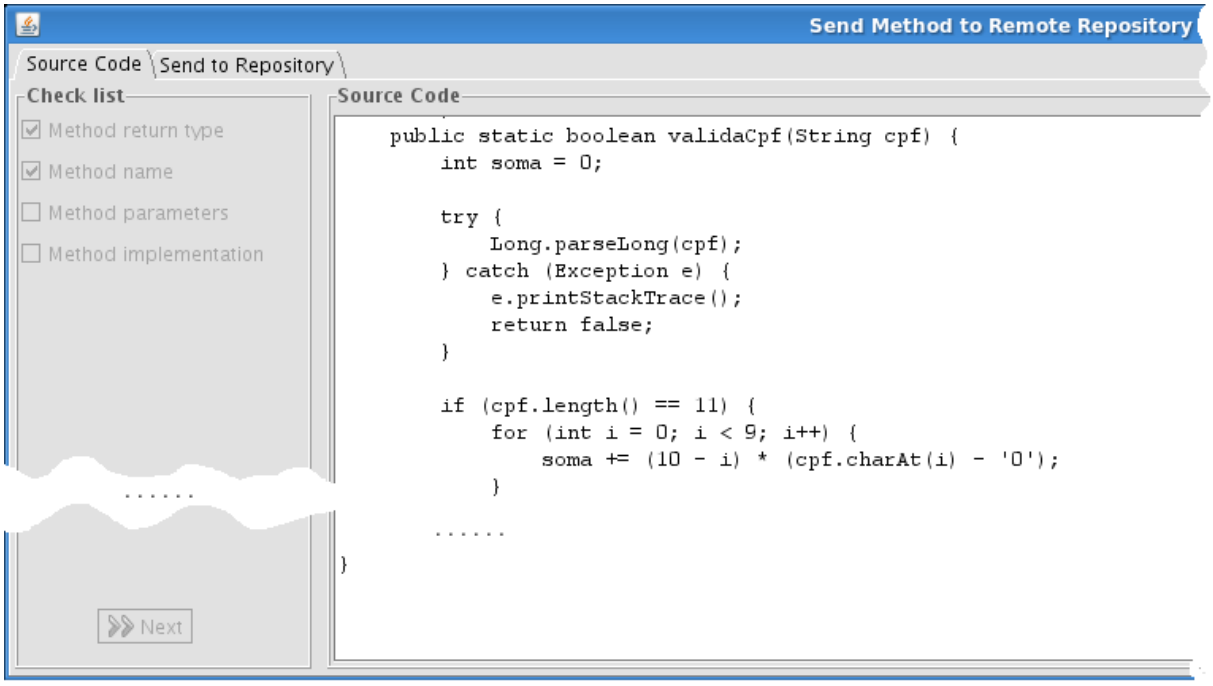

Figura 32: Enviar método para repositório (Coletando informações)

No canto superior esquerdo é possível observar duas abas (*Source Code e Send to Repository*). A primeira representa todo processo de extração de informação, em que o usuário

deverá preencher todas as opções do *Check list* (*Method return type, Method name, Method parameters e Method implementation*). Para isso, o usuário deve selecionar a área onde está presente o código fonte e selecionar a parte desejada, conforme o *Check list*. Em seguida, deve clicar com o botão direito sobre a seleção e escolher a opção equivalente à seleção. Ao realizar essa operação, a opção referente ao *Check list* será marcada como concluída (selecionada).

O procedimento descrito deve ser realizado até que todas as opções estejam marcadas e o botão (`Next`) no canto inferior deste *wizard* seja habilitado. Ao clicar sobre este botão a aba (*Send to Repository*) é ativada e o especialista de domínio deve preencher apenas o campo referente a informação semântica da funcionalidade, como mostra a Figura 33.

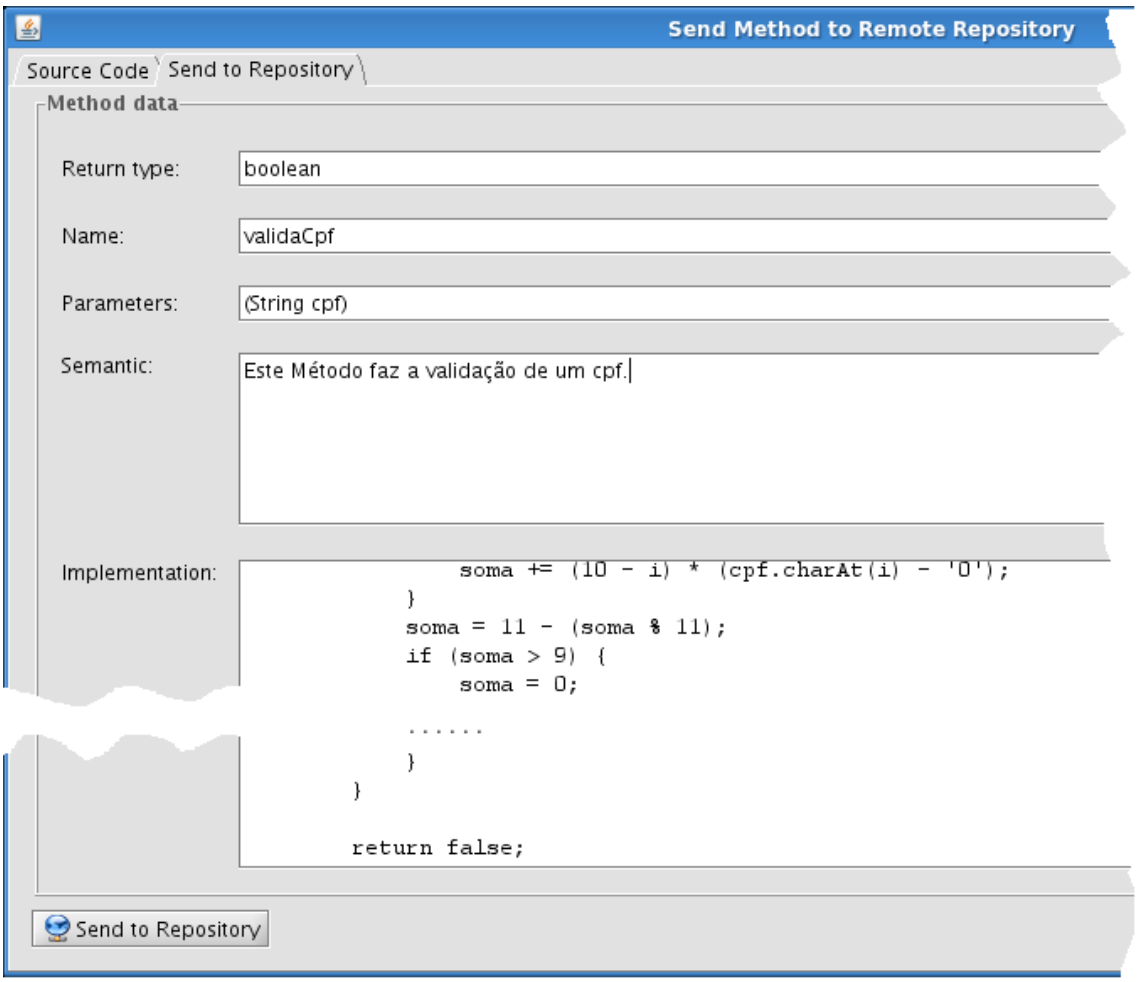

Figura 33: Enviar método para repositório (Inserindo informações nos repositórios remotos)

Ao inserir as informações semânticas, neste caso, meramente ilustrativas, o usuário pode

clicar no botão (`Send To Repository`), para enviar as informações para um repositório de métodos. Este está organizado conforme diagrama, representado na Figura 34.

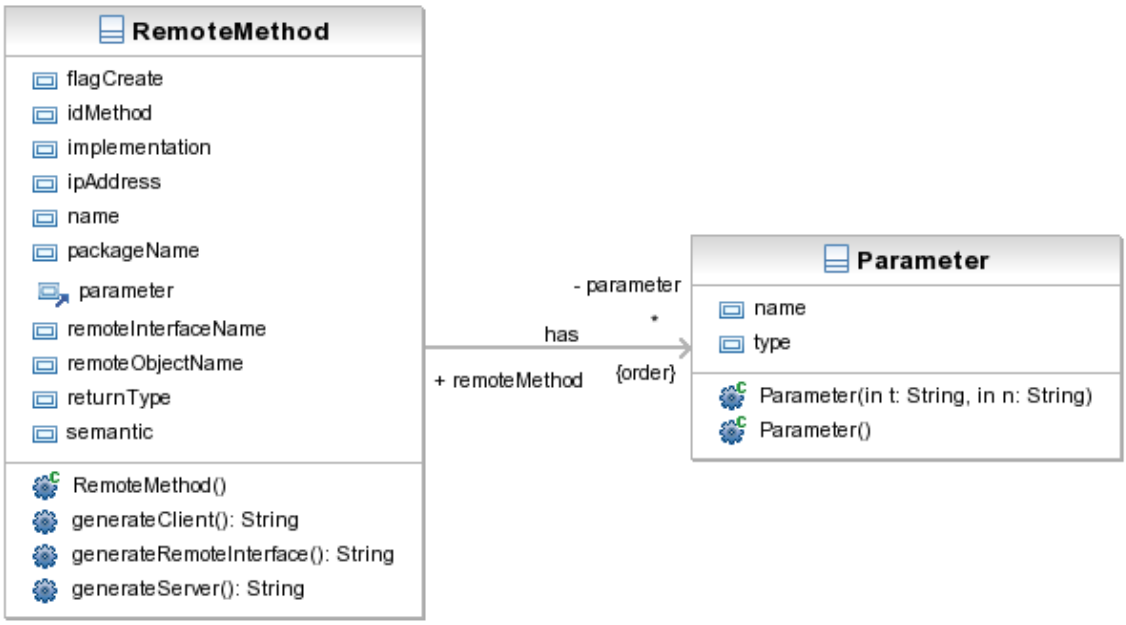

Figura 34: Modelo de classe para repositório de métodos

Como pode-se observar no modelo de classe, um `RemoteMethod` possui vários `Parameters` e pelo *wizard*, apresentado na Figura 33, existe apenas um campo que contém os parâmetros de um método. Antes de realizar a inserção, o modelo apresentado na Figura 34 é instanciado e preenchido corretamente, ou seja, caso o método a ser inserido tenha mais de um parâmetro, este deve ser decomposto em um conjunto de parâmetros e associado ao método remoto que está sendo inserido. As demais informações presentes no modelo são preenchidas automaticamente pelo sistema servidor, onde a aplicação se encontra hospedada.

Sobre a implementação desse subsistema (modelo representado na Figura 34), pode-se dizer que está sendo disponibilizado como um serviço instalado no servidor local de cada domínio de atuação, conforme previsto na MDSR. Este serviço recebe como parâmetro apenas um objeto da classe (`RemoteMethod`) e o serviço faz a persistência dos dados na base de dados. A implementação relativa à persistência de dados não é abordada nesta seção pela não

relevância ao conteúdo aqui apresentado.

Para os clientes, com a ferramenta *ReflectTools*® instalada em cada estação de trabalho, é disponibilizado apenas o WSDL do serviço, que é transformado em um conjunto de objetos *Stubs*. Estes representam um conjunto de objetos internos responsáveis pela comunicação remota entre a ferramenta *ReflectTools*® e o Serviço Web; e pelo fornecimento de uma versão binária dos objetos presentes no modelo apresentado na Figura 34, para que possam ser instanciados e transmitidos para o servidor remoto, onde são inseridos na base de dados equivalente.

As informações geradas nesta etapa, métodos remotos, e no envio de projeto de software para os repositórios constituem uma base de conhecimento ampla que pode ser utilizada no desenvolvimento de novos artefatos e na reconfiguração dos existentes. A seguir, na Seção 5.2.1.2 são apresentados os aspectos referentes às consultas e a geração automática de artefato (objetos remotos) em tempo de execução.

5.2.1.2. *ReflectTools*® - Consultando os repositórios de informação e gerando objetos remotos de maneira automática

Nesta seção é apresentado um procedimento de consulta "exata" na base de dados para os métodos remotos. Este tipo de consulta não prevê a utilização de sinônimos ou palavras correlatas que representam um significado semelhante. Dessa forma, julga-se necessário a presença do especialista de domínio e do engenheiro de software para especificar as características dos artefatos a serem pesquisados e avaliar os resultados obtidos com a execução da consulta, respectivamente. Para realizar uma consulta, a ferramenta disponibiliza um *wizard* aos usuários, como mostra a Figura 35.

Neste *wizard* existem duas áreas (`Template` e `Result`), que representam a entrada de dados para a consulta e o resultado obtido, respectivamente. Na área `Template`, o usuário

preenche as informações, conforme os campos de entrada, sendo todos de preenchimento obrigatório. Para minimizar problemas com a entrada de informações, dois assistentes de preenchimento (`Return Type` e `Parameters`) são disponibilizados. No primeiro, o usuário apenas seleciona o tipo que será adicionado à tela principal. No segundo, o usuário pode adicionar vários parâmetros e tipos conforme necessário. Os demais campos são preenchidos manualmente conforme interesse de busca.

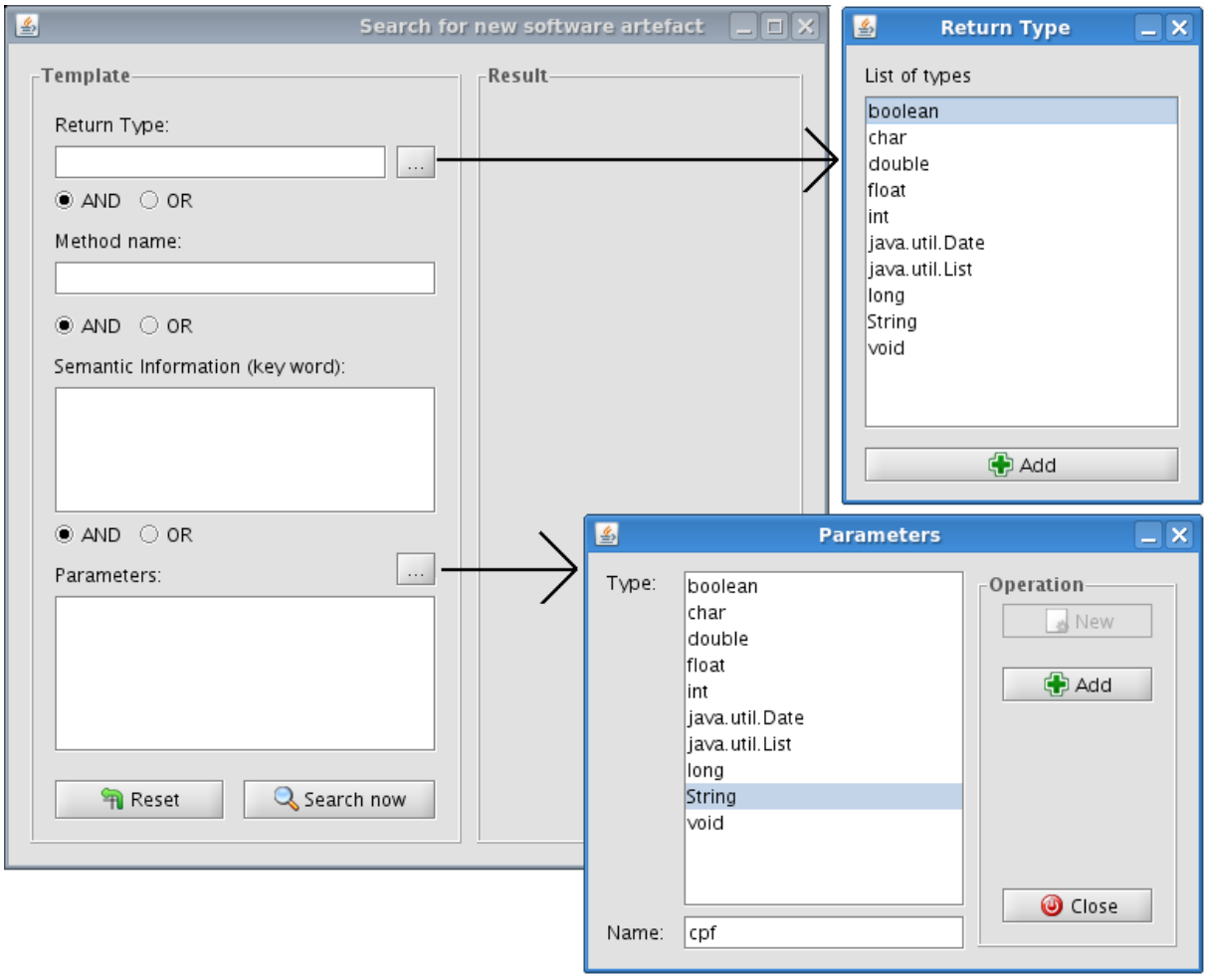

Figura 35: Wizard de consulta de artefatos remotos

Ao finalizar o preenchimento das informações o usuário clica no botão (`Search now`), para que a consulta seja realizada. Caso nenhum artefato seja encontrado, uma mensagem de aviso é exibida ao usuário solicitando a modificação dos critérios de busca ou parâmetros, caso contrário, um ou mais artefatos são exibidos na área (`Result`), como mostra a Figura 36.

Para facilitar o entendimento do processo de consulta, os dados de entrada referem-se ao método remoto cadastrado, apresentado na Figura 33. O usuário preenche os dados conforme apresentado na aba (`Template`) e solicita a pesquisa pelo artefato no repositório de código remoto.

