UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM PARA DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS COM REÚSO DE SOFTWARE BASEADO EM MODELAGEM ESPECÍFICA DE DOMÍNIO E ARQUITETURA ORIENTADA A SERVIÇOS

ALEXANDRE BELLINI

ORIENTADOR: PROF. DR. ANTONIO FRANSCICO DO PRADO

COORIENTADORA: PROFª. DRª. LUCIANA APARECIDA MARTINEZ ZAINA

São Carlos - SP
Junho/2011

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ABORDAGEM PARA DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS COM REÚSO DE SOFTWARE BASEADO EM MODELAGEM ESPECÍFICA DE DOMÍNIO E ARQUITETURA ORIENTADA A SERVIÇOS

ALEXANDRE BELLINI

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software.
Orientador: Dr. Antonio Franscico do Prado
Coorientadora: Drª Luciana A. M. Zaina

São Carlos - SP
Junho/2011

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

B444ad

Bellini, Alexandre.
Abordagem para desenvolvimento de aplicações móveis com reuso de software baseado em modelagem específica de domínio e arquitetura orientada a serviços / Alexandre Bellini. -- São Carlos : UFSCar, 2011.
167 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2011.

1. Ciência da computação. 2. Software - reutilização. 3. Desenvolvimento orientado por modelos. 4. Cuidados médicos. I. Título.

CDD: 004 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Uma Abordagem para Reúso de Software baseado em Modelagem Específica de Domínio e Arquitetura Orientada a Serviços"

ALEXANDRE BELLINI

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Antonio Francisco do Prado
(Orientador - DC/UFSCar)

Profa. Dra. Luciana Aparecida Martinez Zaina
(DC/UFSCar/Sorocaba)

Prof. Dr. Wanderley Lopes de Souza
(DC/UFSCar)

Profa. Dra. Mariza Andrade da Silva Bigonha
(UFMG)

São Carlos
Julho/2011

Dedico este trabalho à minha mãe Izabel, que sempre orou por mim, pela minha vida,
pelo meu sucesso e esteve do meu lado me dando apoio nos momentos
de alegria e de tristeza desta incrível jornada.
Eu te amo, Mãe!

AGRADECIMENTO

Agradeço muito a Deus, por me dar oportunidade, capacidade e forças para superar todas as dificuldades as quais me deparei durante todo o mestrado e conquistar essa vitória em minha vida. Muito obrigado, meu Deus!

A toda minha família, em especial, a minha mãe (Izabel) e meu irmão (Fernando), que com sorrisos e todo amor e carinho me fazem lembrar que a vida não podia ser melhor. E também pelo apoio, força e compreensão nos (muitos) momentos difíceis, que, apesar da distância, sempre estiveram presentes em pensamentos e orações.

Agradeço ao meu orientador, professor Prado, e à minha coorientadora, professora Luciana, que me acolheram, confiaram em mim e me orientaram durante todos estes anos, sem o que eu não teria realizado este trabalho. Além dos diversos papéis que lhe cabem oficialmente, sendo orientadores dedicados, interessados, amigos e competentes.

A minha namorada Erika, por toda sua dedicação, amor, companheirismo, empenho e auxílio durante o período em que esteve comigo no mestrado e com quem sempre pude compartilhar as dores e alegrias deste percurso.

As famílias que me acolheram e me deram morada durante o mestrado. Em especial as famílias Brosque (Tia Fá, Sergião (Ou Serjão?)e meu amigo Marcus), Magalhães (Walter, Vanessa, Natália e Waltinho) e Buzinari (Aparecida, Nildo e Rodrigo), por me acolherem com todo carinho. Deus os abençoe!

Aos amigos e colegas que fiz neste mestrado, pelas horas de estudo, apreensão, descontração e baladas, por terem vivenciando mais intensamente comigo os momentos desta jornada: Du, Lu, Gil, Mah, Hiro, Rafa, Ju, Gui (Baiano), Josue, Gá, Ká, Eduardo, Pablo, Rafael e André. Valeu, pessoal!

A todos os colegas do LaBDES e do PPG-CC, pelo companheirismo, receptividade e eventuais apoios. Também aos professores, pelos ensinamentos, e à secretaria e funcionários (Evelton) do DC. Galera, obrigado!

Aos amigos e companheiros do grupo de Jovens (Junsari), que fizeram da minha estada em São Carlos um momento mais divertido, agradável, prazeroso e principalmente de oração e reflexão, mostrando um Cristo presente e vivo na minha vida. Galera, é TUDO NOSSO!

Ao professor Mauro Biajiz†, por acreditar e dar a oportunidade de cursar aulas como aluno especial do PPG-CC no DC/UFSCar e assim dar início a esse mestrado.

Aos alunos da turma de 2011 da disciplina Desenvolvimento de Software para Web da UFSCar, pelo interesse e participação no experimento realizado neste trabalho.

À CAPES e ao PPG-CC, pelo apoio financeiro.

Enfim… a todos que, de alguma maneira, tiveram uma passagem na minha vida durante esse período e contribuíram para a realização deste sonho e projeto para que tudo desse certo. Que Deus abençoe a todos!

Muito Obrigado!

Há homens que lutam um dia e são bons.

Há outros que lutam um ano e são melhores.

Há os que lutam muitos anos e são muito bons.

Porém, há os que lutam toda a vida.

Esses são os imprescindíveis.

(Bertolt Brecht)

RESUMO

Essa dissertação apresenta uma abordagem para o Desenvolvimento de Aplicações Móveis que enfatiza o Reúso de Software com base na Modelagem Específica de Domínio e na Arquitetura Orientada a Serviços. A abordagem é dividida em duas etapas: *Engenharia de Domínio (ED)* e *Engenharia da Aplicação (EA)*. Na ED são construídos: um metamodelo, que expressa a sintaxe abstrata de uma Linguagem Específica de Domínio de um dado domínio do problema; serviços – denominados Serviços do Domínio – que atendem aos requisitos comuns de diferentes aplicações do domínio do problema para o qual o metamodelo foi construído; e Transformações Modelo-para-Código para reduzir o esforço de desenvolvimento, uma vez que grande parte das tarefas de codificação pode ser encapsulada nas transformações. Na EA são construídas aplicações instanciando-se o metamodelo para apoio à modelagem, bem como reutilizando os Serviços do Domínio. Além disso, as Transformações, construídas na ED, são aplicadas sobre os modelos para gerar grande parte do código das aplicações. O reúso dos artefatos construídos na ED proporciona um ganho de produtividade no desenvolvimento das aplicações do domínio do problema considerado. Para fins de avaliação, a abordagem proposta foi instanciada no domínio de Cuidado de Saúde (*Healthcare*). Uma experimentação da abordagem, seguindo a metodologia experimental, foi conduzida com o intuito de avaliar seu impacto na eficiência de equipes desenvolvendo aplicações para esse domínio. Os resultados evidenciaram que o uso da abordagem proposta colaborou para a redução de tempo no desenvolvimento de aplicações móveis.

Palavras-chave: Reutilização de Software, Desenvolvimento Orientado a Modelos, Modelagem Especifica de Domínio, Arquitetura Orientada a Serviços, Cuidado de Saúde (*Healthcare*).

ABSTRACT

This dissertation presents an approach for the development of mobile applications emphasizing software reuse based on domain-specific modeling and Service Oriented Architecture. The approach is divided into two stages: Domain Engineering (DE) and Application Engineering (AE). In DE, a metamodel, services and Transformations for Model-To-Code are constructed. The metamodel expresses the abstract syntax of *Domain-Specific Languages (DSL)* for a given problem domain and services - known as the Domain Services - meet common requirements of different applications of the problem domain for which the metamodel has been built, and Transformations for Model-To-Code to reduce the development effort since most of the coding tasks can be encapsulated in the transformations. In EA, applications are built by instantiating the metamodel in order to support the modeling and by reusing Domain Services. In addition, the transformations, built in ED, are applied to the models in order to generate most of the application code. The reuse of the artifacts built in ED provides a productivity gain in the development of applications of the problem domain considered. For evaluation purposes, the proposed approach has been instantiated in the HealthCare Domain. An experimentation of this approach, following the experimental methodology, has been conducted in order to evaluate its impact on the efficiency of teams that develop applications for Healthcare Domain. The results showed that the use of the proposed approach has contributed to the reduction of time in developing mobile applications.

Keywords: Software Reuse, Model-Driven Development, Domain-Specific Modeling, Service-Oriented Architecture, Healthcare.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API - *Application Programming Interface*

AST - *Abstract Syntax Tree*

BPEL - *Business Process Execution Language*

CORBA - *Common Object Request Broker Architecture*

DCOM - *Distributed Component Object Model*

DSL - *Domain-Specific Language*

DSM - *Domain Specific Modeling*

EA - *Engenharia de Aplicação*

ED - *Engenharia de Domínio*

EMF - *Eclipse Modeling Framework*

EMP - *Eclipse Modeling Project*

ER - *Entidade Relacionamento*

GMF - *Graphical Modeling Framework*

HTTP - *HiperText Transfer Protocol*

IDE - *Integrated Development Environment*

IDL - *Interface Definition Language*

IIOP - *Internet Inter-ORB Protocol*

Java RMI - *Java Remote Method Invocation*

JET - *Java Emitter Templates*

JSON - *JavaScript Object Notation*

LOC - *Lines of Code*

MDD - *Model-Driven Development*

MIC - *Model Integrated Computing*

MIME-type - *Multipurpose Internet Mail Extensions type*

MVC - *Model-View Controller*

MVCASE - *Multiple-View CASE*

M2C - *Modelo-para-Código*

OMG - *Object Management Group*

PCs - *Personal Computers*

PDA - *Personal Digital Assistant*

REST *- Representational State Transfer*

RPC - *Remote Procedure Call*

SADT - *Structured Analysis and Design Technique*

SPL - *Software Product Lines*

SMTP - *Simple Mail Transfer Protocol*

SOA - *Service Oriented Architecture*

SOAP - *Simple Object Access Protocol*

SOC - *Service Oriented Computing*

SQL - *Structured Query Language*

TI - *Tecnologia da Informação*

TXL - *Turing eXtender Language*

UCF - *Ubiquitous Computing Framework*

UDDI - *Universal Description, Discovery and Integration*

UML - *Unified Modeling Language*

URI - *Uniforme Resource Identifier*

XMI - *XML Metadata Interchange*

XML - *eXtensible Markup Language*

XSD - *XML Schema Definition*

WADL - *Web Application Description Language*

WSDL - *Web Service Description Language*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contextualização

A Engenharia de Software tem procurado avançar para desenvolver e/ou evoluir processos, abordagens, técnicas e métodos para criar software com qualidade e de forma mais ágil, buscando cumprir os prazos e metas dos projetos. Um dos fatores que pode auxiliar no cumprimento dessas metas é o foco no reúso de software durante o processo de desenvolvimento de software (KRUEGER, 1992). Ao longo dos anos, muitos trabalhos têm mostrado que uma forma eficaz de obter os benefícios do reúso de software é adotar um processo para o reúso. Os exemplos desses trabalhos podem ser observados desde o meio acadêmico (e.g. ENDRES, 1993; BAUER, 1993; GRISS, 1995; JOOS, 1994), até pesquisas informais (e. g. FRAKES & ISODA, 1995) e estudos empíricos (e.g. RINE, 1997; MORISIO et al., 2002; ROTHENBERGER et al., 2003).

Ao longo dos anos, diferentes técnicas e abordagens têm sido propostas para apoiar o desenvolvimento com foco no reúso, como, por exemplo, as baseadas em Metamodelagem, Meta Programação, Linha de Produtos (GOMAA, 2004), Componentes de Software, Padrões, *Frameworks* e Serviços. Na reutilização baseada em Serviços destaca-se a Arquitetura Orientada a Serviços (*Service Oriented Architecture – SOA*) (ERL, 2007), que tem seu fundamento no dinamismo para compor a arquitetura de software de acordo com as necessidades de um determinado cenário, prezando a interoperabilidade entre aplicações de plataformas distintas.

A arquitetura orientada a serviços destaca-se por ser um estilo arquitetural para a construção de aplicações de software que utilizam *Serviços* disponíveis em uma rede (MAHMOUD, 2005). Uma SOA básica não trata apenas de Serviços, incluindo o relacionamento entre três tipos de participantes no estilo arquitetural: o provedor de Serviços, o repositório de Serviços e o solicitante de Serviços (cliente). As operações realizadas pelos Serviços envolvem publicação, busca e ligação (PAPAZOGLOU, 2003). Os Serviços Web representam uma das tecnologias capazes de implementar uma SOA. Em princípio, esse Serviço assemelha-se a um componente, contudo o Serviço possui uma série de características que o torna diferenciado, como a forma de comunicação entre eles, o tipo de acoplamento, de interface, de invocação, de reúso, entre outros.

Além do reúso da arquitetura da aplicação, outro enfoque tem sido dado ao provimento de soluções, a partir do domínio do problema, que possam ser reutilizadas em diferentes aplicações. Nesse caso, o desenvolvimento de software e os aspectos reutilizáveis são capturados durante a análise do domínio, que possui o foco em promover o reúso de itens mais abstratos e não somente o código fonte. Dentre esses itens, pode-se citar como exemplos as arquiteturas de software, soluções de projeto, modelos de requisitos e representação do conhecimento de um determinado domínio. No entanto, essa tarefa se mostrou complexa e desafiadora para os desenvolvedores, dependendo da criação de toda uma cultura orientada para a reutilização. Procurando vencer esse desafio, surgiu a reutilização baseada em Metamodelagem, fundamentada no Desenvolvimento Orientado a Modelos (*Model-Driven Development - MDD*)[1] (MELLOR et al., 2003). Apoiando o MDD, a Modelagem Específica de Domínio (*Domain Specific Modeling - DSM*) (KELLYM & TOLVANEN, 2008) visa a tornar a tarefa de modelagem mais aderente às necessidades de um domínio.

A DSM possibilita criar uma linguagem de modelagem em que os conceitos do domínio possam ser representados o mais próximo possível por meio de uma linguagem computacional, o que facilita as tarefas de compreensão do domínio, o melhor entendimento e a comunicação por parte dos especialistas do domínio e dos desenvolvedores de software (KUHNE, 2007). Por meio da DSM os modelos podem expressar conceitos e sintaxe de um domínio, o que possibilita o reúso dos modelos

[1] Existem outros termos encontrados na literatura, tais como MDE (*Model-Driven Engineering*) (SCHMIDT, 2006), MDSD (*Model-Driven Development*) (VÖLTER & GROHER, 2007), ou simplesmente MD* (VÖLTER, 2008), que são usados como sinônimos de MDD.

por várias aplicações do domínio, de forma a trazer benefícios em um projeto de software reutilizável.

Por outro lado, nos últimos anos tem-se observado um rápido crescimento no número de dispositivos móveis, sendo largamente adotados em diferentes áreas, tais como a Saúde, a Educação e o Entretenimento. Além disso, uma mesma aplicação na computação muitas vezes deve ser executada em diferentes arquiteturas. Assim as especificações e os modelos dessa aplicação podem ser reutilizadas para serem implementadas nessas diferentes arquiteturas. Motivados por essas ideias pesquisas tem sido realizadas visando diminuir o tempo e o custo na construção das aplicações. Dentre as pesquisas que visam melhorar o processo de desenvolvimento para auxiliar os desenvolvedores, com objetivo de diminuir o tempo e o custo na construção das aplicações, destacam-se as que têm foco na reutilização de software. Essas abordagens com foco na reutilização permitem desenvolver aplicações com mais agilidade e maior qualidade, na medida em que favorecem o reúso de artefatos previamente validados e testados.

O reúso de software aliado às ideias de DSM e SOA pode ser encarado como uma solução que busca trazer diversos benefícios ao projeto de software, especialmente para o desenvolvimento na computação móvel. Por exemplo, a possibilidade de utilizar a modelagem como ferramenta ativa no desenvolvimento, disponibilizar e reutilizar serviços Web para um conjunto de aplicações de um domínio podem colaborar para inovar e agilizar os processos e abordagens para o desenvolvimento de software na computação móvel.

1.2 Motivação

A necessidade de construir aplicativos de forma eficaz e com maior qualidade são características importantes no desenvolvimento de diferentes aplicações. Para tornar mais ágil a construção de software, técnicas de reutilização de software têm sido pesquisadas e adotadas nos processos de desenvolvimento. Conforme enfatizado por Jim Neighbors (1989) (NEIGHBORS, 1989), a chave do desenvolvimento de software com reutilização é capturada pela análise de domínio que possui o foco abrangente de promover o reúso de itens mais abstratos e não

somente de código. Dentre estes itens podem ser citados como exemplos arquiteturas de software, soluções de projeto, modelos de requisitos, bem como representação do conhecimento de um determinado domínio. No entanto, essa tarefa tem se mostrado complexa e desafiadora para os desenvolvedores, sendo geralmente de alto custo, pois depende da criação de toda uma cultura de apoio no ambiente em que o desenvolvimento para reúso irá ocorrer.

O desenvolvimento de serviço Web tem sido adotado como forma de apoiar a produção de tais aplicações, pois seu emprego pode facilitar o acesso via diferentes dispositivos e prover maior flexibilidade na busca de informações (ERL, 2007). Possibilita também ser utilizado para reúso de software devido a sua característica interoperável e reutilizável. Essas características possibilitam que o desenvolvimento de aplicativos seja independente de plataformas, pois a informação de um serviço Web está disponível em ambientes tais como móvel e desktop, o que permite, por consequência, integrar e compartilhar dados nos sistemas de informação.

Muitos são os desafios para a construção de aplicações móveis. Um desses desafios é melhorar a produtividade, por meio da reutilização de software. Esse novo cenário da computação móvel tem motivado a Engenharia de Software a inovar seus processos e tecnologias de desenvolvimento para suprir as necessidades vigentes dessa nova realidade.

O desenvolvimento de aplicações para diferentes domínios tem demandado a adoção de tecnologias que agilizam as tarefas de desenvolvimento, tornando-o mais eficiente e alinhado aos processos de negócio das organizações. A variedade de serviços ofertados e a popularização da Internet têm demandado a necessidade de oferecer serviços de forma mais flexível, padronizada e extensível, de maneira a facilitar a comunicação entre diferentes aplicações.

Atualmente, a computação móvel está presente não apenas por meio de computadores em sua forma tradicional (e.g.: notebooks), mas principalmente por meio dos aparelhos de telefonia celular, os quais totalizam mais de 4 bilhões no mundo, enquanto o número de *Personal Computers (PCs)* totaliza cerca de 1 bilhão (FOWLER; ASK; GOWNDER, 2010). Estima-se que até 2014 a taxa de penetração de *smartphones* e telefones celulares dotados de capacidades para acesso à Internet no mercado global seja de 90%, excedendo 1,82 bilhão de unidades, ao passo que o número de computadores pessoais ativos chegue em torno de 1,78 bilhão (GARTNER, 2010).

Portanto, com base nessas ideias, a motivação deste trabalho é desenvolver uma Abordagem para Desenvolvimento de Aplicações Móveis com Reúso de Software. O objetivo é auxiliar o desenvolvimento de software por meio de reúso em dispositivos móveis combinando as concepções de serviço Web da SOA e a DSM, focando-se na modelagem das aplicações móveis e na transformação de modelos em código-fonte para diferentes plataformas móveis. Com isso, parte da complexidade envolvida na codificação dos aplicativos móveis fica encapsulada nas transformações, e reúso de serviços Web contribui para redução dos esforços de implementação, diminuição de erros de codificação, aumento da produtividade, qualidade, além de permitir o sucesso no desenvolvimento de aplicações móveis.

1.3 Objetivo

A partir da motivação de pesquisar um processo de desenvolvimento de software orientado para o reúso, este trabalho teve por objetivo geral definir uma abordagem para desenvolvimento de aplicações móveis com reúso de software baseado em DSM e SOA. Pretende-se que a abordagem contribua para tornar mais ágil o desenvolvimento com o uso de modelos, colaborando para o aumento da produtividade e redução dos esforços de desenvolvimento de aplicações móveis.

Para reduzir o tempo de desenvolvimento, a abordagem emprega a geração de código a partir da modelagem das aplicações móveis. Em síntese, o objetivo geral deste trabalho pode ser desmembrado nos seguintes itens:

- Formular uma abordagem baseada em DSM e SOA para simplificar o desenvolvimento com reúso de software no domínio do problema considerado;
- Estabelecer atividades com diretrizes para o desenvolvimento de artefatos reutilizáveis proporcionando um ganho de produtividade no desenvolvimento das aplicações móveis;
- Disponibilizar os artefatos construídos para reúso no desenvolvimento de aplicações móveis, compreendendo: um metamodelo que expresse a sintaxe abstrata de uma linguagem específica de domínio *(Domain-Specific Language - DSL)* para suporte à modelagem; serviços –

denominados Serviços do Domínio – que atendam aos requisitos comuns das diferentes aplicações do domínio do problema para o qual o metamodelo foi construído; e as Transformações de Modelo-para-Código (M2C) utilizadas para gerar parte do código das aplicações móveis.

1.4 Organização

Esta dissertação está organizada em outros cinco capítulos, além deste capítulo introdutório. A Figura 1.1 ilustra a organização geral do trabalho. Os conteúdos de cada capítulo são resumidos a seguir.

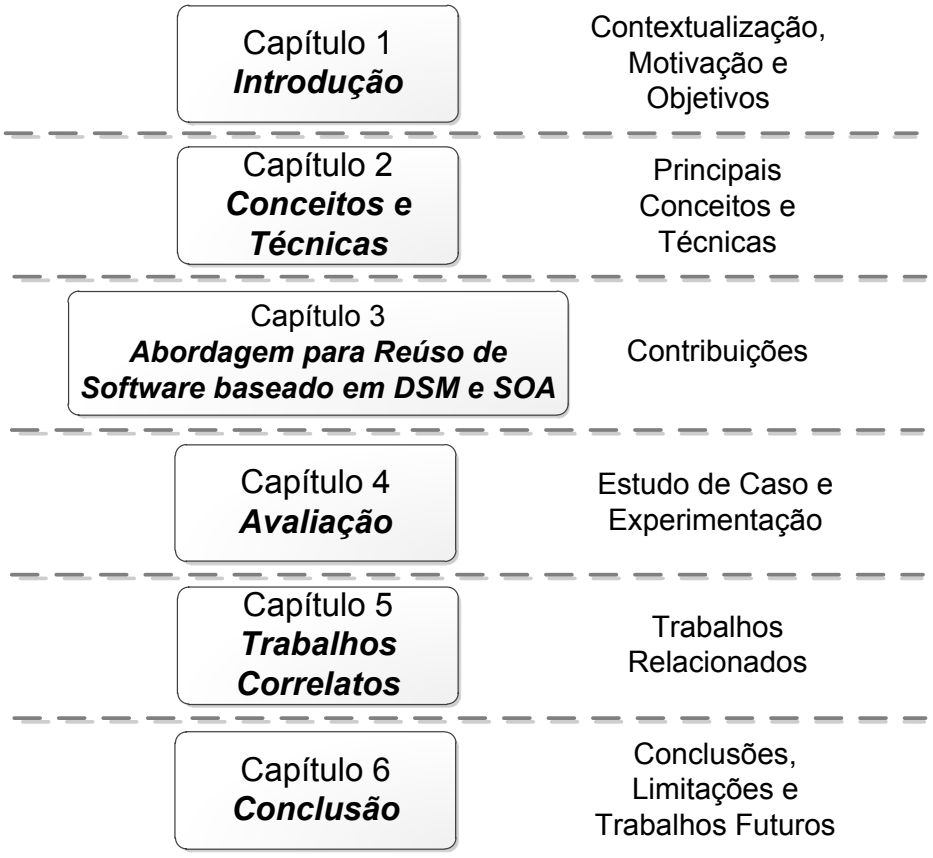

Figura 1.1 – Organização da Dissertação

- O **Capítulo 2** apresenta uma revisão da literatura sobre as tecnologias e conceitos utilizados como base para a definição da abordagem de desenvolvimento de software proposta neste trabalho. São também discutidas as ferramentas do *Eclipse Modeling Project (EMP)*, e da

ferramenta *Multiple-View Computer-Aided Software Engineering (MVCASE)* utilizadas para modelagem das aplicações do domínio do problema e na abordagem proposta;

- O **Capítulo 3** apresenta a Abordagem para o Desenvolvimento de Aplicações Móveis com Reúso de Software baseado em DSM e SOA proposto neste trabalho para suporte à construção de aplicações móveis. As principais ideias em torno da abordagem são discutidas e suas atividades são detalhadas, apresentando as entradas, saídas, controles e os mecanismos que apoiam os desenvolvedores na construção de suas aplicações móveis;
- O **Capítulo 4** aborda a avaliação da proposta;
- O **Capítulo 5** discute alguns trabalhos encontrados na literatura que se relacionam ao trabalho apresentado nesta dissertação; e
- O **Capítulo 6**, por fim, discorre sobre as conclusões desse trabalho, incluindo suas contribuições e limitações, além de apontar direções para futuras pesquisas.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA E TÉCNICA

Este capítulo apresenta os principais conceitos e técnicas nos quais este trabalho se baseia. Considerando que a abordagem proposta objetiva apoiar o desenvolvimento de aplicações móveis por meio do reúso de serviços Web, a Seção 2.2 apresenta os conceitos relacionados à SOA. A Seção 2.3 introduz as concepções de MDD e DSM. A Seção 2.4 discute o EMP e as ferramentas que apoiam a abordagem e auxiliam o reúso de software em dispositivos móveis. A Seção 2.5 apresenta a ferramenta MVCASE para modelagem das aplicações do domínio.

2.1 Arquitetura Orientada a Serviços

O paradigma de Computação Orientada a Serviço, *Service Oriented Computing (SOC)*, e sua respectiva arquitetura de suporte, SOA, são considerados atualmente os mais promissores na área de computação distribuída. Esse paradigma prega a interoperabilidade e o fraco acoplamento entre elementos básicos de software por meio da definição de interfaces neutras e da utilização de protocolos de transporte padronizados. Serviços são unidades lógicas de software, podendo encapsular um simples método ou um grande processo envolvendo múltiplos colaboradores (PAPAZOLOU, GEORGAKOPOULOS, 2003). SOC é um

conceito bastante amplo que representa uma nova geração de plataforma de computação distribuída. Esse conceito engloba vários elementos como princípios de projeto, padrões de projeto, linguagens de programação, hardware, *frameworks*, processos e estratégias organizacionais.

Um serviço Web é uma tecnologia projetada para suportar interações máquina-máquina interoperáveis sobre uma rede. Possui uma interface conhecida como *Web Service Description Language (WSDL)* (W3C, 2007b). Outros sistemas interagem com o serviço Web da maneira definida na sua interface usando mensagens do *Simple Object Access Protocol (SOAP)*, tipicamente transportadas usando o *HiperText Transfer Protocol (HTTP)* com serialização da *eXtensible Markup Language (XML)* e em conjunto com outros padrões relacionados à web (W3C, 2004a), que serão abordados nas subseções a seguir.

Existem diversas definições para SOA na literatura, mas é importante deixar claras algumas definições de SOA que não procedem. SOA não é uma tecnologia, não pode ser reduzida a produtos de software, não é uma solução, não atende todos os desafios tecnológicos aos quais estão submetidos os negócios de hoje, mas sim da forma que a aplicação deve ser construída (PAPAZOGLOU e GEORGAKOPOULOS, 2003).

SOA é, basicamente, um modelo de arquitetura de software no qual toda funcionalidade de negócio é construída com base em funções e processos individualizados denominados serviços. SOA define que os serviços sejam desenvolvidos, implantados e integrados, independentemente da aplicação e da plataforma computacional na qual os serviços serão executados. O serviço deve operar de forma independente de outros serviços, além de possuir uma interface bem definida (ERL, 2007).

SOA se relaciona também com determinadas políticas e conjuntos de práticas que pretendem criar um processo para facilitar a tarefa de encontrar, definir e gerenciar os serviços disponibilizados. O modelo de arquitetura de software beneficia a eficiência, agilidade e produtividade no desenvolvimento de aplicações e está alinhado com os objetivos de SOC (ERL, 2007).

Segundo Erl (ERL, 2005), a adoção da SOA no processo de desenvolvimento de software deve prover as seguintes funcionalidades: Fraco acoplamento, Interoperabilidade, Composição, Reusabilidade, Alta granularidade e Ubiquidade, os quais são assim caracterizados:

- *Fraco acoplamento*: é um conceito fundamental de SOA e dos grandes sistemas distribuídos. Consiste em garantir que cada serviço seja o mais independente possível de outros. Desta forma, ao se modificar um determinado serviço, os outros não deverão ser afetados;
- *Interoperabilidade*: os serviços idealmente devem ser implementados com o uso de tecnologias padronizadas e disponíveis a um grande número de plataformas de software. A utilização de protocolos e aplicação, independentemente da plataforma, resulta em Interoperabilidade entre os serviços. Protocolos proprietários apresentam problemas de interoperabilidade entre softwares desenvolvidos por diferentes fabricantes;
- *Composição*: é a capacidade de compor serviços e possibilitar a execução de funcionalidades complexas dentro de um determinado processo de negócio. Os serviços podem ser compostos para formar novos serviços com um nível maior de abstração e provendo funcionalidades agregadas;
- *Reusabilidade*: é um elemento fortemente adotado na arquitetura orientada a serviços. Por serem módulos independentes de código e fracamente acoplados à aplicação, os serviços podem ser facilmente reutilizados, resultando em um alto nível de produtividade no desenvolvimento de software. O reúso de código gera redução no tempo de desenvolvimento de aplicações, bem como redução nos custos;
- *Alto Grau de Granularidade*: o encapsulamento de funcionalidades no nível de serviço evoca um alto grau de granularidade nos componentes básicos da arquitetura. Para o desenvolvimento de aplicações complexas e extensas a alta granularidade traz vantagens na medida em que detalhes de implementação são deixados à equipe de desenvolvimento responsável por aquele serviço; e
- *Ubiquidade*: os serviços devem ser acessíveis a partir de qualquer lugar e a qualquer momento facilitando a composição de serviços entre empresas.

2.1.1 SOA e os *Web Services*

Os Web Services são a tecnologia mais utilizada para implementar SOA (Erl, 2005). Embora se associe que Web Services e SOA são sinônimos, deve-se deixar claro que é possível implementar aplicações SOA usando tecnologias como *Remote Procedure Call (RPC)* (SINHA, 1997), Java *Remote Method Invocation (RMI)* (MICROSYSTEMS, 2006), *Common Object Request Broker Architecture (CORBA)* (GROUP, 2009), ou *Distributed Component Object Model (DCOM)* (MICROSOFT, 1995), mas essas tecnologias apresentam algumas dificuldades. A Java RMI é muito ligada à linguagem Java e dificilmente se comunica com outras linguagens de programação. A tecnologia CORBA tem o problema de possuir protocolos complexos. A DCOM é uma tecnologia dependente da plataforma e totalmente proprietária. Por esses motivos Web Services se tornou popular, pois tenta resolver esses problemas criando protocolos de comunicação abertos que possam ser facilmente integrados a diferentes plataformas e possui dois requisitos fundamentais: comunica-se via protocolos Internet e o envio e recebimento de dados em um formato padrão (e.g., XML) (ERL, 2007) (KIM & HAN, 2006).

O conceito de Web Services teve seu início em meados de 2000 com a introdução da primeira versão do protocolo SOAP, WSDL Versão 1.1 e de uma versão inicial do servidor *Universal Description, Discovery and Integration (UDDI)*. O Web Service é definido pela publicação de um documento WSDL para descrever o serviço e o cliente acessa esse documento publicado por meio do UDDI ou no provedor de Web Service. Após o cliente receber as informações do serviço, é desenvolvido o software que realizará a troca de informações com o envio de mensagens utilizando o protocolo SOAP via chamada dos Web Services (W3C, 2004a).

A plataforma Web Service de segunda geração (extensões WS-*) busca eliminar algumas brechas relacionadas a serviços de qualidade na plataforma de primeira geração. As áreas de segurança no nível de mensagens, transações de serviços cruzados e troca de mensagens confiáveis são fornecidas pela plataforma de segunda geração. Consistindo de inúmeras especificações que aprimoram o *framework* fundamental da troca de mensagens, essas tecnologias Web Service fornecem um rico conjunto de recursos muito mais complexo, tanto em termos de tecnologia quanto de projeto (ERL, 2007).

A identificação desses Web Services, no entanto, não é uma tarefa simples e imediata. Em geral são utilizadas três abordagens para resolver esse problema: *bottom-up, top-down,* e *middle-out.* A estratégia *bottom-up* é focada na análise dos sistemas existentes, módulos das aplicações legadas e os pacotes das aplicações para selecionar os candidatos viáveis a fornecer uma funcionalidade que suporta o processo de negócio. Por outro lado, a estratégia *top-down* realiza a análise dos modelos de casos de usos de negócio no qual prevê a especificação dos serviços. Essa abordagem é referida como a análise dos processos de negócio para a decomposição do domínio em suas áreas funcionais, subprocessos, subsistemas, e casos de uso. A abordagem *middle-out* consiste na validação e identificação de outros serviços não contemplados pelas abordagens *top-down* e *bottom-up*. A *middle-out* fornece a ligação entre os objetivos de negócio e os da *Tecnologia da Informação (TI)* por meio da rastreabilidade dos serviços diretamente a um objetivo de negócio (PEREPLETCHIKOV et al., 2005) (MEIJLER et al., 2006).

2.1.2 Principais Padrões utilizados pelos *Web Services*

Para que se possa construir uma SOA no contexto de Web Services são utilizados padrões como, por exemplo, o XML. Esse padrão trata-se de uma metalinguagem, ou seja, uma linguagem utilizada para descrever outras linguagens. Essa metalinguagem descreve documentos XML em um formato de texto simples e bastante flexível, elaborado originalmente para atender aos desafios da publicação eletrônica de larga escala. Estes documentos são constituídos de *tags*, que por sua vez podem encapsular *tags* ou outros dados. A extensibilidade e a natureza estruturada de XML permitem sua utilização na comunicação entre diferentes sistemas, que de outra forma não poderiam se comunicar (DAILY, 2003). Devido principalmente a estas características, o formato padrão de troca de dados dos Web Services normalmente são implementados utilizando XML. Para padronizar o formato das mensagens XML, faz-se uso de estruturas bem definidas denominadas Esquemas. O formato mais comum de Esquemas é o XSD, acrônimo de *XML Schema Definition (XSD)*.