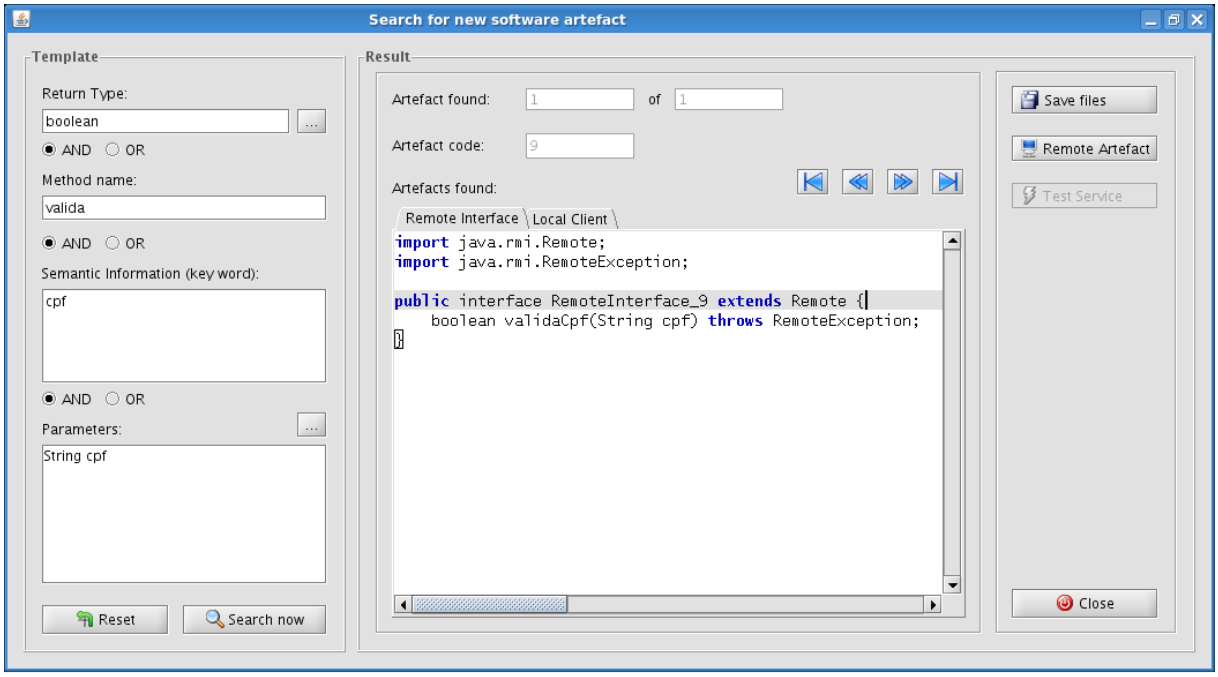

Figura 36: Resultado da consulta no repositório de método

Na parte superior da aba (`Result`) é possível observar algumas informações: número de artefato encontrado, seu código na base de dados remota e as setas navegacionais, que permite o usuário navegar pelos artefatos resultantes da consulta para verificar a melhor opção (solução). Além disso, vale destacar as abas (`Remote Interface` e `Local Client`) que são as interfaces de comunicação remota e a aplicação cliente que irá utilizar a funcionalidade do método remoto presente no servidor de domínio.

Sobre os arquivos em código fonte, pode-se dizer que eles (Servidor Remoto, Interface Remota e Cliente) foram gerados automaticamente pela ferramenta *ReflectTools®*. O Servidor Remoto contém a funcionalidade (`validarCpf`) anteriormente inserida nos repositórios (Figura 32) e, ao ser invocado pela ferramenta *ReflectTools®*, é instalado no servidor de cada domínio.

A Interface Remota representa o mapeamento direto da funcionalidade presente no Servidor Remoto, que será utilizada pelo Cliente.

O arquivos mencionados (Servidor Remoto, Interface Remota e Cliente) estão organizados no formato de um *Template* e apenas a funcionalidade selecionada pelo usuário da ferramenta *ReflectTools*®, no passo anterior – Figura 36, é injetada na estrutura dos arquivos, constituindo uma aplicação cliente-servidor por meio de chamada de métodos remotos.

Após selecionar o artefato remoto que melhor atende às necessidades, o usuário da ferramenta *ReflectTools*® deve executar dois passos antes de tentar executar a aplicação:

1. salvar a aplicação cliente: por padrão e para evitar conflitos com nomes, a ferramenta se encarrega de gerenciar os nomes e diretórios, o usuário deve apenas clicar no botão (`Save file`);
2. verificar se o serviço remoto está em execução: o usuário deve clicar no botão (`Remote Artefact`). Neste caso, a ferramenta *ReflectTools*® verifica, inicialmente, se o serviço (`rmiregistry`) está ativo ou não (Esta tarefa é executada por um módulo específico capaz de detectar os serviços em execução no Sistema Operacional). Em caso afirmativo, ela verifica se o método remoto selecionado pelo usuário encontra-se registrado no Sistema Operacional e, em caso afirmativo, uma referência é encaminhada para a aplicação cliente. Caso contrário, ela executa um processo automático para criação, compilação e execução do método remoto.

Após a execução desses passos, o usuário pode testar a comunicação remota entre os artefatos cliente-servidor clicando no botão (`Test Service`). A Figura 37 mostra o *wizard* que apoia a compilação e execução dos artefatos.

Inicialmente, o usuário deve compilar os arquivos fontes da aplicação cliente, conforme indicado nos campos (`Remote Interface` e `Client`). Caso algum erro no código fonte seja

encontrado nesta etapa, as mensagens indicando o ocorrido serão apresentadas na área (`Output messages`). Caso contrário, nenhum erro encontrado, uma mensagem é exibida ao usuário informando que o processo de comunicação remota pode ser inicializado. Ao clicar no botão (`Run prompt`), a aplicação cliente realiza a comunicação com o objeto remoto e utiliza o método remoto (`validaCpf`) como se fosse local. A resposta de execução pode ser observada na área (`Output messages`).

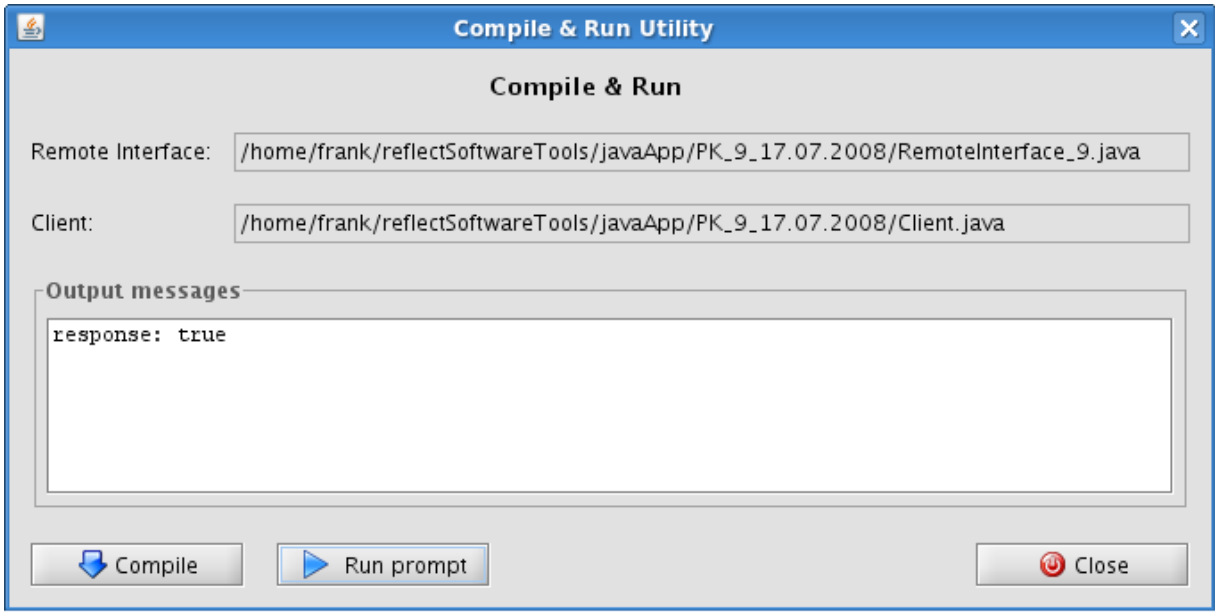

Figura 37: Wizard de compilação e execução

A seguir, na Seção 5.2.1.3, são apresentados alguns os subsistemas da ferramenta *ReflectTools*® que fazem parte das funcionalidades apresentadas nesta seção.

5.2.1.3. Subsistemas

Nesta seção são apresentados os subsistemas que foram desenvolvidos para implementação da ferramenta *ReflectTools*®. Escolheu-se apresentá-los de maneira separada do escopo da ferramenta, pela relevância e importância de cada um deles na execução dos processos automatizados, apresentados na Seção 5.2.1.2.

Os subsistemas que são apresentados nesta seção oferecem suporte para (1)reflexão de

objetos com gerenciamento de informações técnicas (atributos e métodos) e lógicas (dados no momento específico de execução), (2)mapeamento entre objetos JAVA em XML e vice-versa, mapeamento de XML para regras, e finalmente, a (3)reconfiguração dos objetos em tempo de execução sem interrupção de servidores de aplicação (remotos ou *Web*).

O subsistema responsável pelo gerenciamento de informação consiste basicamente em analisar os objetos instanciados em tempo de execução e salvar as informações neles contidas em um arquivo (`.xml`), para que possa ser restaurado no processo de reconfiguração. Dessa forma, oferece garantias de que as execuções vigentes são preservadas e as novas serão atendidas. A Figura 38 mostra o modelo resumido desse subsistema.

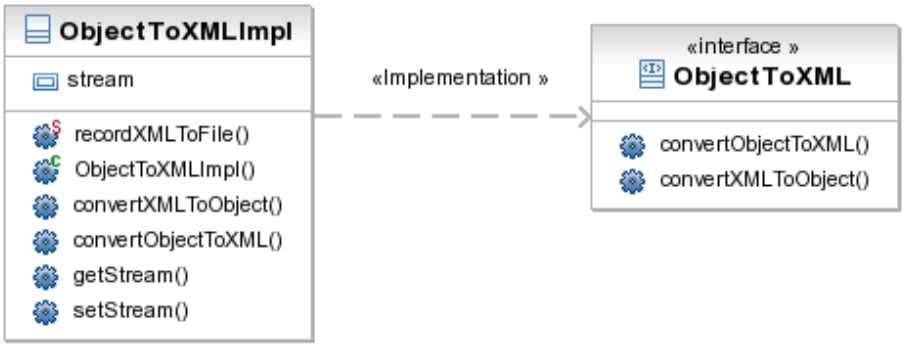

Figura 38: Subsistema de mapeamento de objetos

Basicamente, este subsistema oferece duas funcionalidades à ferramenta *ReflectTools*® (`convertObjectToXML` e `convertXMLToObject`). Estas têm por objetivo, respectivamente, de salvar o estado dos objetos em um arquivo (`.xml`) e realizar o caminho inverso, reconstituindo com as características originais.

Conforme mencionado no Capítulo 4, a padronização de informações, tanto no desenvolvimento de software quanto na recuperação de informações, é um item essencial. Além disso, a minimização da participação humana nesses processos também é considerada, devido a incertezas linguísticas que podem ser geradas. Dessa forma, foi elaborado um subsistema de geração de regras de classificação de objetos para ser utilizado como mecanismo de busca de

artefatos de software em repositórios.

Para mostrar a estrutura do subsistema de geração de regras, deve-se considerar um projeto JAVA composto por um pacote, ou mais, que contém as classes lógicas do sistema, as quais serão utilizadas na elaboração das regras de classificação. Classes auxiliares que implementam requisitos não-funcionais como persistência, por exemplo, não são consideradas. A ferramenta *ReflectTools*® implementa um metamodelo que será instanciado a cada projeto que se pretende gerar as regras de classificação, conforme mostra a Figura 39.

Este metamodelo tem como referência central a classe "`Project`", que possui uma relação (um – para muitos) com a classe "`Clazz`". De maneira lógica pode-se dizer que esta pertence a um pacote "`Package`", pode ou não ter uma classe pai (por meio da auto associação [um – para muitos]), pode implementar uma ou mais interfaces "`Interface`" e, finalmente, cada classe "`Clazz`" possui um conjunto (um – para muitos) de características "`Attributes`" e comportamentos "`Method`". Cada característica possui um tipo de dados que a qualifica e, cada método, possui um tipo de dados de retorno e seus parâmetros.

Destaca-se ainda nesse metamodelo a classe "`ReflectManager`", que é responsável por realizar a extração de características do sistema. Quando um projeto é exportado para os repositórios, essa classe é encarregada de compilar todo código fonte e, a partir do código compilado (`.class`), gerar o arquivo XML, estruturado com o mesmo nome da classe e estrutura de pacote para facilitar sua organização. Essa operação é obrigatoriamente realizada, pois a reflexão é realizada em código compilado. A ferramenta *ReflectTools*® realiza esta operação de maneira automática, sem a percepção do usuário.

Quando o desenvolvedor, na ferramenta *ReflectTools*®, exportar o projeto para o repositórios de informações, o metamodelo - Figura 39, é instanciado com todas as classes lógicas do sistema e as regras de classificação são geradas. No entanto, essa atividade pode ser detalhada em duas etapas:

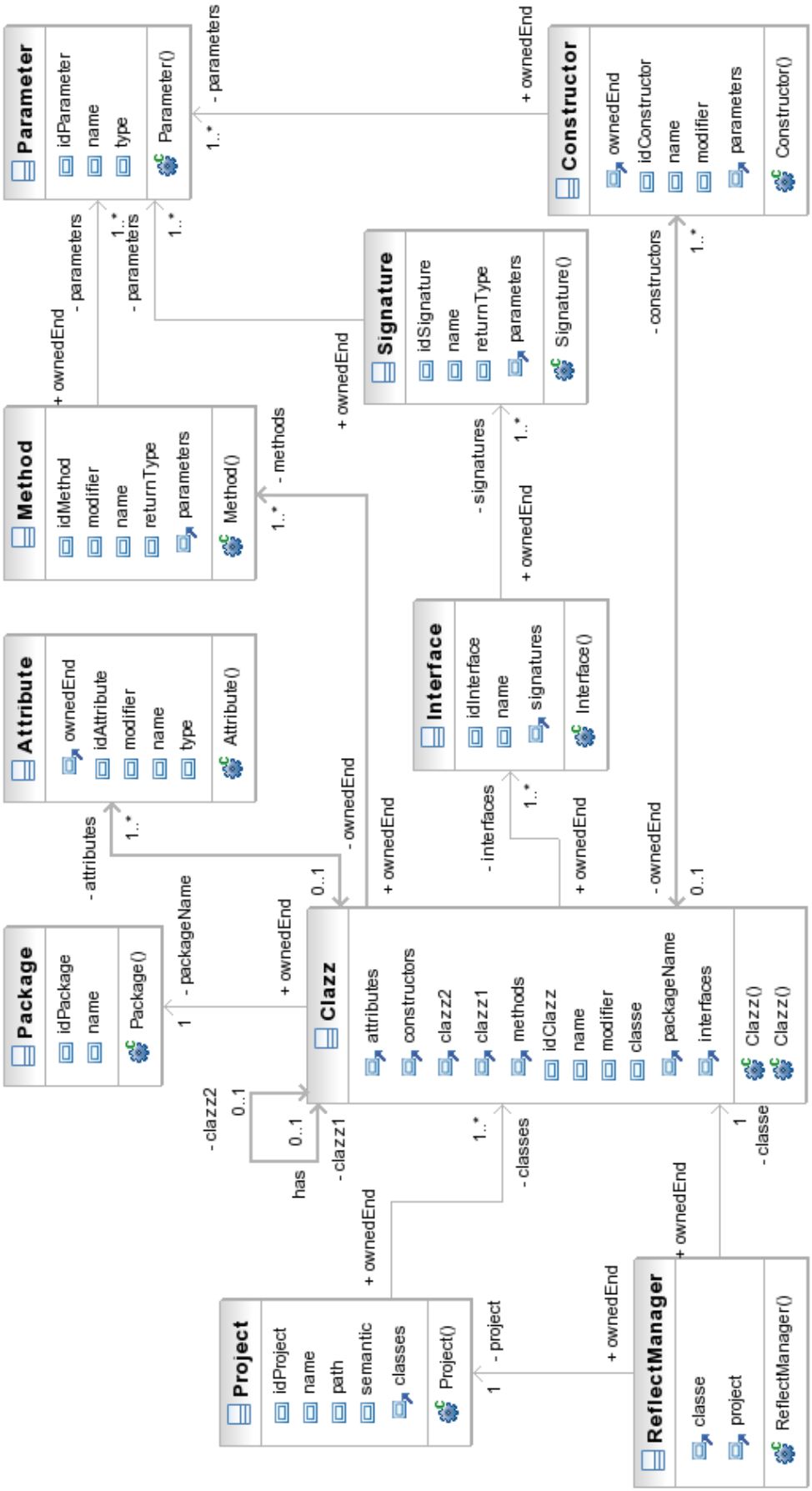

Figura 39: Metamodelo para geração de regras de classificação

(1) o metamodelo é instanciado com o projeto que se pretende exportar e o arquivo XML contendo, de maneira estruturada, as informações das classes lógicas do sistema também é gerado;

(2) baseado nesse arquivo, passo 1, as regras são geradas e formatadas num padrão estabelecido pela ferramenta *ReflectTools*®. Inicialmente, são descritas as informações referentes às características (atributos) e, posteriormente, às funcionalidades (métodos).