Outro padrão utilizado neste contexto é a linguagem de descrição dos serviços que utiliza uma gramática XML para descrever os Web Services. A função do WSDL é fornecer uma documentação e detalhes técnicos dos Web Services que,

além de descrever o serviço, especifica como acessá-lo e quais são as operações ou métodos disponíveis para a chamada e o desenvolvimento do serviço. A implementação do serviço pode se dar utilizando qualquer linguagem de programação e plataforma. Pode-se utilizar ferramentas para geração de código que, a partir da especificação da interface, geram os *stubs* (objeto cliente que oferece implementações dos métodos do serviço remoto) e os *skeletons* do serviço (objeto servidor que recebe as chamadas do cliente e invoca o método do serviço), na linguagem de programação desejada. Resumidamente, um documento WSDL usa os seguintes elementos XML na definição do serviço:

- Tipo (*Type*): contêiner para definições de tipos de dados. Usa algum sistema de definição de tipos como, por exemplo, XSD;
- Operação (*Operation*): descrição abstrata de uma funcionalidade suportada pelo serviço;
- Associação (*Binding*): protocolo de comunicação concreto e especificação da formatação dos dados para um tipo de porta específico;
- Porta (*Port*): um único *endpoint* definido como a combinação de uma associação e um endereço de rede; e
- Serviço (*Service*): uma coleção de portas agrupadas logicamente.

A linguagem WSDL define quatro tipos de interações entre o provedor e o cliente do serviço, baseada na quantidade e direção das mensagens trocadas. É possível que haja somente o recebimento de uma mensagem pelo provedor do serviço, caracterizando uma interação em sentido único (*one-way*). Outra possibilidade é que exista o recebimento e o envio de uma mensagem pelo provedor do serviço, caracterizando uma interação requisição-resposta (*request-response*). Ainda, pode-se ter o envio e recebimento de uma mensagem (*solicit-response*) ou apenas o envio de uma mensagem (*notification*) por parte do servidor.

O SOAP (W3C, 2007c), outro padrão utilizado, é um protocolo de comunicação no nível de aplicação que define a formatação e codificação de mensagens XML a serem enviadas pela rede. O protocolo define um mecanismo leve e simples para troca de informações estruturadas entre pares em um ambiente descentralizado e distribuído (WALSH, 2002). É um protocolo projetado para comunicação na Internet, podendo utilizar outros protocolos (e.g., HTTP ou *Simple Mail Transfer Protocol - SMTP*) para transporte de suas mensagens. O objetivo desse protocolo é prover um meio de comunicação entre aplicações com diferentes

tecnologias, executando em diferentes sistemas operacionais e linguagens de programação. Ao utilizar protocolos padrões da Internet, ele não é bloqueado por *firewalls* nem por roteadores como outros protocolos de aplicação (e.g., *Internet Inter-ORB Protocol - IIOP*).

A mensagem SOAP deve necessariamente conter um elemento envelope. Este por sua vez possui os elementos *header* opcional e *body* mandatório. O elemento corpo representa os parâmetros de entrada e saída do serviço, ou seja, a informação em si sendo transmitida. A mensagem ainda pode conter um elemento de falha (*fault*) que provê informações sobre erros ocorridos durante o processamento da mensagem.

O UDDI (OASIS, 2004) é um mecanismo que visa atender tanto o cliente de Web Service quanto o provedor. A sua tarefa é fornecer ao provedor de Web Services os protocolos necessários para que eles sejam registrados, descobertos e publicados na rede (WALSH, 2002; NEWCOMER, 2002). Funciona como uma agência de registro de Web Services, similar às páginas amarelas de uma lista telefônica. Dado um conjunto de parâmetros de especificação de um serviço (nome, qualidade, localização, entre outros), a agência é responsável por encontrar serviços cadastrados que satisfaçam os parâmetros fornecidos. A agência mantém um banco de dados centralizado dos serviços Web disponíveis na rede classificados por tipo de serviço, informações do provedor, qualidade dos serviços e assim por diante. O protocolo para requisição dessas informações também é definido, sendo este independente de plataforma, pois utiliza SOAP para codificação das mensagens.

2.1.3 Composição de *Web Services*

Os Web Services devem ser projetados a fim de permitir que possam ser utilizados por outros serviços ou aplicações. As novas funcionalidades ou os novos padrões podem ser adicionados aos serviços sem quebrar o contrato com os consumidores atuais do serviço, mantendo as funcionalidades existentes anteriormente e a interoperabilidade (ERL, 2007). A orquestração é um mecanismo de agregação ou composição de serviços visando à obtenção de um novo serviço com funcionalidades mais complexas. É um componente importante da SOA, pois encapsula logicamente serviços de forma transparente ao cliente. A orquestração permite que o desenvolvimento de novos serviços seja realizado de forma adaptável,

flexível e dinâmica às necessidades de negócios, que se alteram frequentemente (CHRIS, 2003).

A orquestração de serviços define um fluxo de execução de um processo, ou seja, por meio de um arquivo de orquestração, que é processado por um programa denominado máquina de orquestração, gerencia as chamadas e os fluxos da composição dos serviços. Nesse arquivo são definidas as chamadas dos serviços componentes necessários à obtenção do novo serviço. Como qualquer linguagem de programação no paradigma imperativo, as linguagens de orquestração possuem atribuição de variáveis, interações e fluxos condicionais. Como a tecnologia amplamente utilizada com a SOA é a de Web Services, as linguagens e tecnologias para composição de serviços que mais se desenvolveram foram aquelas que compõem os Web Services (MILANOVIC & MALEK, 2004).

O objetivo da orquestração é por meio de um processo central gerenciar os Web Services. Esse processo central é responsável por controlar e coordenar a execução dos Web Services envolvidos. Cada Web Service não tem conhecimento de que faz parte de uma composição de serviços, a menos que se tenha acesso ao código dos Web Services envolvidos. Não obtendo esse conhecimento, é impossível saber qual é o fluxo de execução de cada um deles, pois somente as suas interfaces são conhecidas. A função das interfaces é mostrar o comportamento estático do serviço, enquanto a orquestração mostra o comportamento dinâmico do serviço.

A coreografia é um mecanismo de especificação de um protocolo de troca de mensagens entre duas entidades participando em uma interação. A coreografia está inserida no contexto em que as interações entre os serviços são tipicamente de troca de mensagens ponto a ponto, síncronas e assíncronas, envolvendo duas ou mais partes, possivelmente de longa duração e com manutenção de estado. O seu objetivo é especificar o contrato entre duas entidades independentes de software localizadas em ambientes distintos. O desacoplamento é a principal característica, pois possibilita que o serviço seja especificado sem revelar qualquer detalhe sobre sua implementação e provê liberdade quanto à mudança de aspectos privados do serviço sem afetar o protocolo já estabelecido (BEA Systems, 2003).

Os conceitos de coreografia e orquestração são bem próximos, o que causa algumas divergências quanto à aplicação de cada um. Primeiro, os nomes em si já levam à analogia de uma dança coreografada *versus* uma orquestra controlada por um condutor onde os participantes reagem às influências externas como música e

comportamento dos seus pares. A coreografia tem uma natureza colaborativa, isto é, na coreografia é definido um contrato de interação em que ambas as partes colaboram para atingir um objetivo comum. Já a orquestração revela o ponto de vista de uma única entidade interagindo com outras entidades.

2.1.4 Vantagens e Desvantagens dos *Web Services WS-**

Entre as vantagens apresentada pelo WS-* para aplicações Web Service destacam-se:

- *Padrões abertos*: utilizam, ao invés de tecnologias proprietárias, padrões abertos como HTTP, SOAP e UDDI;
- *Flexibilidade*: alterações realizadas nos serviços são mais simples em relação ao sistema como um todo;
- *Custo*: as soluções tradicionais por meio de tecnologia proprietárias apresentam um custo maior, além de que qualquer mudança necessária em sistema que apresenta baixa coesão demanda um esforço maior, acarretando no aumento dos custos de projetos; e
- *Escopo*: cada sistema é tratado de forma individual, uma vez que para torná-lo um serviço é necessário apenas criar uma camada que o encapsule.

Como desvantagens estão:

- Não apresentam um alto desempenho como componentes de sistemas voltados para dentro da empresa, devido ao tamanho e à sobrecarga necessária para o tratamento dos documentos XML, enquanto os objetos distribuídos são mais adequados para a construção de sistemas de alto desempenho com uma arquitetura bem definida; e
- Apresentam problemas quanto aos aspectos de localização dos serviços, tratamento de transação e segurança.

2.2 Representational State Transfer (REST)

O *Representational State Transfer (REST)* foi proposto por Roy Thomas Fielding (FIELDING, 2000; FIELDING e TAYLOR, 2002) em sua tese de doutorado. É definido como um estilo de arquitetura de software para sistemas hipermídia distribuídos, especificado como um conjunto de critérios e padrões de projeto utilizados com o intuito principal de obter maior escalabilidade dos sistemas de software.

REST é baseado na arquitetura ***cliente-servidor*** (FIELDING, 2000), a qual impõe restrições ao seu estilo arquitetural e à forma com que a comunicação é realizada. A comunicação entre o lado cliente e o lado servidor é realizada ***sem estado***. Uma comunicação sem estado significa que cada requisição submetida pelo cliente ao servidor ocorre em completo isolamento. Isto significa que, quando um cliente realiza uma requisição HTTP, esta inclui todas as informações necessárias para que o servidor a processe, sem a necessidade de conhecer os dados das requisições anteriores.

Além disso, toda resposta enviada do servidor ao cliente deve estar explicitamente ou implicitamente rotulada como ***cache*** ou *não-cache*. Se a resposta for colocada em *cache*, o cliente poderá reutilizar esses dados posteriormente, o que evita requisições desnecessárias e sobrecarga da rede. No entanto, esta abordagem pode ocasionar uma diminuição na confiabilidade, uma vez que, com o tempo, os dados podem se tornar desatualizados e obsoletos no *cache*.

Outra característica do REST que o diferencia dos demais estilos arquiteturais para sistemas em rede é a ênfase em uma ***interface uniforme*** entre as partes comunicantes. Essa uniformidade das interfaces de comunicação promove uma simplificação e maior visibilidade na arquitetura global do sistema. A fim de obter uma interface uniforme, várias restrições arquiteturais são necessárias para orientar o comportamento das partes comunicantes. Para o estilo REST são definidas três restrições de interface:

- manipulação de recursos por meio de suas representações abstratas;
- mensagens autodescritivas; e
- identificação de recursos.

Finalmente, REST introduz também uma restrição de sistema em camadas, conforme ilustrado na Figura 2.1. Nessa figura, é possível observar componentes intermediários entre o cliente e o servidor. Ao restringir o conhecimento do sistema a camadas independentes, a complexidade geral do sistema é reduzida em decorrência da separação de interesses entre as camadas. O uso de camadas pode ser útil para o encapsulamento de serviços legados. As camadas intermediárias podem ser utilizadas para melhorar a escalabilidade do sistema, permitindo o balanceamento de carga entre os serviços.

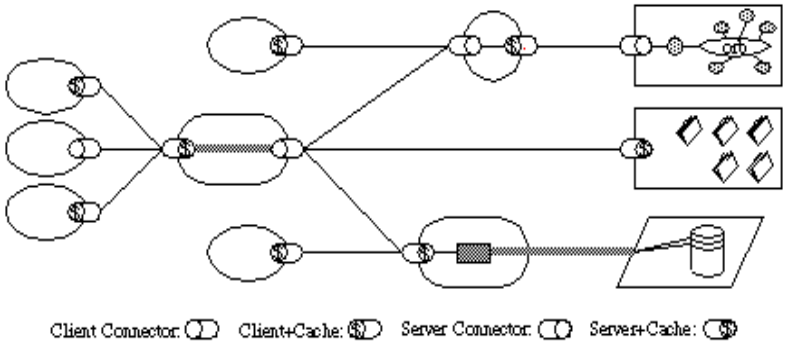

Figura 2.1. Restrição Sistema em Camadas (adaptado de FIELDING, 2000)

2.2.1 Elementos Arquiteturais

O conceito de ***recurso*** é a principal abstração do estilo de arquitetura REST. Enquanto no modelo RPC a comunicação se apoia nas operações que são expostas por meio do protocolo, na arquitetura REST são os recursos que devem ser expostos. Um recurso é qualquer conceito que possa ser alvo de referência de um autor de um hipertexto (FIELDING, 2000), como por exemplo: um documento, uma imagem, uma coleção de objetos não-virtuais (e.g. alunos, clientes) ou até mesmo outros recursos. Alguns recursos são considerados estáticos, pois seus valores não se alteram durante o tempo. Já outros têm o seu valor alterado constantemente ao longo do tempo e são considerados dinâmicos.

Para a identificação dos recursos, é necessário um espaço unificado de nomes e, para isso, a sintaxe universal utilizada é a *Uniforme Resource Identifier (URI)*. Cada recurso deve ter uma URI associada a ele, assim como cada URI deve estar associada a um único recurso. A grande vantagem do uso de interfaces uniformes decorre do fato de a Internet ser uma rede composta não apenas de clientes e servidores, mas possuir elementos intermediários como *firewalls*,

roteadores e *proxies*, responsáveis respectivamente por aumentar a segurança, prover escalabilidade e melhorar o desempenho da rede de maneira geral. O uso de uma semântica padronizada permite sua interpretação por estes elementos otimizando o seu funcionamento.

Uma representação de um recurso é uma sequência de bytes que representam seus dados e informações sobre o recurso, i.e., seus metadados (FIELDING, 2000). Os componentes REST que executam ações sobre um recurso utilizam dessas representações tanto para capturar como para atualizar o estado atual do recurso. Assim, os recursos ficam desacoplados de suas representações de forma que é possível representar um recurso por meio de diferentes formatos, como, por exemplo, um arquivo XML, uma página Web ou uma string *JavaScript Object Notation (JSON)*. JSON é um formato padrão de texto para troca de dados. É baseado em um subconjunto da linguagem de programação *JavaScript*[2] (CROCKFORD, 2006). A estrutura do documento JSON permite sua utilização na comunicação entre diferentes sistemas, pois apresenta uma estrutura padrão simples e possibilita uma redução no volume de dados trafegados.

É possível destacar três componentes principais na arquitetura REST:

a) **User Agent:** utiliza um conector cliente a fim de realizar uma requisição ao servidor e recebe a resposta para este pedido. O exemplo mais comum de *User Agent* é o navegador Web;

b) **Origin Server:** é a fonte das representações dos recursos e responsável por receber e processar as requisições recebidas. Como exemplo, temos os servidores Web (e.g. *Apache TomCat*[3], *GlassFish*[4]); e

c) **Componentes intermediários:** entre os componentes intermediários, temos os *proxies* e *gateways*. Ambos são componentes intermediários capazes de fornecer encapsulamento para interfaces de outros serviços, tradução de dados, melhoria de desempenho ou mesmo mecanismos de segurança. A diferença entre eles é que um *proxy* é um intermediário que pode ser escolhido pelo cliente, já um *gateway* é imposto pela rede ou pelo *Origin Server*.

[2] Mais informações sobre o formato JavaScript em: http://www.w3schools.com/js/default.asp.
[3] Mais informações sobre o formato Apache Tomcat em: http://tomcat.apache.org/.
[4] Mais informações sobre o formato Glassfish em: http://glassfish.java.net/.

Observa-se que ao longo dos anos novos padrões foram sendo criados a fim de facilitar o desenvolvimento de serviços Web. Porém, o desenvolvimento de serviços Web acabou se tornando uma tarefa complexa devido a mais de 70 especificações distintas que cobrem diferentes aspectos (THIES & VOSSEN, 2008). Com o objetivo de minimizar a complexidade no desenvolvimento de serviços Web, foram desenvolvidos os chamados Serviço Web Restful, que serão discutidos a seguir.

2.2.2 Serviços Web Restful

Os serviços Web Restful surgiram como uma forma de simplificar o desenvolvimento de serviços, seguindo os princípios da arquitetura REST, de modo a garantir a mesma portabilidade alcançada pela Web e tornando mais fácil a publicação e utilização desses serviços (PAUTASSO et al., 2008). Assim, os padrões já existentes e amplamente usados na Web como HTTP, XML, URI, entre outros, foram empregados para o desenvolvimento dos serviços Restful, criando uma alternativa para o uso indesejável de uma diversidade de padrões dos Web Service baseado no SOAP.

Os serviços Restful, por seguir os princípios da arquitetura REST, são orientados a recursos e possuem quatro propriedades importantes: *endereçamento*, *comunicação sem estado*, *conectividade* e *interface uniforme* (RICHARDSON & RUBY, 2007). Nas subseções seguintes será detalhado o Restful com as tecnologias Web que tornam possíveis as implementações dessas propriedades.

Uma aplicação é considerada *endereçável* se expõe os aspectos interessantes do seu conjunto de dados como recursos. Portanto, um serviço endereçável deve expor uma URI para cada parte da informação que possa vir a servir. Devido a essa propriedade, é possível acessar os recursos diretamente via URI.

Da mesma forma que a propriedade de endereçamento considera que cada recurso deve possuir sua própria URI, a propriedade de *comunicação sem estado* afirma que cada possível estado do servidor é um recurso e portanto deve possuir sua própria URI, tornando assim cada requisição totalmente desconectada das outras.

Eliminando a necessidade de armazenamento de estado entre as requisições, é possível acrescentar algumas funcionalidades, como, por exemplo, realizar o *bookmark* da URI com o estado desejado para acesso posterior, evitando a necessidade de percorrer dezenas de estados até chegar aonde deseja. É possível ainda realizar o balanceamento de carga das requisições entre diferentes servidores, pois as requisições são totalmente independentes.

As representações dos recursos na maioria das vezes são mais que apenas dados serializados, pois geralmente são hipermídias: documentos com dados e possivelmente hiperlinks para outros recursos. Dessa forma, esses documentos podem conter URIs de outros possíveis estados da aplicação que de alguma forma estão conectados ao recurso que está sendo acessado, seguindo um dos princípios da arquitetura REST que enuncia a “hipermídia como motor do estado da aplicação”. Essa qualidade de permitir a conexão por meio de hiperlinks é chamada de *conectividade* (FIELDING, 2000).

A *interface uniforme* requerida pelos serviços Restful é alcançada por meio dos quatro métodos básicos do protocolo HTTP para as quatro operações mais comuns sobre um recurso. São elas:

1. **PUT:** criar ou sobrescrever um novo recurso;
2. **GET:** recuperar uma representação de um recurso;
3. **POST:** atualizar um recurso já existente; e
4. **DELETE:** excluir um recurso já existente.

Esses métodos são aplicados a uma URI existente para efetuar alguma operação desejada sobre um recurso. Para criar um novo recurso ou sobrescrever um já existente é utilizado o método PUT. Para excluir um recurso o método DELETE é utilizado, e a resposta à requisição pode conter uma mensagem de *status* ou simplesmente nenhuma informação. O método GET retorna uma representação de um recurso no corpo da resposta da solicitação. Em ambos os casos usualmente é necessário incluir a representação do recurso no corpo da requisição a menos que os dados a serem criados ou sobrescritos sejam simples e portanto incluídos como parâmetros da URI. No caso de uma solicitação para sobrescrever os dados, é necessário passar a URI já existente do recurso; no caso de criação de um novo recurso é necessário informar uma nova URI não existente.

O método POST possui um comportamento mais complexo em relação aos demais métodos. Ele é utilizado para criar novos recursos “subordinados”, ou seja,

que dependem de outros recursos já existentes ou ainda para alterar o conteúdo de um recurso sem sobrescrevê-lo. Em geral, quando usado para criação, o método POST é aplicado sobre o recurso "pai" ou também chamado de "produtor", que é responsável por criar o novo recurso e informar a URI gerada via o cabeçalho *Location* contido na resposta HTTP. Quando usado para alteração, este módulo apenas anexa o conteúdo da representação transcrita no corpo da requisição ao recurso.

Em uma arquitetura baseada em recursos é fundamental que todos os métodos HTTP sejam utilizados de acordo com esta interface uniforme, pois dessa forma é possível prever o comportamento de cada requisição. Quando a interface não segue conforme essa especificação, podem ocorrer consequências desastrosas, como, por exemplo, utilizando um método GET para excluir um recurso é possível que os clientes do serviço desejem realizar uma busca do recurso e ao invés disso ele será removido, consequentemente perdendo os dados.

Os recursos REST normalmente são representados por meio de *Multipurpose Internet Mail Extensions type* (*MIME-type*). O tipo do *MIME-type* identifica o tipo de representação contida no corpo de uma solicitação/requisição HTTP. Normalmente são utilizados o JSON[5] ou XML para representar o formato padrão dos documentos. No padrão XML tem-se o uso da linguagem XML para a descrição de dados, enquanto em um padrão JSON utiliza-se o formato do documento de intercâmbio de dados JSON, o qual apresenta uma estrutura simples e possibilita uma redução no volume de dados trafegados.

A Figura 2.2 apresenta o recurso "*patient*" descrito no formato JSON e XML. Considerando que o formato JSON possibilita uma redução no volume de dados trafegados pela rede, ele será utilizado para descrever os recursos acessados por meio dos serviços Web da abordagem proposta.

[5] Mais informações sobre o formato JSON em: http://www.json.org.

JSON	XML

{
“id”: “1”,
“name”: “Alexandre”
“age”: “27”
“bloodtype”: “B+”
}
	<?xml version=”1.0" encoding=”UTF-8"?>
	<patient>
	<id>1</id>
	<name>Alexandre</name>
	<age>27</age>
	<bloodtype>B+</bloodtype>
	</patient>

Figura 2.2. O recurso “*patient*” descrito em formato JSON e XML.

2.2.3 Composição de Serviços Web Restful

A composição de serviços Web Restful consiste em construir um novo recurso com as funcionalidades fornecidas pelo conjunto de recursos existentes (PAUTASSO, 2009) como mostrado na Figura 2.3. O recurso C é um recurso composto, no qual engloba as funcionalidades dos recursos R e S. O recurso composto C pode então manter o seu próprio estado independente e caso ocorra transição neste estado é possível provocar a interação com os seus recursos componentes.

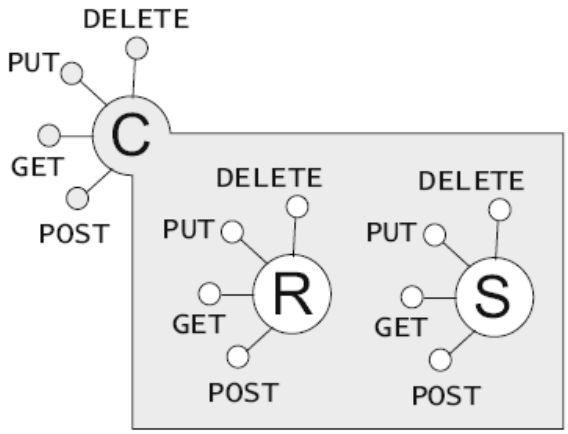

Figura 2.3. Composição de Serviços Restful (PAUTASSO, 2009)

Atualmente já existem algumas abordagens para a composição de serviços Restful como o *Business Process Execution Language (BPEL)* for REST (PAUTASSO, 2008) que propõe uma extensão à linguagem BPEL permitindo o seu uso com uma arquitetura orientada a recursos. Existe também uma abordagem (RAULF et al., 2010) que propõe a utilização da modelagem *Unified Modeling Language (UML)* (OMG, 2009) para criação de serviços Restful compostos.

2.2.4 Descritores de Serviços Web Restful

Existem diversas discussões sobre as interfaces de descrição de serviços Restful. Uma das linguagens mais difundidas disponíveis para isso é a *Web Application Description Language (WADL)*. Com a liberação da WSDL 2.0 tornou-se possível também utilizá-la para descrever um serviço Restful.

A WADL proposta por Marc Hadley (HADLEY, 2006) foi concebida para prover um formato descritivo para aplicações Web, especialmente as que utilizam XML. Em outras palavras, é uma espécie de *Interface Definition Language (IDL)* para aplicações Web (SUN, 2006).

A WADL é um vocabulário XML que expressa o funcionamento de recursos HTTP. Foi nomeada em analogia a conhecida linguagem para descrição de *Web service* – WSDL, utilizada para descrever serviços baseados em SOAP e no estilo RPC. A WADL informa quais são os recursos disponíveis, os métodos permitidos em cada um deles, e os parâmetros de entrada e saída dos serviços. É possível disponibilizar um arquivo WADL que descreva todos os recursos disponíveis pelo serviço criado, funcionando como um manual para o cliente interessado na utilização deste. A Figura 2.4 mostra a estrutura do WADL. Observa-se, por exemplo, que a *tag* "*method"* possui os atributos *name* e *id*, que correspondem ao tipo de requisição e o nome do método. Para a *tag* "*response"* são descritos os tipos de informação, ou seja, a representação contida no corpo de uma solicitação/requisição HTTP.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><?xml-stylesheet type='text/xsl' href='http:/
/www.mnot.net/webdesc/wadl_documentation.xsl'?>
<application xmlns="http://research.sun.com/wadl/2006/10">
    <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Jersey: 1.0.3 04/16/2009 12:07
 AM"/>
    <resources base="http://localhost:9998/">
        <resource path="wadl">
            <method name="GET" id="getWadl">
                <response>
                    <representation mediaType="application/vnd.sun.wadl+xml"/>
                    <representation mediaType="application/xml"/>
                </response>
            </method>
        </resource>
        <resource path="application.wadl">
            <method name="GET" id="getWadl">
                <response>
                    <representation mediaType="application/vnd.sun.wadl+xml"/>
                    <representation mediaType="application/xml"/>
                </response>
            </method>
        </resource>
    </resources>
</application>
```

Figura 2.4. Exemplo WADL

2.2.5 Vantagens e Desvantagens dos Serviços Web Restful

Entre as vantagens apresentadas pela arquitetura REST estão (PAUTASSO et al., 2008) (RICHARDSON e RUBY, 2007):

- *Simplicidade no desenvolvimento*: utiliza os padrões da Web, evitando o uso indesejável de uma diversidade de padrões;
- *Escalabilidade*: permite suporte a *cache* e balanceamento de carga;
- *Possui comunicação sem estado:* torna as requisições independentes, permitindo o *bookmark* de URIs e melhoria da escalabilidade;
- *Utiliza a hipermídia com motor do estado da aplicação:* permite conectar os recursos por meio de hiperlink; e
- *Interface Uniforme:* utiliza os métodos do HTTP para realizar as operações sobre os recursos de maneira uniforme.

Como desvantagens estão:

- *Limitação ao protocolo HTTP: admite-se apenas o uso do protocolo HTTP*;
- *Ausência de suporte para requisições assíncronas: somente requisições síncronas são suportadas*;
- *Não possui repositório de serviços:* não possui um repositório central de serviços, como o UDDI, portanto a busca de serviços deve ser realizada de forma manual; e
- *Alguns proxies "impedem" requisições com o método PUT:* restrições de infraestrutura, como, por exemplo, *firewalls*, impedem o envio de requisições com o método HTTP PUT.

2.3 Desenvolvimento Dirigido a Modelos

O Desenvolvimento Dirigido a Modelos é uma abordagem abstrata de criação de software, que foca na construção de modelos como artefatos integrantes no desenvolvimento e na definição de transformações, a serem aplicadas nestes modelos para geração de outros modelos ou de código-fonte. O sucesso dessa abordagem depende diretamente do modelo gerado no que diz respeito à relevância

e à qualidade da representação dos itens de informação, bem como de suas relações altamente automatizadas, ou seja, uma ferramenta ativa no desenvolvimento (STAHL & VÖLTER, 2006). Neste contexto o modelo não apenas guia as tarefas de desenvolvimento e da manutenção executadas pelos desenvolvedores, mas também é parte integrante do software, assim como o código-fonte, servindo como entrada para ferramentas de geração de código e reduzindo os esforços dos desenvolvedores (KELLYM & TOLVANEN, 2008).

A construção de modelo como uma abstração do software a ser implementado permite que os desenvolvedores foquem seus esforços na modelagem do domínio do problema e não em uma solução ao nível de programação (SCHAUERHUBER, 2006), escondendo possíveis detalhes de implementação, o que possibilita o reúso e a portabilidade.

Porém, apesar de todos os benefícios propostos pela MDD, ainda não é uma solução definitiva para todos os problemas inerentes ao processo de desenvolvimento de software. As vantagens presentes no desenvolvimento dirigido a modelos são obtidas graças à capacidade de automação no processo de geração de código-fonte, por meio de transformações que são aplicadas aos modelos, evitando que os desenvolvedores precisem executar tarefas repetitivas para a geração de código-fonte. Entre as vantagens apontadas por Kleppe et al. (2003), Bahnot et al. (2005) e Mernik, Heering e Sloane (2005) podem ser citadas:

- *Produtividade*: o aumento da produtividade no processo de desenvolvimento de software deve-se à automatização da geração de código a partir de modelos por meio do uso de ferramentas de transformação, incentivando os desenvolvedores a concentrar nas etapas iniciais de requisitos e análise. Além disso, as tarefas repetitivas de codificação são implementadas nas transformações, reduzindo tempo e esforço que podem ser despendidos em tarefas mais importantes;
- *Corretude*: as transformações utilizadas pelo MDD para a geração de código-fonte evitam introduzir erros acidentais por parte dos programadores, evitando, por exemplo, erros de digitação, garantindo dessa forma um código mais correto e livre de erros manuais;
- *Portabilidade*: diferentes transformações podem ser aplicadas em um mesmo modelo para gerar código-fonte para diferentes plataformas;

- *Manutenção*: a manutenção ou evolução do software é realizada diretamente nos modelos, e, aplicando as transformações nos modelos, é então atualizado o código-fonte;
- *Documentação*: no MDD os modelos são os artefatos principais do processo de desenvolvimento de software e por essa razão não se desatualizam, pois o código é gerado a partir desses. Desta forma, a documentação do sistema é mantida sempre atualizada;
- *Comunicação*: no MDD os modelos facilitam a comunicação com os diferentes profissionais, uma vez que são modelos mais simples de compreender do que o código-fonte. Os especialistas do domínio têm o papel ativo na construção do software uma vez que podem participar ativamente na construção do modelo identificando mais facilmente os conceitos do negócio, enquanto os desenvolvedores identificam os elementos técnicos;
- *Reutilização*: o reúso é realizado no âmbito de modelos em vez de no âmbito de código-fonte. A reutilização de modelos por meio de ferramentas de transformações possibilita automaticamente gerar código-fonte para várias plataformas, sem a necessidade da realização de tarefas manuais por parte dos desenvolvedores, o que não ocorre no desenvolvimento tradicional; e
- *Verificações* e *otimizações*: a execução de verificadores semânticos e otimizações automáticas específicas do domínio possibilitam reduzir a ocorrência de erros semânticos e prover implementações mais eficientes.

Segundo os autores Thomas (2004) e Ambler (2003), algumas das principais desvantagens trazidas pelo MDD são:

- *Rigidez*: o software criado por meio MDD possui uma maior rigidez, pois grande parte do código é gerado automaticamente e permanece além do alcance do desenvolvedor;
- *Complexidade*: as ferramentas utilizadas para o MDD, como ferramentas de modelagem, transformadores e geradores de código, possuem uma complexidade maior no seu desenvolvimento e na sua

manutenção, o que acaba adicionando complexidade ao processo de desenvolvimento de software;

- *Desempenho*: os geradores de código na maioria das vezes incluem código adicional ao software, o que acaba degradando o seu desempenho em relação aos códigos escritos à mão; e
- *Alto investimento inicial*: a prática do MDD exige um investimento inicial maior, pois a construção de uma infraestrutura de reutilização orientada a modelos exige mais esforço e tempo.

Segundo Lucrédio (LUCRÉDIO, 2009), o MDD é executado seguindo um processo de desenvolvimento utilizando modelos como parte integrante do código, e combinando ferramentas de modelagem e de transformações e mecanismos para execução das transformações, conforme mostra a Figura 2.5. O engenheiro de software é responsável por construir os modelos usando uma ferramenta de modelagem e também por criar as regras de mapeamento na ferramenta para definir as transformações. Uma vez construídos, estes artefatos servem de entrada para o mecanismo executar as transformações que efetivamente aplica as regras de mapeamento definidas, a fim de gerar novos modelos e/ou código-fonte.

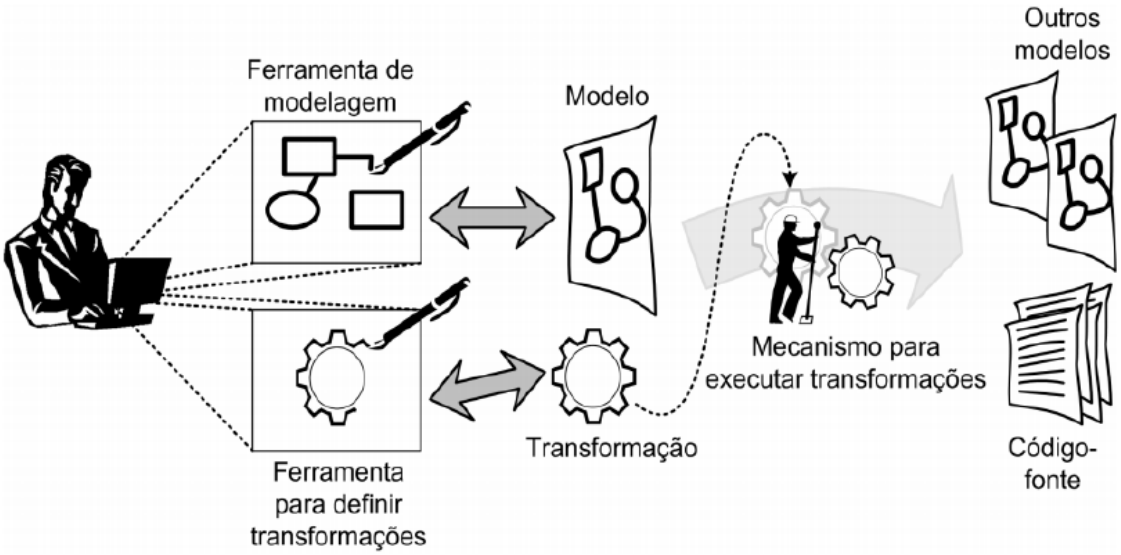

Figura 2.5. Principais Elementos do MDD (LUCRÉDIO, 2009)

Ainda segundo Lucrédio (LUCRÉDIO, 2009), para a aplicação efetiva das tarefas na metodologia MDD são necessários os seguintes elementos:

- *Ferramenta de modelagem*: Os modelos são criados pelo engenheiro de software a fim de descreverem os conceitos do domínio, de acordo com uma DSL. Como os modelos serão utilizados como entrada para a

geração de código-fonte é necessário que estes estejam semanticamente completos e corretos de acordo com a DSL, pois serão processados pelo mecanismo de execução de transformações e não por um ser humano;

- *Ferramenta para definir transformações*: para que os modelos sejam transformados em código-fonte ou mesmo em outros modelos, é necessário definir regras de transformações para efetuar os mapeamentos de modelo para modelo, ou modelo para código-fonte. Portanto, é imprescindível uma ferramenta que permita a definição dessas regras da forma mais natural possível; e
- *Mecanismos para executar transformações*: uma vez que o modelo e as transformações já foram definidas, é preciso um mecanismo que execute essas transformações. Além disso, é importante que esse mecanismo mantenha informações de rastreabilidade, permitindo identificar a origem de cada elemento gerado.

Para que o desenvolvimento orientado a modelos possa ser utilizado de maneira consistente em domínios, pode-se adotar duas técnicas importantes: a DSM e a metamodelagem.

2.3.1 Modelagem Específica de Domínio

A Modelagem Específica de Domínio utiliza uma DSL que é projetada para ser útil a um conjunto específico de tarefas em um dado domínio do problema (SADILEK, 2008) (KELLYM & TOLVANEN, 2008). Uma DSL pode ser definida por meio de um metamodelo, o qual representa o conhecimento do domínio do problema-alvo. Por estarem restritas a domínios específicos, as DSLs normalmente são pequenas, consistindo em um conjunto de abstrações e notações. A DSL possui um poder expressivo capaz de oferecer uma sintaxe concreta e que seja familiar e intuitiva para especialistas destes domínios (VAN DEURSEN et al., 2000). Dessa forma, as DSLs expressam soluções no idioma e no nível de abstração do domínio do problema, reduzindo os esforços de tradução dos conceitos dos desenvolvedores para os especialistas do domínio (LUCRÉDIO, 2009).

O emprego de DSLs para a modelagem de aplicações, em vez de linguagens de modelagem de propósito geral (e.g., UML), possibilita criar modelos mais

específicos e completos. Recursos tais como *frameworks*, padrões de projeto e componentes podem ser incluídos aos modelos, criando uma infraestrutura que permite a execução de transformações M2C, para a geração de uma maior quantidade de código a partir da modelagem. Por não estarem relacionados a uma plataforma específica, os modelos permitem descrever os diferentes aspectos do domínio de uma forma abstrata, o que facilita a transformação dos componentes de interação abstratos, representados nesses modelos em componentes concretos para diversas plataformas-alvo (GREENFIELD et al., 2004).

A DSM utiliza a Metamodelagem como forma de representar seus elementos. Um Metamodelo descreve a estrutura dos modelos e, portanto, é uma das principais características da MDD. O metamodelo é capaz de definir de forma abstrata os construtores de uma linguagem de modelagem e seus relacionamentos, além das regras de modelagem. Dessa forma, o metamodelo e o modelo possuem relação de classe e instância, ou seja, cada modelo é uma instância do metamodelo.

Graças a essas características o uso de um metamodelo se faz necessário para contemplar atividades fundamentais para o MDD tais como: realizar transformação de modelo, construção de uma DSL, geração de código, integração de ferramentas de modelagem a um domínio e validação de modelos (STAHL & VOELTER, 2006).

2.3.2 Eclipse Modeling Project (EMP)

O *Eclipse Modeling Project* é um projeto que disponibiliza diversas especificações e *frameworks* para facilitar a construção de ferramentas que apoiam processos baseados em MDD e na DSM. Esses *frameworks* facilitam a implementação de ferramentas *Computer-Aided Software Engineering (CASE)*, geradores de código, regras de transformação entre modelos, e todos esses apoios são integrados na *Integrated Development Environment (IDE)* Eclipse[6].

Muitas são as ferramentas e *frameworks* que podem ser usadas por meio do Eclipse, porém será focado somente em duas utilizadas nesse trabalho: *Eclipse Modeling Framework (EMF)* e *Java Emitter Templates (JET)*.

O EMF é um *framework* de modelagem para a construção de ferramentas e outras aplicações baseadas em um metamodelo que expressa uma DSL. A partir de

[6] http://www.eclipse.org/.

uma especificação de um modelo descrito em *XML Metadata Interchange (XMI)* (OMG, 2007) o EMF fornece ferramentas para visualização e edição em tempo de execução para produzir um conjunto de classes Java para um modelo. Assim, o EMF permite aos desenvolvedores construir aplicações baseadas em modelos de forma rápida e simples (STEINBERG et al., 2008). Para a transformação de M2C, pode ser utilizado o *framework* JET. Com esse *framework* são definidos *templates* para uma plataforma-alvo, que pode ser uma linguagem de programação, uma linguagem de marcação ou até mesmo um arquivo-texto. Os elementos do modelo são usados como parâmetros para os *templates* e como entrada em ferramentas de transformação para a geração de código-fonte ou outro modelo-alvo.

2.3.3 MVCASE

A MVCASE é uma ferramenta CASE inicialmente desenvolvida em 1997 por alunos de mestrado e evoluída por alunos de iniciação científica do Departamento de Computação da Univerdidade Federal de São Carlos - UFSCar. Em seu último *release* a MVCASE foi reconstruída para suportar a versão 2.1 da UML sendo disponibilizada como um *plugin* no IDE Eclipse. Na sua implementação foram reutilizados os *frameworks* do EMP que facilitam a construção de ferramentas de modelagem. A MVCASE também permite a geração de código e a integração de *frameworks*, tanto nos modelos quanto na geração de código (LUCRÉDIO et al., 2003).

2.4 Considerações Finais

Esse capítulo apresentou uma revisão da literatura a respeito dos principais conceitos e técnicas envolvidas no trabalho elaborado nesta dissertação. Foram discutidos os conceitos relacionados à Arquitetura Orientada Serviços e a Arquitetura REST, incluindo as tecnologias de implementação de Serviço. Além disso, foram apresentados os conceitos da MDD e DSM, aplicados neste trabalho como também as tecnologias e *frameworks* que apoiam os processos baseados em MDD e DSM.

Capítulo 3

ABORDAGEM PROPOSTA

Este capítulo apresenta a proposta deste trabalho de mestrado: uma Abordagem para Desenvolvimento de Aplicações Móveis com Reúso de Software baseado nos conceitos e técnicas apontadas no Capítulo 2. Tendo em vista auxiliar o desenvolvimento de software por meio de reúso em dispositivo móveis, na abordagem proposta são combinadas as concepções de serviço Web da SOA e a as tecnologias e conceitos da DSM, focando-se na modelagem de serviços Web e na transformação de modelos em código-fonte para diferentes plataformas. Com isso, parte da complexidade envolvida na codificação dos serviços Web e do reúso nos aplicativos móveis fica encapsulada nas transformações, o que contribui para redução dos esforços, diminuição de erros de codificação, aumento da produtividade e melhoria no tempo de desenvolvimento das aplicações móveis.

Este capítulo está organizado da seguinte forma: a Seção 3.1 apresenta uma visão geral da abordagem proposta, apresentando seus principais elementos e atividades, detalhados nas seções subsequentes (Seções 3.2 e 3.3); e a Seção 3.4 encerra o capítulo apresentando algumas considerações finais.

3.1 Etapas da Abordagem Proposta

Considerando as ideias apresentadas na revisão bibliográfica, essa dissertação investiga e propõe uma Abordagem para Reúso de Software com o intuito de aumentar a produtividade e diminuir o tempo de desenvolvimento de

aplicações para dispositivos móveis. O reúso de software na abordagem proposta é baseado na DSM e na SOA. Foi elaborada uma linguagem específica de domínio guiada por documentos, diagramas e entrevistas para representar um domínio específico. Um *plug-in* editor de modelos foi desenvolvido com o objetivo de oferecer o suporte computacional, em conjunto com um Gerador de Código construídos especificamente para cada domínio.

Conforme apresentado no diagrama *Structured Analysis and Design Technique (SADT)* (ROSS, 1977) da Figura 3.1, baseando-se nas ideias de reúso de software, na DSM e em SOA, a abordagem de desenvolvimento de software proposto nesse trabalho é realizada em duas etapas: *Engenharia de Domínio (ED)* e *Engenharia de Aplicação (EA)*. Essa divisão em duas etapas origina-se de atuais abordagens da Engenharia de Software conhecidas como Linhas de Produtos de Software (*Software Product Lines - SPL*) (GOMAA, 2004). Nas abordagens de SPL é desenvolvida uma *família de programas* de um domínio, onde na primeira etapa, denominada de *Engenharia de Domínio*, é desenvolvida uma série de artefatos genéricos para o domínio do problema e na segunda etapa, denominada *Engenharia de Aplicação*, utilizam-se os artefatos desenvolvidos na primeira etapa para desenvolver os aplicativos por meio do reúso.

Neste trabalho, a ED possui o mesmo sentido da SPL, ou seja, concentra-se no desenvolvimento de artefatos de software para posterior reutilização. De forma mais geral, a ED constitui-se como uma abordagem de identificação e organização do conhecimento acerca de uma classe de problemas – o domínio do problema – para apoiar sua descrição e solução. Seu objetivo é sistematizar a criação de modelos do domínio, arquiteturas e conjuntos de artefatos de software para dar suporte à construção de aplicações em um domínio de problema particular (BLOIS et al., 2005).

No diagrama da Figura 3.1, os retângulos representam as grandes etapas da abordagem. As setas que incidem pelo lado esquerdo dos retângulos significam as entradas de dados/artefatos, e as setas que saem pelo lado direito representam as saídas geradas em cada atividade. As setas do lado superior representam os controles que orientam a execução de cada atividade. As setas do lado inferior representam os participantes, ferramentas e mecanismos que executam ou automatizam a execução das atividades. Essas duas etapas são interativas e

incrementais, geram diferentes artefatos e são compostas por disciplinas com atividades, técnicas e papéis associados.

A ED é responsável por desenvolver os artefatos reutilizáveis que apoiam o desenvolvimento das aplicações móveis via reúso dos serviços Web. Esses artefatos compreendem uma DSL, representada via metamodelo instanciado para um domínio, e a construção do *kernel* de serviços que atendem o domínio e as transformações M2C apropriadas para a geração de código. Os artefatos provenientes da etapa de ED são armazenados no repositório de artefatos para serem posteriormente reusados.

Na EA o Engenheiro de Software, baseado nas ideias para reúso dos artefatos produzidos na ED, parte dos requisitos da aplicação, e com apoio do Metamodelo Específico do Domínio disponível como um *plugin* no IDE Eclipse, o reúso do *kernel* de Serviços e dos Geradores de Código, desenvolve de forma mais ágil as aplicações móveis do domínio.

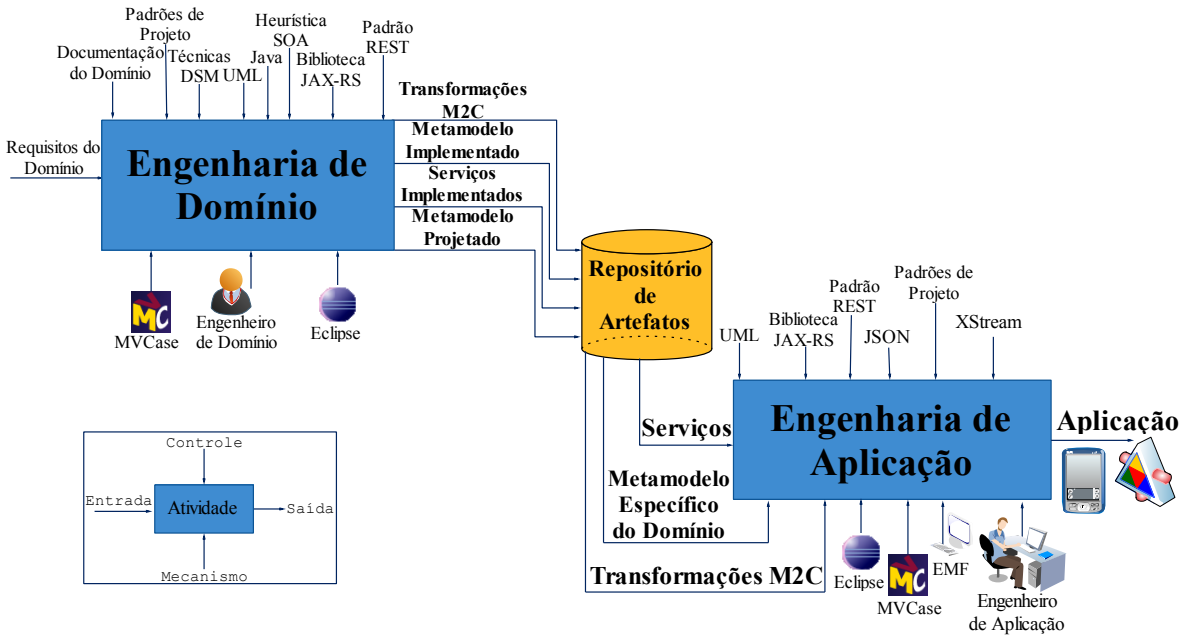

Figura 3.1. Visão geral em alto nível da Abordagem para Desenvolvimento de Aplicações Móveis com Reúso de Software baseado em DSM e SOA

Na abordagem proposta, tanto na ED como na EA, as atividades para construção dos artefatos estendem as disciplinas tradicionais de *Análise*, *Projeto*, *Implementação* e *Testes* da Engenharia de Software, sendo estas adaptadas para atender às necessidades da DSM na construção dos aplicativos para dispositivos móveis.

3.2 Engenharia de Domínio (ED)

Usando a notação de diagramas SADT, a Figura 3.2 mostra as atividades da ED na abordagem: *(1) Especificar Domínio do Problema*, *(2) Projetar Metamodelo do Domínio do Problema*, *(3) Implementar Metamodelo do domínio do Problema*, *(4) Construir Transformações Modelo para Código* e *(5) Implementar Serviço*. A cada ciclo das atividades da ED é gerada uma nova versão do metamodelo, das transformações e dos serviços. Versões mais refinadas são obtidas repetindo-se essas atividades com novos requisitos do domínio do problema.

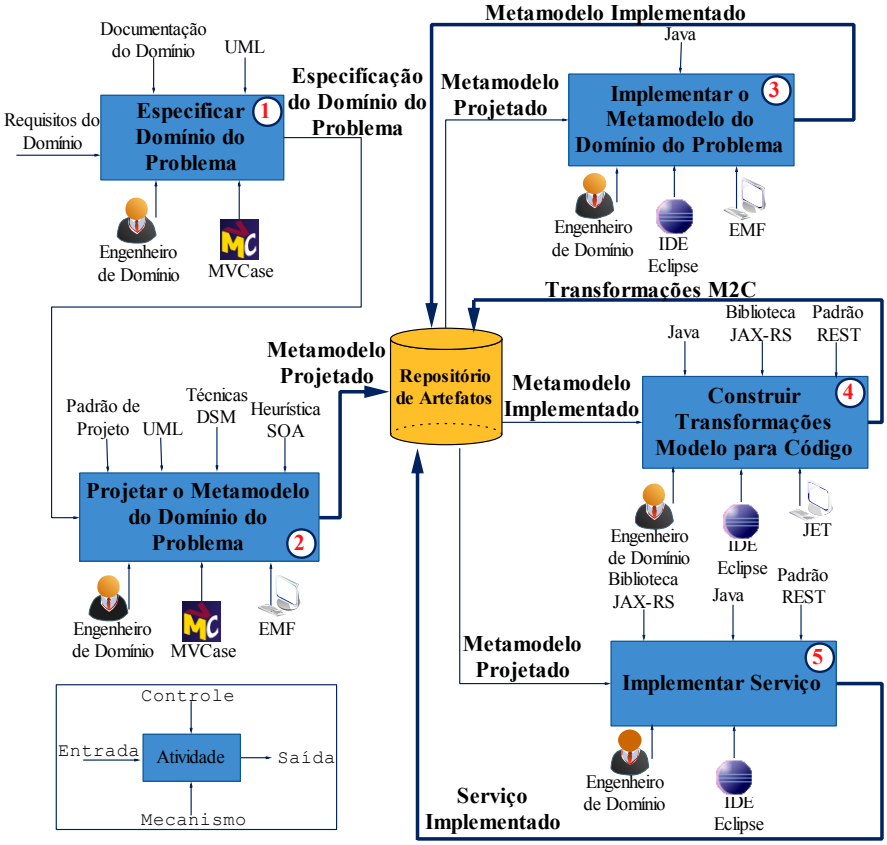

Figura 3.2. Engenharia de Domínio.

3.2.1 Analisar Domínio do Problema

A ED inicia com a análise dos requisitos do domínio do problema para a construção de uma DSL que será representada em um metamodelo. Nesta atividade os requisitos do domínio são elicitados, especificados, analisados e representados em diagramas UML. Documentações e outras fontes de informações, juntamente

com a orientação de um especialista no domínio do problema, auxiliam o Engenheiro do Domínio nesta atividade. Uma ferramenta CASE é o principal mecanismo que apoia a especificação e análise dos diagramas que modelam as diferentes aplicações do domínio do problema. O principal diagrama é o de classes. Contudo, outros diagramas podem ser utilizados para detalhar as especificações como, por exemplo, o diagrama de casos de uso, estado e o de sequência. Analisando e refatorando os diagramas das aplicações do domínio do problema, elaboram-se diagramas de casos de uso, classe e sequencia. O diagrama de classes resultante representa os objetos comuns do domínio, com seus atributos, métodos e relacionamentos. Esse diagrama contém as classes do núcleo comum das aplicações do domínio do problema.

Para melhor compreensão da abordagem proposta, o domínio *Healthcare*, com serviços de atendimento emergencial, será utilizado como exemplo. Para esse domínio realizou-se uma análise a partir dos modelos especificados em projetos desenvolvidos em (LEITE, 2005), (COSTA, 2001), (ITO, et al., 2009), (GAION et al., 2010). O modelo resultante dessa análise foi refinado com orientação de um especialista do domínio, resultando em uma versão inicial do modelo de classes que deu origem ao metamodelo do domínio do problema. Também foram utilizados os seguintes documentos para compreensão do domínio do problema: literatura técnica (normas e padrões, revistas científicas), guia sobre procedimento no atendimento de emergência (STEELE, 1992), glossário médico, entrevistas com especialistas, entre outros. Como resultados dessa atividade foram identificados os atores *Patient*, *Nurse*, *Doctor*, *Driver*, *Pilot*, e as classes *History*, *Address*, *Symptom*, *Ambulance*, *Diagnostic*, *Occurrence*, *Disease*, *Helicopter*, *Procedures*, *Specialty*, *Treatment* e *OperationStation*, representados no diagrama de classes da Figura 3.3. A MVCASE (LUCRÉDIO, 2003), disponível como *plugin* do IDE Eclipse (ECLIPSE, 2011), apoiou essa atividade da abordagem.

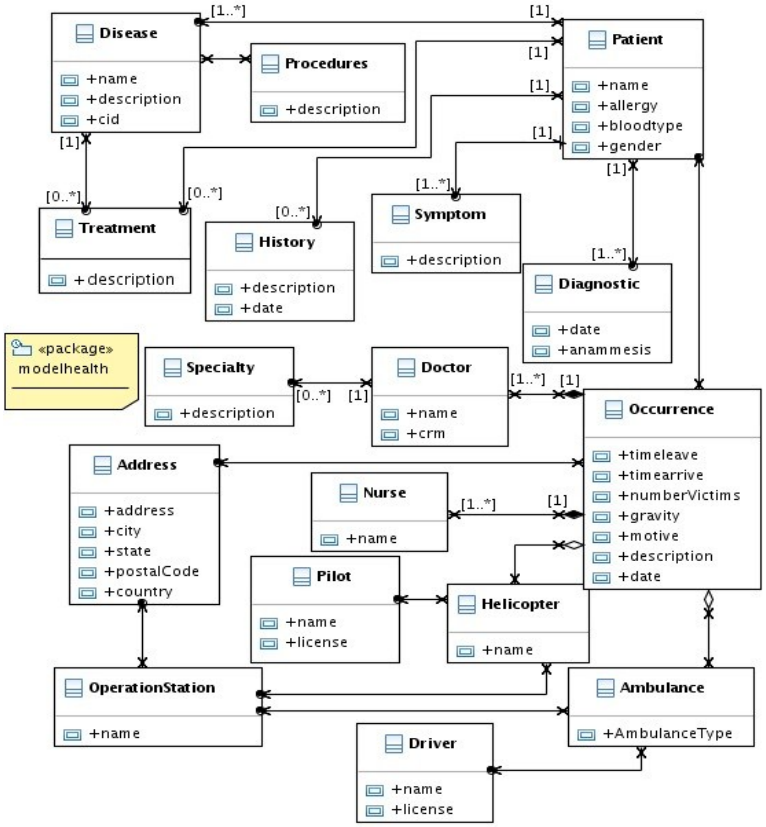

Figura 3.3. Diagrama de Classes UML do Domínio *Healthcare*

Ainda nessa atividade, o Engenheiro de Domínio identifica os serviços do domínio. Para definir os serviços o Engenheiro de Domínio elabora o diagrama de casos de uso com os requisitos funcionais comuns das aplicações do domínio. Esses diagramas produzem uma descrição consistente dos requisitos funcionais das classes e operações que o sistema deve realizar. Essa descrição auxilia o Engenheiro de Domínio nas discussões e no entendimento das funcionalidades com um especialista no domínio. Em seguida, baseado nesses diagramas, especificam-se as características, ou *features* (KANG, 1998), que refatoram as funcionalidades comuns identificadas nos casos de uso. Essas funcionalidades são indicadas como possíveis serviços do domínio do problema. A Figura 3.4 mostra um exemplo de diagrama resultante dessa análise no domínio *Healthcare*. No caso, a *feature* *LookupDisease* foi especificada a partir dos casos de uso *LookupTreatments*, *LookupProcedures* e *LookupSymptoms*. Essa *feature* é responsável pela busca de informações médicas sobre doenças descritas por meio de textos, imagens e vídeos, e dá origem ao serviço *LookupDisease*, com as operações *ResearchDisease*, *FindVideoProcedures*, e *FindVideoSymptoms*.

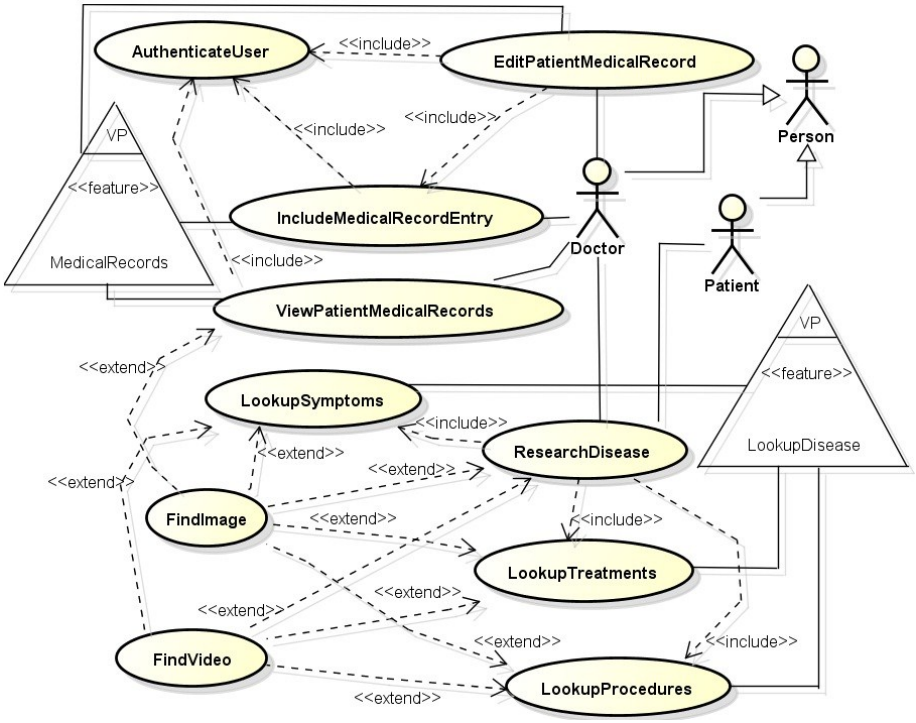

Figura 3.4. *Features* do Domínio *Healthcare*

3.2.2 Projetar Metamodelo do Domínio do Problema

Nesta atividade é projetado o metamodelo do domínio do problema, com base nas especificações resultantes da atividade de análise, e ainda considerando as decisões de projeto definidas. Dentre essas decisões têm-se a plataforma de software, os padrões, *frameworks* e outras tecnologias adotadas para a especificação do metamodelo. O metamodelo é especificado como instância do meta-metamodelo Ecore do *framework* EMF (STEINBERG, 2008). Para melhor explicar a construção do metamodelo *Healthcare*, suas metaclasses foram organizadas em quatro partes: **Parte 1** - Metaclasses da Tecnologia Restful; **Parte 2** - Metaclasses de Serviços *Healthcare*; **Parte 3** - Metaclasses da UML; e **Parte 4** - Metaclasses de Entidades *Healthcare*.