O arquivo de regras gerado está padronizado conforme os critérios estabelecidos pelo *framework* DROOLS (PROCTOR et al., 2009). A Figura 40 mostra um modelo de arquivo XML e seu equivalente mapeamento para regras de classificação.

XML (resumido)

```
<clazz>
<name> Nome-da-Classe </name>
      <atributo1> NAME </atributo1>
            <id> 1 </id>
            <modifier> private </modifier>
            <type> String </type>
      ....
      <construtor> .... </construtor>
      ....
      <Method>
            <idMethod>0</idMethod>
            <modifier>public</modifier>
            <name>getCpf</name>
            <returnType>long</returnType>
            <parameters/>
      </Method>
      ....
</clazz>
```

Regras(resumido)

```
rule "domínio-nome_do_sistema-nome_da_classe"
    when
        template : Template(clazz.atribute1=="NAME" &&
                                        clazz.atribute1.type=="String")
        ....
    then
        template.setObject(new Object());
end
```

Figura 40: Mapeamento de regras para xml (resumido)

Conforme estabelecido na MSDR, as regras geradas nesta etapa são armazenadas num

repositório de regras para serem utilizadas em pesquisas futuras na busca de informações de objetos armazenados nos repositórios de informação. Esse repositório de regras é representado em um conjunto de diretórios organizados pelo domínio e/ou subdomínio, nome do sistema, data de criação e, finalmente, pelos arquivos de regras com extensão (`.drl`). A Figura 41 mostra a estrutura lógica do repositório de regras.

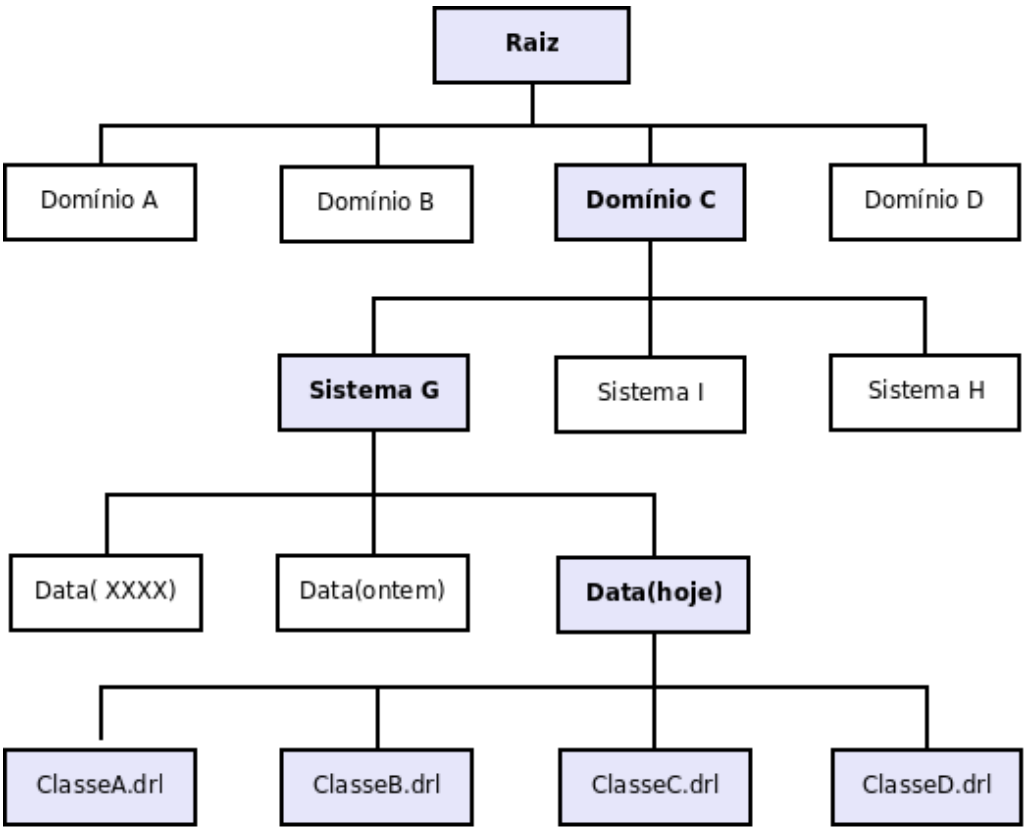

Figura 41: Estrutura do repositório de regras

A estrutura apresentada, Figura 41, é utilizada pela ferramenta *ReflectTools*® e por agentes externos, na busca de informações de artefatos de software, no processo de reconfiguração, tanto na fase de desenvolvimento quanto em tempo de execução.

Conforme apresentado na Figura 41, existem vários domínios presentes no desenvolvimento de software reconfigurável e a busca por informações pode ocorrer de maneira local, dentro do domínio de atuação, ou distribuída, em diferentes domínios, pelos agentes de software. Independente do tipo de pesquisa (local ou distribuída) que se pretende realizar, deve

existir um “mecanismo universal” que permita integrar o sistema de classificação aos artefatos de software que se pretende encontrar.

O sistema de consulta por regras de classificação foi implementado por meio de serviço *Web*, onde o usuário preenche um formulário com as informações da consulta desejada. Em seguida, essas informações são transformadas em regras de seleção e encaminhadas ao repositório de informações para serem processadas.

O servidor recebe essas informações e faz a inferência das informações vindas do solicitante com as existentes no repositório de informações, apresentado na Figura 41, para verificar a presença de um objeto capaz de atender à solicitação requisitada. O resultado dessa operação pode ocorrer de três maneiras:

1. o sistema pode localizar classes/objetos existentes em memória e registrados no sistema operacional e uma referência para o solicitante é enviada;
2. o sistema pode localizar classes/objetos “adormecidos” no repositório de código fonte, que são compilados, registrados no sistema operacional e referenciados de maneira dinâmica para o solicitante; e
3. o sistema não localiza nenhum objeto capaz de atender às necessidades apresentadas e, sendo assim, uma exceção é lançada ao solicitante.

Em ambas as maneiras citadas, são utilizados os conceitos de chamadas remotas de objetos, conforme apresentado na Seção 5.2.1.2. Ao realizar a inferência nesta base, um valor entre uma escala (0-100) é retornado e inserido em um vetor de classificação. Posteriormente, são selecionadas as regras de maior grau de confiabilidade e retornadas ao solicitante para escolha do melhor objeto. O retorno de informações para o solicitante pode ocorrer de duas maneiras: para o engenheiro de software e/ou para um sistema em execução.

Na primeira, detecta-se a presença do ser humano e sua capacidade de raciocínio, com

base nas informações selecionadas e classificadas, podendo analisar e escolher a melhor alternativa para sua necessidade. Pode-se dizer que esse tipo de situação irá ocorrer com maior frequência na fase de desenvolvimento de software, sendo assim, a reutilização será realizada utilizando o código fonte.

A segunda, trata de um processo realizado de maneira automática, em que um agente de software faz a escolha da melhor regra para o sistema em tempo de execução. Dessa forma, esse tipo de seleção é o oposto ao anterior, trabalhando apenas com a versão binária dos objetos, mesmo que estes estejam em repositório de código, pois eles serão compilados e ativados em memória.

Finalmente, é apresentado o subsistema responsável pela reconfiguração dos objetos em tempo de execução. Para facilitar sua apresentação é utilizado como exemplo um sistema com chamada de métodos remotos. O subsistema foi desenvolvido e testado para atuar em vários tipos de aplicações, tais como:

- **locais:** também chamadas de aplicações *desktop*, que possuem componentes hospedados em uma JVM local; e
- ***web* e serviços:** foi testada sua execução no servidor de aplicação *Sun Application Server* 9 com *containers Servlet,* JSP (*JavaServer Pages*) para *web* e *enterprise* para os serviços.

A Figura 42 mostra o modelo de classe para uma aplicação utilizando chamadas de métodos remotos com reconfiguração em tempo de execução. Este modelo está organizado em três camadas (pacotes). A aplicação cliente-servidor está organizada no pacote (`apps`), qualquer outro tipo de aplicação (local, *web*, serviço) seria inserido neste pacote. O pacote (`base`) contém a aplicação lógica com uma interface de acesso definida e as classes responsáveis pela supervisão estão organizadas no pacote (`meta`). A seguir são detalhadas as classes desses

pacotes.

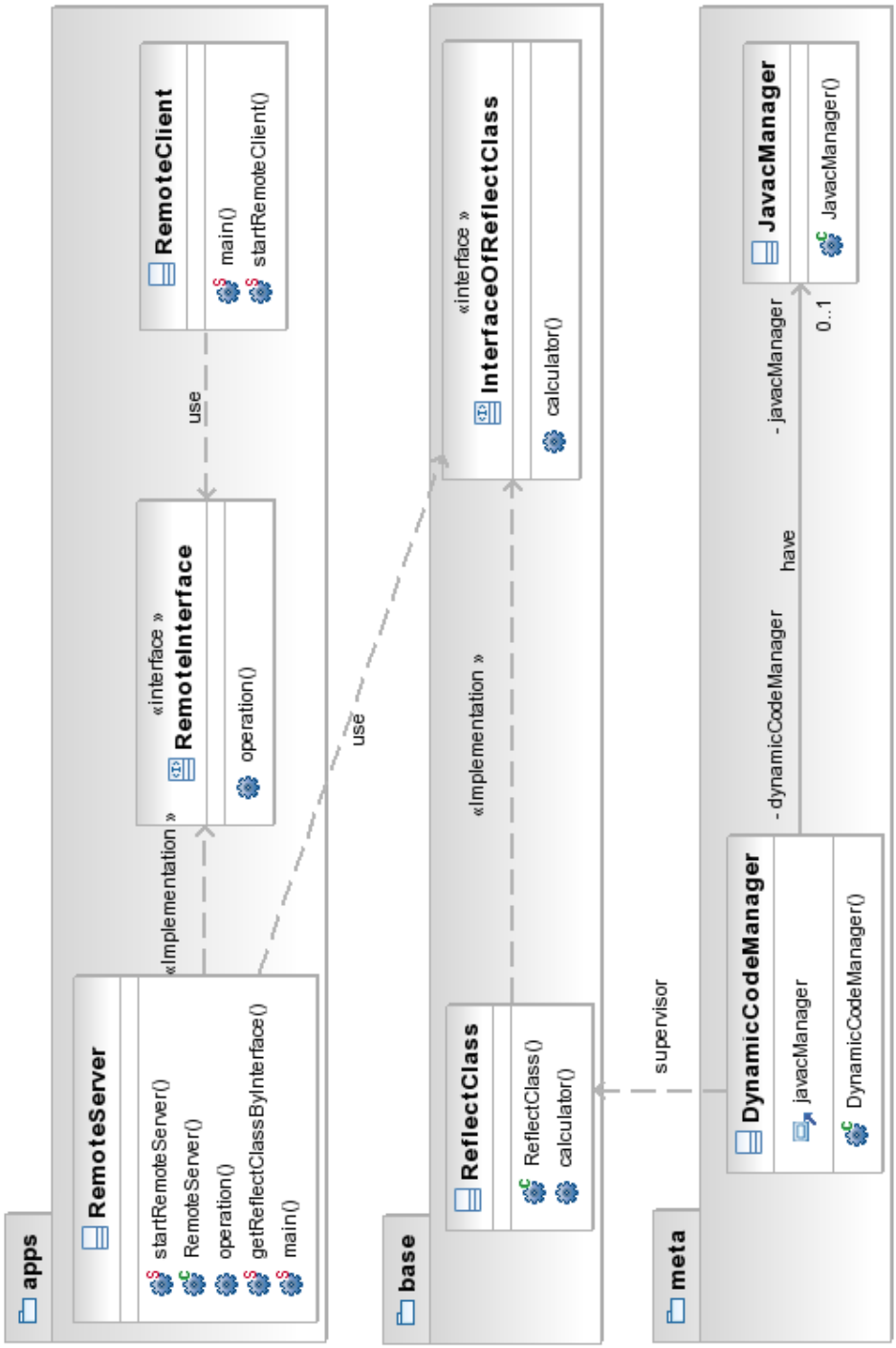

Figura 42: Subsistema de reconfiguração

O pacote (`meta`) possui duas classes essenciais para a reconfiguração dos objetos, `DynamicCodeManager` e `JavacManager`, que são responsáveis por gerenciar a reconfiguração e a recompilação dos objetos em tempo de execução, respectivamente. O acionamento dessas

classes depende da invocação vinda do pacote (`apps`), as classes presentes no pacote (`meta`) analisam se a classe atual (`base`) pode ou não atender as solicitações vindas dos clientes (`apps`).

O pacote (`base`) é composto por uma interface `InterfaceOfReflectClass` e uma classe `ReflectClass`, que representam a aplicação lógica a ser reconfigurada. Pode-se notar também a relação supervisora que a classe `DynamicCodeManager` exerce sobre a classe `ReflectClass`. Além disso, vale ressaltar que o acesso à aplicação é realizado por meio de suas interfaces de comunicação e, dessa forma, a reconfiguração pode ocorrer sem a percepção direta do seu cliente.

O pacote (`apps`) representa, em linhas gerais, as aplicações que podem utilizar uma funcionalidade reconfigurável. A classe `RemoteServer` possui uma referência para a interface `InterfaceOfReflectClass` com a finalidade de utilizar a funcionalidade presente na classe `ReflectClass`. Dessa forma, pode-se dizer que a classe `RemoteServer` encapsula as funcionalidades da classe `ReflectClass` e disponibiliza para seus clientes na forma de uma operação transparente. Eles (clientes) a utilizam como se fosse local e não têm a percepção quanto à distribuição e reconfiguração.

Quanto aos aspectos referentes ao funcionamento da reconfiguração dos objetos, a implementação proposta prevê que novos objetos sejam criados por mecanismos de herança com agregação de características e funcionalidades ou pela modificação e substituição dos executáveis em tempo de execução. No entanto, este último, por se tratar de aplicações distribuídas, requer maior atenção por parte do sistema supervisor, pois pode envolver objetos que estão sendo referenciados por outras aplicações e devem ter seus estados preservados quando um novo objeto, similar a ele é criado, compilado e alocado em memória para atender a nova execução. Nestes casos os subsistemas (JAVA para XML e o inverso) anteriores são essenciais para gerenciamento dos estados dos objetos

Neste contexto ainda, existem situações em que as solicitações vindas dos objetos

clientes não são atendidas pelos provedores de serviços e um novo objeto necessita ser invocado. Para isso, o meta-objeto faz uma consulta nos repositórios de informação, presentes no ambiente de execução reconfigurável, para verificar se existe algum objeto capaz de atender a solicitação requisitada pelo cliente. Caso um objeto seja encontrado, este é compilado e executado pelo sistema para atender a requisição do cliente. Caso contrário, uma exceção é lançada e encaminhada ao cliente da solicitação.

Dessa forma, encerra-se a apresentação da ferramenta *ReflectTools*® e seus subsistemas. A seguir são apresentadas as considerações finais deste capítulo.

5.3. Considerações Finais

Pode-se notar que o desenvolvimento da ferramenta *ReflectTools*® foi fundamental para a MDSR, pois permite que especialistas de domínio, engenheiros de software e desenvolvedores trabalhem no desenvolvimento de software de maneira cooperativa e com padrões de processos definidos. Além disso, destaca-se também o suporte de automatização das tarefas e geração de código padronizado, minimizando as atividades manuais.

Destaca-se também a padronização na geração de informações em consultas e a minimização da participação humana em atividades em que inconsistências ou incertezas linguística podem resultar em problemas quanto à reutilização e reconfiguração dos artefatos desenvolvidos.

Outro fator de grande importância nesta atividade é a definição e a segmentação dos domínios de desenvolvimento, pois permite melhor definição dos artefatos de software e, consequentemente, melhor otimização nos mecanismos de consulta. Essa característica é importante tanto para a busca exata na base de dados quanto nos repositórios de regras de informação, pois os critérios de buscas são baseados nas características e nomes (atributos e métodos) de desenvolvimento.

A técnica de reconfiguração (subsistema) implementada na ferramenta *ReflectTools*® permite que a reconfiguração seja realizada em vários tipos de sistemas: locais, distribuídos, *Web* e serviços. O subsistema desenvolvido interage amigavelmente com máquinas virtuais locais e de servidores de aplicação.

A seguir, no Capítulo 6, Avaliação dos Resultados, são apresentados os resultados obtidos com a aplicação da MDSR no desenvolvimento de software.

CAPÍTULO 6

Avaliação dos Resultados

6.1. CONSIDERAÇÕES INICIAIS

Este capítulo apresenta a avaliação dos resultados obtidos com a aplicação da Metodologia para Desenvolvimento de Software Reconfigurável (MDSR), apresentada no Capítulo 4. Para avaliar as diretrizes propostas na metodologia e a ferramenta *ReflectTools*® foram realizados quatro estudos de casos, com grupos de alunos de graduação e pós-graduação. O objetivo principal foi de simular o ambiente de execução reconfigurável e aplicar a metodologia para o desenvolvimento de artefatos reconfiguráveis.