A Figura 3.5 mostra as metaclasses da **Parte 1**, da tecnologia Restful, utilizadas para construção de serviços na arquitetura REST (FIELDING, 2000). A tecnologia para serviço Web Restful foi adotada neste trabalho porque permite a construção de soluções que sejam aderentes ao estilo arquitetural da Web que utilizam o protocolo HTTP para projetar um sistema hipermídia distribuído, que enfatiza a escalabilidade dos componentes de interação, a interoperabilidade das interfaces, e o desenvolvimento de componentes independentes (WANG et al.,

2010). Além disso, a abordagem com REST tem foco nos recursos que devem ser acessados e utiliza a URI do recurso, além do pequeno conjunto de operações HTTP para o acesso. Assim, com Restful o acesso e a manutenção de serviços Web podem ser mais simples (ALMEIDA, 2011), razão pela qual ele será adotado na abordagem proposta nesse trabalho.

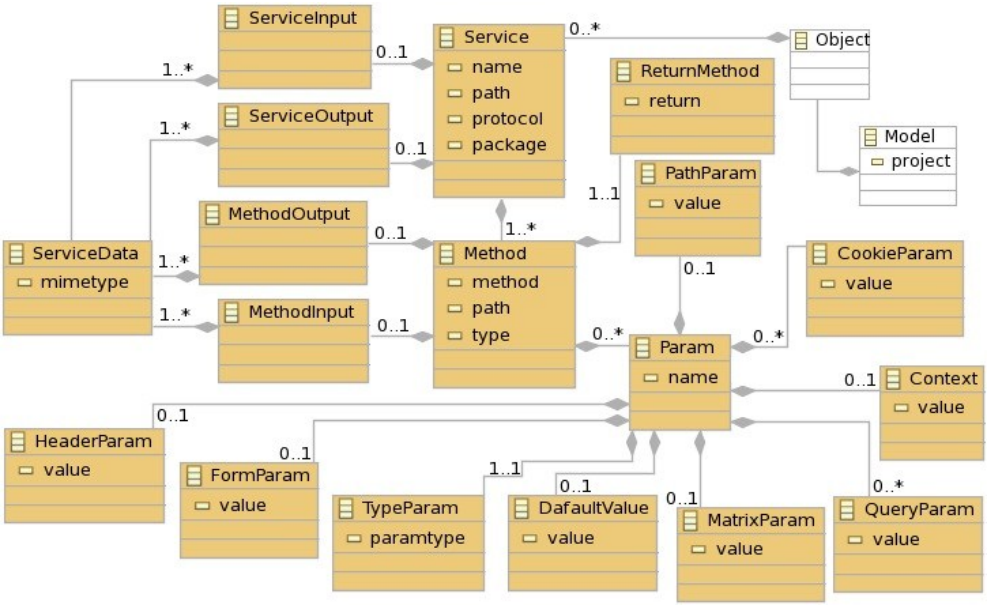

Figura 3.5. Metamodelo Parte 1 - Metaclasses da Tecnologia *Restful*

Conforme mostra a Figura 3.5, foram definidas as metaclasses *Service*, *Method*, *ReturnMethod*, *Param*, *ServiceInput*, *ServiceOutput*, *MethodInput*, *MethodOutput*, *ServiceData*, *PathParam*, *CookieParam*, *Context*, *QueryParam*, *MatrixParam*, *DefaultValue*, *TypeParam*, *FormParam*, e *HeaderParam*. *Service* representa as classes de serviços. Essas metaclasses possuem os meta-atributos *name*, *path*, *protocol* e *package* que representam o nome do serviço, a localização, o tipo de protocolo (HTTP da tecnologia Restful) e o pacote que contém os serviços. *Service* relaciona-se com as metaclasses *ServiceInput* e *ServiceOutput* que representam o tipo de dados da informação recebida e produzida por um serviço. *ServiceInput* e *ServiceOutput* relacionam-se com a metaclasse *ServiceData* que representa o tipo de dados da informação. *ServiceData* possui um meta-atributo *mimetype* que representa o nome do tipo do dado da informação. A metaclasse *Method* representa os métodos dos serviços, e tem os meta-atributos *method*, *path* e *type* que indicam respectivamente o nome do método do serviço, a localização do método do serviço, e o tipo de operação do método (atualização, consulta, inclusão, ou exclusão). As metaclasses *MethodInput* e *MethodOutput* representam o tipo de

dados da informação recebida e produzida por um método, e relacionam-se com a metaclasse *ServiceData*.

Em seguida, o Engenheiro de Domínio projeta a **Parte 2** do metamodelo responsável pelos Serviços do Domínio do Problema. O projeto parte do modelo de *features*, e consiste em elaborar um diagrama em que cada classe representa um serviço com os respectivos protótipos de métodos. As classes são organizadas em pacotes e utilizam estereótipos (*Kernel*, *Service* e *Rest*) para orientar a implementação dos serviços. A Figura 3.6 mostra um diagrama com especificações de serviços para o domínio adotado. No caso, a classe *LookupDisease* do pacote *br.ufscar.dc.healthcare.Resources* foi projetada com base na *feature* *LookupDisease*. Essa classe representa o serviço *LookupDisease* com suas operações. Por exemplo, o método *researchDisease(name: String): Response (A)* implementa uma das suas operações.

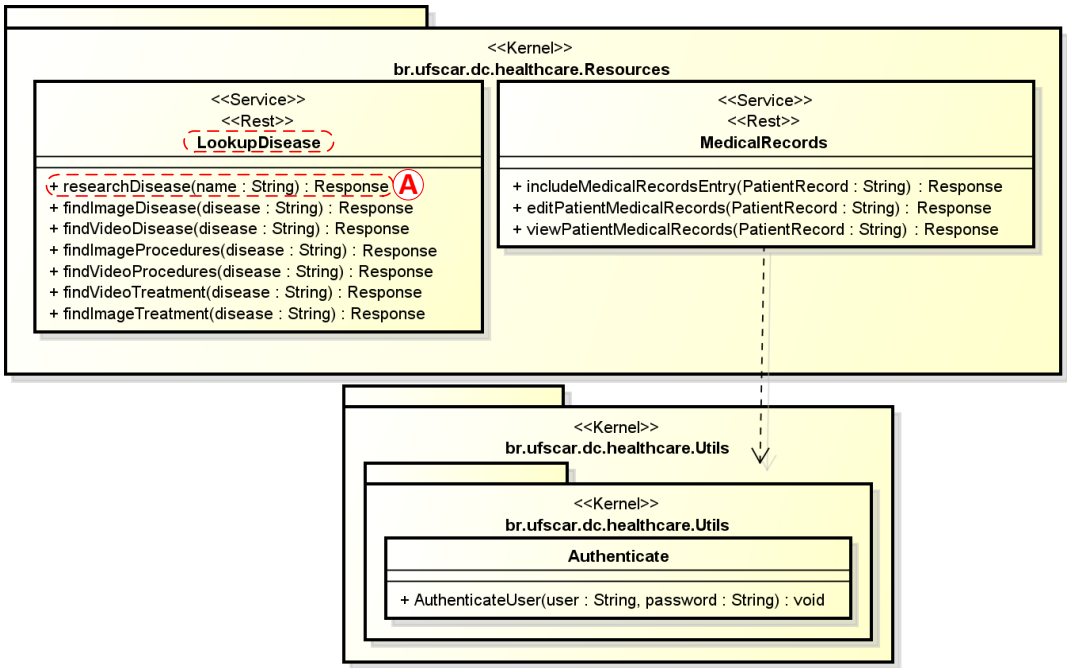

Figura 3.6. Diagrama de classes dos Serviços do Domínio do *Healthcare*

Com base no diagrama de classes de serviços, Figura 3.6, o Engenheiro de Domínio especifica, conforme apresentado na Figura 3.7, as metaclasses dos serviços do domínio *Healthcare*. A metaclasse *ReuseSet* representa o grupo de metaclasses para reúso dos serviços, descritos nas *features*, e representados por meio da metaclasse *ServiceReuse*. Essa metaclasse possui os meta-atributos *name* e *path* que descrevem o nome e o caminho do serviço. Derivadas de *ServiceReuse* têm-se as metaclasses que definem os serviços do domínio específico. Por exemplo,

as metaclasses *Medicalrecords* e *Lookupdisease* representam respectivamente o serviço de registros médicos do paciente e o serviço de busca de informações sobre uma determinada doença. A metaclasse *MethodReuse* representa os métodos de um serviço, com os meta-atributos *name* e *type*. Derivadas de *MethodReuse* têm-se as metaclasses dos métodos. Por exemplo, a metaclasse *Researchdisease* representa uma operação para pesquisa de informações sobre uma determinada doença. *Researchdisease* possui relacionamento com a metaclasse *Param_disease* que representa o parâmetro do método. *Param_disease* possui o meta-atributo *datatype* que define o nome do tipo do parâmetro do método.

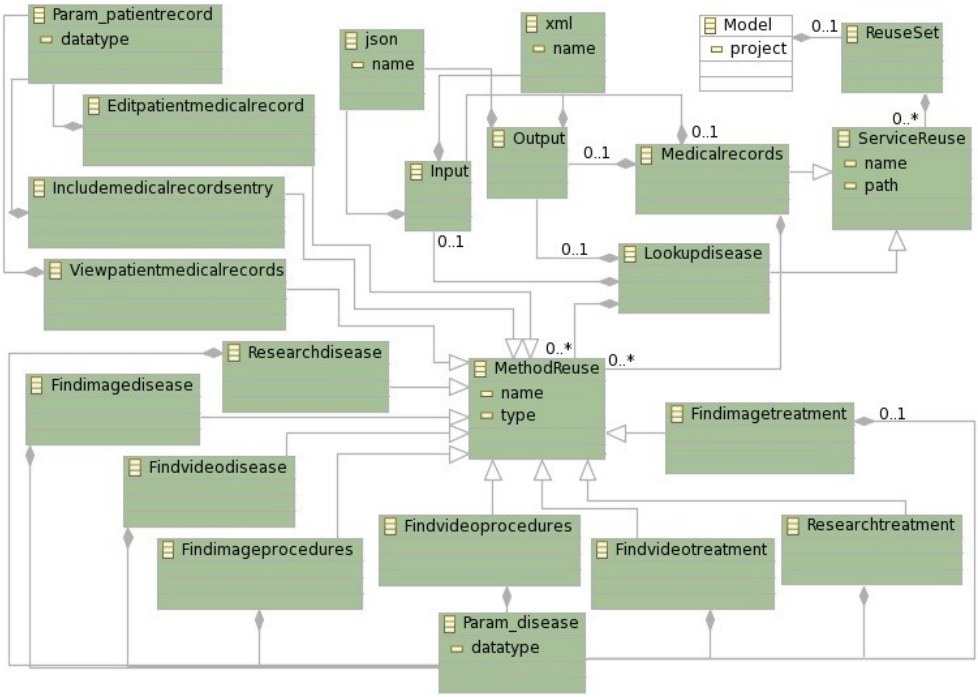

Figura 3.7. Metamodelo Parte 2 - Metaclasses de Serviços *Healthcare*

A **Parte 3** do metamodelo compreende as metaclasses para instanciação de classes na notação UML. Essas metaclasses, Figura 3.8, básicas da UML, são independentes do domínio do problema. As metaclasses da **Parte 3** têm sua raiz na metaclasse *Model*, que representa um modelo em UML. Possui o meta-atributo *project* que define o nome do projeto. *Model* relaciona-se com a metaclasse *Object* que define os objetos pertencentes à aplicação. *Object* relaciona-se com a metaclasse *Element* que descreve o estereótipo (meta-atributo *stereotype*) e o pacote (meta-atributo *package*) de uma classe. Relacionada com *Element* tem-se a metaclasse *Relationship*, que representa os relacionamentos de uma classe, descritos com os meta-atributos *cardinalityIn*, *cardinalityOut*, e *name*. Derivadas da metaclasse *Relationship* têm-se as metaclasses *Aggregation*, *Association*,

Composition, *Dependency*, e *Generalization*, conhecidos da UML. A metaclasse *Class*, derivada de *Element*, é responsável pela instanciação de classes, com seus atributos, métodos e relacionamentos. O meta-atributo *name* descreve o nome de uma classe. *Class* relaciona-se com a metaclasse *Attribute* responsável pela definição dos atributos de uma classe. O meta-atributo *atribute* descreve o nome do atributo de uma classe. *Attribute* relaciona-se com *AttributeType* que representa o tipo dos atributos, cujo nome é descrito pelo meta-atributo *attributetype*. *Class* também relaciona com a metaclasse *Method* que define os métodos de uma classe, cujo nome é descrito pelo meta-atributo *method*. *Method* relaciona-se com a *ReturnType*, que define o tipo de retorno dos métodos de uma classe, cujo tipo é descrito pelo meta-atributo *returntype*. *Method* relaciona-se, ainda, com a metaclasse *Parameter* que representa os parâmetros de um método. O meta-atributo *parameter* representa o nome do parâmetro. *Parameter* relaciona-se com a metaclasse *ParameterType*, que descreve o tipo do parâmetro por meio do meta-atributo *parametertype.*

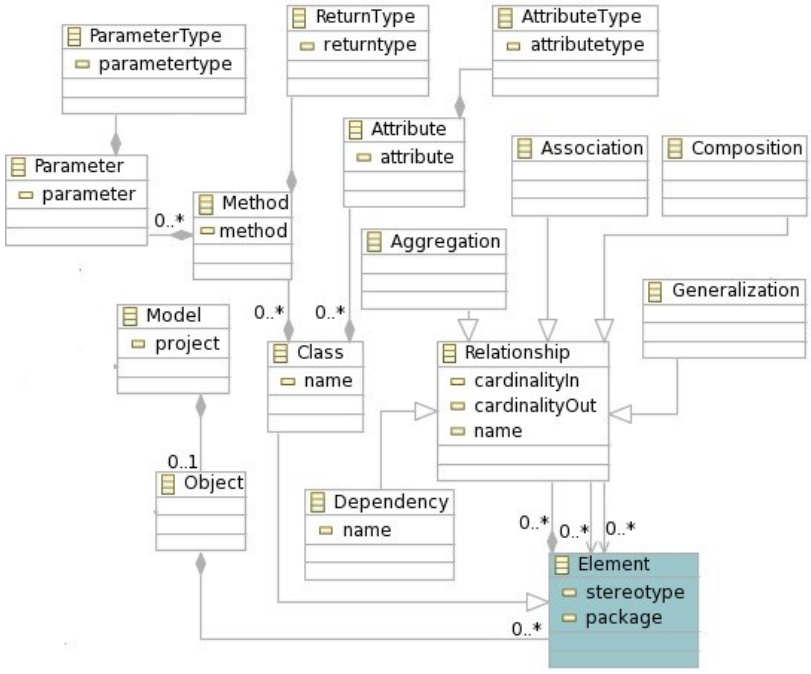

Figura 3.8. Metamodelo Parte 3 – Metaclasses da UML

A **Parte 4** do Metamodelo compreende as metaclasses para instanciação das classes de Entidades do Domínio do Problema. Assim, no metamodelo da **Erro! Fonte de referência não encontrada.**, as metaclasses com o prefixo *Att* descrevem os atributos dessas classes específicas. Por exemplo, a metaclasse *Att_id* representa o atributo que identifica uma classe, e o seu meta-atributo *datatype*

representa o tipo do atributo identificador. As metaclasses de Entidades *Address*, *OperationStation*, *People*, *Nurse*, *Patient*, *Driver*, *Doctor*, *Person*, *Specialty*, *Procedure*, *Treatment*, *Symptoms*, *History*, *Disease*, entre outras, originaram dos modelos de análise do domínio do problema (Figura 3.3).

Essas metaclasses derivam de *Element*, e, portanto, herdam seus meta-atributos, e meta-relacionamentos. Por exemplo, a metaclasse *Patient* descreve a classe Paciente, cujos atributos alergia e tipo sanguíneo são descritos pelas metaclasses *Att_allergy* e *Att_bloodtype*, respectivamente. *History* representa a classe que define o histórico de um paciente. Relaciona-se com a metaclasse *Att_description* que representa a descrição do histórico. *Disease* representa a classe Doença. Relaciona-se com *Att_name*, *Att_cid*, e *Att_description* para definir o nome, a classificação estatística internacional, e a descrição da doença. *Symptoms* representa a classe Sintoma de um paciente. Relaciona-se com *Att_description* que define o atributo de descrição do sintoma. *Specialty* representa as informações de especialidades médicas. *Doctor* representa a classe Médico. Relaciona-se com *Att_crm* e *Att_specialty* que definem o registro e a especialidade do médico. As metaclasses *Treatment* e *Diagnostic* representam as classes com informações do tratamento e diagnóstico médico efetuado por um especialista em um paciente. A metaclasse *Occurrence* descreve a classe Ocorrência que armazena informações de uma ocorrência médica. A metaclasse *Procedures* descreve a classe de um procedimento médico. A metaclasse *Address* representa a classe de endereço, por meio das metaclasses *Att_city*, *Att_state*, *Att_postalcode*, *Att_country*, e *Att_street*. As metaclasses *Nurse* e *Driver* representam as classes Enfermeira e Motorista, respectivamente. A metaclasse *OperationStation* representa a classe de uma estação de controle de emergência médica. O metamodelo provê também meta-*dataTypes*, como, por exemplo, *DataType*, para representar tipos de dados dos meta-atributos.

Concluindo, o resultado dessa etapa da abordagem é o diagrama de metaclasses do metamodelo, implementado conforme será apresentado a seguir.

3.2.3 Implementar Metamodelo do Domínio do Problema

Nesta atividade o Engenheiro de Domínio implementa o metamodelo na linguagem Java conforme as especificações do projeto usando o EMF. A Figura 3.9

mostra um trecho de código do metamodelo do domínio *Healthcare*. Na figura está destacado o código da metaclasse *PatientImpl* que é derivada da metaclasse *PeopleImpl* que implementa a metainterface *Patient*, com os meta-atributos *id* do tipo *integer*, *name*, *allergy*, e *bloodtype* do tipo *String*.

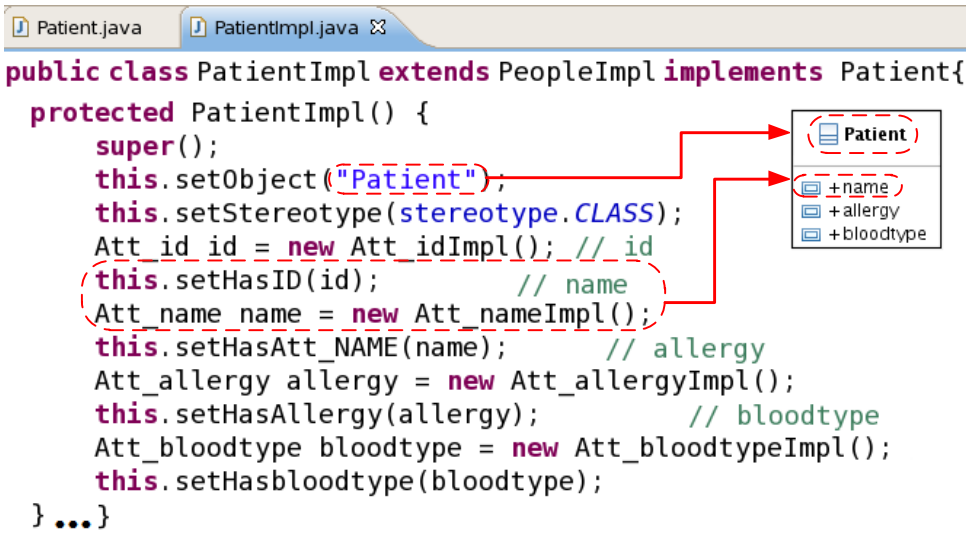

Figura 3.9. Implementação da metaclasse *Patient* do metamodelo *Healthcare*

Baseado no código Java do metamodelo, é gerado um editor de modelos. Esse editor é usado na Engenharia de Aplicação para a modelagem das aplicações. Os modelos gerados são persistidos em XMI, um padrão do *Object Management Group (OMG)* [7] utilizado para representar modelos em XML. Este formato define uma estrutura de documento que considera a relação entre os dados e seus correspondentes *metadados*, o que facilita o mapeamento dos modelos para código, usado pelas transformações M2C.

3.2.4 Construir Transformações de Modelo para Código

Nesta atividade são construídas as transformações a serem aplicadas nas instâncias do metamodelo, ou seja, nos modelos das aplicações construídos na Engenharia de Aplicação, para a geração de código. Na IDE Eclipse o Engenheiro de Domínio utiliza o *framework JET* (Eclipse, 2001), o qual disponibiliza uma biblioteca de marcações (*tags*) para metaprogramação, que implementa comandos condicionais, de laços, formatação, dentre outras funções úteis, para criar os *templates* com as regras de transformações.

[7] http://www.omg.org/.

Um *template* é um arquivo de texto qualquer instrumentado com construções de seleção e expansão de código (CZARNECKI & EISENECKER, 2000). Essas construções realizam consultas em uma entrada, que pode ser um arquivo XMI representando um modelo, e usam as informações resultantes dessa consulta como parâmetro para produzir código personalizado, em qualquer linguagem textual (LUCRÉDIO, 2009). Dessa forma, um *template* geralmente é constituído por partes fixas, as quais sempre são incluídas ao código de saída, e partes variáveis, que dependem das informações contidas no modelo de entrada para serem geradas.

A correta implementação das transformações depende do conhecimento da sintaxe da linguagem em que os modelos de entrada são criados, i.e., seu metamodelo. Dessa forma, o metamodelo *Healthcare* é usado como entrada nesta atividade de modo que as transformações produzidas sejam compatíveis com o metamodelo construído. Assim, por meio do JET, o Engenheiro de Domínio cria os *templates* que interpretarão os modelos das aplicações instanciados a partir do metamodelo para gerar o código das classes de entidades, serviços Web e para o reúso dos serviços no aplicativo móvel.

Na geração do código das aplicações decidiu-se utilizar o padrão *Model-View Controller (MVC)* (KRASNER & POPE, 1988). Esse padrão arquitetural assegura um bom grau de desacoplamento pela separação dos componentes de uma aplicação conforme o aspecto a que se relacionam (persistência de dados, lógica de negócio ou interface do usuário). A arquitetura MVC facilita a manutenção da aplicação, considerando que sua organização modular facilita a adição de novas funcionalidades.

Para que se tenha uma visão das transformações, a Figura 3.10 apresenta a regra de transformação para a geração de código de uma classe Java. No caso, as linhas 1-3 descrevem a classe *Disease* com um atributo *id* do tipo *int*, alvo da transformação. Nas linhas 4-11 tem-se o código JET com as regras para geração de código de uma classe. Essa regra é chamada pelo mecanismo que executa transformações quando, durante a leitura do arquivo de persistência do modelo em XMI, o nó "*hasElements*" é encontrado e é verificado que o elemento é uma classe. Nas linhas 6 e 7 é verificado se existe o elemento "*hasID*" para geração da declaração do atributo na classe (linha 14) com seus métodos assessores *get* e *set* – linhas 15 a 19. As linhas 12-20 apresentam o código gerado da classe *Disease*.

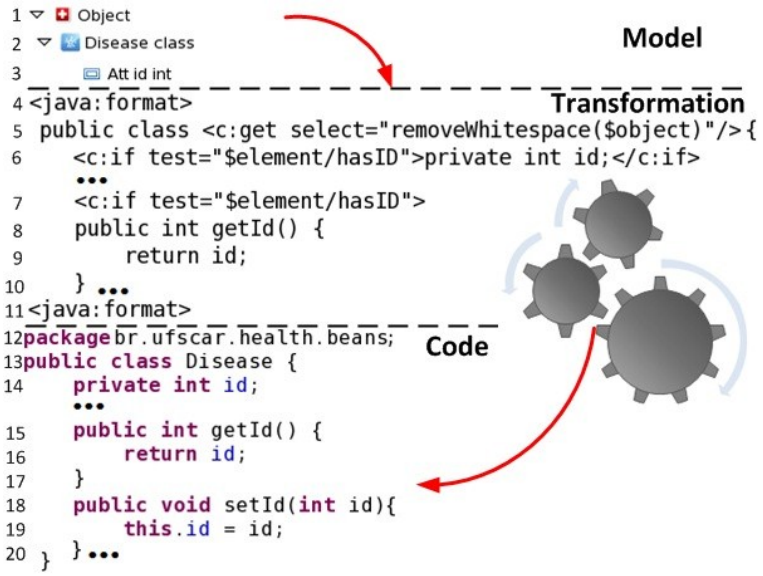

Figura 3.10. Regras de Transformações para geração de código de um serviço

A Figura 3.11 apresenta a regra de transformação para a geração de código de um serviço utilizando a tecnologia Restful. Essa regra é chamada pelo mecanismo que executa transformações quando, durante a leitura do XMI do modelo, o nó "*hasService*" é encontrado. Por exemplo, as linhas 1-8 mostram o serviço *PatientResource* com o método *insertPatient* e seu parâmetro *patient*, modelado e persistido em XMI. As linhas 9-19 apresentam o código JET com as regras para gerar serviço. As linhas 20-27 mostram o código do serviço *PatientResource* com o método *insertPatient* com as anotações da tecnologia Restful. Na linha 10 tem-se a regra para identificar o endereço do serviço por meio do elemento *@path*, dando origem ao código da linha 20, com a anotação *@Path*. Na linha 11 tem-se a regra para verificar o tipo do serviço por meio do elemento *type*, que se for do tipo *retrieve* gera o código *@GET* (linha 24). Nas linhas 14-16 são realizadas iterações sobre o elemento *Input* para gerar os tipos de entrada (*text/plain* e *application/json*) do serviço (linha 21).

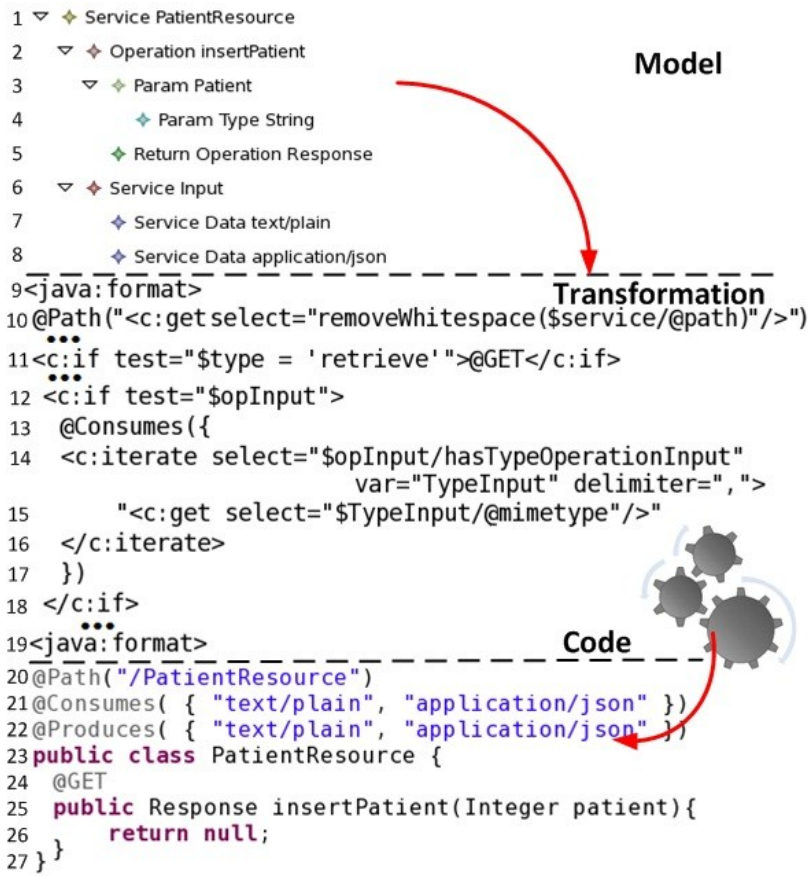

Figura 3.11. Regras de Transformações para geração de código de um serviço

A Figura 3.12 apresenta a regra de transformação para a geração de código do reúso dos serviços do domínio. Essa regra é chamada pelo mecanismo que executa transformações quando, durante a leitura do XMI do modelo, o nó "*hasServiceReuse*" é encontrado. Por exemplo, as linhas 1-8 mostram o modelo do serviço *LookupDisease* com o método *researchDisease*. As linhas 9-18 mostram o código da transformação com as regras para gerar o código cliente do serviço. As linhas 19-24 mostram o código gerado da chamada do serviço *LookupDisease* por meio da operação *researchDisease*. A linha 10 mostra a regra verificando o tipo da operação, e caso seja *retrieve* nas linhas 11-12 é gerado o código da chamada do serviço para esse tipo. Nas linhas 14-17 são realizadas iterações sobre o elemento *input* para gerar os tipos de elementos que o serviço recebe. As linhas 20-21 mostram o código gerado para criar um objeto *HttpGet* que chama vai URL o método *researchDisease* do serviço *LookupDisease*, e na linha 23 tem-se o código que atribui o tipo de dado do serviço.

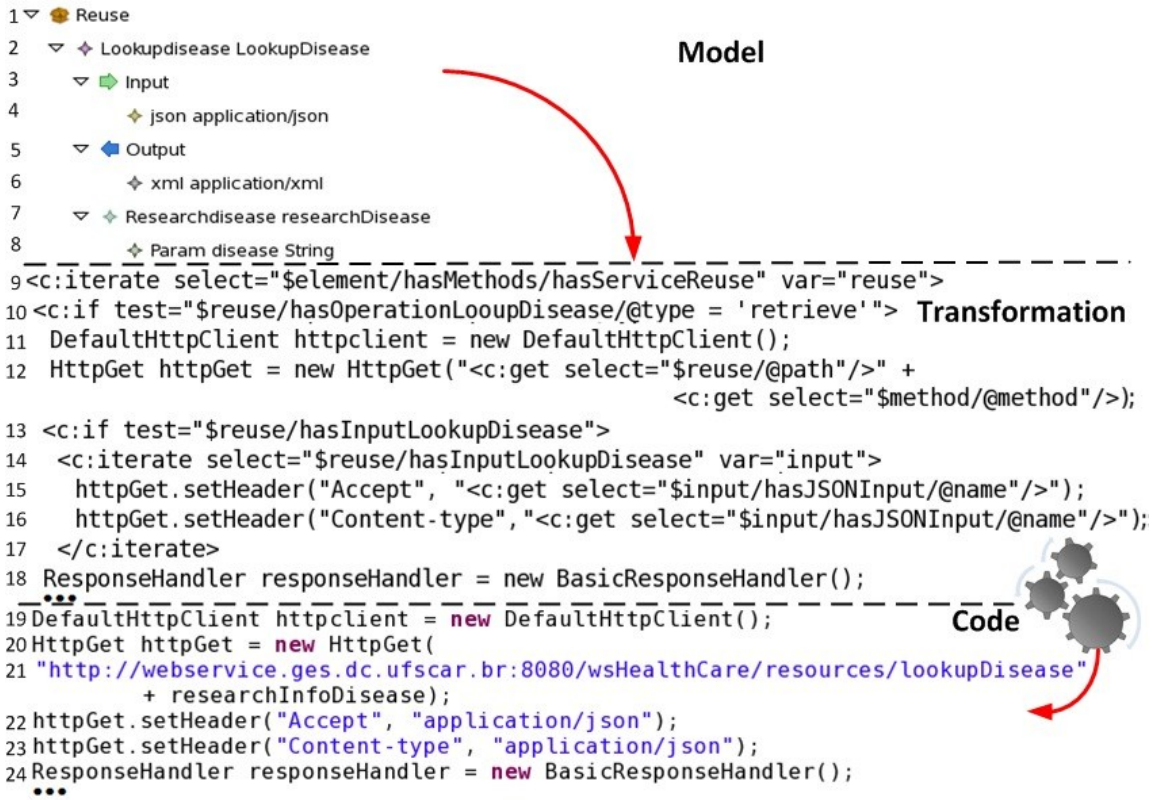

Figura 3.12. Regras de transformações para geração de código de reúso de um serviço

Da mesma forma, foram construídas as demais transformações que completam a geração de código das aplicações instanciadas do metamodelo. Para configuração e funcionamento dos serviços Web, o Engenheiro de Domínio constrói regras de transformação que adiciona as bibliotecas e arquivos de configuração conforme a tecnologia Restful. Assim, têm-se as transformações que geram o código de saída de documentos de configuração XML, e das bibliotecas *JAX-RS* (HADLEY et al., 2009) e *Xstream* (WONG et al., 2004), as quais possuem transformações para serialização de objetos de XML e JSON utilizados pelos serviços.

3.2.5 Implementar Serviço

Nesta atividade são implementados os comportamentos dos serviços especificados no projeto do metamodelo (Figura 3.7). Baseado no estereótipo e nas assinaturas dos métodos do serviço, o Engenheiro de Domínio, na IDE Eclipse, realiza a implementação seguindo a especificação da biblioteca *JAX-RS*. A Figura 3.13 mostra a implementação do serviço *LookupDisease*, destacando os métodos

reserachDisease e *findImageDisease*, cujos protótipos foram especificados na etapa de projeto do metamodelo.

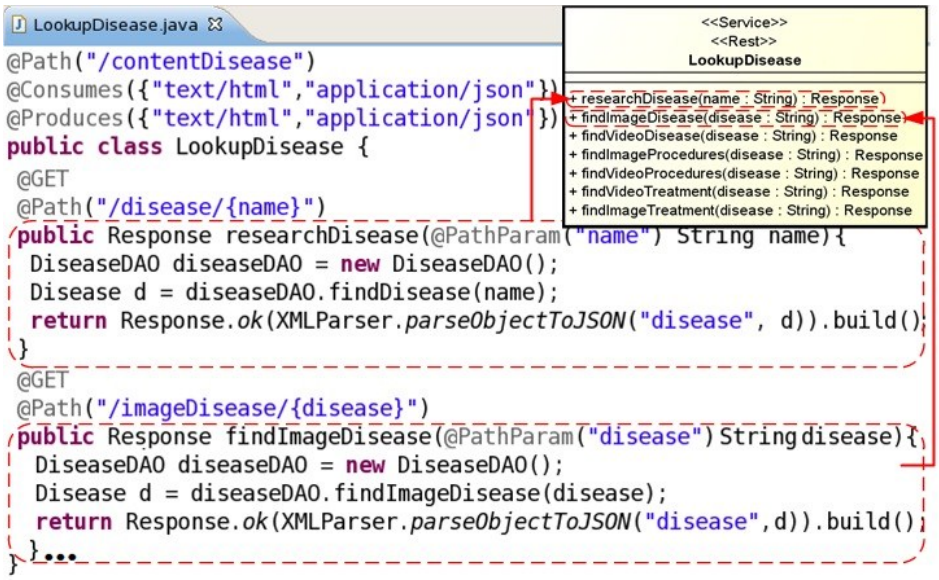

Figura 3.13. Implementação do Comportamento do Serviço LookupDisease

Depois de concluída a Engenharia de Domínio (ED), têm-se os artefatos que apoiarão a construção das aplicações na Engenharia da Aplicação. A ED é executada sempre que for necessário modificar o metamodelo, para atender novos requisitos, corrigir erros ou melhorar sua reutilização no desenvolvimento das aplicações móveis do domínio do problema. As modificações podem implicar na construção de novas transformações e novos serviços.

3.3 Engenharia de Aplicação (EA)

Na EA são construídas as aplicações do domínio específico, reutilizando os artefatos produzidos na ED. Conforme as concepções da DSM, o uso do metamodelo do domínio específico facilita a modelagem das aplicações e as transformações M2C possibilitam a geração de grande parte do código. A geração de código e o reúso de serviços reduzem as tarefas do Engenheiro de Aplicação e torna mais ágil o processo de desenvolvimento das aplicações. Tem-se uma melhoria na produtividade proporcionada pelo reúso dos artefatos, geração de código e pelo fato de os serviços terem sido previamente implementados e testados. Assim, o Engenheiro da Aplicação pode se dedicar a outras tarefas do processo de

desenvolvimento, como, por exemplo, focar na implementação do comportamento das classes e serviços. O diagrama SADT da Figura 3.14 ilustra as atividades da EA: *Analisar*, *Projetar*, *Implementar* e *Testar*. Essas atividades correspondem às disciplinas já conhecidas nos processos de desenvolvimento de software, porém adaptadas para atender ao Reúso baseado na DSM e SOA.

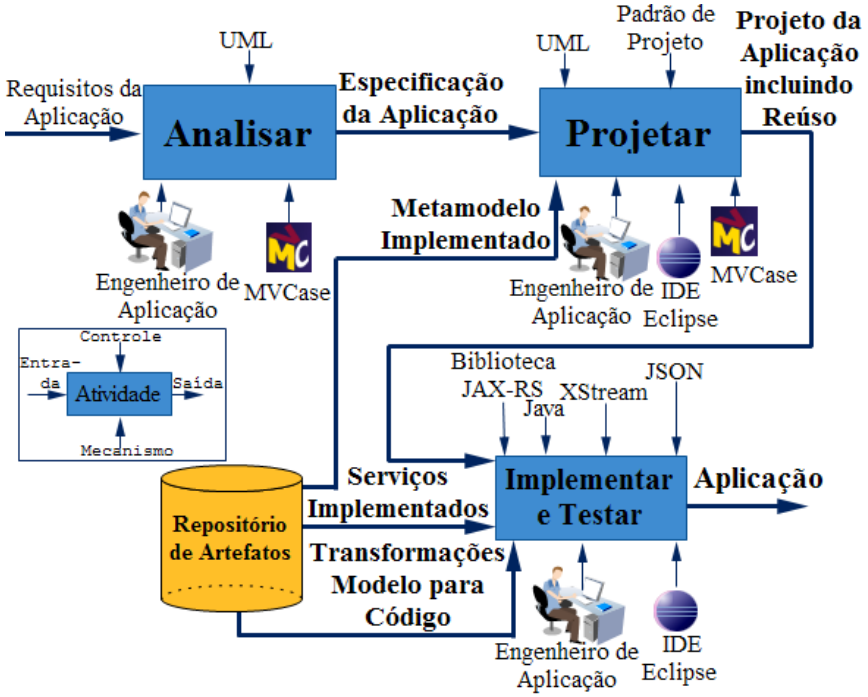

Figura 3.14. Engenharia de Aplicação.

Para ilustrar as atividades da EA serão apresentadas duas aplicações do domínio *Healthcare*, que utilizam serviços de atendimento de emergência, descritas a seguir:

- *Aplicação A* – Com foco em disponibilizar informações sobre o histórico do paciente no atendimento à vítima. Esse aplicativo permite que profissionais da saúde consultem informações sobre a vida do paciente mostrando os tratamentos, doenças e procedimentos efetuados na vítima. Além disso, essa aplicação possui o serviço de registro de observações do paciente para que o profissional da saúde, responsável pelo atendimento do paciente, possa analisá-lo e interferir em casos de necessidade.

- *Aplicação B* – Com foco em disponibilizar informações sobre doenças cardiovasculares e seus respectivos procedimentos. O aplicativo apresenta as informações sobre doenças cardiovasculares e os procedimentos médicos emergenciais. As informações usam recursos textuais e de multimídia, como imagens e vídeos. O foco principal desse aplicativo é o fortalecimento do

conhecimento, orientação e sistematização dos processos inclusos no cuidado aos pacientes de doenças cardiovasculares.

A seguir são detalhadas as atividades da EA, apresentando o desenvolvimento dessas aplicações como estudo de caso.

3.3.1 Analisar

Nesta atividade, inicialmente, são identificados, elicitados, analisados e especificados os requisitos da aplicação. Para a especificação, o Engenheiro de Aplicação utiliza as técnicas UML com o auxílio da ferramenta MVCASE. Dentre as técnicas utilizadas têm-se o diagrama de casos de uso e o diagrama de classes. A Figura 3.15 mostra um diagrama de casos de uso que especifica os requisitos identificados para as aplicações. Salienta-se que o caso de uso *ResearchDisease* é utilizado pelas duas aplicações.

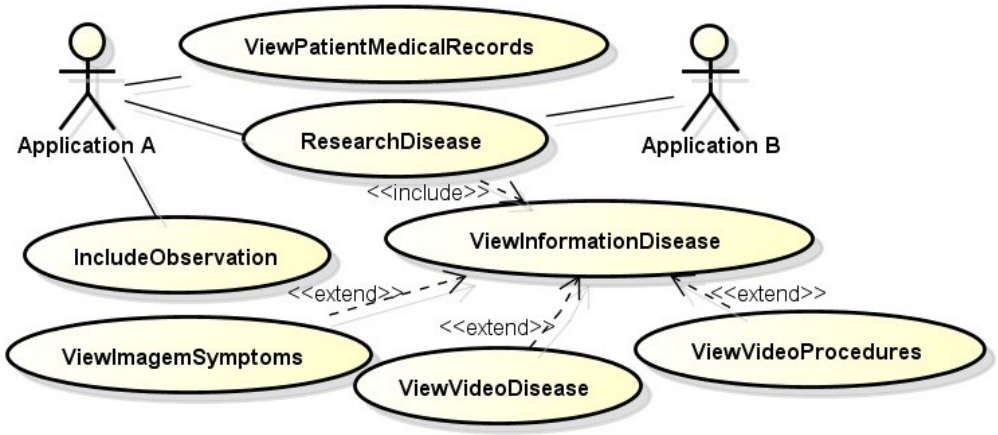

Figura 3.15. Casos de uso das aplicações

Na *Aplicação A* foram identificadas e especificadas as classes *Patient*, *Disease*, *Symptom*, *History*, *Treatment*, *Diagnostic*, *HealthEletronicRecord*, e *Observation*, e na *Aplicação B* as classes *Disease*, *Symptom*, *Treatment*, e *CallEmergency*. A Figura 3.16 mostra, por exemplo, o modelo de classes da *Aplicação A*. Uma vez especificados e analisados os requisitos, o processo continua conforme apresentado na próxima atividade da abordagem.

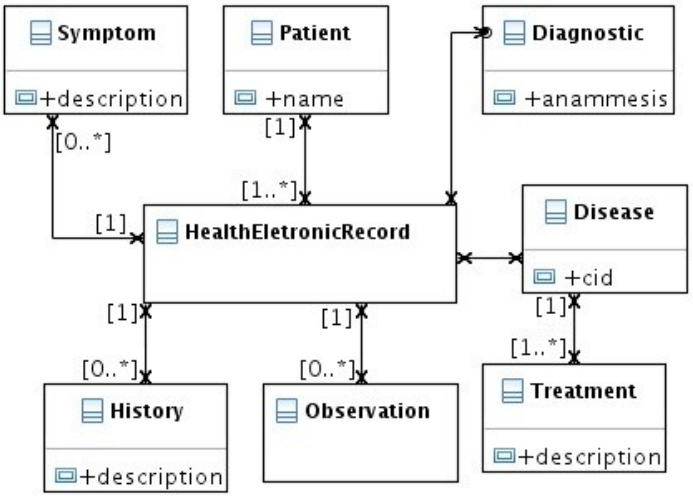

Figura 3.16. Modelo de Classes da Aplicação A

3.3.2 Projetar

Nesta atividade as especificações da aplicação são refinadas, considerando as tecnologias, plataformas de hardware e software, e outras decisões de projeto. Por exemplo, para as aplicações apresentadas, adotou-se a *Application Programming Interface (API)* Android[8] 2.2 como plataforma *Mobile*.

O Engenheiro de Aplicação refina os modelos da *Análise*, reutilizando o metamodelo pela instanciação de classes de entidades e serviços, conforme os requisitos da aplicação. A Figura 3.17 mostra o reúso do metamodelo nessa etapa de projeto. Na direita da Figura 3.17 tem-se o editor de modelos do metamodelo disponível como um *plugin* no IDE Eclipse, e na esquerda tem-se o projeto das aplicações.

Com base nos modelos de classes da *Análise* foram identificadas e reutilizadas do metamodelo as classes *Patient*, *Disease*, *Symptoms*, *History*, *Treatment*, e *Diagnostic* na *Aplicação A*, e as classes *Disease*, *Symptom*, e *Treatment* na *Aplicação B*. O Engenheiro de Aplicação realiza essa tarefa usando os recursos de *drag-and-drop* do editor de modelos. As classes *Disease*, *Symptom* e *Treatment* evidenciam o reúso pela instanciação das classes desenvolvidas na ED. Outro ponto a ressaltar é que o Engenheiro da Aplicação pode, conforme os requisitos específicos de cada aplicação, adicionar ou modificar os atributos das classes instanciadas do metamodelo. Outras classes mais específicas da aplicação

[8] http://developer.android.com/sdk/index.html.

também podem ser adicionadas como a *HealthEletronicRecord* e *Observation* na *Aplicação A* e *CallEmergency* na *Aplicação B*.

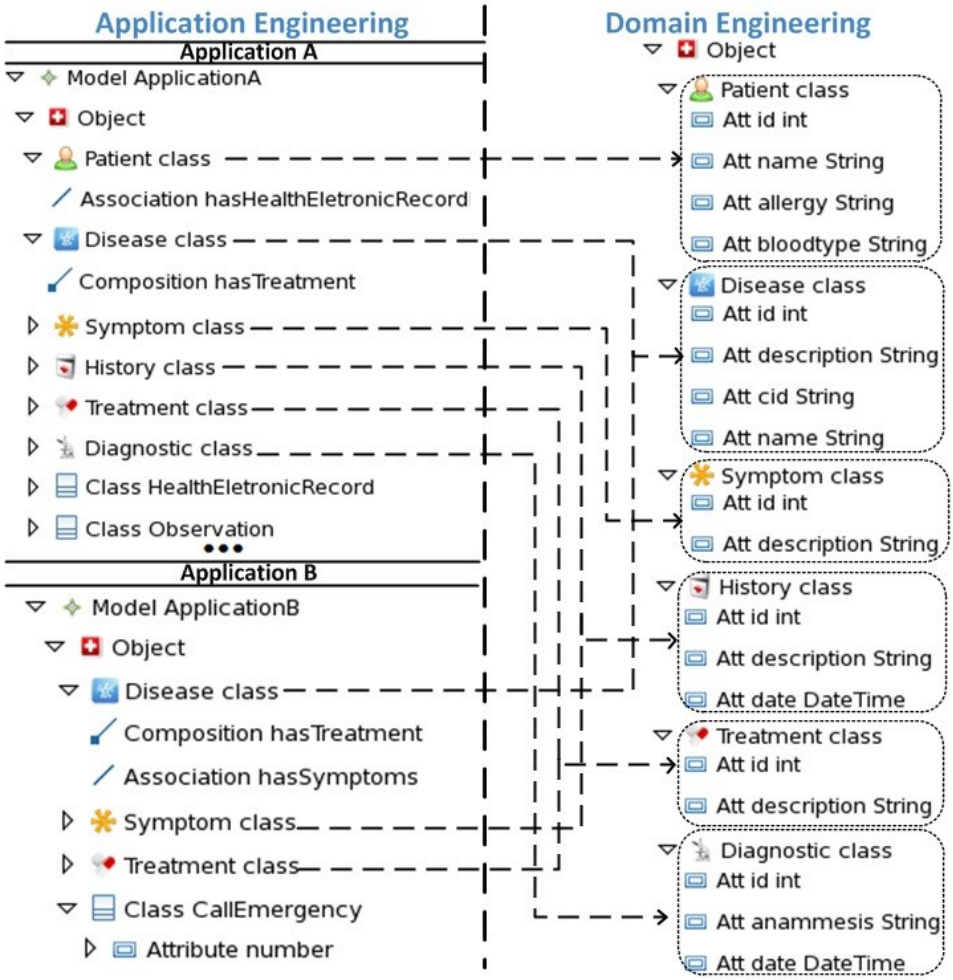

Figura 3.17. Reúso na Modelagem das Aplicações

Outra tarefa desta atividade da abordagem é o projeto dos serviços. A Figura 3.18 mostra o reúso de serviços providos pelo metamodelo. Na direita da Figura 3.18 são mostrados os serviços e respectivos métodos a serem reutilizados e na esquerda têm-se as aplicações, destacando o reúso dos serviços. Conforme mostra a Figura 3.18, na *Aplicação A*, a classe *PatientResource* possui o método *insertObservation* que reutiliza o método *includeMedicalRecordentry* do serviço *MedicalRecords*. A classe *TreatmentDisease* têm os métodos *findDisease*, *findTreatment*, *findVideoTreatment*, que reutilizam os métodos *reseachDisease*, *researchTreatment*, *findVideoTreatment*, respectivamente, do serviço *LookupDisease*. Na *Aplicação B*, a classe *DiseaseResource* possui os métodos *researchInfoDisease*, *ImageDisease*, e *VideoDisease*, que reutilizam os métodos *researchDisease*, *findImageDisease*, e *findVideoDisease* do serviço *LookupDisease*.

Note que esses serviços disponibilizam outros métodos que não foram reutilizados pelas aplicações.

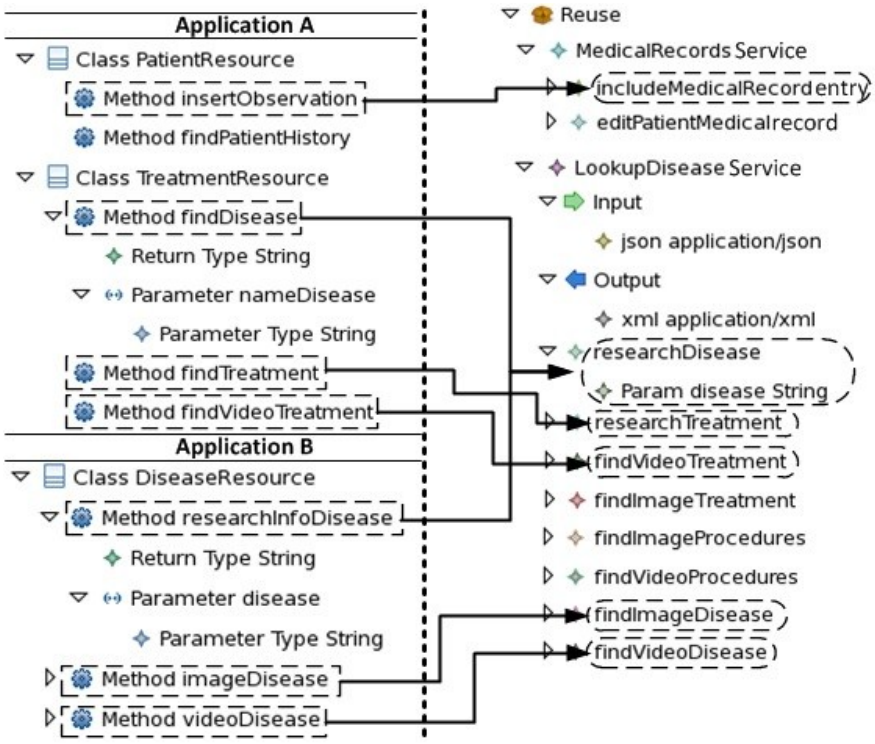

Figura 3.18. Reúso no projeto dos Serviços

A reutilização dos serviços reduz a necessidade de implementação dos métodos, e proporciona agilidade no desenvolvimento do projeto das aplicações.

3.3.3 Implementar e Testar

Nesta etapa da abordagem a aplicação é implementada e testada. Parte da implementação é obtida pela geração de código e pela reutilização de serviços construídos na ED. Outra parte que completa a implementação das aplicações móveis (i. e. interface) é realizada manualmente pelo Engenheiro da Aplicação.

No IDE Eclipse, o Engenheiro de Aplicação auxiliado pelo *framework JET*, executa as transformações M2C para a geração parcial do código, baseado no projeto da aplicação. A Figura 3.19 mostra um trecho de código gerado da classe *DiseaseResource*, da *Aplicação B*. Além da estrutura da classe com seus atributos e relacionamentos, as transformações geram também o código para a reutilização dos serviços. Por exemplo, os métodos *researchInfoDisease*, *imageDisease*, *videoDisease* da classe *DiseaseResource* chamam os métodos *reseachDisease*, *findImageDisease*, *findVideoDisease*, respectivamente do serviço *LookupDisease*.

Ainda na IDE Eclipse o Engenheiro de Aplicação completa a implementação conforme as necessidades específicas da aplicação.