Os alunos envolvidos foram organizados aleatoriamente em grupos de três a cinco pessoas, com os papéis definidos, conforme previsto na metodologia, cujo objetivo foi de simular os núcleos de desenvolvimento e os domínios de atuação presentes no Ambiente de Execução Reconfigurável (AER). Cada grupo selecionou um sistema a ser desenvolvido com domínio de atuação distinto, para que a MDSR pudesse ser aplicada e avaliada, assim como a ferramenta *ReflectTools*®.

Este capítulo está organizado da seguinte maneira: na Seção 6.2 são apresentadas as

formações dos grupos e as descrições dos sistemas utilizados; a Seção 6.3 apresenta a aplicação da metodologia e os artefatos produzidos; na Seção 6.4 são apresentadas as comparações dos resultados obtidos e, finalmente, na Seção 6.5 são apresentadas as considerações finais.

6.2. Os Núcleos de Desenvolvimento e os Sistemas Desenvolvidos

Antes de iniciar a aplicação das diretrizes da MDSR, foi necessário ministrar um treinamento das ferramentas de desenvolvimento de software e os *plugins*, utilizados como suporte. Julgou-se necessário este tipo de conduta pelas necessidades da MSDR e pela natureza dos sistemas que se pretendia desenvolver. Todos os sistemas que foram desenvolvidos utilizaram diagramas UML, linguagem de programação JAVA e banco de dados relacional. *Frameworks* adicionais também foram utilizados, em que pode-se destacar o *Hibernate* com recurso de anotações para persistência de dados.

Apesar dos estudos de casos serem conduzidos com alunos de graduação e pós-graduação, a diversidade de conhecimento entre eles é similar, assim como a falta de conhecimento nas ferramentas (Eclipse e *Netbeans*) e na linguagem de programação JAVA. Dessa forma, julgou-se adequado realizar o treinamento nessas ferramentas e linguagem de programação. Nos quatro estudos de casos realizados, o treinamento foi realizado de maneira gradativa, sendo aplicado conforme necessidade da MDSR e da implementação. A seguir, nas Seções 6.2.1 a 6.2.4, são apresentados os estudos de casos realizados.

6.2.1. Estudo de Caso 1

O primeiro estudo de caso foi realizado durante o ano 2006, com alunos de graduação. Eles foram organizados em sete grupos e cada um deles com um sistema a ser desenvolvido. Cada grupo foi responsável por elaborar um documento de requisitos com as seguintes seções: visão geral do sistema; descrição dos requisitos funcionais; consultas e relatórios; e requisitos

não-funcionais. Como resultado, os seguintes documentos de requisitos foram obtidos:

1. **Gerenciamento de uma Biblioteca:** que tem por objetivo gerenciar os empréstimos e devolução das obras presentes na biblioteca pelos seus leitores, além disso, este também prevê o gerenciamento de leitores e obras. O sistema também controla as atividades realizadas pelos funcionários;
2. **Gerenciamento de um Hotel:** cuja finalidade é gerenciar as reservas e estadias de hóspedes no hotel, assim como os possíveis consumos e serviços utilizados na estadia. Outra característica desse sistema é o gerenciamento de acompanhantes aos hóspedes. O sistema também controla as atividades realizadas pelos funcionários;
3. **Gerenciamento de uma Locadora de Carros:** tem por finalidade gerenciar a reserva e retirada de carros da locadora, os serviços contratados pelos clientes, assim como os funcionários que atenderam determinada reserva/retirada. O sistema também permite realizar um controle financeiro quanto aos pagamentos realizados pelos clientes;
4. **Controle de Produção para uma Pizzaria:** tem por objetivo fazer o gerenciamento da produção de uma pizzaria, verificando os ingredientes utilizados e alertando sobre suas quantidades no estoque. O sistema também controla as atividades realizadas pelos funcionários;
5. **Controle de Pagamento e Recebimento de Contas:** tem por finalidade o gerenciamento das contas e seus respectivos pagamentos pelos seus proprietários;
6. **Revisão de Artigos de uma Revista Eletrônica:** tem por objetivo controlar o processo de revisão de artigos, por vários autores, para uma revista eletrônica; e
7. **Controle de uma Distribuidora de Produtos Genéricos:** tem por finalidade fazer apenas o controle de estoque de produtos, informando quais foram os pedidos realizados pelos clientes e os respectivos produtos selecionados.

Os documentos de requisitos não serão apresentados na íntegra nesta seção por motivo de espaço. De posse do documento de requisitos de cada grupo, chamados de agora em diante de núcleo de desenvolvimento, foi iniciada a modelagem dos sistemas. Essa etapa foi realizada na ferramenta Eclipse, versão 3.2, utilizando o *plugin Omondo*. Os modelos produzidos são os diagramas de classe e sequência, que representam, respectivamente, a lógica do sistema e troca de mensagens entre objetos na execução de uma funcionalidade. Ainda nessa etapa, os engenheiros de software realizam a inserção das *tags* de documentação (JAVADOC) das funcionalidades por eles modeladas. Vale ressaltar nessa etapa a importância do *plugin* utilizando, pois, a partir do modelo UML, foi gerado o código JAVA equivalente, que aumenta consideravelmente a produtividade e a padronização proposta neste trabalho.

Quando este estudo de caso foi realizado não existia na proposta da metodologia a presença de um especialista de domínio, conforme especificado na versão atual. A padronização dos nomes dos artefatos era intuitiva e não seguia qualquer orientação sobre domínio de atuação, e a reutilização também ocorria por intuição. Diante disso, para aprimorar o desenvolvimento e, consequentemente, aumentar a reutilização e reconfiguração dos artefatos, foi inserida a figura de um especialista de domínio, conforme descrito no Capítulo 4.

A seguir, na Tabela 2, é apresentado um quadro resumo contendo os nomes dos sistemas modelados, o número de classes geradas e seus respectivos nomes. Nessa etapa, cada núcleo realizou a interpretação do documento de requisitos e modelagem do sistema de maneira individual, não foram explorados recursos de reutilização de artefatos (classes, componentes e/ou serviços) entre os diferentes núcleos do ambiente.

Em seguida, foram desenvolvidos os Modelos Entidades-Relacionamento (MER) para os diagramas de classe da etapa anterior. Esses modelos também foram confeccionados na ferramenta Eclipse, versão 3.2, utilizando o *plugin Azzurri Clay*. Nessa etapa, merecem destaque o mapeamento de objetos para entidades conforme diretrizes do *framework Hibernate*

(HIBERNATE, 2009) e a geração automática dos *scripts* SQLs, a partir do MER gerado anteriormente.

Tabela 2: Lista de sistema e classes lógicas geradas

Lista de Sistema e Classes Lógicas Geradas		
Data do desenvolvimento:	**Setembro a outubro de 2006**	
Nome do Sistema	**Número de Classes**	**Nome das Classes**
Gerenciamento de uma Biblioteca	10	`CategoriaLeitor, CopiaObra, Funcionario, Locacao, Pessoa, CategoriaObra, Emprestimo, Leitor, Obra, Reserva`
Gerenciamento de um Hotel	13	`Acomodacao, CategoriaItens, Estadia, Funcionario, ItensConsumo, Pessoa, TipoAcomodacao, Brasileiros, Consumo, Estrangeiros, Hospede, Locacao, Reserva`
Gerenciamento de uma Locadora de Carros	10	`Aluguel, Carro, Cliente, Funcionario, Reserva, Brasileiro, Categoria, Estrangeiro, Locacao, Servicos`
Controle de Produção para uma Pizzaria	6	`Cliente, Funcionario, Pedido, Pessoa, Produto, ProdutoPedido`
Controle de Pagamento e Recebimento de Contas	5	`Agencia, Bancos, Conta, Documentos, Pessoa`
Revisão de Artigos de uma Revista Eletrônica	6	`Administrador, Autor, Avaliacao, Pessoa, Revisor, Trabalhos`
Controle de uma Distribuidora de Produtos Genérica	4	`Cliente, Pedido, Pedido_Produto, Produto`

É importante observar, neste momento, que tanto os diagramas de lógica (classe) e entidades (MER) não serão apresentados neste capítulo, pois são vários sistemas em desenvolvimento e os espaços por eles exigidos são considerados elevados.

Na fase de mapeamento foram observados dois fatores importantes:

1. o primeiro refere-se às regras de mapeamento objeto-entidade, que se mostraram de

difícil entendimento pelos desenvolvedores na fase de confecção do modelo físico e, consequentemente, retardando a velocidade de desenvolvimento. No entanto, esta dificuldade pode ser amenizada por um subsistema que será encarregado de gerar a estrutura de armazenamento equivalente para cada classe modelada. Esse subsistema encontrava-se em fase de desenvolvimento quando este estudo de caso foi realizado, no entanto, diante da evolução e facilidade da tecnologia de mapeamento (*framework hibernate* com anotações), sua evolução foi interrompida e este *framework* incorporado ao desenvolvimento; e

2. o segundo fator é uma consequência do primeiro, que representa o benefício da geração de código SQL, a partir do modelo MER, pois facilita a padronização no desenvolvimento.

Após geradas as estruturas lógicas (classes) e de armazenamento (entidades), iniciou-se a etapa de criação dos métodos de persistência dos objetos. Inicialmente, foram criados como parte da lógica dos sistemas, ou seja, uma classe lógica possui um método que foi implementado com o objeto que gera uma *String* como retorno, que representa um SQL de inserção de um objeto/entidade, por exemplo. No entanto, neste trabalho, essa funcionalidade pode ser eliminada pelos *frameworks* de requisitos não-funcionais, *Hibernate*, por exemplo.

As Figuras 43 e 44 mostram, respectivamente, trechos de código com o desenvolvimento de requisitos não-funcionais nas classes lógicas, com sua chamada de execução e o código com a utilização do *framework Hibernate* para persistência de objetos. Na Figura 43, o código fonte encontra-se espalhado pelos métodos da classe, dificultando futuras manutenções desse sistema, pois eles devem ser avaliados a cada atributo adicionado ou retirado da classe. Além disso, podem retardar o desenvolvimento, devido à codificação realizada para cada operação. Enquanto que na Figura 44, o código apresentado é “enxuto” e

otimizado, pois um objeto do *framework* é responsável por receber um objeto lógico e fazer sua persistência no Banco de Dados. Além disso, destaca-se a velocidade de desenvolvimento, pois os desenvolvedores ficam isentos da implementação dessas funcionalidades (geração do código SQL).

Método original desenvolvido dentro da Classe Funcionário para realizar uma inserção deste:

```
public String insereFuncionario() {

      String sql = "INSERT INTO FUNCIONARIO (cargo, codigoPessoa,
                  dataContratacao, dataDemissao, salario, setor) VALUES ('";
            sql = sql + this.getCargo()           + " ', ' ";
            sql = sql + this.getDataContratacao() + " ', ' ";
            sql = sql + this.getDataDemissao()    + " ',  ";
            sql = sql + this.getSalario()         + " , ' ";
            sql = sql + this.getSetor()           + " ') ";


            return sql;
} // fim do método
```

Código de utilização:

```
// Código para criação e atribuição de valores ao objeto
Funcionario funcionario = new Funcionario();
funcionario.setCargo("entregador");
funcionario.setCodigoPessoa(p.getCodigoPessoa());
// ...
// demais atribuições
// ...
funcionario.setTelefone("3333-4444");
// Chamada ao método da classe que gera o código SQL para inserção no
// Banco de Dados
String sql = funcionario.insereFuncionario();
conexao.persistentConexao(sql);
```

Figura 43: Desenvolvimento sem comprometimento com organização de código

Métodos de persistências são disponibilizados pelo *framework* de requisitos não-funcionais.

```
// Declaração e criação do objeto

FrmPersistent persistent = new FrmPersistent();

// Métodos para criar a conexão com o banco de dados

persistent.getConexao();
```

Código de utilização:

```
// O Código para criação do objeto é o mesmo do apresentado na Figura 43

// Código para realizar a persistência de objetos

persistent.save(funcionario);
```

Figura 44: Desenvolvimento utilizando o framework de persistência

Ainda sobre a diferença entre o desenvolvimento apresentado nas Figuras 43 e 44, cabe salientar que a ideia principal da metodologia proposta neste trabalho é fazer do software um produto em linha de montagem, em que sua concepção inicial (artefato = classe | componente | aspecto | serviço) é a mais "pura" possível, onde somente requisitos funcionais, e os requisitos de infraestrutura, não-funcionais, (distribuição, otimização, segurança, persistência e reflexão) são adicionados, quando necessário, conforme representa a Figura 45.

Figura 45: Objeto em linha de montagem

Seguindo o mesmo princípio, de linha de montagem, o próximo passo é incorporar os

recursos de distribuição da aplicação. Esta ocorre de maneira independente, sem misturar as funcionalidades lógicas com a distribuição. Da mesma forma é realizada a agregação dos recursos de segurança e otimização, pois, em uma aplicação distribuída, alguns cuidados com acesso às informações que trafegam pela rede e, também, a velocidade de comunicação entre objetos deve ser considerada. Finalmente, os recursos de reflexão, que permitem à aplicação auto-reconfiguração, criação de novas características, funcionalidade ou até mesmo novas classes, também utilizam o mesmo princípio. Esse recurso também é incorporado de maneira opcional pelos desenvolvedores, livre de qualquer envolvimento com a lógica funcional do artefato que se pretende reconfigurar.

Após finalizar o processo de montagem do software, pode-se dizer que um novo artefato foi desenvolvido, sendo assim, deve ser inserido nos repositórios de informações para que possa ser reutilizado em projeto futuro. Para tornar essa possibilidade viável, é necessário que sejam extraídas informações para os respectivos repositórios. Por motivos de espaço, é apresentada apenas a listagem de todas as classes desenvolvidas nos sistemas apresentados neste estudo de caso e um trecho da documentação da classe (`Pessoa`), como mostra a Figura 46.

Para cada projeto, desenvolvido em cada núcleo, existe uma documentação referente que mostra as seguintes informações: pacote em que a classe está inserida; o nome da classe; qual(is) é (são) seu(s) construtor(es); seus atributos e tipos; seus métodos com tipos de retorno e parâmetros. Todas estão associadas a uma descrição inserida pelos engenheiros de software e desenvolvedores na etapa de concepção/desenvolvimento. Sobre esta etapa, vale destacar que a documentação de todo artefato desenvolvido na fase de análise não era supervisionada por um especialista de domínio, conforme apresentado no Capítulo 4. Diante dos problemas encontrados com a realização dos estudos de casos desenvolvidos e apresentados neste capítulo, a metodologia passou por um processo de re-estruturação desde sua concepção original, sendo personagens (especialista de domínio) e etapas (fases de desenvolvimento) adicionados. Na

Seção 6.3, considerações finais, são destacadas as principais contribuições de cada estudo de caso para a evolução da MDSR.

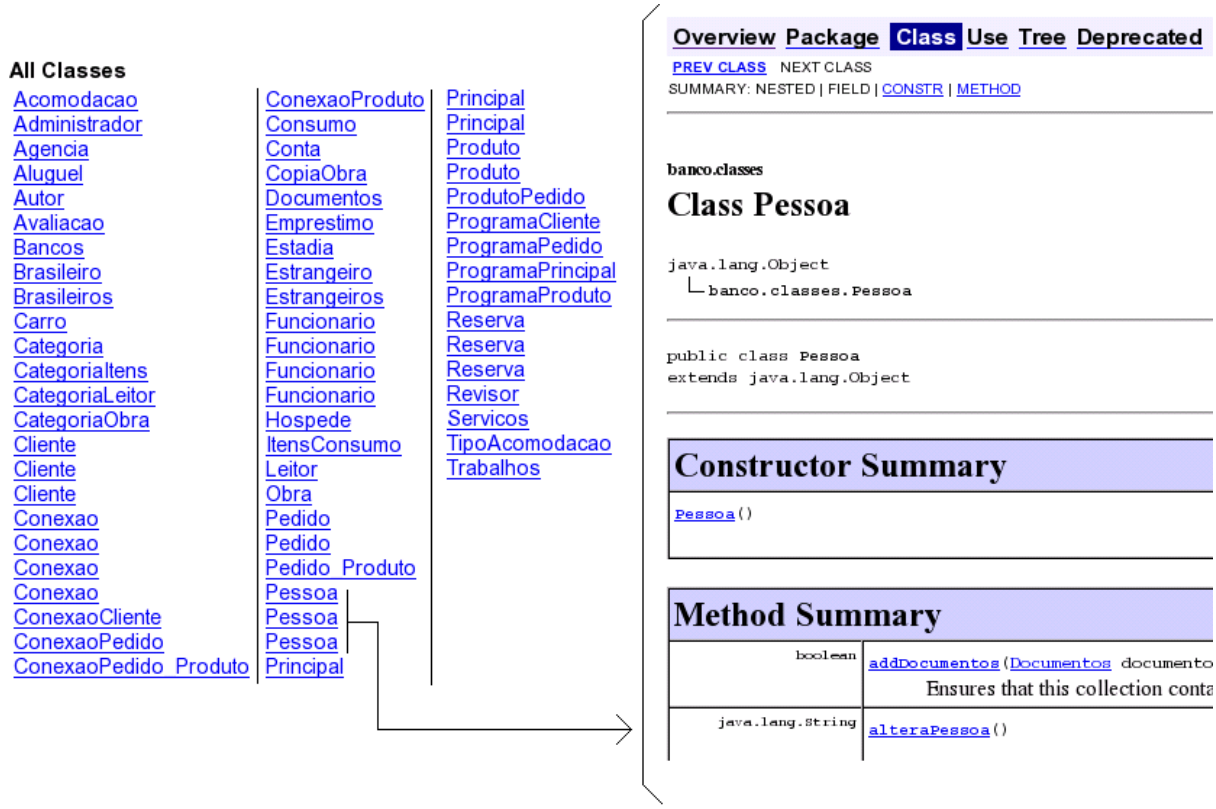

Figura 46: Lista de classe e trecho de documentação JAVADOC

Com o acompanhamento dos resultados foram obtidos alguns indicativos de grande importância, que justificam a relevância e objetivo deste trabalho. Conforme mencionado no início desta seção, cada núcleo possui uma especificação de um sistema com domínios distintos e, foram aplicadas as diretrizes da MDSR original, para que os sistemas fossem implementados.

No desenvolvimento dos sistemas, não houve nenhuma troca de informação entre os núcleos e cada um modelou e implementou seu sistema de maneira restrita ao domínio de atuação. Essa situação foi proposital para avaliar a redundância que eles poderiam gerar e, consequentemente, mostrar que um ambiente de desenvolvimento com reutilização de software pode minimizar o esforço na etapa de desenvolvimento. Essa situação pode ser facilmente constatada na Figura 47.

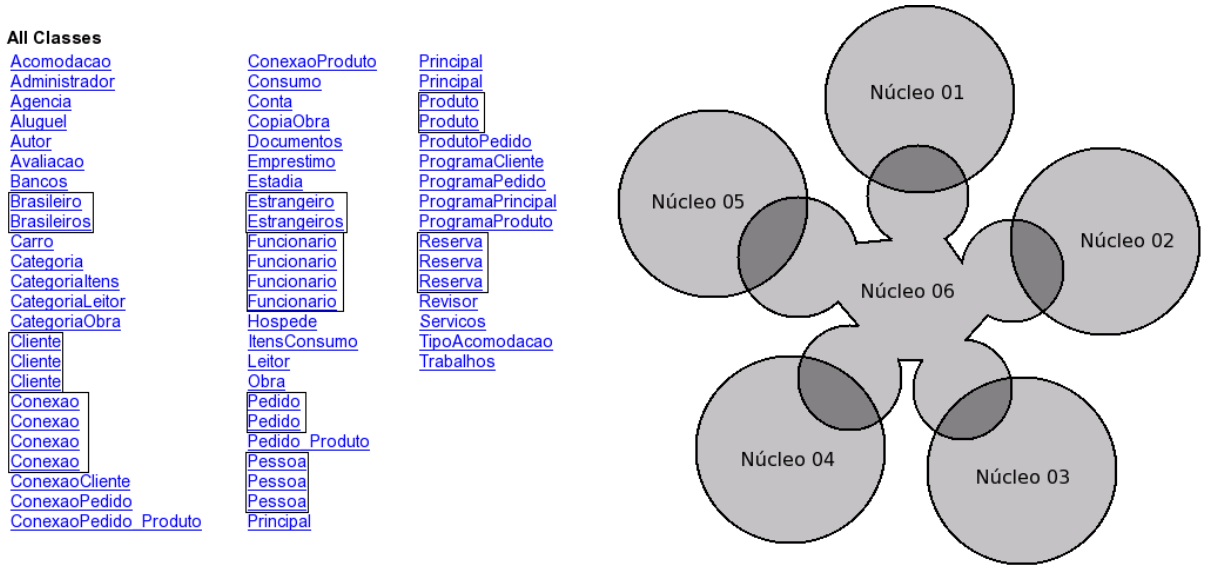

Figura 47: Listagem de classe e o reúso entre os núcleos

Com a documentação gerada, pode-se observar, pelos quadros, que existem várias classes de mesmo nome ou apenas divergindo pelo plural do seu nome. Esses quadros mostram que elas foram projetadas em núcleos diferentes e que, talvez no processo de desenvolvimento diferenciado (MDSR), poderiam ser reutilizadas na sua totalidade ou por meio de pequenas adaptações, minimizando o esforço de desenvolvimento e proporcionando maiores garantias quanto à funcionalidade, pois já são unidades de software bem definidas e testadas. Essa situação pode ser visualizada também na Figura 47, em que o núcleo 6, ao desenvolver sua aplicação, pode fazer uso de parte dos demais núcleos, diminuindo sua carga de produção e aumentando a reusabilidade dos artefatos já existentes.

Como o desenvolvimento central deste trabalho é a construção de aplicações reconfiguráveis e, para isso, julgou-se necessário a criação de uma metodologia que auxiliasse tanto engenheiros de software quanto desenvolvedores na padronização de desenvolvimento de novas aplicações, na reutilização das existentes e na integração de serviços existentes/novos. Essas características são alcançadas somente pela segmentação de interesses no desenvolvimento e pela padronização da troca de mensagens entre os artefatos (objetos, componentes e serviços).

Ainda no contexto de reutilização e reconfiguração, destaca-se a criação de ferramentas de apoio no ambiente de execução reconfigurável. Dentre elas, vale ressaltar que a reflexão pode ser aplicada no banco de dados para coletar informações sobre as estruturas de armazenamento (*Hibernate*) e, quando necessário, podem ser utilizadas pelos desenvolvedores para a verificação da integridade entre objetos e entidades, quando um comando de persistência é acionado. Além disso, podem ser utilizadas para adaptação/reconfiguração de artefatos existentes.

Esta seção apresentou em mais detalhes as etapas de condução do estudo de caso realizado. Seu principal objetivo foi avaliar as diretrizes da MDSR, centralizando as atividades no desenvolvimento de artefatos de software "puros", com ênfase nas atividades de reúso dos artefatos produzidos em diferentes domínios de atuação. Neste estudo de caso não houve a aplicação direta de mecanismos de reconfiguração dos artefatos. A seguir são mostrados os demais estudos de casos (2, 3 e 4), no entanto, as etapas que foram apresentadas nesta seção são omitidas, apenas destacando aquelas que apresentam contribuição significativa em resultados ou para modificação da metodologia apresentada no Capítulo 4.

6.2.2. Estudo de Caso 2

O segundo estudo de caso foi realizado em 2007, com alunos de graduação. Eles foram organizados em 10 grupos e cada um deles com um sistema a ser desenvolvido. Cada grupo foi responsável por elaborar um documento de requisitos com as mesmas seções do estudo de caso 1. Como resultado, os seguintes documentos de requisitos foram obtidos:

1. **Gerenciamento de Posto de combustível:** tem por objetivo gerenciar a venda de combustível para os clientes. O sistema pode gerenciar a venda de produtos da loja de conveniência e qual funcionário atendeu o cliente;
2. **Controle de Tráfego Aéreo:** permite gerenciar o tráfego aéreo, gerenciando o

número de escalas e rotas de um ponto de origem a um ponto de destino.

3. **Gerenciamento de Consulta Médica:** permite o gerenciamento de consultas em um consultório médico. Para realizar o agendamento de uma consulta, o sistema pode solicitar o convênio médico ou não e, ainda, associa o funcionário que registrou a consulta. Finalmente, qual médico atendeu o paciente e qual foi a prescrição médica;
4. **Gerenciamento de Agência de Viagem:** permite realizar a venda de pacotes turísticos para os clientes. Para isso, o sistema possui um conjunto de pacote e pontos turísticos cadastrados, para que o cliente possa selecionar o desejado. O sistema registra qual funcionário da agência atendeu o cliente;
5. **Gerenciamento de Entrega para *FastFood*:** permite que o cliente realize um pedido para uma loja de *FastFood*. Para isso, o sistema deve possuir seus dados previamente cadastrados. Um funcionário registra o pedido e encaminha para o setor de entrega para que seja enviado ao cliente;
6. **Submissão e Avaliação de Artigo em Congresso:** permite realizar o gerenciamento no processo de submissão de artigos para congresso. Os autores submetem os artigos, que são encaminhados para a comissão de avaliação para emissão de um parecer. Em seguida, os artigos aceitos são convocados para apresentação no evento;
7. **Gerenciamento de Agendamento de Tarefas:** permite realizar o gerenciamento de tarefas por um solicitante. O sistema recebe uma ordem de serviço, que será apresentada para os responsáveis pela execução. Ao executar a tarefa um registro, deve ser inserido no sistema para informar ao solicitante quem a executou, horário, atividades realizadas e atividades pendentes;
8. **Logística de Transporte para Transportadora:** permite realizar o gerenciamento de transportes por uma transportadora. O cliente faz a solicitação de envio de uma

mercadoria de um ponto de origem a um ponto de destino. O sistema deve otimizar a entrega para o cliente e para a transportadora, retornando a melhor opção para efetuar o frete, dentro das regras e condições de custo e prazos de entrega;

9. **Gerenciamento de Conta Bancária *On-Line*:** permite aos clientes de um banco administrar as operações financeiras realizadas em sua conta corrente. O cliente pode registrar os pagamentos a serem efetuados e verificar a movimentação financeira por um período definido; e
10. **Controle de Acesso a Salas:** permite realizar controle de acesso em determinados locais de uma empresa. O sistema faz a leitura, por meio de um sensor, das credenciais do funcionário e libera ou não o seu acesso ao local. O sistema permite verificar a rastreabilidade de um funcionário, mostrando os locais que teve acesso e quanto tempo permaneceu naquele local.

Para realizar este estudo de caso foram realizados os mesmos passos referentes ao anterior. No entanto, os mesmos não serão apresentados nesta seção, sendo destacadas apenas as informações relevantes à condução do mesmo. A modelagem, lógica e de dados, foi realizada na Ferramenta Eclipse, assim como a aplicação das regras de mapeamento objeto-relacional. Para melhorar a implementação dos sistemas foram introduzidos alguns padrões de projeto, DAO, *Data Access Object, Singleton, Facade* e *Proxy* (GAMMA, 1994).

Os padrões de projeto fornecem melhor segmentação da aplicação e isolamento das funcionalidades em requisitos funcionais e não-funcionais. Além disso, permitem melhor desempenho quanto ao gerenciamento de memória dos servidores de aplicação, pois, se tratando de aplicações distribuídas reconfiguráveis, a administração dos recursos de aplicação deve ser otimizada, devido ao número de acesso que eles podem possuir ao longo de sua execução. A implementação desses padrões ocorreu da seguinte maneira:

1. **oculta:** quando os desenvolvedores receberam um subsistema com funcionalidade definida. Por exemplo, tem-se o gerenciamento de conexão com o banco de dados, em que foi projetado um subsistema chamado "Fábrica de Conexão", cuja responsabilidade seria retornar uma conexão para o solicitante. Para otimizar o uso de memória, foi utilizado o padrão *Singleton*, que garante que existirá apenas uma instância da Classe responsável por fornecer essa conexão; e
2. **declarada:** quando os desenvolvedores aplicaram os padrões para facilitar o isolamento, agrupamento e mudança de acesso das funcionalidades para um outro sistema ou dentro do próprio sistema em desenvolvimento.

Para conduzir este estudo de caso foi elaborado um plano de ação, que era composto por um conjunto de atividades, cujo objetivo foi de corrigir os erros do estudo de caso anterior (1) e/ ou melhorar as atividades executadas. Os grupos de alunos envolvidos foram bem heterogêneos quanto à experiência com a linguagem de programação JAVA e seus recursos de programação.

Apesar de o estudo de caso ser conduzido de maneira orientada quanto à execução das atividades, houve um direcionamento quanto ao estudo da linguagem JAVA pelos grupos. O principal objetivo foi de proporcionar um nivelamento entre eles, para que a velocidade de desenvolvimento ocorresse mais similar possível. No entanto, constatou-se grande dificuldade na aplicação do treinamento da linguagem de programação e de seus recursos adicionais como padrões e *frameworks*.

Ao avaliar os resultados parciais da implementação, outros problemas foram surgindo, dentre eles, destaca-se a falta de padronização nos nomes dos artefatos. Inicialmente, foi introduzido o procedimento de nomenclatura, porém o mesmo não foi aplicado na confecção dos modelos de lógica (classe) e de dados. Os grupos que apresentaram estes problemas tiveram que refazer os modelos e codificação, pois, conforme apresentado na MDSR e na Ferramenta

ReflectTools®, esses nomes são essenciais para identificação dos artefatos (nome, atributos e métodos). Dessa forma, o desenvolvimento foi retardado e o cronograma de execução dos projetos foi comprometido.

Neste estudo de caso foram observados dois fatores de grande relevância e que contribuíram significativamente para o desenvolvimento da MDSR. São eles:

1. diante dos fatos ocorridos com a padronização dos nomes, julgou-se necessário criar um mecanismo que auxiliasse os núcleos desenvolvedores a atribuir os nomes para os artefatos de software que estão sendo elaborados. Diante disso, decidiu-se inserir na MDSR a figura de um especialista de domínio, que é o encarregado de nomear artefatos, atributos e métodos, conforme contexto de atuação, segundo conceitos de rede semântica. Além disso, tem como responsabilidade a revisão dos projetos elaborados, desenhos de lógica (Diagrama de Classe) e de dados (MER), antes que a atividade de implementação seja inicializada; e
2. na etapa de implementação foi possível introduzir conceitos "modernos" de programação, como serviços *Web*. Em alguns grupos somente, devido a maturidade e experiência na linguagem JAVA, o treinamento teve melhor aproveitamento e, após a implementação do sistema, algumas funcionalidades e recursos de validação de informação foram identificados, isolados e transformados como serviços *Web*. Os núcleos passaram a utilizar essas funcionalidades por meio do consumo de serviço. A reconfiguração foi aplicada utilizando esse recurso de programação e os sistemas reagiram com naturalidade, não detectando qualquer mecanismo de interceptação e modificação da funcionalidade. Dessa forma, foi possível observar que a técnica desenvolvida e apresentada no Capítulo 5, Seção 5.2.1.3, mostrou-se válida para a reconfiguração em tempo de execução. A Figura 48 mostra a estrutura das aplicações com o subsistema de reconfiguração.

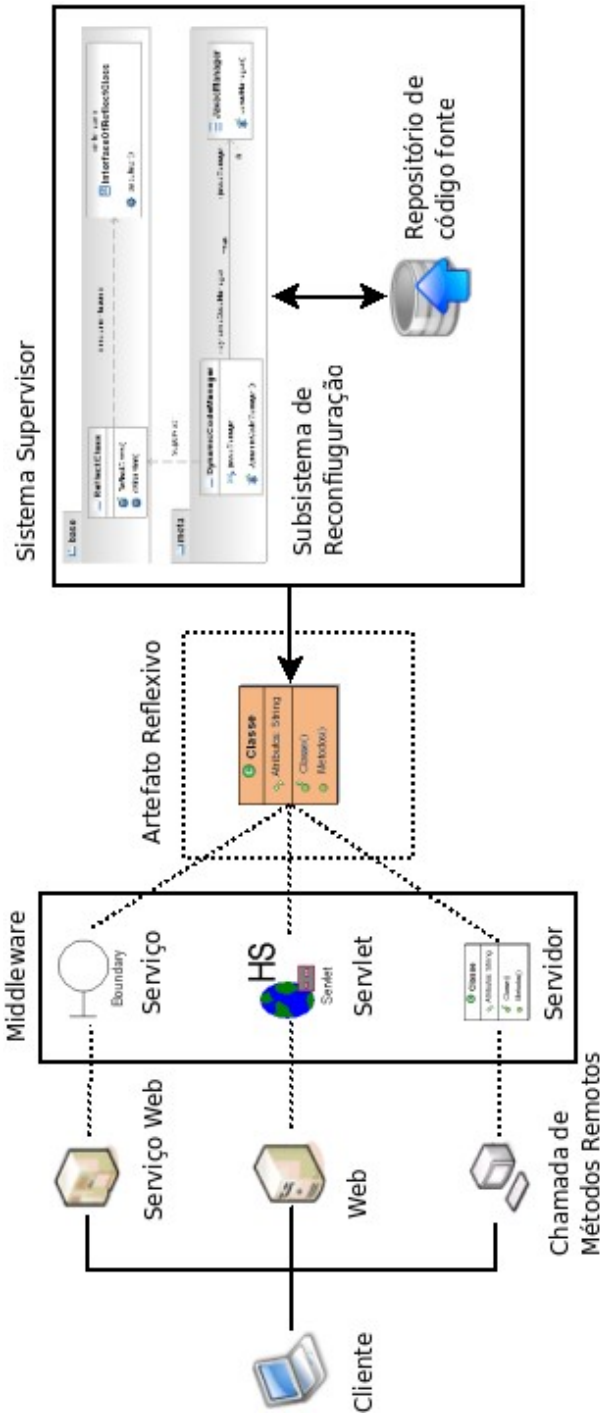

Figura 48: Estrutura da aplicação utilizando reconfiguração

A Figura 48 mostra a estrutura dos sistemas desenvolvidos neste estudo de caso, destacando a reconfiguração das funcionalidades. Pode-se observar que uma aplicação cliente solicita uma funcionalidade presente em um artefato de software (Artefato Reflexivo), por meio de qualquer *middleware* de acesso (Serviço, *Web* e/ou Chamada de Métodos Remotos). Este

artefato é supervisionado por subsistema encarregado de gerenciar a execução de suas funcionalidades, ou seja, se o artefato existente pode ou não atender a solicitação do cliente. Em caso positivo, a resposta é enviada imediatamente, caso contrário, o subsistema se encarrega de consultar os repositórios existentes (em domínio local ou distribuído) e adaptar o artefato reflexivo para atender a solicitação do cliente.