```
public class DiseaseResource {
 public String researchInfoDisease(String nameDisease) {
  DefaultHttpClient httpclient = new DefaultHttpClient();
  HttpGet httpGet = new HttpGet(URL + "disease/" + nameDisease);
  httpGet.setHeader("Accept", "application/json");
  httpGet.setHeader("Content-type", "application/json");
  ResponseHandler responseHandler = new BasicResponseHandler();
  try {
      return httpclient.execute(httpGet, responseHandler);
  •••
 }
 public String imageDisease(String nameDisease) {
  DefaultHttpClient httpclient = new DefaultHttpClient();
  HttpGet httpGet = new HttpGet(URL + "imageDisease/" + nameDisease);
  httpGet.setHeader("Accept", "application/json");
  httpGet.setHeader("Content-type", "application/json");
  ResponseHandler responseHandler = new BasicResponseHandler();
  try {
      return httpclient.execute(httpGet, responseHandler);
  •••
 }
 public String videoDisease(String nameDisease) {
  DefaultHttpClient httpclient = new DefaultHttpClient();
  •••
 }
```

Figura 3.19. Trecho de Código da Classe *DiseaseResource* da Aplicação B

Ainda nessa etapa da abordagem, são realizados os testes com a aplicação, os quais fornecem os *feedbacks* que indicam o retorno ou não às atividades anteriores da EA. Na abordagem são realizados testes caixa preta (PRESSMAN, 2004) para avaliar o comportamento externo do software, porém a abordagem não é restrita à utilização de somente esse tipo de teste, podendo ser utilizados outros tipos de teste que auxiliem na verificação, validação e execução da aplicação móvel.

Os testes das aplicações A e B foram executados em um dispositivo móvel *HTC A8181* com API *Android 2.2*. A Figura 3.20 mostra duas interfaces da execução da *Aplicação A* para cadastro de pacientes, e duas interfaces da *Aplicação B* que consulta informações no servidor, sobre a doença *stroke*, retornando sua descrição, tratamento, imagem e vídeo.

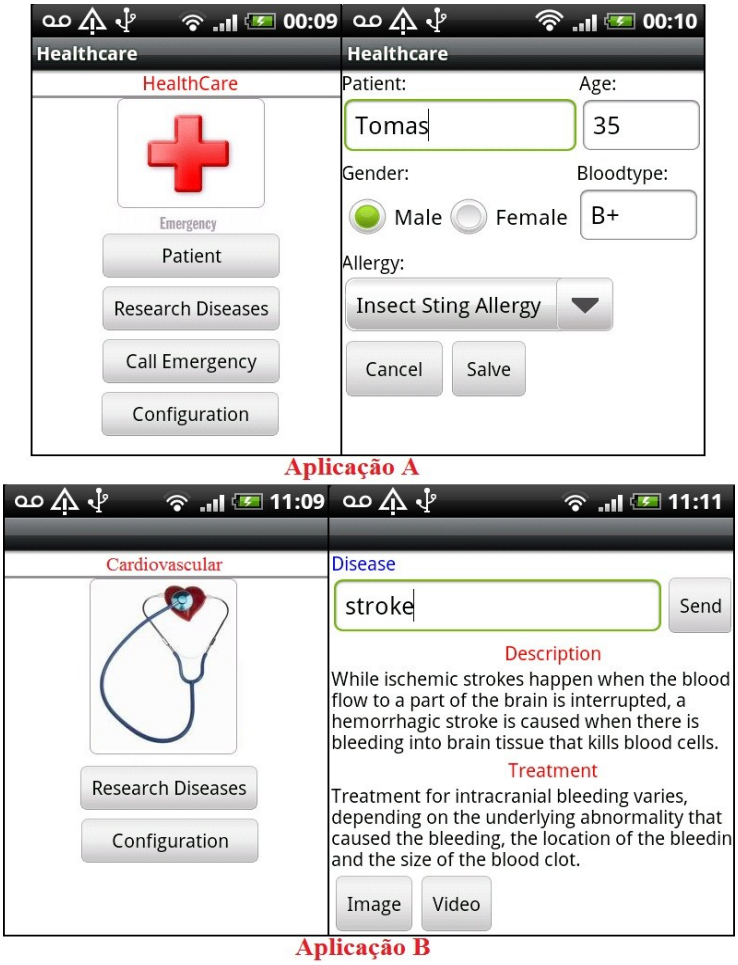

Figura 3.20. Interface da Execução das aplicações no dispositivo móvel

3.4 Considerações Finais

Este capítulo apresentou as duas principais etapas da abordagem proposta para desenvolvimento de aplicações móveis via reúso. A ED engloba as atividades para a construção de artefatos reutilizáveis que apoiam o desenvolvimento na etapa da EA. Esses artefatos incluem um metamodelo, que expressa a sintaxe abstrata de uma linguagem específica de domínio de um dado domínio do problema; serviços Web – denominados Serviços do Domínio – que atendem aos requisitos comuns das diferentes aplicações do domínio do problema para o qual o metamodelo foi construído bem como às transformações M2C.

As atividades da EA estendem as disciplinas de Análise, Projeto, Implementação e Testes do ciclo de vida de desenvolvimento dos processos de

engenharia software convencionais, porém adaptadas para atender o Reúso dos artefatos produzidos na ED.

Capítulo 4

VALIDAÇÃO DA PROPOSTA

Este capítulo descreve o experimento realizado na Universidade Federal de São Carlos para avaliação da abordagem apresentada. Os objetivos relacionados à execução de experimentos em Engenharia de Software são: a avaliação de processos, abordagens de desenvolvimentos, modelos, teorias entre outros (TRAVASSOS et al., 2002).

Na próxima subseção apresenta-se o experimento conduzido que avaliou o impacto na eficiência de equipes que desenvolveram a aplicação móvel via reúso dos serviços Web e o serviço Web utilizando a abordagem proposta, dirigida a modelos, em comparação a um processo baseado no ciclo de vida tradicional do software (PRESSMAN, 2004).

4.1 Experimentação da Abordagem Proposta

A experimentação da abordagem para o Desenvolvimento de Aplicações Móveis com Reúso de Software baseado em DSM e SOA seguiu as atividades experimentais, conforme definido em Travassos et al., 2002, e foi conduzida no primeiro semestre de 2011. O experimento baseia-se em um estudo comparativo entre o uso da abordagem proposta para a construção de Aplicações Móveis utilizando serviços Web e o uso do processo tradicional de desenvolvimento de software, sem a utilização de MDD, para o mesmo fim. O objetivo do experimento foi comparar os efeitos de ambas as abordagens de desenvolvimento em relação à

eficiência das equipes, em termos de tempo despendido e a produtividade, ou seja, a quantidade total de linhas de código produzidas. A seguir são detalhadas as fases do desenvolvimento da experimentação.

4.1.1 Desenvolvimento da Experimentação

Foram utilizadas três aplicações móveis para o domínio *Healthcare*, elaboradas especialmente para a execução do experimento. As aplicações móveis, denominadas **Aplicação A**, **Aplicação B**, e **Aplicação C**, foram desenvolvidas com o mesmo nível de complexidade entre si. Foi também elaborada uma aplicação Web, denominada **wsNurse** com o objetivo de testar a eficiência e a produtividade do uso da abordagem pelos participantes para o desenvolvimento de um serviço Web. Todas as aplicações foram projetadas e implementadas para avaliar a construção e reutilização dos Serviços Web do domínio *Healthcare*.

A *Aplicação A*, Apêndice E, possibilita a visualização de detalhes das ocorrências através de dispositivos móveis aos funcionários de uma determinada estação de operação, de maneira que estes possam se preparar adequadamente durante o percurso a um atendimento de emergência. Esses funcionários, munidos de um dispositivo móvel, devem visualizar a lista com o título das ocorrências. Essa aplicação ainda disponibiliza funcionalidades para listar a(s) vítima(s) em situação de emergência, de forma que possam conhecer com antecedência o perfil da vítima. Com isso, os funcionários atendentes podem visualizar e buscar, por meio de um dispositivo móvel, as informações sobre o atendimento e da(s) vítimas(s) de qualquer lugar e a qualquer momento, auxiliando na prestação dos serviços de atendimento de emergência.

A *Aplicação B,* Apêndice H, possibilita a visualização de detalhes dos pacientes através de dispositivos móveis aos médicos, de maneira que estes possam se preparar adequadamente durante o percurso a um atendimento de emergência. Esses médicos, munidos de um dispositivo móvel, devem visualizar a lista com o nome dos pacientes. Além de conhecerem os detalhes de um paciente vitimado, algumas vezes esses médicos necessitam de informações a respeito do histórico do paciente que está registrado em sua clínica médica, de forma a efetuar uma análise e acompanhamento do paciente durante a evolução de suas doenças. Assim, esses médicos, munidos de um dispositivo móvel, devem também ser

capazes de acessar uma lista dos históricos de atendimentos do paciente. Com isso, os médicos poderão visualizar e buscar, através de um dispositivo móvel, de qualquer lugar e a qualquer momento, informações sobre os pacientes, de forma a auxiliar na prestação dos serviços de atendimento de emergência.

A *Aplicação C*, Apêndice K, exibe uma lista de doenças e seus detalhes, de maneira que os funcionários de uma estação de operação possam se preparar adequadamente durante o percurso para socorrer o paciente vitimado. Além de conhecerem os detalhes das doenças, esses funcionários também necessitam de informações mais específicas a respeito dos sintomas associados às doenças, de forma que as medicações e procedimentos a serem prestados durante o socorro da vítima não interfiram na evolução de seu tratamento. Assim, outra funcionalidade necessária refere-se à apresentação de uma lista de sintomas associados à doença visualizada. Com isso, os funcionários poderão visualizar, através de um dispositivo móvel, de qualquer lugar e a qualquer momento, informações sobre doenças e seus sintomas, de forma a auxiliar na prestação dos serviços de atendimento de emergência.

A Aplicação *wsNurse* possui um serviço web que exibe uma lista de enfermeiras do hospital em que ocorre o atendimento de emergência.

No experimento, a avaliação da abordagem foi orientada para a etapa da EA. Para viabilizar o experimento, as interfaces gráficas (Figura 4.1) e as regras de negócio específicas das aplicações foram também previamente implementadas. Assim, os grupos receberam os projetos parcialmente implementados para serem importados ao seu ambiente de desenvolvimento. Dessa forma, a tarefa dos participantes foi a modelagem das classes de entidades das aplicações, e a construção e reúso de serviços Web para cumprir com os requisitos específicos de cada aplicação. Adotou-se para os cenários que os ambientes seriam desenvolvidos com base na tecnologia Restful executando sobre o servidor de aplicação *Apache TomCat*[9]. As aplicações móveis para consumir os serviços foram construídas com o Android SDK (GOOGLE, 2011) para serem executadas a partir de um celular. O IDE Eclipse foi adotado como ambiente de desenvolvimento para os projetos do experimento.

A atribuição dos participantes nos grupos foi realizada de maneira *desbalanceada* para refletir equipes com número variado de membros. Os

[9] http://tomcat.apache.org/.

participantes dos grupos selecionados para aplicar a abordagem proposta seguiram as atividades da etapa de EA. Para esses grupos foram disponibilizados todos os artefatos de suporte da abordagem produzidos previamente na etapa de ED, que operacionalmente foram: um *plugin* editor de modelos para a modelagem das classes de entidade e dos serviços Web com base no metamodelo; um conjunto de classe de entidades e de serviços Web do domínio *Healthcare*; e as transformações M2C para geração de código. Os grupos que aplicaram o desenvolvimento utilizando o ciclo de vida tradicional realizaram as disciplinas tradicionais de Projeto, Implementação e Testes da Engenharia de Software. Esses grupos não se beneficiaram do reúso dos artefatos da ED e tiveram a tarefa adicional de desenvolver os modelos das entidades, sem apoio do metamodelo, além de construir os serviços a serem reutilizados pelas aplicações.

Figura 4.1. Interface das Aplicações desenvolvidas no Experimento

Durante a execução do experimento, o experimentador solicitou que cada grupo registrasse os dados coletados em um formulário (a hora de início e fim da realização de cada atividade executada), bem como as quantidades de linhas de código geradas automaticamente e codificadas manualmente. Na contabilização das linhas de código, foi considerado apenas o código referente às classes de entidades, dos serviços Web e o código das classes no Android para a chamada dos serviços Web. No caso dos grupos do ciclo de vida tradicional, os quais não utilizaram transformações para geração de código com base em modelos, considerou-se como código gerado aquele criado automaticamente pelo ambiente de desenvolvimento (e.g. *templates* de classes).

Tanto para os grupos da abordagem proposta quanto para os grupos do ciclo de vida Tradicional, a atividade de Projeto compreendeu a modelagem das classes de entidades e dos serviços Web (obrigatória no caso da abordagem), bem como a tomada de decisões de tecnologias, padrões e *frameworks* adicionais que os grupos julgassem necessários para a construção das aplicações.

A Implementação envolveu a codificação das classes de entidades, dos serviços Web e das classes no Android para a chamada dos serviços Web. Finalmente, a atividade de Testes englobou a verificação da execução dos serviços Web e das aplicações usando o emulador do dispositivo móvel disponível no Android SDK. Cada grupo recebeu um material de apoio apropriado com diretrizes que os orientou na execução do experimento. As fases experimentais realizadas no estudo são descritas nas subseções seguintes.

4.1.2 Planejamento do Experimento

A abordagem proposta foi realizada conforme apresentado a seguir (CIRILO, 2011):

1) *Seleção do contexto.* O experimento foi realizado em um ambiente universitário, sendo aplicado com estudantes de graduação no Laboratório de Ensino do Departamento de Computação da *Universidade Federal de São Carlos (UFSCar)*, no âmbito da disciplina *Desenvolvimento de Software para Web*.

2) *Formulação das Hipóteses.* Foram elaboradas três hipóteses para o experimento em relação ao efeito da abordagem de desenvolvimento no resultado. Para a formulação das hipóteses, foram consideradas as seguintes métricas:

- τ_w – Tempo total gasto pela equipe para o desenvolvimento do serviço Web;
- $Þ_w$ – Produtividade da equipe em termos de linhas de código produzidas (*Lines of Code - LOC*) na implementação do serviço Web por unidade de tempo ($Þ_s = LOC / \tau_s$);
- $\mu_{\tau w}$ – Tempo médio despendido pelas equipes para o desenvolvimento do serviço Web;
- $\mu_{Þw}$ – Produtividade média das equipes no desenvolvimento do serviço Web;
- τ_a – Tempo total gasto pela equipe para o desenvolvimento da aplicação móvel com reúso dos serviços;
- $Þ_a$ – Produtividade da equipe em termos de LOC na implementação da

Aplicação Móvel com reúso dos serviços Web por unidade de tempo ($Þ_a = LOC / \tau_a$);

- $\mu_{\tau a}$ – Tempo médio despendido pelas equipes para o desenvolvimento da Aplicação Móvel com reúso dos serviços Web; e
- $\mu_{Þa}$ – Produtividade média das equipes no desenvolvimento da Aplicação Móvel com reúso dos serviços Web.

A hipótese nula e suas correspondentes alternativas referentes à construção das Aplicações Móveis com reúso dos serviços Web são:

- **Hipótese nula (H_{0w}):** em geral, não há diferença entre equipes utilizando a abordagem proposta e equipes utilizando a abordagem baseada no ciclo de vida tradicional para a construção das Aplicações Móveis com reúso dos serviços Web, com relação à eficiência (ε) da equipe.

$$H_{0w}: \varepsilon_{ReúsoSoftware} = \varepsilon_{Tradicional} \Rightarrow \mu_{\tau wReúsoSoftware} = \mu_{\tau wTradicional} \text{ e } \mu_{ÞwReúsoSoftware} = \mu_{ÞwTradicional}$$

- **Hipótese alternativa (H_{1w}):** equipes utilizando a abordagem proposta para a construção das Aplicações Móveis com reúso dos serviços Web são, em geral, mais eficientes do que equipes utilizando o ciclo de vida Tradicional de desenvolvimento.

$$H_{1w}: \varepsilon_{ReúsoSoftware} > \varepsilon_{Tradicional} \Rightarrow \mu_{\tau wReúsoSoftware} < \mu_{\tau wTradicional} \text{ e } \mu_{ÞwReúsoSoftware} > \mu_{ÞwTradicional}$$

Hipótese alternativa (H_{2w}): equipes utilizando o ciclo de vida Tradicional de desenvolvimento para a construção das Aplicações Móveis com reúso dos serviços Web são, em geral, mais eficientes do que equipes utilizando a abordagem proposta.

$$H_{2w}: \varepsilon_{ReúsoSoftware} < \varepsilon_{Tradicional} \Rightarrow \mu_{\tau wReúsoSoftware} > \mu_{\tau wTradicional} \text{ e } \mu_{ÞwReúsoSoftware} < \mu_{ÞwTradicional}$$

A hipótese nula e suas correspondentes alternativas referentes à construção do serviço Web são:

- **Hipótese nula (H_{0a}):** em geral, não há diferença entre equipes utilizando a abordagem proposta e equipes utilizando a abordagem baseada no ciclo de vida Tradicional para a construção do serviço Web, com respeito à eficiência (ε) da equipe.

$$H_{0a}: \varepsilon_{ReúsoSoftware} = \varepsilon_{Tradicional} \Rightarrow \mu_{\tau aReúsoSoftware} = \mu_{\tau aTradicional} \text{ e } \mu_{ÞaReúsoSoftware} = \mu_{ÞaTradicional}$$

- **Hipótese alternativa (H_{1a}):** equipes utilizando a abordagem proposta para a construção do serviço Web são, em geral, mais eficientes (ε) do que equipes

utilizando o ciclo de vida Tradicional de desenvolvimento.

$$H_{1a}: \varepsilon_{ReúsoSoftware} > \varepsilon_{Tradicional} \Rightarrow \mu_{\tau aReúsoSoftware} < \mu_{\tau aTradicional} \text{ e } \mu_{ÞaReúsoSoftware} > \mu_{ÞaTradicional}$$

- **Hipótese alternativa (H_{2a}):** equipes utilizando o ciclo de vida tradicional de desenvolvimento para a construção do serviço Web são, em geral, mais eficientes do que equipes utilizando a abordagem proposta.

$$H_{2a}: \varepsilon_{ReúsoSoftware} < \varepsilon_{Tradicional} \Rightarrow \mu_{\tau aReúsoSoftware} > \mu_{\tau aTradicional} \text{ e } \mu_{ÞaReúsoSoftware} < \mu_{ÞaTradicional}$$

3) *Seleção das variáveis.* A variável dependente selecionada para o experimento foi a *eficiência da equipe*. As variáveis independentes foram a *abordagem proposta* e as *tecnologias de desenvolvimento*. Uma vez que o objetivo era investigar as consequências do uso da abordagem na eficiência da equipe, a *abordagem de desenvolvimento proposta* foi estabelecida como o fator que receberia tratamentos distintos e cujo efeito deveria ser observado na variável dependente. As demais variáveis independentes foram mantidas constantes durante a execução do experimento, conforme segue:

- Aplicações: *Aplicação A*, *Aplicação B*, *Aplicação C* e *wsNurse*;
- Ambientes de desenvolvimento: IDE Eclipse;
- Tecnologias de desenvolvimento: Java, Restful, Android SDK, *JavaScript Object Notation (JSON)* e *XStream*.

4) *Seleção dos participantes.* A seleção dos participantes foi feita via *amostragem não probabilística por conveniência* (WOHLIN et al., 2000) selecionando os indivíduos disponíveis mais próximos para participarem do experimento. Participaram do estudo 33 alunos de graduação dos cursos de Bacharelado em Ciência da Computação e Engenharia da Computação da UFSCar, matriculados na disciplina *Desenvolvimento de Software para Web*.

5) *Projeto do Experimento.* O projeto do experimento descreve como os testes experimentais são estruturados e executados. O experimento foi planejado *em blocos* (WOHLIN et al., 2000) a fim de assegurar que o efeito do fator *nível de experiência dos participantes* não interferisse nos resultados dos tratamentos do fator *abordagem de desenvolvimento proposta*. Dessa forma, foram divididos em 11 grupos (G_i), de modo que cada grupo possuísse, tanto quanto possível, médias semelhantes de nível de experiência. A experiência dos participantes foi avaliada pelo *Formulário de caracterização dos participantes* (Apêndice A) – documento utilizado para capturar a experiência profissional e acadêmica nos assuntos

relacionados ao estudo (e.g. Java, biblioteca JAX-RS, biblioteca *XStream*, UML, JSON, Restful, Android SDK). Esse formulário foi entregue a cada participante semanas antes da execução do experimento, de forma que fosse possível planejar o estudo antecipadamente. O gráfico da Figura 4.2 mostra os níveis de experiência individuais de cada aluno (participante), bem como o nível médio de experiência dos grupos (rótulos sobrepostos às barras adjacentes referentes aos participantes de um mesmo grupo), conforme as informações apresentadas pelos alunos em seus respectivos formulários de caracterização. Esses níveis foram obtidos quantificando as ponderações entre o grau de conhecimento e os tempos de experiência de cada assunto reportado pelos participantes (P_i) nos formulários.

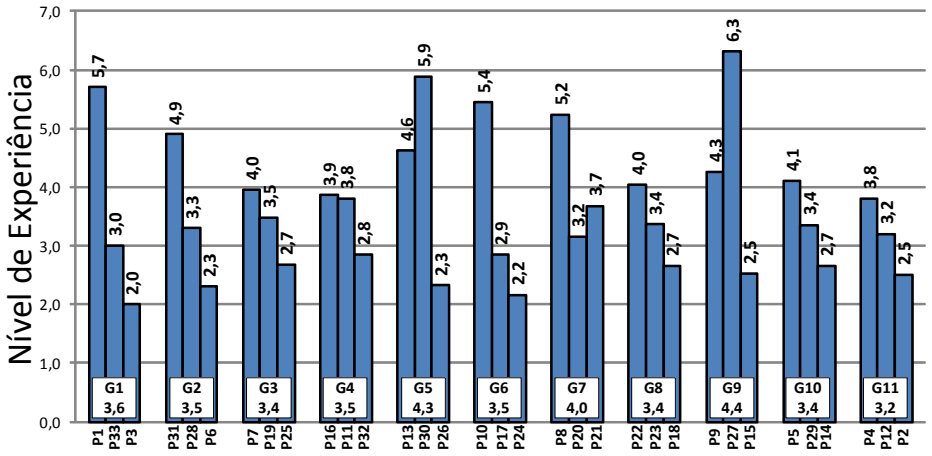

Figura 4.2. Níveis de experiência individuais dos participantes e experiência média dos grupos

6) *Tipo do Projeto.* O tipo do projeto do experimento foi de um fator (abordagem de desenvolvimento) *com dois tratamentos* (abordagem proposta e o Ciclo de Vida Tradicional) *completamente randomizados* (WOHLIN et al., 2000). Nesse tipo de projeto um mesmo objeto a ser manipulado durante o estudo é utilizado em todos os tratamentos e os participantes ou grupos são aleatoriamente atribuídos a cada tratamento. No caso do experimento da abordagem proposta, as aplicações *Aplicação A, Aplicação B, Aplicação C e wsNurse* foram desenvolvidas utilizando ambos os processos de desenvolvimento, sendo seis grupos atribuídos para utilizar o Ciclo de Vida Tradicional e cinco grupos atribuídos para utilizar a Abordagem Proposta distribuindo as aplicações aleatoriamente para cada grupo. Tabela I**Erro! Fonte de referência não encontrada.** apresenta o resultado dessa distribuição.

7) *Instrumentação.* Os materiais necessários para auxiliar os participantes no experimento foram previamente planejados, compreendendo a definição dos *objetos* que seriam manipulados, as *diretrizes* para orientar os grupos na execução dos processos, bem como os *instrumentos para a coleta de dados e medição*.

Tabela I. Distribuição Aleatória dos Grupos aos Tratamentos

Grupos	Tratamento 1: Ciclo de vida Tradicional	Tratamento 2: Reúso de Software baseado em DSM e SOA	Aplicação
G1	X		A
G2	X		B
G3	X		C
G4		X	C
G5		X	C
G6		X	B
G7	X		B
G8	X		C
G9		X	A
G10	X		B
G11		X	C

4.1.3 Execução do Experimento

A execução do experimento foi efetuada em três atividades:

1) *Preparação.* Nesta atividade os materiais definidos na instrumentação do experimento foram efetivamente elaborados, a saber:

a) *Objetos***.** O módulo serviço Web denominado *wsNurse* foi projetado e implementado, com exceção das Classes Entidades e dos serviços Web. O projeto parcial contendo as funcionalidades internas da aplicação foi empacotado para ser distribuído aos grupos juntamente com o *script Structured Query Language (SQL)* de criação do banco de dados utilizado. As três aplicações móveis foram projetadas e implementadas, com exceção das classes entidades e das classes responsáveis por efetuar a chamada aos serviços Web. Aos grupos de Reúso de Software baseado em DSM e SOA foi entregue o projeto do módulo serviço Web bem como um projeto do aplicativo móvel, escolhido aleatoriamente, além, dos artefatos de apoio à abordagem, que inclui o editor de modelos para a modelagem das classes de entidade e dos serviços Web, um conjunto de classe de entidades e de serviços Web do domínio *Healthcare*, e as transformações M2C para geração de código. Para os grupos do Ciclo de Vida Tradicional foram entregues somente o projeto do módulo de serviço Web e um projeto de aplicação móvel escolhido aleatoriamente.

b) *Diretrizes.* Os seguintes documentos foram produzidos para serem utilizados no estudo: 1) *Formulário de caracterização dos participantes* (Apêndice A), para obtenção da experiência acadêmica sobre os assuntos relacionados diretamente ao experimento; 2) *Formulário de consentimento* (Apêndice B), para aprovação e ciência dos participantes dos objetivos do estudo e das condições de participação; 3) *Descrição da tarefa* (Apêndice C e Apêndice D), com as instruções da sua execução; e 4) *Descrição das aplicações* (Apêndice E, Apêndice H, Apêndice K) *e material de apoio* (Apêndice F, Apêndice G, Apêndice I, Apêndice J, Apêndice L e Apêndice M), contendo a descrição de cada uma das aplicações móveis que reutiliza serviços Web juntamente com o módulo Web para o desenvolvimento do serviço, bem como foram preparados os tutoriais para cada uma das aplicações em versões especializadas em cada um dos processos de desenvolvimento.

c) *Instrumentos para Coleta de Dados.* Medições em experimento são efetuadas com os dados que são coletados. Em experimentos envolvendo atividades executadas por seres humanos, os dados geralmente são coletados manualmente por meio de formulários ou entrevistas. Assim, para a coleta de dados do experimento, foi elaborado o *Formulário de coleta de dados* (Apêndice N e Apêndice O), no qual os grupos deveriam preencher todas as informações requeridas durante a execução do experimento. Além disso, foram elaborados também dois tipos de *Formulário de avaliação qualitativa* (Apêndice P e Apêndice Q) para cada um dos processos de desenvolvimento, nos quais cada participante individualmente, após a execução do experimento, deveria reportar sua percepção sobre a facilidade de utilização da abordagem proposta e por meio do Ciclo de Vida Tradicional.

Para evitar que o tempo de aprendizagem de como utilizar a abordagem e o ciclo de vida Tradicional interferisse no tempo de desenvolvimento do experimento, foram realizados treinamentos como forma de aquecimento (*warm-up*) com todos os participantes do experimento ao longo de duas semanas que antecederam a aplicação do experimento. Conforme mostra a Tabela II, durante os treinamentos o módulo Web para o desenvolvimento de serviços Web e uma aplicação móvel para o reúso dos serviços Web foram utilizados como objetos para a construção de serviços Web e do reúso de serviços Web usando ambos os processos de desenvolvimento. Os treinamentos foram conduzidos de forma que, no fim da fase de treinos, todos os

participantes estivessem suficientemente aptos para executar quaisquer dos processos de desenvolvimento.

Tabela II. Fases do experimento

			Processo de desenvolvimento	
			Ciclo de vida tradicional	**Abordagem para o Desenvolvimento de Aplicações Móveis com Reúso de Software baseado DSM e SOA**
Aplicação	*1ª Fase (Treinamentos)*	*Módulo serviços Web e Aplicação Móvel*	Todos os participantes (P1 a P33) – 1ª e 2ª semanas	Todos os participantes (P1 a P33) – 2ª semana
	2ª Fase (Execução)	*Módulo serviços Web e Aplicação Móvel A*	G1 – 3ª semana	G9 – 3ª semana
		Módulo serviços Web e Aplicação Móvel B	G2, G7, G10 – 3ª semana	G6 – 3ª semana
		Módulo serviços Web e Aplicação Móvel C	G3, G8 – 3ª semana	G4, G5, G11 – 3ª semana

2) *Execução.* Os grupos executaram os testes experimentais conforme o projeto experimental. Foram coletados os dados pelos grupos na execução do experimento referente ao desenvolvimento do módulo Web e da aplicação móvel como são mostrados na Tabela III e Tabela IV. O significado dos acrônimos dos dados pode ser visualizado na Tabela V.

Os dados mostrados nas Tabela III e Tabela IV estão organizados em dois blocos referentes aos processos de desenvolvimento usados como tratamento. Na parte superior das tabelas são listados os dados referentes aos grupos que executaram a abordagem proposta, e na parte inferior têm-se os dados dos grupos que executaram o ciclo de vida tradicional. As colunas "LOC Total (ι_0)", "Tempo Total (τ_0)" e "Produtividade ($Þ_0$)" representam, respectivamente, a quantidade total de linhas de código produzidas pelo grupos para implementação do serviço Web, o tempo total despendido para a construção do serviço no módulo Web e a produtividade de cada grupo em termos de linhas de código produzidas por hora. Além disso, as colunas "LOC Total (ι_1)", "Tempo Total (τ_1)" e "Produtividade ($Þ_1$)" representam, respectivamente, a quantidade total de linhas de código produzidas

pelo grupos para o desenvolvimento do aplicativo móvel, o tempo total despendido para a construção do aplicativo móvel via reúso dos serviços e a produtividade de cada grupo em termos de linhas de código produzidas por hora. Essas informações foram calculadas apurando os dados coletados pelos grupos após a realização do experimento.

Tabela III. Dados coletados para o desenvolvimento do serviço Web

	Grupo	HIAProj	HTAProj	HIAImpl	HTAImpl	HIATeste	HTATeste	LCGAuto	LCGManual	LOC Total (ι_0)	Proble-mas	Tempo Total (τ_0)	Produti-vidade (P_0)
Reúso de Software	G4	08:49	09:16	09:17	09:30	09:32	10:29	73	26	99	00:55	01:40	59
	G5	09:00	09:23	09:33	09:33	09:40	10:10	73	29	102	00:30	01:10	87
	G6	08:52	09:16	09:16	09:16	10:16	10:17	55	13	68	00:41	01:25	48
	G9	09:00	09:25	09:35	09:35	09:35	10:19	71	27	98	00:30	01:19	74
	G11	09:40	10:10	10:10	11:00	11:00	11:05	70	30	100	00:40	01:25	71
	Média	**00:25**		**00:16**		**00:27**		**68**	**25**	**93**	**00:39**	**01:23**	**68**
Ciclo de vida Tradicional	G1	08:55	09:06	09:07	09:22	09:23	09:56	49	34	83	00:30	01:01	82
	G2	09:00	09:02	09:05	09:15	09:20	10:22	40	42	82	00:52	01:11	60
	G3	08:58	09:20	09:20	09:31	09:32	10:30	40	41	81	00:50	01:32	53
	G7	09:03	09:06	09:06	09:30	09:30	10:05	40	25	65	00:30	01:02	63
	G8	08:55	09:01	09:02	09:10	09:11	09:20	40	54	94	00:03	00:18	313
	G10	08:57	09:10	09:10	09:35	09:35	10:50	29	49	78	01:31	01:40	47
	Média	**00:08**		**00:15**		**00:45**		**40**	**41**	**81**	**00:42**	**00:42**	**103**

Tabela IV. Dados coletados para o desenvolvimento da aplicação móvel

	Grupo	HIAProj	HTAProj	HIAImpl	HTAImpl	HIATeste	HTATeste	LCGAuto	LCGManual	LOC Total (ι_1)	Proble-mas	Tempo Total (τ_1)	Produti-vidade (P_1)
Reúso de Software	G4	10:40	10:45	10:45	10:57	10:57	11:19	48	0	48	00:00	00:39	74
	G5	10:12	10:22	10:22	10:23	10:24	10:39	153	1	154	00:00	00:27	342
	G6	10:25	10:38	10:38	10:44	10:45	11:00	164	1	165	00:00	00:35	283
	G9	10:20	10:34	10:39	10:40	10:40	10:42	173	1	174	00:00	00:22	475
	G11	11:05	11:15	11:15	11:20	11:20	11:22	40	0	40	00:00	00:17	141
	Média	**00:10**		**00:05**		**00:11**		**116**	**1**	**116**	**00:00**	**00:28**	**263**
Ciclo de vida Tradicional	G1	10:00	10:04	10:05	10:24	10:25	11:32	82	46	128	00:00	01:32	83
	G2	10:24	10:26	10:28	10:38	10:40	11:10	65	48	113	00:10	00:46	147
	G3	10:30	10:47	10:48	11:05	11:05	11:18	60	40	100	00:10	00:48	125
	G7	10:05	10:08	10:08	10:30	10:30	11:00	70	40	110	00:23	00:55	120
	G8	09:21	09:26	09:27	09:45	09:45	10:59	5	44	49	01:10	01:32	32
	G10	09:40	09:50	09:51	10:15	10:15	10:30	64	60	124	00:00	00:39	191
	Média	**00:05**		**00:18**		**00:37**		**58**	**46**	**104**	**00:17**	**01:01**	**118**

Tabela V. Legenda dos dados coletados

Descrição do Dado	Acrônimo
Hora de Início da Atividade de Projeto	HIAProj
Hora de Término da Atividade de Projeto	HTAProj
Hora de Início da Atividade de Implementação	HIAImpl
Hora de Término da Atividade de Implementação	HTAImpl
Hora de Início da Atividade de Testes	HIATeste
Hora de Término da Atividade de Testes	HTATeste
Linhas de Código Geradas Automaticamente	LCGAuto
Linhas de Código Geradas Manualmente	LCGManual

As linhas rotuladas como “Média” mostram as médias de tempo gasto pelos grupos da abordagem proposta e do ciclo de vida tradicional, para o desenvolvimento do serviço Web e para o aplicativo móvel, na execução de cada uma das atividades dos processos de desenvolvimento, bem como o número médio de linhas de código geradas tanto manualmente quanto automaticamente. Também

são mostradas as médias do total geral de linhas de código, do tempo total despendido ($\mu_{\tau 0}$ - para o desenvolvimento do serviço Web e $\mu_{\tau 1}$ - para o desenvolvimento da aplicação móvel) e da produtividade dos grupos (μ_{P0} - para o desenvolvimento do serviço Web e μ_{P1} – para o desenvolvimento da aplicação móvel).

Os grupos foram instruídos a relatar os problemas técnicos encontrados durante a execução do experimento. Para tanto, no formulário de coleta de dados foram colocados campos apropriados para serem preenchidos com a descrição dos dados. A Tabela VI detalha os dados preenchidos pelos grupos relacionados aos problemas encontrados, indicando o tempo em que cada problema foi solucionado pelos grupos. O tempo total gasto por cada grupo para solução dos problemas encontra-se sumarizado na coluna "Problemas" da Tabela VI**Erro! Fonte de referência não encontrada.**. Esse tempo não foi descontado dos tempos totais dos grupos uma vez que problemas de ordem técnica podem ocorrer em qualquer desenvolvimento.

Tabela VI. Problemas técnicos enfrentados pelos grupos

Grupo	Descrição do Problema	Identi-ficação	Reso-lução	Tempo Gasto
G1	O servidor de aplicação TomCat não esta iniciando em uma das máquinas.	09:25	09:55	00:30
G2	Problema de endereçamento. Estava chamando wsDoctorCare no browser ao invés de wsNurse.	09:30	10:00	00:30
	Erro para carregar a página inicial (*http Status 404*)	10:00	10:22	00:22
	Problema ao executar no Android, não estava carregando o link, pois ocorria página não encontrada.	11:00	11:10	0:10
G3	Problema de endereçamento do projeto wsNurse.	09:40	10:20	00:40
	Problema ao executar no Android, não estava carregando o link, pois ocorria página não encontrada.	11:07	11:17	00:10
G4	Erro para carregar a página inicial (*http Status 500*)	09:32	10:27	00:55
G5	Problema de endereçamento no servidor de aplicações TomCat.	09:40	10:10	00:30
G6	Projeto do serviço não executava no TomCat.	09:31	10:12	00:41
G7	Erro para carregar a página inicial (*http Status 500*)	09:30	10:00	00:30
	Importação da Classe Java.util.Data errada no serviço.	10:37	11:00	00:23
G8	Problemas com a nomeação com a Base de Dados	09:13	09:16	00:03
	Problema com a implementação da Classe (implements Serializable)	09:48	10:13	00:25
	Problema ao carregar o link do serviço no android.	10:14	10:59	00:45
G9	Problema de endereçamento. Estava chamando wsDoctorCare no browser ao invés de wsNurse.	09:45	10:01	00:16
	Erro para carregar a página inicial (*http Status 500*)	10:01	10:15	00:14
G10	Erro para carregar a página inicial (*http Status 500*)	09:18	10:49	01:31
G11	Problema com o contexto do projeto wsNurse.	10:15	10:55	00:40

3) *Validação dos Dados*. Após o término do experimento, foi verificado se os dados registrados pelos grupos eram razoáveis, bem como se foram coletados de forma correta, ou seja, os participantes seguiram todos os passos do experimento, preencheram todos os formulários de acordo com as instruções do experimentador e entregaram as aplicações móveis implementadas e testadas. Durante a apuração, observou-se uma diferença de início do projeto Web para construção do serviço reportada pelo grupo G11 em relação aos demais grupos. O grupo reportou que encontrou problemas com o computador utilizado no experimento, alegando que não havia os softwares necessários para completar o ambiente de desenvolvimento proposto no experimento. Conforme orientado ao grupo, foi solicitada a instalação dos softwares necessários e devido a isso foi constatada a demora em iniciar o projeto. Ao analisar a média de tempo de teste do projeto Web na construção do serviço, constatou-se uma demora plausível devido ao problema de contexto web da aplicação reportada pelos grupos. Conforme orientado aos grupos, foi necessária uma análise pelo experimentador para verificar o erro causado, mas, após solucionar o problema, foi passada a todos os grupos a solução e assim puderam continuar o experimento.

4.1.4 Análise e Interpretação dos Resultados

Com base nos dados das Tabela III e Tabela IV, alguns aspectos interessantes foram observados entre os grupos que utilizaram a abordagem para o Desenvolvimento de Aplicações Móveis com Reúso de Software baseado em DSM e SOA e os grupos que aplicaram o ciclo de vida tradicional. A Figura 4.3 apresenta a distribuição média dos esforços dos grupos entre as atividades executadas nos processos para o desenvolvimento do serviço Web. Nota-se que, apesar de os grupos da Abordagem proposta (Reúso) terem gasto maior tempo na atividade de projeto (em média 25 min) do que os grupos do ciclo de vida tradicional (Tradicional) (em média 8 min), na atividade de Teste houve uma redução no tempo (em média 27 min para Reúso e 45 min para Tradicional) devido à utilização de geradores de código, o que diminui a probabilidade de erros de codificação dos serviços web.

Analisou-se também que o esforço de modelagem realizado na atividade de Projeto não foi bem aproveitado pelos grupos da abordagem para Reúso. Conforme

ilustrado na Figura 4.3, o tempo médio total para os grupos de Reúso (em média 1h e 23 min) e dos grupos Tradicional (1h e 9 min) demonstrou que não houve um aproveitamento viável para o uso da abordagem. A partir da análise preliminar desses dados, um aspecto importante observado entre os grupos é o desenvolvimento de somente um serviço Web com um método, ou seja, devido à baixa complexidade do serviço Web não foi válido aplicar a abordagem.

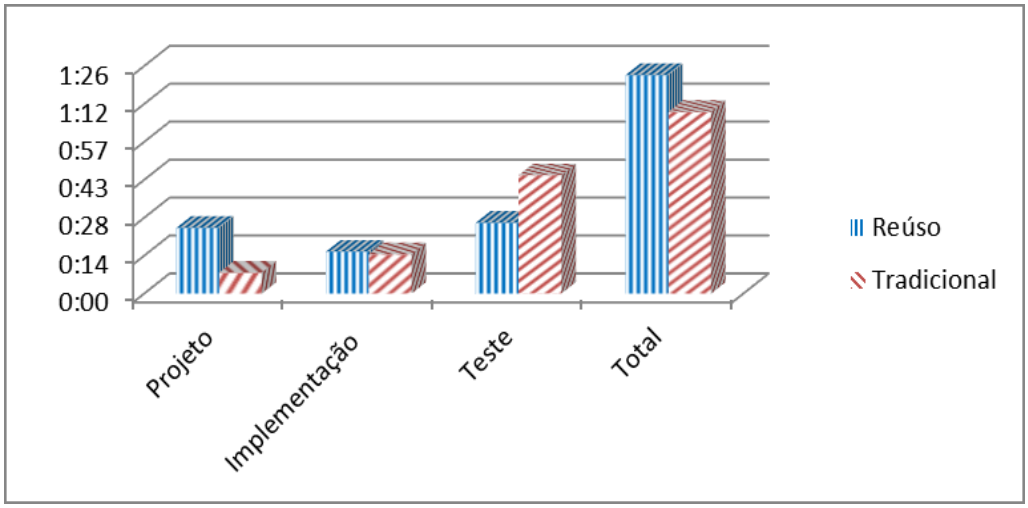

Figura 4.3. Distribuição média dos esforços por atividades do Serviço Web

A Figura 4.4 apresenta a distribuição média dos esforços dos grupos entre as atividades executadas nos processos para o desenvolvimento da Aplicação Móvel. Nota-se que, apesar de os grupos da Abordagem para Reúso terem gasto menos tempo na implementação (em média 5 min) do que os grupos do ciclo de vida tradicional (em média 18 min), os primeiros foram mais produtivos (em média de 263 LOC/h) do que os segundos (em média de 116 LOC/h). A média de linhas de código totais implementadas pelos grupos da abordagem para Reúso foi de 116 LOC, ao passo que os grupos do ciclo de vida tradicional produziram, em média, apenas 104 LOC. Essa relação inversamente proporcional entre tempo despendido e quantidade de linhas de código geradas pode ser justificada pelo uso das transformações M2C na abordagem para Reúso de Software. Os dados comprovam que o esforço de desenvolvimento pode ser significativamente reduzido sem afetar negativamente a produtividade da equipe, uma vez que grande parte das tarefas de codificação pode ser encapsulada nas transformações.

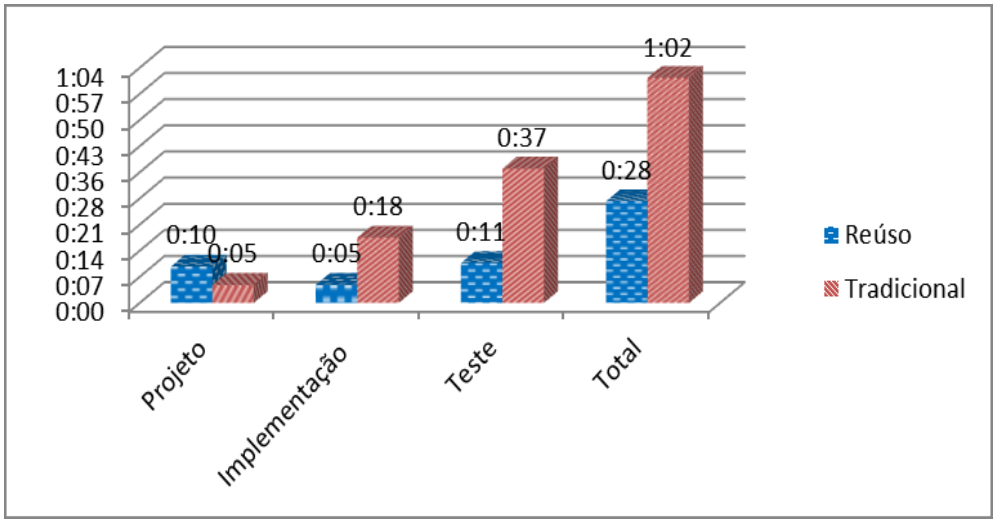

Figura 4.4. Distribuição média dos esforços por atividades da Aplicação Móvel

Outro aspecto importante que se pôde observar nos dados coletados nas atividades de desenvolvimento da Aplicação Móvel foi a relação entre o tempo total gasto pelos grupos que utilizaram a abordagem proposta (em média 28 min) e o tempo total pelos grupos do ciclo de vida tradicional (em média 1h 2 min) que reduziu significativamente o tempo de desenvolvimento da aplicação móvel quando aplicada a abordagem proposta. Isto indica que a abordagem para Reúso de Software colaborou na simplificação do desenvolvimento das aplicações móveis via reúso de serviços Web, para os grupos que aplicaram o desenvolvimento na Aplicação A, Aplicação B ou Aplicação C, o que sugere que a abordagem possui um conjunto de serviços, denominado de Serviços do Domínio, que atendem aos requisitos comuns das diferentes aplicações do domínio do problema.

Concluídas essas observações preliminares, passou-se para as tarefas de obtenção de conclusão do estudo, envolvendo a caracterização dos dados por meio da *estatística descritiva*, a *redução do conjunto de dados* e o *teste das hipóteses*. Essas tarefas são descritas a seguir:

a. *Estatística Descritiva.* Este tarefa é responsável com a representação e processamento numérico do conjunto de dados, representando-o graficamente para melhorar o entendimento de como os dados estão distribuídos bem como ajudar a identificar *outliers* (dado anormal ou atípico para o conjunto de dados da amostra).

A Figura 4.5 apresenta os gráficos de caixa (*boxplot*) que mostram a dispersão dos tempos totais (Figura 4.5 (a)) e das produtividades (Figura 4.5 (b)) dos grupos participantes do experimento para as amostras do serviço Web.

No *boxplot*, a linha sobre cada caixa marca a mediana do conjunto de dados e representa o segundo quartil (Q2) da distribuição. As partes inferior e superior das caixas são delimitadas, respectivamente, pelos quartis inferior (Q1) e superior (Q3). As hastes que se estendem acima e abaixo das caixas representam o limite teórico dentro do qual provavelmente são encontrados todos os dados da amostra se a distribuição for normal. A haste inferior se estende do quartil inferior até o menor valor não inferior a Q1 – 1,5*IQR, em que IQR é o intervalo interquartil (Q3 – Q1). Já a haste superior se estende do quartil superior até o maior valor não superior a Q3 + 1,5*IQR. Os valores que se encontram abaixo ou acima desses limites são representados como pontos individuais no gráfico, sendo caracterizados como *outliers*.

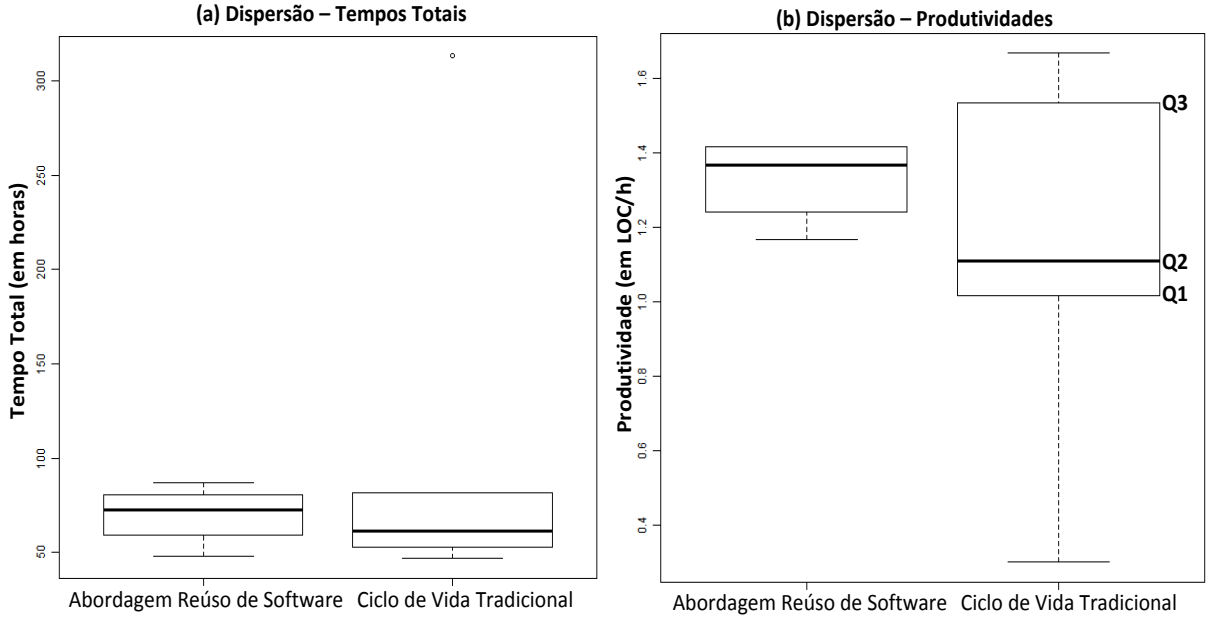

Figura 4.5. Estatística descritiva dos dados das amostras do serviço Web

Conforme se observa nos gráficos de caixa da Figura 4.5, para as amostras de tempo total referente à construção do serviço, o tempo do grupo G4 (1:40h) foi caracterizado como *outlier*, e portanto removido do conjunto para assegurar a correção do teste. Por esse motivo os graus de liberdade do teste para essas amostras foram menores. A partir desses gráficos também é possível observar claramente que não houve uma melhoria considerável no uso da abordagem proposta para o desenvolvimento de um serviço Web, por não obter uma melhora significativa na implementação e na produtividade.

A Figura 4.6 apresenta os gráficos de caixa (*boxplot*) que mostram a dispersão dos tempos totais (Figura 4.6 (a)) e das produtividades (Figura 4.6 (b)) dos grupos participantes do experimento para as amostras da Aplicação Móvel.

Conforme se observa nos gráficos de caixa da Figura 4.6, nenhum *outilier* foi identificado no conjunto de dados da aplicação móvel. A partir desses gráficos também é possível observar claramente a tendência dos grupos da abordagem para o Desenvolvimento de Aplicações Móveis com Reúso de Software em concluir a implementação de forma mais rápida e com maior produtividade, ao contrário dos grupos que não aplicaram a abordagem, que tenderam a ser mais lentos e menos produtivos.

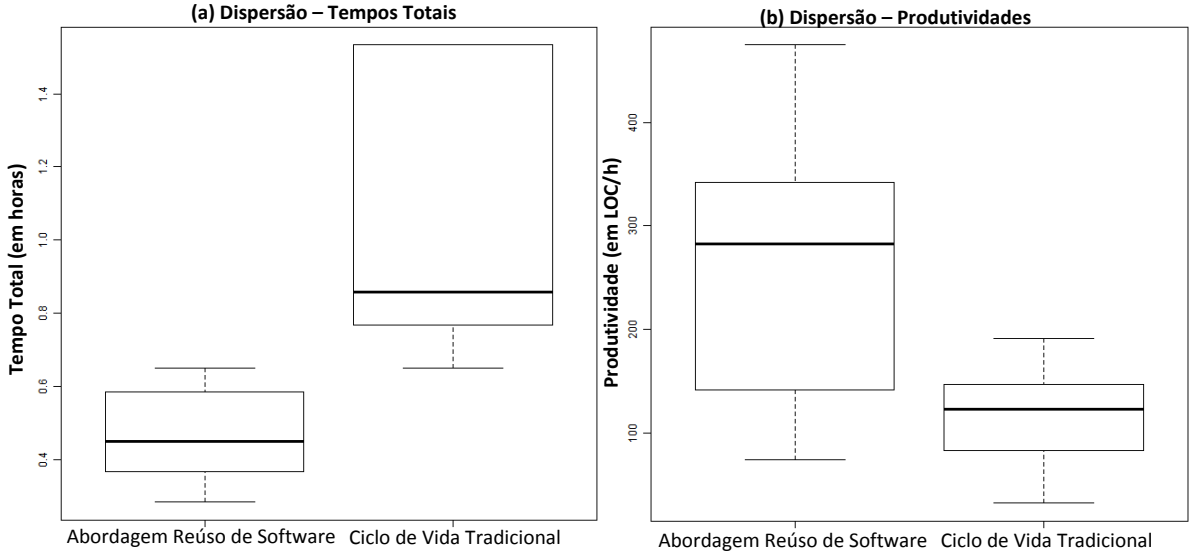

Figura 4.6. Estatística descritiva dos dados das amostras da Aplicação Móvel

A verificação da normalidade da distribuição dos dados das amostras foi feita com o teste não-paramétrico *Shapiro-Wilk* (MONTGOMERY, 2000), conforme Tabela VII e Tabela VIII.

Tabela VII. Normalidade para Serviço Web

	Dados das amostras do Serviço Web			
	Produtividade	**p-value**	**Tempo**	**p-value**
Reúso	0,94356	0,6761	0,84807	0,2199
Tradicional	0,60004	0,0004	0,92365	0,5320

Tabela VIII. Normalidade para os dados da Aplicação Móvel

	Dados das amostras da Aplicação Móvel			
	Produtividade	**p-value**	**Tempo**	**p-value**
Reúso	0,97119	0,8828	0,96161	0,8191
Tradicional	0,98155	0,9590	0,80088	0,0598

b. *Redução do Conjunto de Dados.* Os métodos estatísticos para o teste das hipóteses para obtenção de bons resultados dependem da qualidade dos dados de entrada. Se os dados sobre os quais um método estatístico é aplicado estiverem incorretos, consequentemente os resultados do método também o estarão. É necessário, portanto, remover todos os dados redundantes e os que podem levar a resultados incorretos. Falhas no conjunto de dados geralmente ocorrem por conta de erros durante a transcrição dos dados, ou então pela presença de *outliers*, que são como "ponto fora da curva", i.e., são muito menores ou muito maiores do que se poderia esperar em relação aos demais dados do conjunto.

Conforme constatado na atividade de estatística descritiva, um *outlier* foi encontrado nas amostras do serviço Web, sendo necessária a sua remoção para a obtenção de bons resultados estatísticos. Além disso, todos os formulários com os dados coletados pelos grupos foram cuidadosamente revisados e nem um erro de transcrição foi observado. Portanto, houve necessidade somente da redução do conjunto de dados do serviço Web.

É importante notar que esta atividade tem correlação com a atividade "Validação dos Dados" da fase de Execução do estudo, cujo objetivo foi identificar dados inválidos com base na execução do experimento, como, por exemplo, determinar se os grupos participaram seriamente do estudo fornecendo dados confiáveis. Nesta atividade, a redução dos dados foca na identificação de *outliers* e baseia-se não apenas na forma com que o estudo foi executado, como também na análise dos resultados obtidos nos formulários de coleta de dados considerando, para isso, a estatística descritiva.

c. *Teste das Hipóteses.* A proposta desta atividade foi verificar com algum grau de significância α se era possível rejeitar as hipóteses nulas (H_{0w} e H_{0a}) em favor de alguma das hipóteses alternativas (H_{1w} ou H_{2w}, e H_{1a} ou H_{2a}) com base no conjunto de dados obtido.

O grau de significância reflete o risco de se rejeitar a hipótese nula, sendo esta verdadeira. Para o teste das hipóteses, foi aplicado o teste chamado *t-test* (MONTGOMERY, 2000), o qual se baseia na ideia de que, quando se procura por diferenças entre duas amostras X e Y, deve-se julgar a diferença entre suas médias considerando a dispersão ou variabilidade dos dados que as compõem. Isto porque, quanto maior a variabilidade dos dados das amostras comparadas, maiores as chances desses dados estarem sobrepostos na distribuição normal, mesmo que a

diferença das médias se mantenha constante. A Equação (1) apresenta a fórmula do *t-test*. Trata-se de uma razão: no numerador tem-se a diferença das médias, e no

$$t_0 = \frac{\bar{x} - \bar{y}}{S_p \sqrt{\frac{1}{n} + \frac{1}{m}}} \qquad (1)$$

$$S_p = \sqrt{\frac{(n-1)S_x{}^2 + (m-1)S_y{}^2}{n+m-2}} \qquad (2)$$

denominador tem-se a medida da variabilidade ou dispersão dos dados. Essa medida é obtida com base nas variâncias das amostras (S_x^2 *e* S_y^2) e no número de itens de dados em cada amostra (*n* e *m*), conforme cálculo apresentado na Equação.

Após o cálculo de t_0 deve-se verificar a *tabela padrão de distribuição de probabilidade estatística* ***t*** *de* ***Student*** (Anexo A) (GOSSET, 1947) para descobrir se a razão calculada é suficientemente grande de modo que se possa atestar que é improvável que a diferença amostral tenha sido mera causalidade. Estabelece-se, portanto, um grau de significância α que represente o "ponto de corte" ou "nível de risco" com o qual é permitido afirmar que há diferença entre as amostras e rejeitar a hipótese nula. Por exemplo, tomando-se α = 0,10, significa que se assume um risco de 10% em se encontrar uma diferença significativa entre as médias das amostras, mesmo que essa diferença fosse por acaso (falso positivo). O nível de confiança do resultado do teste neste caso seria de 90%. Também é necessário estabelecer o *grau de liberdade (gl)* para o teste, o qual corresponde à soma do número de dados nas amostras menos dois (m + n – 2).

Assim, tendo t_0, α e gl, observa-se o valor de t na tabela padrão para determinar se o valor de t_0 é grande o suficiente para ser significativo. O valor t verificado na tabela reflete a probabilidade de ter sido obtida a diferença observada entre as médias das amostras numa situação em que as eficiências dos grupos de ambos os processos se equiparam, i.e., a hipótese nula é verdadeira. Se essa probabilidade for muito pequena, então pode-se concluir que o resultado observado no estudo é estatisticamente relevante. Neste caso, se o valor t da tabela referente ao nível de risco escolhido e ao grau de liberdade for inferior ao t_0 calculado, a hipótese nula é rejeitada. Caso contrário, a hipótese nula não é rejeitada e nenhuma conclusão poderá ser obtida do experimento.

Os resultados do teste sobre os dados de tempo e produtividade coletados durante o experimento são apresentados na Tabela IX. A análise dos dados deste

estudo foi realizada utilizando o sistema *R*[10] e o software *RKWard*[11], uma interface gráfica para o *R*.

Tabela IX. Resultados do Teste das Hipóteses

	Amostra	t_0	Grau de Significância (α)	Graus de Liberdade (gl)	Tabela Padrão ($t_{\alpha/2,\ gl}$)
Desenvolver Aplicação Web	$Þ_w$	-0,8224	0,88	9	0,8079
	τ_w	1,0061	0,69	8	1,0035
Desenvolver Aplicações Móveis	$Þ_a$	1,9592	0,17	9	1,9349
	τ_a	-3,231	0,03	9	2,9982

Os resultados do experimento foram divididos nas tarefas “Desenvolver Aplicação Web” e “Desenvolver Aplicações Móveis”. Na tarefa Desenvolver a Aplicação Web, notou-se o elevado valor para o grau de significância referente às amostras de produtividade ($Þ_w \rightarrow \alpha = 0{,}88$) e tempo ($\tau_w \rightarrow \alpha = 0{,}69$). Foi concluído que o alto grau de significância para estes casos foi afetado pela baixa complexidade da aplicação Web, i.e., o fato de desenvolver somente um serviço para ser reutilizado na aplicação Web não impactou de forma significativa na eficiência dos grupos com o uso da abordagem. Na tarefa de Desenvolver Aplicações Móveis com reúso dos serviços, evidenciou-se uma significativa melhora na produtividade ($Þ_a \rightarrow \alpha = 0{,}17$) e no tempo ($\tau_a \rightarrow \alpha = 0{,}03$), principalmente em decorrência de uma complexidade ligeiramente maior das aplicações experimentais.

Com o resultado, considerando valores acentuados para os graus de significância para as amostras de $Þ_w$ e τ_w do desenvolvimento da aplicação Web, concluiu-se que não foi possível rejeitar a hipótese nula $\mathbf{H_{0w}}$. Em relação ao desenvolvimento das aplicações móveis, a conclusão foi de que a hipótese nula $\mathbf{H_{0a}}$ pode ser parcialmente rejeitada. Para o caso das amostras de τ_a, o teste evidencia, a 3% de significância, uma diferença acentuada entre os grupos que aplicaram a abordagem proposta e os grupos que aplicaram a abordagem do ciclo de vida Tradicional. Já em relação às amostras de $Þ_a$, o teste indica diferenças de produtividade entre esses grupos a 17% de significância, o que dificulta obter conclusões mais precisas a esse respeito. Com os dados atuais, tem-se apenas que a hipótese alternativa $\mathbf{H_{1a}}$ tende a ser validada em detrimento da hipótese $\mathbf{H_{2a}}$. Contudo, visto o pequeno porte das aplicações, não foi possível rejeitar $\mathbf{H_{0a}}$ por completo. No entanto, com base em uma análise dos dados coletados, pode-se inferir que, para os casos de projetos maiores, em que o nível de complexidade dos

[10] http://www.r-project.org.
[11] http://rkward.sourceforge.net.

serviços Web tende a ser maior, a tendência é que diferenças mais acentuadas poderão ser observadas nas amostras de tempo e produtividade da aplicação de ambas as abordagens. Isto possibilitará obter conclusões mais significativas para uma validação mais abrangente das hipóteses de pesquisa.

Considerando o experimento com relação às amostras de produtividade e tempo das equipes no desenvolvimento das aplicações móveis, dado o alto grau de significância do teste para as amostras de produtividade, não se pôde obter uma conclusão mais precisa em relação ao impacto da abordagem na produtividade das equipes desenvolvendo aplicações móveis. Por outro lado, com relação às amostras de tempo, o teste indica que as equipes utilizando a abordagem proposta tendem a gastar menos esforços no desenvolvimento das aplicações móveis. Assim, conclui-se que para este caso a hipótese nula pode ser parcialmente rejeitada. Para as amostras de Produtividade e Tempo do desenvolvimento do serviço Web observou-se um alto grau de significância em ambas as amostras, o que dificulta rejeitar a Hipótese Nula. Porém, observou-se que esse fato foi ocasionado pela baixa complexidade do serviço Web, i.e., o fato de desenvolver somente um serviço para ser reutilizado não impactou de forma significativa na eficiência dos grupos com o uso da abordagem.

Enfim, considerando que o experimento foi realizado *in-vitro* sob condições controladas, é importante ressaltar que as conclusões a respeito dos resultados observados neste trabalho se restringem ao escopo de desenvolvedores de software em ambiente universitário no qual o estudo foi conduzido. Por questões de validade, para expandir a generalização do fenômeno observado para um contexto mais amplo, é necessário que novos estudos com um número maior de equipes sejam realizados em ambientes *in-vivo*, comparando-se também o uso da abordagem proposta em relação a outras abordagens de desenvolvimento, tais como abordagem de Desenvolvimento Orientado a Serviços e Linhas de Produto de Software, por exemplo. Isto permitirá obter uma validação mais abrangente das hipóteses de pesquisa. A extensão deste experimento em ambiente industrial, portanto, torna-se importante já que outros fatores ausentes no ambiente acadêmico poderão ser controlados e estudados.

4.1.5 Ameaças à Validade

Uma das discussões principais envolvendo os resultados de um experimento é determinar quão válidos eles são. Por isso, é fundamental considerar a questão da validade da avaliação desde a fase de planejamento de forma a guiar a execução do experimento no intuito de obter uma validade adequada dos resultados. Validade adequada, neste caso, significa que os resultados são válidos para a população de interesse, tanto para o subconjunto populacional que executou o estudo, quanto para uma população mais ampla para a qual os resultados podem ser generalizados (WHOLIN et al., 2000).

Segundo Wohlin et al. (2000), as principais ameaças que podem colocar em risco a validade de um experimento são: *de conclusão*, *interna*, *de construção* e *externa* (WHOLIN et al., 2000).

a) *Validade de Conclusão.* De acordo com as questões que afetam a habilidade de tirar conclusões corretas a respeito do efeito dos tratamentos nas variáveis dependentes (e.g. escolha do método estatístico adequado para análise, cuidados tomados na execução e na medição do experimento). No caso do estudo, em se tratando de um projeto do tipo um fator com dois tratamentos, o teste estatístico *t-test* foi adotado. Esse teste é o mais recomendado para projetos desse tipo, em que o objetivo é comparar as médias obtidas nos resultados de dois tratamentos distintos. O *t-test* usualmente requer dados normalmente distribuídos para comparação. O teste de normalidade de *Shapiro-Wilk* (**Erro! Fonte de referência não encontrada.** e **Erro! Fonte de referência não encontrada.**) assegura a correção da análise em vista do pequeno tamanho do conjunto amostral de dados, que dificulta obter uma conclusão precisa a respeito de sua normalidade. Por exemplo, alguns dos dados do serviço Web (**Erro! Fonte de referência não encontrada.**) não atingiram o nível de 5% de significância para a produtividade (0,0004). Entretanto, conforme indicado por Wohlin et al. (2000), mesmo que essas amostras não estejam em sua distribuição normal, o *t-test* é robusto o suficiente para suportar alguns desvios dessa pré-condição. Particularmente, sendo um teste paramétrico, sua potência é, em geral, maior do que a de testes não-paramétricos, o que também justifica seu uso para obter resultados mais precisos no estudo. Porém, para o teste de normalidade dos dados da aplicação móvel foi positivo, com amostras do estudo a 5% de significância.

b) *Validade Interna.* De acordo com as questões que afetam a habilidade de assegurar que os resultados foram, de fato, obtidos em decorrência dos tratamentos e não por uma coincidência, ou pelo efeito de outro fator que não foi medido ou não se pôde controlar, e.g., modo como os participantes são selecionados, agrupados, recompensados e tratados; situação em que ocorre o experimento. No caso do estudo, buscou-se realizar o experimento com estudantes da área de Computação de seus cursos de graduação, os quais já possuíam certo nível de experiência com desenvolvimento de software, conforme demonstrado em seus formulários de caracterização. Assim, admitiu-se que eles são representativos para a população dos desenvolvedores de software. Ainda, os estudantes foram agrupados adequadamente conforme seus níveis de experiência para que os grupos ficassem homogêneos, evitando, dessa forma, discrepâncias na velocidade com que cada grupo aprendia a executar os processos à medida que interagiam com o experimento. Nenhum tipo de auxílio ou benefício quanto à nota da disciplina foi oferecida aos estudantes de modo a evitar que se comportassem com empenho anormal durante o estudo e não criar expectativas para obtenção de vantagens neste sentido. O estudo também foi realizado em uma única sessão e simultaneamente com todos os grupos, de forma que todos os participantes estivessem sob as mesmas circunstâncias durante a execução do experimento.

c) *Validade de Construção.* De acordo com as questões que afetam a habilidade de generalizar o resultado do experimento ao conceito ou teoria envolvidos no estudo, e.g., definição adequada das medidas e tratamentos. No estudo realizado, o objetivo era comparar dois processos de desenvolvimento em relação a seu impacto na eficiência da equipe. Para os propósitos do estudo, uma equipe eficiente é aquela que produz mais código em uma hora e finaliza a implementação em menos tempo. Assim, os dados para coleta e os tratamentos foram definidos de modo que fosse possível realizar adequadamente a análise do efeito na eficiência das equipes em conformidade com os objetivos do estudo. A fim de evitar interações dos participantes voltadas para maximização ou minimização das métricas de interesse (produtividade e tempo despendido), também evitou-se divulgar exatamente quais métricas seriam analisadas quando os participantes foram informados dos objetivos; além disso, evitou-se que eles se comportassem de maneira diferente do comum, buscando se empenhar melhor ou agir de forma

negligente mais do que de costume apenas para obterem resultados que favorecessem ou prejudicassem o experimento.

d) *Validade Externa.* Com relação à validade externa do estudo realizado, o experimento foi conduzido em um laboratório informatizado com equipamentos adequados, bem como com ferramentas e tecnologias de desenvolvimento atualizadas comumente empregadas em ambientes industriais, tais como o IDE Eclipse e a linguagem de programação Java. Em relação às questões temporais, o experimento foi planejado de forma que os estudantes pudessem desempenhá-lo dentro de um período de quatro horas, evitando que os resultados fossem afetados em decorrência de tédio ou desgaste excessivo dos participantes.

4.2 Considerações Finais

Este capítulo apresentou a avaliação da abordagem proposta. A avaliação seguiu a metodologia experimental, em que se avaliou o impacto do uso da abordagem para o Desenvolvimento de Aplicações Móveis com Reúso de Software baseado na Modelagem Específica de Domínio e Arquitetura Orientada a Serviços e de um processo baseado no ciclo de vida tradicional do software na eficiência da equipe de desenvolvimento.

Os resultados mostraram perdas de eficiência no desenvolvimento de serviços Web, usando a abordagem proposta. Isto se deve à baixa complexidade dos serviços usados nas aplicações. Contudo, deve-se considerar que grande parte desta tarefa pode ser executada na ED e não na EA. Portanto, com a abordagem proposta, esse esforço é realizado uma única vez na ED, ficando os serviços disponíveis para reúso, a partir do metamodelo.

No desenvolvimento da aplicação móvel os resultados foram bastante favoráveis ao uso da abordagem. Portanto, os resultados obtidos com a avaliação do trabalho vão ao encontro das expectativas iniciais da abordagem desenvolvida, que incluem redução dos esforços, aumento da produtividade e melhoria pelo reúso de serviços Web.

Capítulo 5

TRABALHOS RELACIONADOS

Na literatura há diversos trabalhos relacionados como o processo de desenvolvimento de software usando os conceitos e tecnologias da DSM e da SOA, o que demonstra a sua importância nessa área de pesquisa. Neste capítulo são apresentados os trabalhos que se relacionam de diferentes perspectivas com a pesquisa apresentada nesta dissertação.

5.1 Trabalhos Encontrados na Literatura

O trabalho de Wee Siong Ng et al. (NG et al, 2009) propõem o uso da SOA para aplicações distribuídas em diferentes plataformas. O objetivo desse trabalho foi construir uma plataforma de serviços de cuidados de saúde (*Healthcare*) denominada *Service-oriented Architecture for m-Healthcare (SOAMOH)* para que serviços possam ser acessados por meio de dispositivo móveis a partir de qualquer localidade e a qualquer momento. As vantagens tecnológicas proposta com o uso dessa plataforma é um sistema escalável que possibilita projetar e compor novas aplicações e serviços formando assim um repositório. Para o domínio *Healthcare* as vantagens são que por meio de dispositivos móveis possam acessar serviços de monitoramento em tempo real da saúde dos pacientes para observação dos especialistas, serviços que disponibilizam registro de saúde pessoal armazenado em um repositório para acesso pelos médicos e outros serviços que possam ser compostos para disponibilizar cuidados de saúde. Nesse sentido, esta dissertação

também propõe o uso SOA para a construção de um repositório de serviços para o domínio *Healthcare*. Porém, a abordagem, proposta promove reúso de serviços disponíveis em um metamodelo. Os serviços estão disponíveis por meio das concepções da DSM com o foco na atividade de análise. Assim é possível a modelagem de várias aplicações do domínio *Healthcare* e a geração de código para aplicações móveis.

No trabalho de Marin e Lalanda (MARIN & LALANDA, 2007) é elaborada uma ferramenta, chamada *DoCoSOC*, que utiliza uma DSL, que possibilita a composição automática de serviços em diversos contextos. Com a *DoCoSOC* pretende-se facilitar a modelagem dos serviços específicos do domínio do problema. Possui ainda um gerador de código que automatiza parte da construção dos serviços Web especificados nos modelos. Porém, este trabalho apresenta uma abordagem para reúso e um grupo de serviços de um domínio específico que compõem o metamodelo e, portanto, se apresenta como um diferencial em relação ao trabalho de Marin e Lalanda.

Outra pesquisa que se relaciona com o trabalho proposto nesta dissertação é o de Santana et al. (SANTANA et al., 2009). Nessa pesquisa é proposta uma abordagem para o desenvolvimento de software na computação ubíqua, baseado na DSM em Linhas de Produtos de Software. A abordagem inicia-se com os requisitos do domínio do problema e utilizando o conceito de *features* identifica os *assets* que constituirão o *core asset* da Linha de Produto na Computação Ubíqua. A partir dos requisitos e orientado por técnicas da DSM elabora-se um metamodelo, que suporta a modelagem de diversos produtos do domínio do problema. O metamodelo foi especificado com o EMF e em uma arquitetura Cliente e Servidor. Um *framework*, denominado *Ubiquitous Computing Framework (UCF)* é responsável pelos aspectos de adaptação de conteúdo e ciência de contexto da Computação Ubíqua, sendo um dos *assets* do núcleo da Linha de Produto. Com base no metamodelo e um gerador de código para a plataforma JME são desenvolvidos os produtos. Dessa forma, por meio do metamodelo foi possível reutilizar o conhecimento do domínio do problema na construção dos produtos, automatizar e tonar mais ágil grande parte das atividades do desenvolvimento dos produtos da LPS com a geração de código e disponibilizar uma ferramenta como apoio computacional para suportar as atividades. Embora as etapas utilizadas nesse trabalho para o desenvolvimento do metamodelo sejam semelhantes ao da abordagem apresentada nesta dissertação, o

mesmo não é baseado na SOA. Os serviços Web são especificados no metamodelo e possibilita o reúso dos mesmos para um domínio específico na construção de aplicações móveis. Além disso, outro diferencial apresentado nesta dissertação são as entidades do domínio no metamodelo. Com essas entidades na atividade de modelagem das aplicações móveis é possível criar modelos baseado em uma sintaxe concreta, familiar e intuitiva para os especialistas do domínio. Dessa forma, os modelos expressam soluções no idioma e no nível de abstração do domínio, reduzindo os esforços de tradução dos conceitos dos desenvolvedores para especialistas do domínio.

O trabalho de Cao et al. (CAO et al., 2004) demonstram uma abordagem de Metamodelagem para Web Service, baseado nos princípios da *Model Integrated Computing (MIC)*. A abordagem utiliza a *Generic Modeling Environment (GME)* para a criação dos modelos específicos de domínio. O processo apresentado é baseado na representação de *Entidade Relacionamento (ER)* como forma de derivar e evoluir a Metamodelagem. Os autores mostram que utilizar ER é uma forma intermediária para construir metamodelo e evita a natureza *ad-hoc*. A abordagem proposta nesta dissertação difere daquela proposta por Cao et. al. (CAO et. al., 2004) porque além de propor uma abordagem para Metamodelagem busca o reúso de serviços em aplicações móveis. Além disso, a abordagem utiliza os conceitos e técnicas da DSM por meio da ferramenta EMF na atividade de Metamodelagem. Outra diferença da abordagem é a geração parcial de código para aplicações móveis por meio das instancias do metamodelo.

Czarnecki et al. (CZARNECKI et al., 2005), discutem a combinação das ideias de desenvolvimento orientado a modelos e linhas de produto de software (LPS), ressaltando que essas duas linhas são complementares. Neste sentido, os autores propõem a combinação da modelagem de *feature*, com o uso de *templates* de características, responsáveis pelo mapeamento automático das características para modelos que as implementam, com base em transformações de modelos. Esse trabalho, apesar de relevante, considera apenas a modelagem de características e sua representação em modelos, deixando de lado outras etapas do processo. Além disso, não aborda SOA como uma maneira complementar para o desenvolvimento por meio de suas características de interoperabilidade e reúso.

5.2 Considerações Finais

Este capítulo apresentou os trabalhos associados ao desenvolvimento utilizando as concepções de Modelagem Específica de Domínios e Arquitetura Orientada a Serviço que possuem correlação ao trabalho desenvolvido nesta pesquisa para a formulação de uma abordagem que define atividades e artefatos para apoio ao desenvolvimento via reúso. Os trabalhos apresentados relacionados demonstram a grande preocupação e os esforços da comunidade acadêmica em contribuir com essas áreas, a fim de tornar mais ágil o desenvolvimento de aplicativos móveis.

A abordagem proposta é baseada em diversas características dos trabalhos correlatos, apresentando, porém, suas próprias contribuições por meio da adequação e evolução dos trabalhos em que está baseado. Mais precisamente, os trabalhos citados evidenciam os esforços das pesquisas usando Metamodelagem e Serviços Web para melhorar e facilitar o desenvolvimento de software. Da mesma forma, a abordagem proposta neste trabalho busca utilizar a modelagem de software como ferramenta ativa no desenvolvimento baseado na reutilização de software e geração de código.

Capítulo 6

CONCLUSÃO

Nos últimos anos, muitos processos e abordagens têm sido pesquisados para suprir as dificuldades ou melhorar o desenvolvimento para as novas arquiteturas e técnicas que apoiem o reúso de software. A definição de uma abordagem adequada, portanto, torna-se essencial para fornecer aos desenvolvedores o suporte necessário para o cumprimento de prazos considerando as diferentes plataformas e tecnologias sem perder a qualidade do software.

Diante disso, neste trabalho foi proposta uma abordagem para o Desenvolvimento de Aplicações Móveis com Reúso de Software baseado na DSM e na SOA. A abordagem define atividades e artefatos que auxiliam na modelagem e geração de código das aplicações móveis e de serviço Web. Esses artefatos compreendem um metamodelo do Domínio do Problema que expressa a sintaxe abstrata de uma DSL para apoio à modelagem e transformações M2C para geração de código. Na abordagem também são desenvolvidos os Serviços do Domínio que atendem aos requisitos comuns das diferentes aplicações do domínio do problema para o qual o metamodelo foi construído. Esses artefatos são construídos previamente na abordagem em uma etapa de ED e reutilizados durante a EA para simplificar o desenvolvimento de aplicações móveis por meio do reúso de serviços Web e geração de código.

6.1 Contribuições e Síntese dos Principais Resultados

Esta dissertação apresentou uma abordagem para o Desenvolvimento de Aplicações Móveis com foco em Reúso baseado em DSM e SOA. A abordagem foi avaliada com o desenvolvimento de aplicações do domínio *Healthcare* com serviços de Atendimento Emergência, mas pode ser utilizada no desenvolvimento de software de outros domínios de aplicações. Foi construído um metamodelo contendo metaclasses cujas instanciações geram modelos de aplicações desse domínio. Um conjunto de serviços desse domínio foi também especificado, implementado, ficando disponível para o reúso. Além disso, transformações de M2C foram construídas para apoiar a geração de código das aplicações.

A avaliação da abordagem em um estudo de caso possibilitou representar os conceitos do domínio das aplicações para atendimento de emergência na modelagem, a partir da qual, aplicando-se as transformações M2C adequadas, foi gerado o código parcial das classes de entidades e dos serviços Web para aplicação móvel. Observou-se ainda que a representação dos serviços de domínio via modelos colaborou para a redução do tempo de implementação das aplicações, facilitada pelo reúso do metamodelo e das transformações. Além disso, os serviços de domínio construídos na ED permitiram o reúso e proporcionaram que o foco do desenvolvimento durante a EA fosse mantido nos requisitos de negócio da aplicação.

Para validação do trabalho foi realizada uma experimentação da abordagem, na qual grupos de estudantes executaram a etapa de EA, enquanto outros grupos seguiram as disciplinas do ciclo de vida tradicional, para a construção das aplicações móveis. Os dados coletados durante o estudo referentes ao tempo gasto e ao total de linhas de código implementadas pelos grupos de ambos os processos serviram de base para análise. Os resultados obtidos, mesmo que em contexto universitário, evidenciaram o potencial da abordagem proposta em colaborar para o aumento da eficiência de equipes no desenvolvimento de aplicações móveis que reutilizam serviço Web em termos de tempo despendido e produtividade. Em geral, os grupos que aplicaram a abordagem para o desenvolvimento da aplicação móvel foram mais rápidos e produtivos do que os grupos que não usaram a abordagem proposta. Os resultados evidenciaram que, com o uso da Abordagem para o Desenvolvimento de

Aplicações Móveis com Reúso de Software, os esforços de desenvolvimento podem ser significativamente reduzidos. Esta redução vem do emprego de modelos específicos do Domínio do Problema, no caso o domínio *Healthcare* para esse trabalho, que facilita a tradução e comunicação dos conceitos e sintaxes do domínio do problema, e também do uso de transformações M2C, que encapsulam grande parte das tarefas de codificação das classes de entidade, serviço Web e o reúso nas aplicações móveis.

Em síntese, ao explorar os fundamentos de DSM e SOA, a abordagem apresentada neste trabalho contribui para a Engenharia de Software nas seguintes áreas:

a) Reúso de Software: por meio dos artefatos metamodelo do domínio; as transformações M2C e os Serviços do Domínio produzidos na ED podem ser reutilizados em diferentes aplicações durante a EA em um domínio específico. Além disso, há indicações de vantagens em aspectos como: reutilização, redução no tempo de implementação, interoperabilidade e manutenibilidade.

b) Modelagem Específica de Domínio (DSM): um metamodelo do domínio de cuidado de saúde (*Healthcare*), com serviços Web de atendimento emergencial construído, facilita a comunicação, modelagem e o desenvolvimento das aplicações desse domínio. A geração parcial do código colabora para a redução dos esforços de desenvolvimento. A grande vantagem da DSM é o reúso no nível de modelagem, o que possibilita que uma aplicação pode ser implementada para diferentes plataformas a partir dos mesmos modelos aplicando-se as transformações M2C apropriadas; o emprego da DSM na abordagem também possibilita que as aplicações de um domínio de problema sejam modeladas com reúso por meio de serviços cujas especificações, implementações e testes são realizadas na ED.

c) Arquitetura Orientada a Serviços (SOA): a modelagem de serviços para um domínio específico e as transformações M2C possibilitam a melhoria na tarefa de construir serviços Web, já especificados no metamodelo.

d) Transformações M2C: as transformações, construídas na ED, apoiam a abordagem, gerando parte do código das aplicações móveis reutilizando os serviços. No processo de escrita das transformações o Engenheiro de Domínio utiliza *templates* usando uma biblioteca de marcações (*tags*), o que propicia a geração de código mais legível e simplifica essa tarefa, muitas vezes difícil e complexa em outros sistemas de transformação como o Draco (ABRAHÃO & PRADO, 1999),

Turing eXtender Language (TXL) (THURSTON & CORDY, 2006), e outros, que utilizam a *Abstract Syntax Tree (AST)*.

e) Apoio Computacional: o metamodelo e as transformações M2C desenvolvidas na ED tornaram-se operacionais com um *plugin* da IDE Eclipse que apoia grande parte das atividades dos Engenheiro de Aplicação.

Com base na pesquisa apresentada nesta dissertação, foram publicados, em colaboração com diferentes pesquisadores, os seguintes artigos:

MARCONDES, C.; **BELLINI, A.**; CIRILO, C. E.; FERRAZ, V. R. T.; DUQUE, J. L.; ANNIBAL, L. P.; ARAUJO, J. G.; DURELLI, R. S. A Low Cost Positioning and Visualization System using Smartphones for Emergency Ambulance Service. In: ACM/IEEE 32nd International Conf. on Software Engineering (ICSE 2010), 2010, Cape Town. Proceedins of 2nd Workshop on Software Engineering in Health Care, 2010.

BELLINI, A.; PRADO, A. F.; ZAINA, L. A. M. Top-Down Approach for Web Services Development. In: Fifth International Conference on Internet and Web Applications and Services, 2010, Barcelona. WSSA-Web Services-based Systems and Applications. Los Alamitos, CA, USA : IEEE Computer Society, 2010. v. 0. p. 426-431.

BELLINI, A.; CIRILO, C. E.; PRADO, A. F.; ZAINA, L. A. M.; SOUZA, W. L. A Service Layer for building GSM Positioning Systems in e-Health Domain. In: U-Media - International Conference on Ubi-Media Computing, 2011, São Paulo. International Conference on Ubi-Media Computing, 2011. p. 7-12.

CIRILO, C. E.; **BELLINI, A.**; PRADO, A. F.; ZAINA, L. A. M. Desenvolvimento de Sistemas Sensíveis ao Contexto usando Web Services. In: Profa. Dra. Simone das Graças Domingues Prado; Prof. Dr. Elvio Gilberto da Silva; Prof. Ms. Roque Maitino Neto; Prof. Ms. Marcelo José Storion. (Org.). VII Escola Regional de Informática - São Paulo/Oeste 2010 - Livro dos Minicursos apresentados no Evento. Bauru: SBC - Sociedade Brasileira da Computação, 2010, v. , p. 83-111.

6.2 Dificuldades e Limitações

As principais limitações são relacionadas com:

- Construção do metamodelo: tarefa que exige do desenvolvedor um bom conhecimento do domínio do problema.
- Construção das Transformações: tarefa que exige o conhecimento e escrita de *templates* para a geração parcial do código. O Engenheiro de Domínio precisa conhecer tanto a linguagem de origem quanto a linguagem-alvo da transformação.

- Identificação dos Serviços: essa tarefa exige que o Engenheiro de Domínio identifique os serviços que atendam a diferentes aplicações do domínio do problema.
- Conhecimento da plataforma constituída dos *frameworks* EMF, *Graphical Modeling Framework (GMF)* e JET, para a construção dos artefatos da ED.

Embora essas tarefas sejam mais difíceis de serem realizadas, elas são executadas uma única vez na ED para cada domínio de problema. Além disso, o Engenheiro de Domínio pode iniciar com um metamodelo menor, um pequeno conjunto de transformações e serviços, e evoluir conforme as novas necessidades no desenvolvimento das aplicações do domínio do problema.

6.3 Trabalhos Futuros

Com base no desenvolvimento deste trabalho, algumas oportunidades de melhoria, tanto na abordagem quanto do suporte computacional, podem ser destacadas. Além disso, novas necessidades de pesquisa foram identificadas. Os possíveis trabalhos futuros envolvem:

- A inclusão no metamodelo de novos componentes do domínio do problema e a construção de notações gráficas para tornar a modelagem mais abrangente e amigável.
- A sistematização dos testes dos artefatos reutilizáveis produzidos na ED de forma a maximizar a correção de defeitos e evitar que estes se propaguem nas aplicações desenvolvidas na EA.