A operação de adaptação (reconfiguração) é realizada de maneira transparente, sem que a aplicação cliente saiba que mudanças no artefato reflexivo tenham sido realizadas. Além disso, o sistema supervisor também é encarregado de gerenciar as execuções e preservar os estados das solicitações vigentes, sem que haja perda de informações no caso de reconfiguração de uma funcionalidade.

Para finalizar, os sistemas desenvolvidos neste estudo de caso são, basicamente, voltados para *Web* com a combinação de Serviços *Web* e Chamadas de Métodos Remotos. Em alguns deles houve uma mescla de tecnologias, pois os recursos comuns foram identificados e transformados em unidades reutilizáveis.

Este estudo de caso teve por objetivo principal avaliar as mudanças na MDSR, desde sua concepção inicial até as constatações, e os resultados obtidos com a realização do estudo de caso 1. A seguir, Seção 6.2.3, são apresentados os resultados do estudo de caso 3.

6.2.3. Estudo de Caso 3

O terceiro estudo de caso foi realizado em 2008, com alunos de pós-graduação. Eles foram organizados em 4 grupos e cada um deles com um sistema a ser desenvolvido. Neste estudo de caso, o número de grupos foi reduzido para preservar a estrutura interna dos grupos elaborados nos estudos de casos anteriores. Cada grupo foi responsável por elaborar um documento de requisitos com as mesmas seções do estudo de caso 1. Como resultado, os seguintes documentos de requisitos foram obtidos:

1. **Gerenciamento de Distribuidora de Produtos:** permite realizar o gerenciamento de venda de produtos para clientes. Ao realizar uma venda para um cliente, o sistema registra o funcionário que a realizou. O sistema também é responsável por controlar o estoque dos produtos e enviar uma ordem de compra para os fornecedores;
2. **Gerenciamento de Sistema Acadêmico:** permite apenas o gerenciamento de alunos e as disciplinas que o mesmo cursou num semestre. O aluno pode consultar seu desempenho escolar por meio de um boletim *on-line*. Este sistema não foi implementado com todas as funcionalidades que retratam sua realidade, devido à sua complexidade;
3. **Gerenciamento de Posto de Combustível:** tem por objetivo gerenciar a venda de combustível para os clientes. O sistema pode gerenciar a venda de produtos da loja de conveniência e qual funcionário atendeu o cliente. Basicamente, é o mesmo sistema do apresentado no estudo de caso 2; e
4. **Gerenciamento de Granja:** permite controlar variações dos sensores de temperatura, umidade e luminosidade. Conforme variação apresentada pelos sensores dispositivos como ventiladores, *spray* e *spots* são acionados para normalizar o ambiente. O sistema registra a data e horário das medições dos sensores em um tempo pré-estabelecido.

Estes grupos apresentaram maior maturidade para absorção das diretrizes existentes na metodologia, principalmente na fase inicial de modelagem, que requer padronização quanto aos nomes dos artefatos (classes), características (atributos) e funcionalidades (métodos). Os grupos apresentaram pouca experiência com a linguagem de programação JAVA e seus recursos.

A condução deste estudo de caso foi a mesma dos anteriores. Os grupos receberam

treinamento sobre as ferramentas de modelagem UML. Em seguida, foram introduzidas as regras de mapeamento segundo o *framework* de persistência *Hibernate* (HIBERNATE, 2009), com anotações. Este recurso prevê que o mapeamento objeto-relacional ocorra dentro do código fonte da aplicação, orientada a objetos por meio de meta-informações, que descrevem o funcionamento da classe (orientada a objetos), como uma entidade no banco de dados (modelo relacional).

Os demais passos para realização são os mesmos aos apresentados nos estudos de casos 1 e 2. No entanto, com a utilização do *framework Hibernate* (HIBERNATE, 2009) para persistência de dados, apesar da pouca experiência dos grupos, a atividade de implementação mostrou-se menos "onerosa" quanto ao tempo e esforço de programação. Essa facilidade ocorreu devido à automatização que o mesmo oferece na geração dos códigos SQL para persistência dos objetos na base de dados e vice-versa. Sobre esta característica é importante mencionar que este *framework* também utiliza recursos de reflexão computacional para recuperar as características dos objetos em tempo de execução e gerar o código SQL de maneira dinâmica.

Outra inovação neste estudo de caso foi propor aos grupos, após realizada a modelagem dos sistemas, que realizassem, de maneira manual, a identificação das funcionalidades comuns e as transformassem em serviços *Web*, para serem reutilizadas na etapa de desenvolvimento do sistema. Essa atividade teve por objetivo mostrar aos grupos que a reutilização pode ocorrer em várias etapas do processo de desenvolvimento de software. Eles passaram a consumir um serviço existente na rede e não desenvolver uma funcionalidade que pode existir no AER.

Destaca-se ainda neste estudo de caso a participação de um novo personagem presente na MDSR, o especialista de domínio, que orientou os grupos na escolha dos nomes dos artefatos na fase de confecção dos modelos e, consequentemente, na identificação das funcionalidades redundantes transformadas em serviços.

As dificuldades encontradas com a realização deste estudo de caso concentram-se, basicamente, na implementação, pois, inicialmente, os grupos apresentaram dificuldades em identificar e organizar algumas funcionalidades como serviços. O conceito de reúso de software estava assimilado, porém a visualização de como ele (serviço) seria utilizado na implementação não estava sendo assimilado pelos grupos. Esses problemas são facilmente resolvidos com treinamentos na tecnologia de implementação.

Outra constatação importante neste estudo de caso refere-se à minimização dos problemas quanto aos nomes gerados na etapa de elaboração dos modelos. Este fato está relacionado a dois fatores importantes: a maturidade das equipes, por serem alunos de pós-graduação com pouca experiência na tecnologia empregada no desenvolvimento, porém com uma visão mais ampla quanto ao entendimento dos processos e metodologias de desenvolvimento de software.

Este estudo de caso teve por objetivo avaliar as mudanças inseridas na MDSR, com a realização do estudo de caso 2 e com a adição de recursos tecnológicos: *framework hibernate* e os serviços *Web*. A seguir, Seção 6.2.3, são apresentados os resultados do estudo de caso 4.

6.2.4. Estudo de Caso 4

O quarto estudo de caso foi realizado em 2008, com alunos de graduação. Eles foram organizados em 12 grupos e cada um deles com um sistema a ser desenvolvido. Cada grupo foi responsável por elaborar um documento de requisitos com as mesmas seções do estudo de caso 1. Como resultado, os seguintes documentos de requisitos foram obtidos:

1. **Gerenciamento de Imobiliária de Imóveis:** permite o gerenciamento (venda e locação) de imóveis dos proprietários para um inquilino. Ao efetuar uma venda/locação o sistema deve registrar o imóvel, o inquilino, o fiador e o funcionário que realizou a operação. O sistema deve registrar os fiadores para a venda/locação

dos imóveis;

2. **Gerenciamento de uma Pizzaria:** deve registrar o pedido (Pizza(s)) de um cliente. O sistema também é responsável por controlar o estoque dos produtos e quantificar cada item para que uma pizza seja montada. Ao terminar uma pizza, realiza-se o procedimento de embalagem, para que a mesma seja encaminhada aos entregadores para encaminhar às residências dos clientes;
3. **Gerenciamento de uma Oficina Mecânica:** deve registrar o orçamento de um serviço para um cliente. Quando o cliente aprovar o orçamento, o mesmo será encaminhado para execução. O sistema também é responsável por gerenciar o número de peças e o estoque das mesmas na realização dos orçamentos e serviços;
4. **Agência de Turismo:** deve realizar a reserva de pontos turísticos e hotéis para os clientes. O sistema possui tanto pontos turísticos quanto hotéis cadastrados nele. Ao selecionar uma locação desejada, o sistema deve informar a disponibilidade, assim como o valor a ser pago no período da estadia;
5. **Gerenciamento Escolar:** permite o gerenciamento de matrícula de alunos em turmas. Além disso, deve associar o professor com a aula que irá lecionar. O sistema armazena as notas dos alunos nas disciplinas (boletim escolar);
6. **Gerenciamento de Biblioteca:** permite o gerenciamento de empréstimos de materiais (livros, revistas, CDs, etc) da biblioteca para os alunos de uma escola. Ao registrar um empréstimo, o sistema deve emitir um comprovante que contém o nome do aluno e os itens que estão sendo emprestados. O atraso na devolução do empréstimo gera uma penalização em multa. Os alunos em débito não podem efetuar novos empréstimos;
7. **Gerenciamento de Posto de Combustível:** tem por objetivo gerenciar a venda de combustível para os clientes. O sistema pode gerenciar a venda de produtos da loja

de conveniência e qual funcionário atendeu o cliente. Basicamente, este é o mesmo sistema do apresentado nos estudos de casos 2 e 3;

8. **Gerenciamento de uma Locadora de Vídeo:** permite realizar o empréstimo dos itens da locadora (fitas, DVD's, CD's, cartuchos) para os clientes. Ao registrar um empréstimo, o sistema deve emitir um comprovante que contém o nome do cliente e os itens que estão sendo emprestados. O atraso na devolução do empréstimo gera uma penalização em multa. Os clientes em débito não podem efetuar novos empréstimos;
9. **Gerenciamento de uma Clínica Médica:** permite o gerenciamento de consultas em um consultório médico. Para realizar o agendamento de uma consulta, o sistema pode solicitar o convênio médico ou não e, ainda, associa o funcionário que registrou a consulta. Finalmente, qual médico atendeu o paciente e qual foi sua prescrição. Este sistema possui as mesmas funcionalidades do apresentado no estudo de caso 2;
10. **Venda de Produtos:** permite realizar o gerenciamento de venda de produtos para clientes. Ao realizar uma venda para um cliente, o sistema registra o funcionário que a realizou. O sistema também é responsável por controlar o estoque dos produtos e enviar uma ordem de compra para os fornecedores;
11. **Gerenciamento de uma Transportadora:** permite realizar o gerenciamento de transportes por uma transportadora. O cliente faz a solicitação de envio de uma mercadoria de um ponto de origem a um ponto de destino. O sistema deve otimizar a entrega para o cliente e para a transportadora, retornando o melhor veículo que poderá efetuar o frete, dentro das regras e condições de custo e prazos de entrega. Este sistema possui algumas particularidades em relação ao apresentado no estudo de caso 2;
12. **Controle de Passagens Aéreas:** permite gerenciar a venda de passagens aéreas,

realizando alocação dos passageiros em voos comerciais ou fretados, escalas e rotas de um ponto de origem a um ponto de destino.

Apesar dos grupos de trabalho terem recebido o treinamento na fase inicial de modelagem, alguns problemas foram identificados: nomes dos artefatos (classes), características (atributos) e funcionalidades (métodos) não foram padronizados conforme estabelece as diretrizes da MSDR. Apesar de existir um especialista de domínio responsável por gerenciar essa fase, os nomes foram inseridos com abreviaturas, siglas e junção por meio do caractere (_). Também houve muita confusão quanto ao uso dos nomes com as iniciais em letra maiúscula e plural. Sendo assim, várias revisões foram realizadas para adequar os projetos à especificação da MDSR.

Outro fator significativo na condução do estudo de caso, deu-se na fase de treinamento da linguagem JAVA. Pelas dificuldades observadas nos estudos de casos anteriores, *frameworks* adicionais não foram abordados como recurso de desenvolvimento neste estudo de caso. Essa decisão foi tomada, pois julgou-se, naquele momento, que as dificuldades de aprendizado seriam significativas para utilizá-las de maneira adequada. Sendo assim, poderiam ocorrer atrasos na execução dos cronogramas e outras fases poderiam não ser executadas.

Alguns grupos tiveram grandes dificuldades na implementação dos artefatos e outros facilidades. Isso se deve ao conhecimento prévio na linguagem de programação JAVA e metodologias de desenvolvimento de software. Alguns grupos tiveram dificuldades para otimizar a implementação, principalmente em utilizar recursos das IDEs na geração automática de código fonte.

A reutilização de software foi utilizada de uma maneira diferenciada neste estudo de caso. Quando os grupos estavam elaborando os modelos, algumas funcionalidades foram julgadas necessárias e identificadas por uma pessoa externa (responsável por conduzir o estudo

de caso). Em seguida, essa pessoa implementou essas funcionalidades, utilizando o recurso de distribuição por chamadas de métodos remotos. Na etapa de implementação, apenas as interfaces remotas dessas funcionalidades foram fornecidas aos grupos, que deveriam utilizá-las em seus respectivos projetos.

De acordo com a conduta apresentada, constatou-se que alguns grupos implementaram facilmente a integração dos métodos remotos com a aplicação que estavam desenvolvendo. Em contrapartida, outros grupos apresentaram muita dificuldade em utilizar esse recurso. A conclusão obtida sobre este fato está relacionada à experiência que alguns membros dos grupos, que obtiveram sucesso, possuíam em relação à programação de aplicações distribuídas.

Outro fator significativo para a realização deste estudo de caso foi a avaliação do subsistema de reconfiguração sobre ambientes distribuídos, que utilizam chamadas de métodos remotos. Os métodos foram reconfigurados em tempo de execução, sem a percepção da aplicação cliente, conforme apresentado no estudo de caso 2 (Figura 48).

Diante dos fatos apresentados neste estudo de caso e com a experiência adquirida nos estudos de casos anteriores, pode-se concluir que a aplicação da MDSR está relacionada a um "impacto cultural" entre os desenvolvedores, ou seja, deve haver uma mudança na maneira de pensar dos membros das equipes de desenvolvimento de software, para que as atividades previstas numa metodologia sejam executadas sem comprometer as atividades posteriores. Sendo assim, como medida imediata importante para a realização deste trabalho é necessária a elaboração de um manual de capacitação contendo as diretrizes da MDSR e suas ferramentas, recursos tecnológicos por ela suportados, além dos manuais de treinamento na linguagem de programação JAVA e *frameworks* adicionais por ela utilizados.

6.3. Considerações Finais

Com a aplicação dos estudos de casos realizados foi possível avaliar as diretrizes

propostas na MDSR, desde sua concepção inicial até a fase final, descrita no Capítulo 4. No entanto, pode-se observar que novos recursos foram adicionados como forma de facilitar o desenvolvimento das atividades por ela previstas.

Um fator de grande importância para a condução destes estudos de casos, foi a padronização nas etapas de aplicação da MDSR em todos os estudos de casos realizados. Entende-se que esse tipo de conduta auxilia a identificação de indicadores para a melhoria da MDSR e que os resultados obtidos possam oferecer melhores parâmetros quanto a aplicabilidade e tomadas de decisões.

Destacam-se nesses estudos de casos a inclusão de *frameworks* para persistência de objetos e a figura de um especialista de domínio, que contribuem, respectivamente, com a velocidade de desenvolvimento e com a padronização da informação. Este último pode ser considerado como um item vital para o subsistema de reconfiguração desenvolvido neste trabalho.

Outro fator relevante foi a aplicação do subsistema de reconfiguração, que foi utilizado tanto em aplicações que utilizaram recursos de chamadas de métodos remotos quanto em aplicações que utilizaram serviços *Web*. Em ambos os casos, a reconfiguração mostrou-se transparente à aplicação cliente, ou seja, não houve a percepção direta quanto à execução da aplicação. Finalmente, vale mencionar que este subsistema também foi testado e validado para ser utilizado em aplicações *Web* tradicional.

CAPÍTULO 7

Conclusão e Trabalhos Futuros

7.1. CONSIDERAÇÕES INICIAIS

Na busca em atender a evolução de sistemas com a utilização de recursos computacionais mais recentes, capazes de se modificar no ambiente de execução mediante a mutação dos requisitos para os quais foram concebidos, de maneira automática ou pela intervenção humana, técnicas e recursos de adaptação/reconfiguração de software foram estudados. Com isso, uma metodologia de apoio ao desenvolvimento de software reconfigurável e um ambiente de execução reconfigurável apoiado por ferramentas, também foram elaborados neste trabalho.

Este trabalho teve uma concepção inicial para a MDSR, que foi sendo lapidada no decorrer da aplicação dos estudos de casos e testes realizados em paralelo, conforme apresentou o Capítulo 6. Até chegar ao modelo final, apresentado no Capítulo 4, foram coletados e avaliados os resultados em relação à metodologia, os mecanismos de comunicação distribuídos e, principalmente, a técnica de reconfiguração que foi desenvolvida.

Este capítulo está organizado da seguinte maneira: a Seção 7.2 apresenta as conclusões obtidas com a realização deste trabalho; a Seção 7.3 as atividades futuras que podem ser desenvolvidas; a Seção 7.4 apresenta as limitações e dificuldade encontradas no desenvolvimento deste trabalho; e, na Seção 7.5, as publicações realizadas durante seu período de desenvolvimento.

7.2. Conclusões Obtidas

Como resultados obtidos do desenvolvimento deste trabalho, bem como a aplicação dos estudos de casos realizados, destacaram-se:

1. A metodologia desenvolvida mostrou-se viável para o desenvolvimento de aplicações distribuídas reconfiguráveis. Destaca-se neste item, a evolução da metodologia ocorrida ao longo da aplicação dos estudos de casos, pois novos recursos foram a ela incorporados, conforme as necessidades/problemas que foram identificadas.
2. Outro fator relevante referente à metodologia é o isolamento dos requisitos em funcionais e não-funcionais. Dessa forma, os artefatos produzidos encontram-se independentes da lógica de programação e recursos adicionais podem ser a eles incorporados, como um processo de linha de montagem de software pela Ferramenta *ReflectTools*®.
3. Destaca-se a automatização das tarefas pela Ferramenta *ReflectTools*®, pois sem ela o desenvolvimento das aplicações dependeria ainda mais dos seres humanos e, conforme relatado nos estudos de casos, vários problemas foram observados pela falta da padronização, tanto no desenho dos modelos quanto na implementação em código fonte.
4. A Técnica de Reconfiguração desenvolvida mostrou-se genérica quanto à sua

utilização em aplicações locais, com chamada de métodos remotos e com serviços *Web*. Dessa forma, permite aos desenvolvedores maior flexibilidade quanto a escolha da implementação a ser realizada e os clientes que as utilizariam.

5. Destacam-se também as metodologias de desenvolvimento de software, que podem ser empregadas no desenvolvimento softwares reconfiguráveis. Os produtos gerados por essas metodologias são encapsulados em artefatos reconfiguráveis transparentes aos clientes que os utilizam.
6. A presença do *framework Hibernate* para persistência de objetos em banco de dados relacional, também merece destaque neste trabalho. Inicialmente, pensou-se em implementar algo semelhante para geração das bases de dados e dos *scripts* SQL. Com a evolução deste trabalho e estudo aprofundado sobre seu funcionamento, comprovou-se que ele utiliza todos os princípios de reflexão computacional empregados na técnica de reconfiguração aqui desenvolvida. Além disso, pode-se afirmar que se adequa perfeitamente ao desenvolvimento proposto na metodologia, no entanto, tem como requisito fundamental o treinamento a ser realizado para que seus recursos e facilidades sejam mais bem aproveitados.
7. Outro fator interessante, constatado com a realização dos estudos de casos, foi uma comparação entre os recursos de distribuição, chamada de métodos remotos e serviços *Web*. O primeiro apresentou maior facilidade de entendimento e implementação pelos grupos, porém com restrição quanto à sua utilização, pois somente aplicações escritas em JAVA podem realizar comunicação com essas aplicações. Para eliminar essa limitação, o protocolo IIOP pode ser utilizado para garantir interoperabilidade entre clientes e servidor, no entanto, haveria um aumento significativo na complexidade da aplicação. O segundo mostrou-se de difícil entendimento quanto ao funcionamento, mas de fácil implementação pela IDE

Netbeans. Além disso, tem como vantagem a interoperabilidade, ou seja, qualquer cliente poderia "consumir" o serviço que foi implementado.

8. Os estudos de casos realizados mostraram-se bem diferenciados quanto aos resultados obtidos. Isso deve-se à maturidade dos grupos, pois aqueles que possuíam maior experiência na área de desenvolvimento de software obtiveram melhores resultados na aplicação das diretrizes da metodologia, utilização da ferramenta *ReflectTools*®, na implementação dos sistemas e na utilização de recursos adicionais, como *frameworks* de persistência de objetos, por exemplo.
9. Destaca-se a implementação por serviços, que foi utilizada praticamente em todos os subsistemas, pois são funcionalidades que são acessadas remotamente pelos desenvolvedores em "tempo real". O recurso mostrou-se de fácil desenvolvimento e integração com a Ferramenta *ReflectTools*®.
10. Outro mecanismo facilitador na implementação dos subsistemas e da Ferramenta *ReflectTools*®, foi o mapeamento de classes em arquivos XML e desses arquivos para as regras de classificação. Foi um recurso de fácil implementação, pois existem vários *frameworks* capazes de transformar objetos em XML e vice-versa. Esta foi uma característica bastante empregada no desenvolvimento deste trabalho.

7.3. Trabalhos Futuros

Como atividades futuras após a realização deste trabalho destacam-se:

1. A implementação e instanciação dos sistemas para outras linguagens de programação, como por exemplo *.Net*, *Delphi*, entre outras. Todos os sistemas desenvolvidos neste trabalho utilizaram a linguagem JAVA em 100% da implementação. As técnicas de disponibilização dos sistemas como serviço *Web*, por exemplo, podem utilizar o desenvolvimento do serviço e a técnica de reconfiguração

em JAVA, porém o cliente que consome e utiliza uma aplicação reconfigurável pode ser escrito em uma linguagem de programação qualquer, desde que ela seja capaz de consumir esse tipo de aplicação.

2. Uma atividade consequente ao item anterior (1) é a aplicação de um estudo de caso envolvendo os serviços instanciados e os novos sistemas que seriam desenvolvidos, reutilizando partes desses sistemas, funcionalidades como serviços ou chamadas de métodos remotos. O objetivo seria explorar a interoperabilidade das linguagens de programação e do ambiente de desenvolvimento reconfigurável na confecção de novas aplicações.
3. Outra atividade futura corresponde a confecção de um manual de utilização da metodologia e da Ferramenta *ReflectTools*®, pois enquanto os treinamentos estavam sendo desenvolvidos, os grupos de trabalho mostraram entendimento do processo. Porém, quando as orientações e execuções das atividades passaram a ser executadas por eles, problemas começaram a ser apresentados. Um guia de consulta *on-line* poderia ter evitado uma série de problemas simples, porém vitais para a reconfiguração da aplicação.
4. Instanciar a técnica de reconfiguração para outras linguagens que possuem o recurso de reflexão computacional, para realizar estudos de casos comparativos quanto ao desempenho das aplicações e do desenvolvimento das novas.
5. Aprimorar os mecanismos de consultas, implementar outros que permitam realizar busca por artefatos com resultados mais significativos, sem a inferência de um conjunto de dados de entrada e um resultado final exato.
6. Implementação de sistema de agentes móveis e inteligentes para localização e escolha da melhor solução. No estado atual de desenvolvimento foram utilizados serviços *Web* como mecanismos de distribuição dos subsistemas e utilização dos

recursos por eles disponibilizados, principalmente, a consulta remota dos artefatos nos repositórios.

7. Elaborar um subsistema de consulta e cadastro, utilizando ontologia para classificação e recuperação dos artefatos de software no AER.

A seguir, na Seção 7.4, são apresentadas as limitações e dificuldades encontradas para a realização deste trabalho.

7.4. Limitações e Dificuldades

Esta seção apresenta as dificuldade e limitações encontradas com a realização deste trabalho. Dentre elas, podem ser destacadas:

1. Uma das dificuldades enfrentadas no desenvolvimento deste trabalho foi a necessidade de investigação em diversas áreas do conhecimento, pois o trabalho desenvolvido envolve várias áreas da computação. No entanto, pelo próprio objetivo do trabalho, em propor uma metodologia e um ambiente para o desenvolvimento de software reconfigurável apoiado por ferramentas, vários recursos tecnológicos e metodologias tiveram que ser estudos para a realização deste.

2. Outra dificuldade a ser relatada neste trabalho é a falta de maturidade nos recursos tecnológicos e linguagens de programação. No desenvolvimento deste, observou-se que novos recursos foram surgindo com o passar do tempo e, consequentemente, foram incorporados gradativamente às diretrizes da metodologia e à ferramenta *ReflectTools®.*

3. Para validar e verificar os processos estabelecidos na MDSR, foram realizados ao longo dos anos 4 estudos de casos e, como dificuldade, pode-se citar a falta de padronização das informações quando os artefatos estão sendo desenvolvidos.

Apesar de existirem padrões, tanto estabelecidos na MDSR quanto nas linguagens de programação (JAVA), houveram problemas na elaboração dos modelos de classes e de dados. Isso motivou a elaboração de um manual de instrução, como trabalho futuro, para a aplicação da MDSR. A ferramenta *ReflectTools*® também requer a elaboração de um manual de utilização de software, pois houveram dificuldades entre os grupos quanto à sua utilização, mesmo após terem recebido treinamento;

4. Como limitação pode-se dizer que a MDSR foi elaborada para atender, inicialmente, sistemas desenvolvidos em JAVA. Além disso, a ferramenta *ReflectTools*® e o AER foram projetados para atender sistemas desenvolvidos nesta linguagem de programação. O subsistema de reconfiguração, núcleo deste trabalho, está limitado a gerenciar a reconfiguração em artefato desenvolvidos em JAVA somente;
5. A busca de informações por artefatos de software nos diversos domínios de atuação é realizada de maneira estática via *web services,* pois houve dificuldade de integração entre a implementação desenvolvida neste trabalho e o *framework* de desenvolvimento de agentes disponíveis quando este texto foi elaborado. A varredura de domínios é realizada, de maneira similar, a uma tabela de rota.

A seguir, na Seção 7.5, são apresentados os trabalhos desenvolvidos e publicados com o desenvolvimento deste trabalho.

7.5. Lista de trabalhos publicados

Esta seção apresenta os artigos publicados durante o período de desenvolvimento deste trabalho. São eles:

1. AFFONSO, F. J.; ROSSI, E. G.; RODRIGUES, E. L. L., **Reconfiguração de objetos distribuídos utilizando regras para classificação e recuperação de**

informações. Multiciência (ASSER), v. 9, p. 38-53, 2008;

2. ROSSI, E. G.; AFFONSO, F. J.; RODRIGUES, E. L. L., **Ambiente de apoio ao desenvolvimento de software reconfigurável baseado em agentes de software e busca inteligente de informações.** In: 7 Congresso Nacional de Pesquisadores, 2008, São Carlos. 7 Congresso Nacional de Pesquisadores. São Carlos : UNICEP, 2008. v. 7.;
3. AFFONSO, F. J.; RODRIGUES, E. L. L., **Diretrizes para desenvolvimento de software reconfigurável**. São Carlos, 2008 (Relatório Técnico);
4. AFFONSO, F. J.; RODRIGUES, Evandro L. L.,. **Processamento de Imagem Distribuído e Orientado a Serviço usando JAVA**. 3º Workshop de Visão Computacional, 2007;
5. AFFONSO, F. J.; RODRIGUES, E. L. L., **Metodologia para desenvolvimento de software para um ambiente de execução reconfigurável**. Multiciência (ASSER), v. 8, p. 114-128, 2007;
6. AFFONSO, F. J.; ROSSI, E. G.; RODRIGUES, E. L. L., **Armazenamento de Imagens Médicas utilizando Serviço de Diretórios**. In: 5º Congresso Nacional de Pesquisadores, 2006, São Carlos. 5º Congresso Nacional de Pesquisadores, 2006;
7. AFFONSO, F. J.; ROSSI, E. G.; RODRIGUES, E. L. L., **Reconfiguração de Aplicações Distribuídas utilizando RMI-Remote Method Invocation.** In: 5º Congresso Nacional de Pesquisadores, 2006, São Carlos. 5º Congresso Nacional de Pesquisadores, 2006;
8. QUEIROZ, A. C; OLIVETE, C.; AFFONSO, F. J. ; RODRIGUES, E. L. L., **Estimação da idade óssea baseada no método de EKLOF & RINGERTZ usando redes neurais artificiais.** 2º Simpósio de Instrumentação e Imagens Médicas, São Pedro - SP, 2005;

9. AFFONSO, F. J. ; QUEIROZ, A.C.; OLIVETE, C.; RODRIGUES, E. L. L., **Armazenamento e busca de imagens médias em sistemas voltados para WEB com centralização da informação.** 1º Workshop de Visão Computacional, 2005;

10. OLIVETE, C.; QUEIROZ, A. C.; AFFONSO, F. J. ; RODRIGUES, E. L. L., **Estimativa Simplificada da idade óssea - uma proposta de metodologia de automatização do processo para auxilio do diagnóstico.** 1º Workshop de Visão Computacional, 2005;

11. QUEIROZ, A. C.; OLIVETE, C.; AFFONSO, F. J. ; RODRIGUES, E. L. L., **Metodologia para extração de característica automáticas da mão usando assinatura vertical.** 1º Workshop de Visão Computacional, 2005.

Referências

AHMED, K. Z. & UMRYSH, C. E. **Desenvolvendo aplicações comerciais em java com j2ee e uml.** 1 ed. ISBN: 85-7393-240-6. ed. Ciência Moderna, 2002.

AKKAWI, F. et. al. **Dynamic weaving for building reconfigurable software systems**, in Proceeding of the Workshop of the Midwest Society for Programming Languages and systems, Bloomington, Indiana, 2002.

ANDERSON, R. et al. **Professional xml.** 1 ed. ISBN: 85-7393-116-7. ed. Ciência Moderna, 2001.

AOYAMA, M. **New age of software development:** how component-based software engineering changes the way of software development?. Proceedings of International Workshop on Component-Based Software Engineering, Kyoto, Japan, Abril 1998.

APACHE-AXIS. **Site oficial do projeto apache axis.** Disponível em: <http://ws.apache.org/axis/>, Acesso em: 04 de fev. de 2009.

ARSHAD, N. et. al. **Deployment and dynamic reconfiguration planning for distributed software systems**, in Proceedings 15th IEEE International Conference on Tools with Artificial Intelligence, 2003.

ASPECTJ, **Site oficial do projeto aspectj.** Disponível em: <http://www.eclipse.org/aspectj/>, Acesso em: jan. de 2009.

AZZURRI, C. **Site oficial do plugin azzurri clay.** Disponível em:

<http://www.azzurri.jp/en/clay/index.html>, Acesso em: 23 de fev. de 2009.

BEDER, D. M. **Uma arquitetura de software baseada em padrões de projeto para o desenvolvimento de aplicações confiáveis.** Tese de Doutorado. IC-UNICAMP. Campinas-SP, 2001.

BERNARDES, M. C. **Avaliação do uso de agentes móveis em segurança computacional.** Dissertação de Mestrado. ICMC/USP. São Carlos-SP, 1999.

BIGUS, J. P. & BIGUS, J. **Constructing intelligent agents using java.** 2 ed. ISBN: 0-47-39601-X. New York: Wiley Computer Publishing, 2001.

BOND, M. et al. **Aprenda j2ee em 21 dias.** Tradução João Eduardo Nobrega 1 ed. ISBN: 85.346.1488-1. São Paulo: Person Education do Brasil, 2003.

BOOCH et al. **UML: guia do usuário.** 2 ed. Editora Campus, 2005.

BROWNE, P. **Give your business logic a framework with drools.** Disponível em: <http://www.onjava.com/pub/a/onjava/2005/08/03/drools.html> Acesso em: 11 de fev. 2009a.

BROWNE, P. **Using drools in your enterprise java application.** Disponível em <http://www.onjava.com/pub/a/onjava/2005/08/24/drools.html> Acesso em: 11 de fev. 2009b.

CASTRO, P. A. D. **Um paradigma baseado em algoritmos genéticos para o aprendizado de regras fuzzy.** Dissertação de Mestrado. PPGCC/DC-UFSCAR. São Carlos-SP, 2004.

CATALYSIS. **Site oficial do método catalysis**. Disponível em: <http://www.catalysis.org> Acesso em: 06 de mar. 2009.

CBDI. **Site oficial do service oriented architecture practice portal.** Disponível em:

<http://www.cbdiforum.com> Acesso em: 23 de jan. 2009.

CHEN, X. **Extending rmi to support dynamic reconfiguration of distributed systems.** In Distributed Computing Systems, ISBN:0-7695-1585-1, IEEE Computer Society, Washington, DC, USA, 2002.

CHIBA, S. et al. **Using hotswap for implementinf Dynamic AOP Systems.** In ECOOP'03 Workshop on Advancing the State of the Art in Runtime Inspection (ASARTI), 2003.

CHIBA, S. **Getting started with javassist** Disponível em <http://www.csg.is.titech.ac.jp/~chiba/javassist> Acesso em: 23 de jan. 2009.

CHIBA, S. **Javassist - a reflection-based programming wizard for java.** In Proceedings of the ACM OOPSLA'98 Workshop on Reflective Programming in C++ and Java, Outubro, 1998.

CHIBA, S. **Load-time structural reflection in java.** In ECOOP 2000 - Object-Oriented Programming, LNCS 1850, Springer Verlag, page 313-336, 2000.

D'SOUZA, D. F., WILLS, A. C. **Objects, components, and frameworks with UML:** the catalysis(sm) approach. 1 ed. ed. Addison-Wesley Object Technology Series. 1998.

DANTAS, A. & BORBA, P. **Adaptability aspects: an architectural pattern for structuring adaptive applications with aspects**. In Third Latin American Conference on Pattern Languages of Programming, SugarLoafPLoP?'03, Porto de Galinhas, PE, Brazil, pages 12-15, Agosto, 2003.

DANTAS, A. et al **Using aspects to structure small devices applications**. In First Workshop on Reuse in Constrained Environments at the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'03, Anaheim,

CA, USA, Outubro, 2003.

DEITEL, H. M. & DEITEL, P. J. **Java, como programar**. tradução Edson Furnankiewicz. 3. ed. Porto Alegre : Bookman. 2001.

DEITEL, H. M. & DEITEL, P. J. **Java, como programar**. tradução Edson Furnankiewicz. 6. ed. São Paulo: Pearson Prentice Hall. 2005.

DEITEL, H. M. et al. **Advanced java 2 plataform how to program.** ISBN: 0130895601. 1. ed., ed. Prentice Hall. 2001.

DONG, W. **Dynamic reconfiguration method for web service based on policy**. International Symposium on Electronic Commerce and Security, 2008.

ECLIPSE. **Site oficial da ferramenta eclipse**. Disponível em: <http://www.eclipse.org> Acesso em: 29 de jan. 2009.