- Novos estudos experimentais com a abordagem para Reúso de Software baseado na DSM e SOA em *Healthcare* para extensão do metamodelo com outros componentes visando atender outras aplicações desse domínio. Outros aspectos também poderão ser estudados, tais como análise de Reusabilidade, Quantificação de Reúso e Métricas de Software (i.e. métricas de coesão, métricas de acoplamento, tamanho das classes em termos de métodos e atributos). Além disso, a inclusão de mais serviços no metamodelo reduzirá os esforços da EA, proporcionando resultados ainda mais favoráveis

para o uso da abordagem proposta, comparada com o ciclo de vida tradicional de desenvolvimento.

- A inclusão de módulos de orquestração e composição de serviços do domínio, visando à redução dos esforços de implementação com serviços fornecidos por terceiros.

Por fim, apesar do foco deste trabalho ter sido mantido no Domínio *Healthcare*, a abordagem pode ser aplicada para atender a outros domínios de problemas. A Engenharia de Domínio com o desenvolvimento de DSLs e Transformações precisa ser testada em outros domínios de aplicações. Assim, este trabalho prossegue com pesquisas para definir um processo mais abrangente para apoiar os desenvolvedores de aplicações móveis via o reúso com base na DSM e SOA.

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAHÃO, S. M.; PRADO, A. F. "**Web-Enabling Legacy Systems Through Software Transformations**" In. International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems, p. 87, 1999.

ALMEIDA, R. A. P. BLUEYOU: Uma Plataforma De Comunicação Ciente De Contexto Baseada Em Serviços Para Computação Móvel. **Dissertação de Mestrado**. Universidade Federal de São Carlos. Departamento de Computação, São Carlos, SP, 2011.

AMBLER, S. W. Agile model driven development is good enough. **IEEE Software**, v. 20, n. 5, p. 71-73, 2003.

BAHNOT, V. et. al. Using domain-specific modeling to develop software defined radio components and applications. In: **The 5th OOPSLA Workshop on Domain-Specific Modeling**, San Diego USA. [S.1.: s.n.], 2005.

BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems. **Business Process Execution Language for Web Services Version 1.1**, Maio 2003.

BERNERS-LEE, T.; MASINTER, L.; McCAHILL, M. **Uniform resource locators (URL)**. Dec. 1994. Disponível em: http://www.ietf.org/rfc/rfc1738.txt Acesso em: Jun. 2011.

BLOIS, A. P.; WERNER, C. M. L.; BECKER, K. Towards a components grouping technique within a Domain Engineering process. In: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, 2005, Porto, Portugal. **Proceedings… IEEE Computer Society**, 2005, p. 18-25.

CAO, F.; BRYANT, B.R.; ZHAO, W.; BURT, C.C.; RAJE, R.R.; OLSON, A.M.; Auguston, M. "A meta-modeling approach to Web services", **Proceedings. IEEE** International Conference on Web Services, vol., no., pp. 796- 799, July 2004.

CIRILO, C. E. Model Driven RichUbi - Processo Dirigido a Modelos Para a Construção de Interfaces Ricas de Aplicações Ubíquas Sensíveis ao Contexto. **Dissertação de Mestrado**. Universidade Federal de São Carlos. Centro de Ciências Exatas e de Tecnologia, São Carlos, SP, 2011.

CHRIS, P. **Web Services Orchestration and Choreography**. Computer, 2003.

CONRADI, R.; BASILI, V.; CARVER, J.; SHULL, F.; Travassos, G. **A Progmatic Document Standards for na Experience Library: Roles, Documents, Contents and Structure**, Technical Report CS-TR-4235, University of Maryland, April 2001.

COSTA, C. G. A. "**Desenvolvimento e Avaliação Tecnológica de um Sistema de Prontuário Eletrônico do Paciente, Baseado nos Paradigmas da World Wide Web e da Engenharia de Software**". 2001. Disponível em: <http://cutter.unicamp.br/document/?code=vtls000231024>. Acesso em: Jun. 2011.

CROCKFORD, D. **The application/json Media Type for JavaScript Object Notation (JSON)**. Disponível em: <http://www.ietf.org/rfc/rfc4627.txt?number=4627>. Acesso em: 18 Jun. 2011.

CZARNECKI, K. et. al. Model-driven software product lines. In: *20th Annual **ACM SIGPLAN** Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2005).* San Diego, CA, USA: ACM, 2005. P. 126-127.

DAILY, P. G., 2003, "**XML Basics and Benefits**". Disponível em <http://www.intranetjournal.com/articles/200312/ij_12_08_03a.html>. Acesso em: 17 junho 2011.

ECLIPSE Foundation. EMFT JET Developer Guide. Disponível em:<http://help.eclipse.org/>. Acesso em: Jun. 2011.

ENDRES, A. **Lessons Learned in an Industrial Software Lab**, In: IEEE Software, Vol. 10, No. 05, September, 1993, pp. 58-61.

ERL, T.: **SOA Principles of Service Design**. The Prentice Hall Service-Oriented Computing Series from Thomas Erl. Prentice Hall PTR, 2007.

ERL, T.: **Service-Oriented Architecture: Concepts, Technology & Design**. The Prentice Hall Service-Oriented Computing Series from Thomas Erl. Prentice Hall PTR, 2005.

FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. 2000. Dissertation (PhD) - University of California, Irvine, 2000. Disponível em: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. Acesso em: Junho 2011.

FOWLER, S.; ASK, J. A.; GOWNDER, J. P. **Engaging Smartphone Users**. Disponível em: <http://www.forrester.com/rb/Research/engaging_smartphone_users/q/id/56086/t/2>. Acesso em: 15 fev. 2010.

FRAKES, W.B.; ISODA, S. **Success Factors of Systematic Software Reuse**, In: IEEE Software, Vol. 12, No. 01, January, 1995, pp. 14-19.

GAION, S.; MININEL, S.; VATTA, F.; UKOVICH, W. "**Design of a domain model for clinical engineering within the HL7 Reference Information Model**". In: IEEE Workshop on Health Care Management (WHCM), Venice-Italy, pp. 1-6, 2010.

GARTNER. **Gartner Highlights Key Predictions for IT Organisations and Users** in 2010 and Beyond, 2010. Disponível em: http://www.gartner.com/it/page.jsp?id=1278413. Acesso em: 31 jan 2011.

GOMAA, H. **Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures**. Redwood City, CA, USA: Addison Wesley Longman, p. 736, 2004.

GOOGLE. **Google Android**. Disponível em: <http://code.google.com/intl/en/android/>. Acesso em: Jun. 2011.

GOSSET, W. S. **"Student's" Collected Papers**. Biometrika Office, 1947.

GREENFIELD, J.; SHORT, K.; COOK, S.; KENT, S. "**Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools**". Wiley, 2004.

GRISS, M.L. **Software Reuse Experience at Hewlett-Packard**, In: Proceedings of the 16th IEEE International Conference on Software Engineering (ICSE), Sorrento, Italy, ACM Press, May, 1994, pp. 270.

GROUP, O. M. **Common Object Request Broker Architecture (Corba)**. Disponível em: <http://www.corba.org/>. Acesso em: 20 Jun. 2011.

HADLEY, M.; SANDOZ, P. **JAX-RS: Java™ API for RESTful Web Services**, Version 1.1, Sun Microsystems, Inc., 2009. Disponível em: <http://jcp.org/aboutJava/communityprocess/mrel/jsr311/index.html>. Acesso em: Abr. 2011.

HADLEY, M. Web Application Description Language (WADL), Sun Microsystems, 2006. Disponível em: <http://labs.oracle.com/techrep/2006/smli_tr-2006-153.pdf>. Acesso em: Jul. 2011.

Introducing **JSON**. Disponível em: <http://www.json.org/>. Acesso em: Abr. 2011.

ITO, M.; MARTINI, J. S. C.; IOCHIDA, L. C. "**A Modelagem de Negócio com UML de uma Central de Monitoração de Diabéticos, Revista Eletrônica de Sistemas de Informação**", Vol. 5, No 3, 2009. Disponível em: <http://revistas.facecla.com.br/index.php/reinfo/article/view/232/>. Acesso em: Jun. 2011.

JOOS, R. **Software Reuse at Motorola**, In: IEEE Software, Vol. 11, No. 05, September, 1994, pp. 42-47.

KANG K. C.; KIM S.; LEE J.; KIM K.; SHIN E.; HUH, M. **Form: A feature-oriented reuse method with domain-specific reference architectures**. Ann. Softw. Eng., 5:143–168, 1998.

KELLY, S.; TOLVANEN, JP. "**Domain-Specific Modeling: Enabling full code generation**", John Wiley & Sons, 2008, pp. 427.

KRASNER, G.; POPE, S. "**A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80**", JOOP, v.1, n.3, pp. 26-49, 1988.

KRUEGER, C. W. Software reuse. **ACM Computing Surveys**, v 24, n. 2, p.131-183, 1992.

KIM S.; HAN, S. **Performance comparison of DCOM, CORBA and Web service**. In: Parallel and Distributed Processing Techniques and Applications, 2006, pp. 106-112.

KUHNE, T. **Making modeling languages fit for model-driven development**. In: Fourth International Workshop on Software Language Engineering, Nashville, USA. Grenoble, France: megaplanet.org, 2007. ISBN not assigned.

LEITE, M. E. **"System of information for clinical research: case study of IPEC/FIOCRUZ"**, 2005. Disponível em: <http://bases.bireme.br/cgi-bin/wxislind.exe/iah/online/?IsisScript=iah/iah.xis&src=google&base=LILACS&lang=p&nextAction=lnk&exprSearch=422215&indexSearch=ID>. Acesso em: Abr. 2011.

LUCRÉDIO, D.; ALVARO, A.; ALMEIDA, E. S.; PRADO, A. F. "MVCASE tool - working with design patterns". **Proceedings...** Latin American Conf. Pattern Lang. of Programming, 2003.

MAHMOUD, H. **Service-oriented architecture (SOA) and web services: The road to enterprise application integration (EAI)**. Sun Technical Articles, 2005. Disponível em: http://java.sun.com/developer/technicalArticles/WebService/soa/ (Acessado em 21 de junho de 2011).

MARKS, E. A.; BELL, M., 2006, "**Service-Oriented Architecture: a planning and implementation guide for business and techonology**", John Willey & Sons Inc.

MARIN, C.; LALANDA, P. "DoCoSOC- Domain Configurable Service-Oriented Computing", Services Computing, 2007. SCC 2007. **IEEE International Conference on ,** vol., no., p.52-59, 9-13 July 2007.

MEIJLER D. T.; KRUITHOF, G.; NICK, V. B. "**Top Down Versus Bottom Up in Service-Oriented Integration: An MDA-Based Solution for Minimizing Technology Coupling**". In ICSOC November 24 2006, pp. 484-489.

MELLOR, S. J.; CLARK, A. N.; FUTAGAMI, T. "Model-driven development", **IEEE Software**, v. 20, n. 5, 2003, pp. 14-18.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. **ACM Computing Surveys**, v. 37, n. 4, p. 316-344, dez. 2005. ISSN 0360-0300.

NEWCOMER, E. **Understanding Web Services - XML, WSDL, SOAP, and UDDI**. Addison Wesley, 2002.

MICROSOFT, **The Component Object Model specification**, October 1995.

MICROSYSTEMS, S. **Remote Method Invocation**. Disponível em: <http://java.sun.com/javase/6/docs/technotes/guides/rmi/index.html>. Acesso em: 20 Jun. 2011.

MILANOVIC, N.; MALEK, M. 2004. "Current Solutions for Web Service Composition". **IEEE Internet Computing**.

MORISIO, M.; EZRAN, M.; TULLY, C. **Success and Failure Factors in Software Reuse**, In: IEEE Transactions on Software Engineering, Vol. 28, No. 04, April, 2002, pp. 340-357.

MONTGOMERY, D. C. **Design and Analysis of Experiments**. 5 ed., Wiley, 2000.

NEIGHBORS, J. M. D. **A method for engineering reusable software systems**. In: Biggerstaff, T. J., Perlis, A. J. (ed). Software Reusability, Vol. 1, Concepts and Models. Addison-Wesley, 1989, pp. 295-319.

NG, W. S.; TEO, J.C.M.; ANG, W. T.; VISWANATHAN, S.; THAM, C. K. Experiences on developing SOA based mobile healthcare services. In: Services Computing Conference. APSCC. **IEEE Asia-Pacific**. 2009, pp. 498-501.

OASIS, **Advancing open standards for the information society**. Inc.: UDDI Version 3.0.2. 2004. Disponível em: <http://uddi.org/pubs/uddi_v3.htm>. Acesso em: 17 Junho 2011.

OMG. **Unified Modeling Language (OMG UML) Infrastructure Version 2.2**. 2009. Disponível em: <http://www.omg.org/>. Acesso em: Jun. 2011.

OMG, MOF 2.0/XMI Mapping Version 2.1.1. 2007. Disponível em: <http://www.omg.org/>. Acesso em: Jun. 2011

PAPAZOGLOU M. P.; GEORGAKOPOULOS D. "**Service-oriented computing: Introduction**". Communications of the ACM, 46(10): pp. 24-28, Outubro 2003.

PAPAZOGLOU, M. P.; HEUVEL; WILLEM-JAN, 2007. "**Service oriented architectures: approaches, technologies and research issues**", VLDB Journal, Springer-Verlag.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. **Restful web services vs. "big"' web services: making the right architectural decision**. In: Proceeding of the 17th international conference on World Wide Web, WWW '08, New York, NY, USA: ACM, 2008, pp. 805-814.

PAUTASSO, C. **Composing restful services with jopera**. In: Proceedings of the 8th International Conference on Software Composition, SC '09, Berlin, Heidelberg: Springer-Verlag, 2009, pp. 142-159. Disponível em http://dx.doi.org/10.1007/978-3-642-02655-3_11.

PAUTASSO, C. **Bpel for rest**. In: Proceedings of the 6th International Conference on Business Process Management, BPM '08, Berlin, Heidelberg: Springer-Verlag, 2008, pp. 278-293.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach**. 6 ed., McGraw-Hill Science, 2004.

RAUF, I.; RUOKONEN, A.; SYSTA, T.; PORRES, I. **Modeling a composite restful web service with uml**. In: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, ECSA '10, New York, NY, USA: ACM, 2010, pp. 253-260.

RICHARDSON, L.; RUBY, S. **Restful web services**. O'Reilly Media, 2007.

RINE, D.C. **Success Factors for software reuse that are applicable across Domains and businesses**, In: ACM Symposium on Applied Computing, San Jose, California, USA, ACM Press, March, 1997, pp. 182-186.

ROSS, D. T. Structured analysis: a language for communicating ideas. **IEEE Transactions on Software Engineering**, v. 3, n. 1, p. 16-34, 1977.

ROTHENBERGER, M.A.; DOOLEY, K.J.; KULKARNI, U.R.; NADA, N. **Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices**, In: IEEE Transactions on Software Engineering, Vol. 29, No. 09, September, 2003, pp. 825-837.

SADILEK, D. A. "**Prototyping Domain-Specific Language Semantics**", In: Proceedings of the Companion to the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications. Orlando, Flórida: ACM, 2008, pp. 895-896.

SANTANA, E. Z. et. al. "**Modelagem Específica de Domínio em Linhas de Produto de Software na computação ubíqua**". In: III Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS), Natal, 2009.

SATYANARAYANAN, M.; Pervasive Computing: Vision and Challenges. In. **IEEE Personal Communications**, pp. 10-17, Vol. 8, August 2001.

SCHAUERHUBER, A.; WIMMER, M.; KAPSAMMER, E. **Bridging existing web modeling languages to model-driven engineering: a metamodel for webml**. In: Workshop proceedings of the sixth international conference on Web engineering, ICWE '06, New Yourk, NY, USA: ACM, 2006 (ICWE '06).

SCHMIDT, D. C. Guest editor's introduction: Model-driven engineering. **IEEE Computer**, v. 39, n. 2, p. 22-32, 1994.

SINHA, P. K. **Remote Procedure Calls**. In: ANDERSON, J. B. (Ed.). Distributed Operating Systems: Concepts and Design. New York: IEEE Press, 1997. p. 742. cap. 4.

STEELE, S. B. **Emergency Dispatching: A Medical Communicator's Guide**. Prentice Hall; Facsimile edition (August 28, 1992).

STEINBERG, D.; BUDINSKY, F.; PATERNOSTRO, M.; MERKS, E. **EMF – Eclipse Modeling Framework**, Addison-Wesley, 2008.

SUN Developer Network. **The Source for Java Developers**. Disponível em: <http://java.sun.com/>. Acesso em: 23 mar 2010.

THIES, G.; VOSSEN, G. **Modelling web-oriented architectures**. In: Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling - Volume 96, APCCM '09, Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2009, pp. 97-106 (APCCM '09).

THOMAS, D. MDA: Revenge of the modelers or uml utopia? **IEEE Software**, v. 21. n.3. p. 15-17, 2004.

THURSTON A.; CORDY, J.R. "Evolving TXL", **Proceedings…** SCAM 2006 - IEEE 6th International Workshop on Source Code Analysis and Manipulation, Philadelphia, pp. 117-126, 2006

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. A. G.. **Introdução à Engenharia de Software Experimental**, Relatório Técnico RT-ES-590/02, Programa de Engenharia de Sistemas e Computação, UFRJ, 2002.

VAN DEURSEN, A.; KLINT, P.; VISSER, J. "**Domain-specific languages: an annotated bibliography**", ACM SIGPLAN Notices, Junho 2000, pp. 26-36.

VÖLTER, M. **MD* Best Practices**. Dezembro 2008. Disponível on-line em: <http://www.voelter.de/>. Acesso em: 29 Jun 2011.

VÖLTER, M.; GROHER, I. Product line implementation using aspect-oriented and model-driven software development. In: Proceedings of the 11th International Software Product Line Conference, 2007, Kyoto, Japan. **Proceedings…** IEEE Computer Society, 2007. p. 233-242.

WALSH, A. E., editor. **UDDI, SOAP, and WSDL: The Web Services Specification** Reference Book. Prentice Hall, 2002.

WANG, H.; ZHANG, B.J.; LIU, X.Z.; LUO, D.Z.; ZHONG, S.B. A Novel WebService Architecture Based on REST. In. **Journal Advanced Materials Research**, pp. 143-144, 2010.

WONG, E. Y. C.; CHAN A. T. S.; LEONG, H. V. Xstream: A Middleware for Streaming XML Contents over Wireless Environments. **IEEE Trans. Softw. Eng**. 30, pp. 918-935, 2004.

WOHLIN, C.; RUNESON, P.; HOST, M.; OHLSSON, M.; REGNELL, B.; WESSLEN, A. **Experimentation in Software Engineering: an introduction**, Kluwer Academic Publishers, 2000.

W3C. **Web Services Architecture**. 2004a. Disponível em: <http://www.w3.org/TR/ws-arch> Acesso em: 20 Jun 2011.

W3C. **Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language**. 2007b. Disponível em: <http://www.w3.org/TR/wsdl20>. Acesso em: 20 Jun 2011.

W3C. **Simple Object Access Protocol (SOAP) Version 1.2 Part 1: Messaging Framework (Second Edition)**. 2007c, Disponível em: <http://www.w3.org/TR/soap12-part1/>. Acesso em: 20 Jun 2011.

Zelkowitz, M.; Wallace, D. "Experimental Models for Validating Technology", **IEEE Computer**, Vol. 31, N. 5, pp. 23-31, 1998.

Apêndice A

FORMULÁRIO DE CARACTERIZAÇÃO DO PARTICIPANTE

Este formulário tem por objetivo caracterizar sua experiência acadêmica, pessoal e profissional em relação à Ciência da Computação. Por favor, responda a TODAS as questões o mais fielmente possível. Toda informação fornecida é confidencial, sendo que seu nome e quaisquer outros meios de identificação não serão divulgados em nenhuma hipótese.

1) Dados do participante

Registro acadêmico: _____

Idade: ____

2) Formação acadêmica

( ) Graduação

( ) Mestrado

( ) Doutorado

Ano de Ingresso: _____ Mês/Ano de Conclusão (ou previsão de conclusão): ____/_____

3) Experiência profissional

3.1) Assinale a opção que melhor reflita seu grau atual de experiência com as tecnologias listadas a seguir, considerando a escala de 5 pontos:

0 = nenhum

1 = estudei em aula ou em livro

2 = pratiquei em projetos em sala de aula

3 = usei em projetos pessoais ou na indústria

4 = uso em grande parte dos projetos que realizo

3.1.1) Linguagem de Programação Java	0	1	2	3	4
3.1.2) Biblioteca JAX-RS	0	1	2	3	4
3.1.3) Biblioteca XStream	0	1	2	3	4
3.1.4) Unified Modeling Language (UML)	0	1	2	3	4
3.1.5) JavaScript Object Notation (JSON)	0	1	2	3	4
3.1.6) Banco de Dados MySQL	0	1	2	3	4
3.1.7) IDE Eclipse	0	1	2	3	4

3.1.8) Java Emitter Templates (JET)	0	1	2	3	4
3.1.9) Eclipse Modeling Framework (EMF)	0	1	2	3	4
3.1.10) Android SDK	0	1	2	3	4
3.1.11) Restful	0	1	2	3	4

3.2) Qual das opções a seguir melhor descreve sua experiência anterior em relação ao desenvolvimento de software na prática?

Tenho desenvolvido software por conta própria (sozinho) ( )	Tenho desenvolvido software como membro de equipe durante cursos que realizo ( )	Tenho desenvolvido software como membro de equipe, na indústria, há menos de 2 anos ( )	Tenho desenvolvido software como membro de equipe, na indústria, há 2 anos ou mais ( )

3.2.1) Por favor, detalhe sua resposta indicando o número de meses de experiência relevantes tanto em projetos desenvolvidos durante algum curso, como em projetos desenvolvidos na indústria.

a) Quantos meses de experiência você teve como gerente de projeto de software:

Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

b) Quantos meses de experiência você teve como analista de sistemas:

Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

c) Quantos meses de experiência você teve como desenvolvedor (programação):

Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

3.2.2) Como você distribuiria o tempo gasto em sua experiência com <u>programação</u>?

_____ % em esforço individual (sozinho)

_____ % em desenvolvimento conjunto com outras pessoas (equipe)

_____ % na supervisão de outros programadores (gerência)

100% ← TOTAL

3.3) Qual sua habilidade em...

a) ... desenvolver software com o Android SDK?

( ) Especialista ( ) Avançado ( ) Médio ( ) Básico ( ) Nenhum

Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

b) ... desenvolver aplicativos com Serviços Web (WebServices)?

( ) Especialista ( ) Avançado ( ) Médio ( ) Básico ( ) Nenhum

Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

c) ... modelar aplicativos com UML?

( ) Especialista ( ) Avançado ( ) Médio ( ) Básico ( ) Nenhum

Meses de experiência: ____ (como aluno em um curso) ____ (na indústria)

4) Experiência com os conceitos e técnicas empregados no experimento

Esta seção será utilizada para compreender sua familiaridade com os conceitos e técnicas que serão utilizados nas atividades do experimento.

4.1) Assinale na tabela a seguir a opção que melhor reflita seu grau atual de experiência com os itens relacionados, considerando a escala de 4 pontos:

0 = Eu não tenho familiaridade com este assunto. Eu nunca fiz isto.

1 = Já li ou estudei sobre isto. Conheço os conceitos e/ou técnicas.

2 = Eu utilizo isto algumas vezes em projetos pessoais ou na indústria, mas não sou um(a) especialista.

3 = Eu sou muito familiar com este assunto. Eu me sentiria confortável fazendo isto.

4.1.1)	Desenvolvimento Dirigido a Modelos (MDD)	0	1	2	3
4.1.2)	Desenvolvimento de Web Services	0	1	2	3
4.1.3)	Desenvolvimento de aplicações para dispositivos móveis com o Android SDK	0	1	2	3

Obrigado por sua colaboração!.

Apêndice B

FORMULÁRIO DE CONSENTIMENTO

Experimento

Esse experimento visa avaliar a aplicação da abordagem para desenvolvimento de aplicações móveis que enfatiza o reúso de software com base na *Modelagem Específica de Domínio (DSM)* e *Arquitetura Orientada a Serviços (SOA)* para apoio à construção de aplicações móveis que enfatiza o reúso de software.

Idade

Eu declaro ser maior que 18 (dezoito) anos de idade e concordar em participar do experimento conduzido por Alexandre Bellini na Universidade Federal de São Carlos (UFSCar).

Procedimento

Esse experimento ocorrerá em uma única sessão, que incluirá o desenvolvimento de aplicativos móveis e serviços para o domínio *Healthcare*. O desenvolvimento das aplicações e serviços se dará com ou sem a aplicação da abordagem para desenvolvimento de aplicações móveis que enfatiza o reúso de software com base na DSM e SOA, conforme determinado pelo experimentador. Eu entendo que, uma vez que o experimento tenha sido concluído, os trabalhos que desenvolvi, bem como os dados coletados, serão estudados visando analisar a aplicação dos procedimentos e técnicas propostos.

Confidencialidade

Estou ciente de que toda informação coletada nesse experimento é confidencial, e meu nome ou quaisquer outros meios de identificação não serão divulgados. Da mesma forma, me comprometo a não comunicar meus resultados aos demais participantes e/ou a outros grupos enquanto não terminar o experimento, bem como manter sigilo das técnicas e documentos apresentados que fazem parte do experimento.

Benefícios e liberdade de desistência

Eu entendo que os benefícios que receberei desse experimento se limitam ao aprendizado do material que é distribuído e apresentado. Eu compreendo que sou livre para realizar perguntas a qualquer momento, solicitar que qualquer informação relacionada à minha pessoa não seja incluída no experimento, ou comunicar minha desistência de participação. Eu entendo que participo livremente com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

Pesquisador responsável

Alexandre Bellini

Programa de Pós-Graduação em Ciência da Computação – PPG-CC/DC/UFSCar

Professor responsável
Prof. Dr. Antonio Francisco do Prado
Programa de Pós-Graduação em Ciência da Computação – PPG-CC/DC/UFSCar

Ao preencher e assinar este formulário dou plena ciência e consentimento com os termos acima expostos.

Nome (em letra de forma): _____ **RA:** _____

Assinatura: _____

Data: ____/_____/_____

Apêndice C

DESCRIÇÃO DA TAREFA – GRUPO ABORDAGEM DE REÚSO COM DSM E SOA

Instruções

Este experimento avaliará a aplicação da abordagem de reúso utilizando a *Modelagem Específica de Domínio (DSM)* e a *Arquitetura Orientada a Serviços (SOA)* para o desenvolvimento de aplicações móveis para o Domínio de Cuidados de Saúde (*Healthcare*). A análise dos resultados será baseada nos dados coletados durante a execução das atividades propostas no experimento. Assim, de forma que o experimentador possa melhor avaliar os resultados obtidos, nos formulários entregues ao seu grupo preencha corretamente todos os dados relacionados ao horário de início e término de cada atividade, bem como os dados relacionados ao número de linhas de código implementadas.

Da mesma maneira, caso encontre algum problema de ordem técnica durante a execução de determinada atividade, como configuração do IDE, configuração de máquina, etc., anote os horários de identificação e solução do problema juntamente com sua descrição no formulário apropriado.

Pergunte e comente tudo o que julgar necessário.

Tarefa

Você recebeu a descrição de uma aplicação móvel para o Domínio de *Healthcare*. Uma vez que o foco da abordagem avaliada está no reúso dos serviços, as regras de negócio da aplicação a ser desenvolvida e suas interfaces gráficas já encontram-se implementadas. **A tarefa de seu grupo é desenvolver serviços utilizando RESTful e a aplicação móvel indicada para o seu grupo para consumir os serviços usando o *Android SDK*. Para tanto, construa os modelos dos serviços e da aplicação, e em seguida gere o código parcial com o uso das transformações. Complemente o código gerado até a finalização da aplicação.**

Utilize o material de apoio para auxiliá-lo durante a construção dos serviços e da aplicação móvel. Importe para o seu ambiente de desenvolvimento o projeto dos serviços e da aplicação que já estão parcialmente implementados. Tendo que os requisitos da aplicação já foram identificados, inicie a execução da abordagem a partir da disciplina *Projeto*. Para cada atividade realizada anote os dados temporais e de linhas de código no formulário apropriado.

No fim do experimento, compacte o projeto dos serviços e da aplicação móvel desenvolvida e envie por email para alexandrebli@gmail.com, usando como assunto o seguinte padrão: *"GRUPO [Nº de seu grupo] - Experimento"*.

Apêndice D

DESCRIÇÃO DA TAREFA – GRUPO ABORDAGEM TRADICIONAL

Instruções

Este experimento avaliará a aplicação da abordagem de reúso utilizando a *Modelagem Específica de Domínio (DSM)* e a *Arquitetura Orientada a Serviços (SOA)* para o desenvolvimento de aplicações móveis para o Domínio de Cuidados de Saúde (*Healthcare*). A análise dos resultados será baseada nos dados coletados durante a execução das atividades propostas no experimento. Assim, de forma que o experimentador possa melhor avaliar os resultados obtidos, nos formulários entregues ao seu grupo preencha corretamente todos os dados relacionados ao horário de início e término de cada atividade, bem como os dados relacionados ao número de linhas de código implementadas.

Da mesma maneira, caso encontre algum problema de ordem técnica durante a execução de determinada atividade, como configuração do IDE, configuração de máquina, etc., anote os horários de identificação e solução do problema juntamente com sua descrição no formulário apropriado.

Pergunte e comente tudo o que julgar necessário.

Tarefa

Você recebeu a descrição de uma aplicação móvel para o Domínio de *Healthcare*. Uma vez que o foco da abordagem avaliada está no reúso dos serviços, as regras de negócio da aplicação a ser desenvolvida e suas interfaces já encontram-se implementadas. **A tarefa de seu grupo é desenvolver serviços utilizando RESTful e a aplicação móvel indicada para o seu grupo para consumir os serviços usando o *Android SDK*, seguindo as disciplinas tradicionais de *Análise*, *Projeto*, *Implementação* e *Testes* da Engenharia de Software.**

Utilize o material de apoio para auxiliá-lo durante a construção dos serviços e da aplicação móvel. Importe para o seu ambiente de desenvolvimento o projeto dos serviços e da aplicação que já estão parcialmente implementados. Tendo que os requisitos da aplicação já foram identificados, inicie a execução da abordagem a partir da disciplina *Projeto*. Para cada atividade realizada anote os dados temporais e de linhas de código no formulário apropriado.

No fim do experimento, compacte o projeto dos serviços e da aplicação móvel desenvolvida e envie por email para alexandrebli@gmail.com, usando como assunto o seguinte padrão: *"GRUPO [Nº de seu grupo] - Experimento"*.

Apêndice E

DESCRIÇÃO DA APLICAÇÃO A

Descrição da Aplicação - Aplicação A

A *Aplicação A* é uma aplicação que permite a **visualização de ocorrências de atendimentos emergenciais** previamente cadastradas na base de dados da unidade de atendimentos de emergência de uma determinada cidade. A Estação de Operação X da unidade, ao receber uma chamada de ocorrência, realiza o registro da ocorrência, armazenando-o na base de dados. A ocorrência deverá conter a data, os horários de saída e chegada ao local da ocorrência, o número de vítimas, a gravidade da ocorrência, a descrição do fato ocorrido, e o motivo da ocorrência. Os funcionários da unidade de atendimentos de emergência, responsáveis pela execução do atendimento, muitas vezes necessitam conhecer os detalhes das ocorrências atuais enquanto dirigem-se para o atendimento, de forma que possam se planejar apropriadamente durante o caminho e chegar ao local da emergência com toda a instrumentação necessária preparada para socorrer a vítima. Contudo, na maioria das vezes, conforme a gravidade da situação, a fim de assegurar o sucesso do atendimento, esses funcionários necessitam partir para o atendimento tão logo fiquem sabendo do local da emergência sem mesmo aguardar o término da chamada recebida pelo operador. Logo, a necessidade vigente é a disponibilização dos detalhes das ocorrências por meio de dispositivos móveis aos funcionários, de maneira que estes possam se preparar adequadamente durante o percurso. Esses funcionários, munidos de um dispositivo móvel, devem visualizar a ***lista com o título das ocorrências***. Essa aplicação ainda disponibiliza funcionalidades para ***listar a(s) vítima(s)*** em situação de emergência, de forma que possam conhecer com antecedência o perfil da vítima. Com isso, os funcionários atendentes podem visualizar e buscar, por meio de um dispositivo móvel, as informações sobre o atendimento e

da(s) vítimas(s) de qualquer lugar e a qualquer momento, auxiliando na prestação dos serviços atendimento de emergência.

Apêndice F

MATERIAL DE APOIO DA APLICAÇÃO A – GRUPO ABORDAGEM DE REÚSO COM DSM E SOA

I - DESENVOLVIMENTO DO SERVIÇO

1. **<u>Importar o projeto parcial do Web Service no Eclipse</u>**

   Importe o projeto *wsNurse* que acompanha este material. Siga os seguintes passos:

1. No IDE Eclipse vá no menu *File* e escolha a opção *Import*...
2. Na janela que se abre, vá à categoria *General* e selecione a opção *Existing Projects Into Workspace*. Pressione o botão *next*.
3. Na próxima janela, clique no botão *Browse* e selecione a pasta raiz do projeto *wsAmbulance* salvo em seu computador.
4. Nessa mesma janela selecione a opção *Copy Projects Into Workspace*.
5. Finalmente, pressione o botão *Finish*. O IDE Eclipse importará o projeto Web de nome *wsAmbulance* ao *workspace* atual.
6. Corrija quaisquer erros de importação de bibliotecas. Clique com o botão direito sobre a pasta raiz do projeto. Selecione a opção Build *Path > Configure Build Path...* Na janela que se abre remova as bibliotecas conflitantes e adicione novamente (Bibliotecas com possíveis erros: *Apache TomCat, MySQL*, biblioteca do *RESTful*).

   A estrutura do projeto importado será como a **Figura F.1** abaixo:

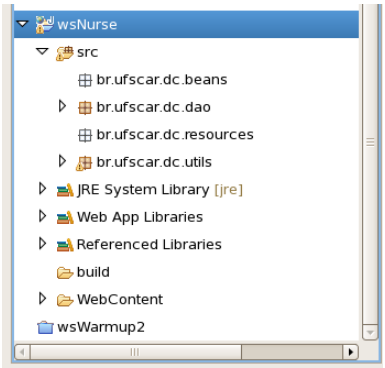

Figura F.1 - Estrutura do projeto importado

2. <u>Criar o banco de dados acessado pelo Serviço</u>

Antes de iniciar a implementação do serviço, crie o banco de dados que será utilizado:

1. Na plataforma Windows, para acessar o terminal do MySQL e criar o banco, deve-se ir em Iniciar *> Programas > MySQL > MySQL Server 5.x > MySQL Command Line Client*.
2. Com o terminal aberto, entre com o seguinte comando: ***mysql –u [usuário_root] –p [senha_usuário-root]***
3. Em seguida, cole o *script* SQL que se encontra no arquivo Nurse_BD.sql que acompanha este material.

As regras de negócio para manipulação do banco de dados já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.dao* do projeto importado*.* As funcionalidades para manipulação e conversão de objetos JSON e XML encontram-se prontas no pacote *br.ufscar.dc.utils.*

Na classe ***FactoryConnection*** do pacote *br.ufscar.dc.dao*, altere as variáveis ***USER*** e ***PASSWORD***, atribuindo, respectivamente, os dados de usuário e senha locais para acesso ao banco de dados. <u>Exemplo</u>: USER = “root”; PASSWORD=”123456”;.

3. <u>Modelar o Serviço</u>

Para criar o modelo da aplicação, utilize o *plugin* editor de modelos construído na etapa de Engenharia de Domínio da abordagem Reúso com DSM e SOA:

1. Na pasta raiz do projeto *wsNurse* crie uma pasta chamada “*Modelo*” (clique com o botão direito sobre a pasta raiz, opção *New > Folder*)
2. Clique com o botão direito sobre a pasta *Modelo* recém-criada e selecione a opção *New > Other...*
3. Na janela que se abre, selecione a opção *EmergencyMetamodel Model* da categoria *Example EMF Model Creation Wizards*. Em seguida clique em *Next*.
4. Na próxima janela, nomeie o modelo com o nome *ws_model.emergencymetamodel* Clique em *Next*.
5. Na janela seguinte escolha a opção **Model** para o campo *Model Object*. Em seguida pressione o botão *Finish*. O modelo será criado, tendo como nó raiz o elemento *Model*.
    a. Clique com o botão direito do mouse sobre o elemento *Model* do modelo criado e selecione a opção “Show properties view”. Na caixa de propriedades do elemento *Model*, atribua o nome *wsNurse* para a propriedade “*Project”,* que indica a que projeto o modelo está vinculado.
    b. Clique novamente com o botão direito sobre o elemento *Model*, escolha a opção New Child. Em seguida selecione o elemento *Object* que aparece no menu popup que se abre. Será criado um novo elemento *Object* no modelo, filho do elemento *Model*.
    c. Agora, clique sobre o elemento Object e adicione o elemento *Nurse*, atribua os seguintes valores:
        i. Package: br.ufscar.dc.beans;
        ii. Stereotype: bean.
    d. Adicione um elemento *Attribute* dentro do elemento *Nurse*. Na barra de propriedades, atribua os seguintes valores:
        i. Attribute: age.

e. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “Integer”.

f. Adicione um elemento *Attribute* dentro do elemento *Nurse*. Na barra de propriedades, atribua os seguintes valores:

   i. Attribute: gender.

g. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “String”.

h. Adicione um elemento *Attribute* dentro do elemento *Nurse*. Na barra de propriedades, atribua os seguintes valores:

   i. Attribute: dateBirth.

i. Adicione um elemento Attribute Type dentro do elemento Attribute. Na barra de propriedades, selecione o valor “Java.sql.Date”.

j. Agora, adicione um elemento *Service* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:

   i. Name: NurseResource;

   ii. Package: *br.ufscar.dc.resources;*

   iii. *Path: contentNurse*

   iv. *Protocol: Rest.*

k. Adicione um elemento *Service Input* ao elemento *Service*. Dentro do elemento *Service Input* recém-criado, adicione um elemento *Service Data*. Na propriedade “Mimetype” deste último, selecione o valor “application/json”.

l. Adicione um elemento *Service Output* ao elemento *Service*. Dentro do elemento *Service Output* recém-criado, adicione um elemento *Service Data*. Na propriedade “Mimetype” deste último, selecione o valor “application/json”.

m. Adicione um elemento *Method* Service ao elemento *Service*. Preencha as propriedades deste elemento da seguinte forma:

   i. Method: listNurses;

   ii. Path: listNurses;

   iii. Type: retrieve.

n. Adicione um elemento Return Method ao elemento Method Service. Na propriedade “Return” deste último selecione o valor “Response”.

4. <u>Executar as Transformações Modelo para Código</u>

Dica Importante: sempre salve todas as alterações no modelo antes de executar as transformações, de forma que essas alterações sejam refletidas no código gerado.

1. Clique com o botão direito sobre o modelo do serviço e selecione a opção *Run As > Run Configurations*.
2. Na caixa do lado esquerdo da janela que se abre, dê um clique duplo sobre a opção *JET Transformation*. Do lado direito aparecerão as opções para execução de transformações.
3. No campo *Transformation Input*, clique sobre o botão *Browse...* e selecione o arquivo do modelo do serviço.

4. No campo *ID* selecione a transformação que deseja aplicar sobre o modelo (*br.ufscar.dc.helth.codegeneratorJEEWS*) e em seguida pressione o botão *Run*. A transformação será executada, sendo o código gerado dentro do projeto *wsNurse* (conforme definido na propriedade "Project" do elemento Model).

5. Implementar o Serviço

1. A classe implementada deverá ficar semelhante à mostrada na Figura F.2.

**Ao criar os métodos e atributos indicados neste tutorial, assegurem-se de obedecer ao padrão utilizado na nomenclatura adotada, tais como letras maiúsculas, minúsculas, etc.*

```
import java.sql.ResultSet;
@Path("/contentNurse")
@Consumes( { "application/json" })
@Produces( { "application/json" })
public class NurseResource {

    @GET
    @Path("/listNurses")
    public Response listNurses() {
        NurseDAO nurseDAO = new NurseDAO();
        ResultSet rs = nurseDAO.listNurses();
        List<Nurse> listNurses = new ArrayList<Nurse>();
        try {
            while (rs.next()) {
                Nurse n = new Nurse();
                n.setId(rs.getInt(1));
                n.setName(rs.getString(2));
                n.setAge(rs.getInt(3));
                n.setGender(rs.getString(4));
                n.setDateBirth(rs.getDate(5));
                listNurses.add(n);
            }
            return Response.ok(
                    XMLParser.parseObjectToJSON("listNurses", listNurses))
                    .build();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return Response.ok(XMLParser.parseObjectToJSON("listNurses", null))
                .build();
    }
}
```

Figura F.2 - Código da Classe NurseResource

6. Testar o Serviço

1. Após a implementação do serviço, teste-o no browser. Para isto, clique com o botão direito sobre a pasta raiz do projeto wsNurse e escolha a opção *Run As… > Run on Server.*
2. O browser será aberto. Teste a operação para listar as enfermeiras registradas no banco de dados, digitando a seguinte URL: http://localhost:8080/wsNurse/resources/contentNurse/listNurses

II - DESENVOLVIMENTO DA APLICAÇÃO MÓVEL

1. Importar o projeto parcial da aplicação móvel no Eclipse

1. Importe o projeto *AndroExpA* que acompanha este material. Siga os mesmos passos do passo "Importar o projeto parcial do Web Service no Eclipse" apresentado anteriormente.
2. Corrija quaisquer erros de importação de bibliotecas referentes à SDK do Android. Vá ao menu *Window > Preferences*. Na janela que se abre, selecione na parte esquerda a opção "Android". Caso o campo SDK Location esteja vazio, clique no botão "Browse…" e indique a

localização da pasta “android-sdk” que se encontra no diretório do Eclipse. Após selecionar a pasta, clique em OK.

A estrutura do projeto importado será como a **Figura F.3** abaixo:

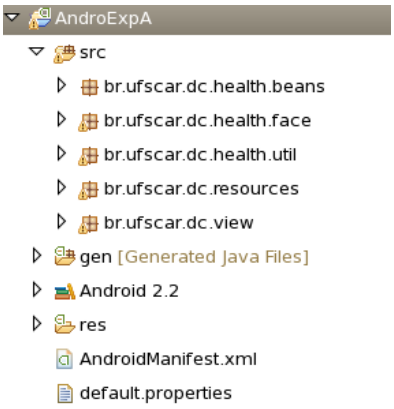

Figura F.3 - Estrutura do projeto Android importado

As interfaces gráficas da aplicação móvel já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.health.face* e *br.ufscar.dc.view* do projeto importado. As funcionalidades para formatação de objetos de dados do tipo Date encontram-se prontas no pacote *br.ufscar.dc.health.util.*

2. <u>Modelar o Serviço</u>

Para criar o modelo da aplicação, utilize o *plugin* editor de modelos construído na etapa de Engenharia de Domínio da abordagem Reúso com DSM e SOA:

6. Na pasta raiz do projeto ***AndroExpA*** crie uma pasta chamada “*Modelo*” (clique com o botão direito sobre a pasta raiz, opção *New > Folder*)
7. Clique com o botão direito sobre a pasta *Modelo* recém-criada e selecione a opção *New > Other...*
8. Na janela que se abre, selecione a opção *EmergencyMetamodel Model* da categoria *Example EMF Model Creation Wizards*. Em seguida clique em *Next*.
9. Na próxima janela, nomeie o modelo com o nome *android_model.emergencymetamodel* Clique em *Next*.
10. Na janela seguinte escolha a opção **Model** para o campo *Model Object*. Em seguida pressione o botão *Finish*. O modelo será criado, tendo como nó raiz o elemento *Model*.
    a. Clique com o botão direito do mouse sobre o elemento *Model* do modelo criado e selecione a opção “Show properties view”. Na caixa de propriedades do elemento *Model*, atribua o nome *AndroExpA* para a propriedade “*Project”,* que indica a que projeto o modelo está vinculado.
    b. Clique novamente com o botão direito sobre o elemento *Model*, escolha a opção New Child. Em seguida selecione o elemento *Object* que aparece no menu popup que se abre. Será criado um novo elemento *Object* no modelo, filho do elemento *Model*.
    c. Adicione um elemento *Occurrence* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:
        i. Package: br.ufscar.dc.health.beans;
        ii. Stereotype: bean.

d. Adicione um elemento *Class* e atribua os seguintes valores:
    i. Name: Victim;
    ii. Package: br.ufscar.dc.health.beans;
    iii. Stereotype: bean.

e. Adicione um elemento *Attribute* dentro do elemento *Class*. Na barra de propriedades, atribua os seguintes valores:
    i. Attribute: id.

f. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “Integer”.

g. Adicione um elemento *Attribute* dentro do elemento *Class*. Na barra de propriedades, atribua os seguintes valores:
    i. Attribute: name.

h. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “String”.

i. Adicione um elemento *Attribute* dentro do elemento *Class*. Na barra de propriedades, atribua os seguintes valores:
    i. Attribute: age.

j. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “Integer”.

k. Adicione um elemento *Attribute* dentro do elemento *Class*. Na barra de propriedades, atribua os seguintes valores:
    i. Attribute: anammesis.

l. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “String”.

m. Clique novamente com o botão direito sobre o elemento *Model*, escolha a opção New Child. Em seguida selecione o elemento *Reuse Set* que aparece no menu popup que se abre. Será criado um novo elemento chamado *Reuse Set* no modelo, filho do elemento *Model*.
    i. O elemento Reuse Set contém os serviços juntamente com seus métodos do Domínio Healthcare para reúso pela aplicações.

n. Adicione um elemento *Class* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:
    i. Name: OccurrenceResource;
    ii. Package: br.ufscar.dc.resources;
    iii. Stereotype: reuse.

o. Adicione um elemento *Method* dentro do element *Class* recém criado. Na barra de propriedades, atribua os seguintes valores:
    i. Has Operation Service: Listoccurrences listOccurrences;
    ii. Has Service Reuse: Occurrenceresource OccurrenceResource;
    iii. Method: listOccurrences;

iv. Stereotype: reuse.

p. Adicione um elemento *Return Type* dentro do elemnto Method recém criado. Na barra de propriedades, selecione para a propriedade *ReturnType* o valor “String”.

q. Adicione um elemento *Method* dentro do element *Class*. Na barra de propriedades, atribua os seguintes valores:

i. Has Operation Service: Listvictimstooccurrence listVictimsToOccurrence;

ii. Has Service Reuse: Occurrenceresource OccurrenceResource;

iii. Method: listVictimsToOccurrence;

iv. Stereotype: reuse.

r. Adicione um elemento *Return Type* dentro do elemnto Method recém criado. Na barra de propriedades, selecione para a propriedade *ReturnType* o valor “String”.

s. Adicione um elemento *Parameter* dentro do elemnto Method recém criado. Na barra de propriedades, atribua o seguinte valore:

i. Parameter: idOccurrence

t. Adicione um elemento *Parameter Type* dentro do elemnto *Parameter* recém criado. Na barra de