EGE, R. K. **Storing java objects in any database** In Technology of Object-Oriented Languages and Systems, Pages: 312-321, 1999a.

EGE, R. K. **Object-oriented database access via reflection.** In Computer Software and Applications Conference (COMPSAC '99), Pages: 36-41, 1999b.

ELMASRI, R. E. & NAVATHE, S. **Sistemas de Banco de Dados**, ISBN: 8588639173, Ed. Addison-Wesley, 2005.

ERL, T. **Service-Oriented Architecture (SOA): Concepts, Technology, and Design.** ISBN: 0131858580. Ed. Prentice Hall, 2005.

ERL, T. **Service-Oriented Architecture: A Field Guide to Integrating XML and Web**

Services, ISBN: 0131428985. Ed. Prentice Hall, 2004.

FERNANDES, A. P. & LISBÔA, M. L. B. **Reflective Implementation of an object recovery design pattern.** VII Congresso Argentino de Ciencias de la Computación. El Calafate, República Argentina, 2001.

FERNANDES, A. P. **Reflexão computacional**. Disponível em: <http://attila.urcamp.tche.br/~acauan/art_ccei_rc.html> Acesso em: 26 de jan. 2009.

FILHO, S. F. P. **Avaliação de ambientes servidores para agentes móveis.** Dissertação de Mestrado. ICMC/USP. São Carlos-SP, 2001.

FORMAN, I. R. & FORMAN, N. **Java reflection in action**. ISBN: 1932394184. 1 ed. United State: Manning Publication, 2004.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. ISBN-10: 0201633612. Ed. Addison-Wesley Professional Computing Series, 1994.

GIARRATANO, J. C. & RILEY, G. D. **Expert systems: principles and programming**. 3 ed. ISBN 0-534-95053-1. Boston: PWS Publishing Company, 1998.

GOMAA, H. & HUSSEIN, M., **Software reconfiguration patterns for dynamic evolution of software architectures**, in Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture - WICSA, 2004.

GRADERICKI, J. & LESIECKI, N. **Mastering aspectj: aspect-oriented programming in java** 1 ed. ISBN: 0-471-4310104-4. Indiana: Wiley Publishing, 2003.

GROSSO, W. **Java rmi -** designing and building distributed application. 1. ed. ISBN: 1565924525. ed. O'Reilly Media. 2001.

GUSTAVSSON, J., et. al. **Runtime evolution as an aspect**, in First International Workshop on Foundations of Unanticipated Software Evolution, Barcelona, Espanha 2004.

HAYKIN, S. **Redes neurais: princípios e práticas.** tradução: Paulo Martins Angel, 2 ed. ISBN: 85-7307-718-2. Porto Alegre: Bookman, 2001.

HEINEMAN, G. **Adaptation of software components**, 2nd International Workshop on Component-Based Software Engineering, in association with the 21st International Conference on Software Engineering, Los Angeles, CA, Junho. 1999.

HEINEMAN, G. **An evaluation of component adaptation techniques**, Disponível em <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.7360> Acesso em: 02 de mar. 2009.

HIBERNATE. **Site oficial do projeto hibernate.** Disponível em: <http://www.hibernate.org> Acesso em: 15 de jan. 2009.

IBM-RATIONAL. **Site oficial da ibm-rational**. Disponível em <http://www-01.ibm.com/software/br/rational/>. Acesso em: 18 de fev. 2009.

IBM. **Site oficial da ibm**. Disponível em <http://www.ibm.com>. Acesso em: 18 de jan. 2009.

JACOBSON, I. et al. **Object-oriented software engineering -** a use case driven approach. Addison - Wesley. ACM Press. p. 289-312. 1997.

JAVAWORLD. **Site oficial do javaworld -** rmi-iiop over iiop. Disponível em <http://www.javaworld.com/javaworld/jw-12-1999/jw-12-iiop.html> Acesso em: 18 de fev. 2009.

JAXB. **Site oficial do projeto jaxb**. Disponível em: <https://jaxb.dev.java.net/>, Acesso em: 20

de fev. de 2009.

JAVASSIST. **Site oficial do projeto javassist**, Disponível em: <http://www.csg.is.titech.ac.jp/~chiba/javassist/>, Acesso em: 01 de abr. de 2009.

JBOSS-AOP. **Site oficial do projeto jboss-aop -** jboss aop - aspect-oriented framework for java. jboss aop reference documentation Disponível em: <http://labs.jboss.com/portal/jbossaop/docs/1.5.0.GA/docs/aspect-framework/reference/en/html/index.html> Acesso em: 07 de fev. 2009.

JESS. **Site oficial do projeto jess.** Disponível em <http://www.jessrules.com/> Acesso em: 11 de mar. 2009.

JINI. **Site oficial do JINI.** Disponível em <http://jan.newmarch.name/> Acesso em: 18 de jan. 2009.

JPA, **Site oficial da API JPA.** Disponível em <http://java.sun.com/javaee/technologies/persistence.jsp> Acesso em: 20 fev. de 2009.

JUNIT. **Site oficial do projeto de teste junit.** Disponível em:<http://www.junit.org>, Acesso em: 04 de fev. de 2009.

KASTEN, E. P. et al. **Separating introspection and intercession to support metamorphic distributed systems**. In Proceedings of the 22nd International Conference on Distributed Computing Systems ICDCS, , 2002.

KHOSLA, R. & DILLON, T. **Engineering intelligent hybrid multi-agent systems.** ISBN: 079239982X. Boston, Dordrecht, London: Kluwer Academic Publishers, 1997.

KICZALES, G., et al. **An overview of aspectJ.** 15th European Conference on Object Oriented

Programming (ECOOP). Springer, Junho, 2001.

KICZALES, G., et al. **Aspect-oriented programming.** 15th European Conference on Object Oriented Programming (ECOOP). Springer, Junho, 1997.

KIM, D. K. & BOHNER, S. **Dynamic reconfiguration for java applications using aop,** Southeastcon - IEEE, 2008.

KIM, J. A. et al. **Component adaptation using pattern components.** In Systems, Man, and Cybernetics, Tucson, AZ, USA, 2001.

KLEINODER, J. & GOLM, M. **MetaJava: an efficient run-time meta architecture for JavaTM** In Object-Orientation in Operating Systems, 1996.

KON, F. & CAMPELL, R. H. **Supporting automatic configuration of component-based distributed systems.** In 5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'99), páginas: 175-187. San Diego, 1999.

KRAFZIG, D. et al. **Enterprise SOA**. service oriented architecture best practices. 1 ed. Prentice Hall, 2005.

KUCHARMA, P., **Software architecture design patterns in java.** ISBN: 0-8493-2142-5. Ed. Auerbach, 2004.

LANCE, D. B. & OSHIMA, M. **Progrmming and deploying java mobile agents with aglets.** 2 ed. ISBN 0-201-32582-9. Massachusetts: Addison Wesley, 1998.

LISBÔA, M. L. B. **Reflexão computacional no modelo de objetos.** In Tutorial do II Simpósio Brasileiro de Linguagens de Programação (SBLP), Campinas, São Paulo, Setembro, 1997.

LUCENA, P. **SemanticAgent, uma plataforma para o desenvolvimento de agentes inteligentes**. Dissertação de Mestrado. ICMC/USP. São Carlos-SP, 2003.

MAES, P. **Concepts and experiments in computational reflection.** ACM SIGPLAN Notices, Orlando: ISSN:0362-1340. 1987.

MCKINLEY, P. K. et al. **Composing adaptive software.** IEEE Computer, Volume 7, pags. 56-64, 2004.

MEDEIROS, E. S. **Desenvolvendo software com uml 2.0.** ISBN: 85-346-1529-2. São Paulo: Pearson Makron Books, 2004.

MILADI, M.N. et. al. **A uml rule-based approach for describing and checking dynamic software architectures,** International Conference on Computer Systems and Applications - AICCSA, 2008.

NAKAGAWA, E. Y, **Uma contribuição ao projeto arquitetural de ambientes de engenharia de software**, Tese de Doutorado ICMC-USP, 2006.

NETBEANS-UML, **Site oficial da ferramenta *Netbeans* com plugin UML.** Disponível em: <http://www.netbeans.org/features/uml/index.html>. Acesso em: 12 de fev. de 2009.

NETBEANS, **Site oficial da ferramenta *Netbeans*.** Disponível em:<http://www.netbeans.org> Acesso em: 20 de mar. de 2009.

OASIS-SOA, **Site oficial do grupo oasis-organization for the advancement of structured information standards.** Disponível em: <http://www.oasis-open.org/committees/tc_cat.php?cat=soa>. Acesso em: 18 de fev. de 2009.

OMG. **Site oficial da omg**. Disponível em <http://www.omg.org>. Acesso em: 18 de fev. 2009.

OMONDO. **Site oficial do plugin uml omondo.** Disponível em: <http://www.omondo.com> Acesso em: 07 de jan. 2009.

OPENORB, **Site oficial do projeto openorb community.** Disponível em: <http://openorb.sourceforge.net>. Acesso em 11 de jan. de 2009.

OREIZY, P. & TAYLOR, R.N. **On the role of software architectures in runtime system reconfiguration**, in Proceedings., Fourth International Conference on Configurable Distributed Systems, 1998.

PFLEEGER, S. L. **Engenharia de software, teoria e prática**, ISBN: 85-87918-31-1, 2. ed. Person Education, 2004.

PIRES, M. G. **Aprendizado genético de funções de pertinência na modelagem nebulosa**. Dissertação de Mestrado. PPGCC/DC-UFSCAR. São Carlos-SP, 2004.

PRESSMAN, R. **Engenharia de software.** ISBN 85-86804-57-6. 5. edição. ed. Mc Graw Hill Interamericana do Brasil. 2006.

PRESSMAN, R. **Software engineering.** ISBN 0-07-285318-2. 6. edição. ed. Mc Graw Hill, 2005.

PREVITALI, S. C. & GROSS, T. R. **Dynamic updating of software systems based on aspects**, in Proceedings of the 22nd IEEE International Conference on Software Maintenance, Pennsylvania, USA, 2006.

PREVITALI, S. C. **Dynamic updates: another middleware service?** in Proceedings of the 1st workshop on Middleware-application interaction: in conjunction with Euro-Sys 2007, Lisboa, Portugal, 2007.

PROCTOR, M. et al. **Drools documentation.** Disponível em: <http://downloads.jboss.com/drools/docs/4.0.7.19894.GA/html/index.html> Acesso em: 12 de fev. 2009.

REZENDE, A. M. P. & SILVA, C. C. **Programação orientada a aspectos em java.** ISBN: 85-7452-212-0. 1 ed. Rio de Janeiro : Brasport, 2005.

REZENDE, S. O. **Sistemas inteligentes: fundamentos e aplicações.** 1 ed. ISBN 85-204-1683-7. Barueri, São Paulo: Manole, 2003.

RICHMOND, M., NOBLE, J. **Reflections on remote reflection.** In Proceedings of the 24th Australasian Conference on Computer Science, ISSN:1530-0900, ACM International Conference Proceeding Series; Vol. 11, 2001.

RUSSEL, S. NORVIG, P. **Artificial intelligence: a modern approach.** 2 ed. ISBN 0-13-790395-2. New Jersey: Prentice Hall, 2003.

SIEGEL, J. **Corba 3 fundamentals and programming.** 2. ed. John Wiley Consumer. 2000.

SIEGEL, J. **Corba fundamentals and programming.** John Wiley and Sons. 1996.

SILVA, F. J. S. **Adaptação dinâmica de sistemas distribuídos.** Tese de Doutorado. IME-USP, São Paulo-SP, 2003.

SILVA, F. J. S. et al. **Desenvolvendo software adaptável para computação** móvel In Terceiro Workshop de Comunicação sem Fio (WCSF 2001), Recife, 2001.

SILVA, F. J. S. et al. **Dynamic Adaptation of Distributed Systems.** In Object-Oriented Technology ECOOP 2002 Workshop Reader, LNCS 2548, Malaga, Espanha, 2002.

SOMMERVILLE, Y. **Engenharia de software,** ISBN: 8588639289, 8 ed. Person Education,

2008.

STELTING, S. & MAASSEN, O., **Applied java patterns.** ISBN: 0-13-093538-7. 1 ed. California: Sun Microsystems Press, 2002.

SUN. **Site oficial da sun microsystem.** Disponível em:<http://www.sun.com>, Acesso em: 04 de fev. de 2009.

SUN-APPLICATION-SERVER. **Site oficial do servidor sun application server.** Disponível em:<http://www.sun.com/software/products/appsrvr/>, Acesso em: 04 de fev. de 2009.

SUN-DEVELOPERS. **The lifecycle of an rmi server and dynamic class loading in rmi.** Disponível em <http://java.sun.com/developer/JDCTechTips/2001/tt0227.html> Acesso em: 07 de mar. 2009.

SUN-JAVA-IDL. **Site oficial da sun microsystem**. Disponível em <http://java.sun.com/j2se/1.5.0/docs/guide/idl/index.html>. Acesso em: 18 de jan. 2009.

SUN-JAVA. **Site oficial da sun microsystem**. Disponível em <http://java.sun.com/javase/6/docs/>. Acesso em: 18 de mar. 2009.

SUN-JAVABEANS. **Site oficial da sun microsystem.** Disponível em: <http://java.sun.com/j2se/1.5.0/docs/guide/beans/index.html> Acesso em: 07 de mar. 2009.

SUN-JEE. **Site oficial da sun microsystem.** Disponível em: <http://java.sun.com/javaee/technologies/javaee5.jsp> Acesso em: 07 de jan. 2009.

SUN-REFLECT. **Site oficial da sun microsystem.** Disponível em: <http://java.sun.com/javase/6/docs/technotes/guides/reflection/index.html> Acesso em: 07 de fev. 2009.

SUN-RMI-IIOP. **Site oficial da sun microsystem**. Disponível em <http://java.sun.com/javase/6/docs/technotes/guides/rmi-iiop/index.html>. Acesso em: 10 de fev. 2009.

SUN-RMI. **Site oficial da sun microsystem**. Disponível em <http://java.sun.com/javase/6/docs/technotes/guides/rmi/index.html>. Acesso em: 18 de fev. 2009.

TATSUBORI, M. et al. **A bytecode translator for distributed execution of "legacy"java software.** In ECOOP 2001 -- Object-Oriented Programming, LNCS 2072, Springer Verlag, 2001.

TAVARES, D. M. **Avaliação de técnicas de captura para sistemas detectores de intrusão.** Dissertação de Mestrado. ICMC/USP. São Carlos-SP, 2002.

UML. **Site oficial da UML - Unified Modeling Language**, Disponível em: <http://www.uml.org/>, Acesso em: 01 de abr. de 2009.

W3C-SOA. **Site oficial do w3c, world wide web consortium -** web services architecture. Disponível em <http://www.w3.org/2002/ws/> Acesso em: 10 de fev. 2009.

WATSON, M. **Intelligent java applications for the internet and intranets.** 1a ed. ISBN: 1-55860-420-0. San Francisco, California: Morgan Kaufmann Publishers, 1997.

WATSON, M. **Practical artificial intelligence in java.** Disponível em: <http://www.markwatson.com/opencontent/javaai_lic.htm> Acesso em: 02 de mar. 2009.

WEISS, G. M. **Adaptação de Componentes de Software para o Desenvolvimento de Sistemas Confiáveis.** Dissertação de Mestrado. IC-UNICAMP, Campinas-SP 2001.

WERNER, C. M. L. & BRAGA, R. M. M. **Desenvolvimento baseado em componentes.** SBES - Simpósio Brasileiro de Engenharia de Software. Universidade Federal do Rio de Janeiro,

2000.

XML, **Site oficial do projeto xml.** Disponível em: <http://www.w3c.org/XML/>, Acesso em: mar. de 2009.

YANG, Z. et. al. **An Aspect-Oriented Approach to Dynamic Adaptation**, in Proceedings of the first workshop on Self-healing systems, SC, USA, 2002.

UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

METODOLOGIA PARA DESENVOLVIMENTO DE SOFTWARE RECONFIGURÁVEL APOIADA POR FERRAMENTAS DE IMPLEMENTAÇÃO: UMA APLICAÇÃO EM AMBIENTE DE EXECUÇÃO DISTRIBUÍDO E RECONFIGURÁVEL

FRANK JOSÉ AFFONSO

SÃO CARLOS - SP

2009

FOLHA DE JULGAMENTO

Candidato: Bacharel **FRANK JOSÉ AFFONSO**

Tese defendida e julgada em 26/05/2009 perante a Comissão Julgadora:

APROVADO

Prof. Dr. **EVANDRO LUÍS LINHARI RODRIGUES (Orientador)**
(Escola de Engenharia de São Carlos/USP)

APROVADO

Prof. Associado **IVAN NUNES DA SILVA**
(Escola de Engenharia de São Carlos/USP)

APROVADO

Profª. Drª. **ELISA YUMI NAKAGAWA**
(Instituto de Ciências Matemáticas e de Computação/USP)

Aprovado

Prof. Dr. **FRANCISCO JOSÉ MONACO**
(Instituto de Ciências Matemáticas e de Computação/USP)

Rosana Braga — Aprovado

Profª. Drª. **ROSANA TERESINHA VACCARE BRAGA**
(Instituto de Ciências Matemáticas e de Computação/USP)

Prof. Associado **GERALDO ROBERTO MARTINS DA COSTA**
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica e
Presidente da Comissão de Pós-Graduação

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo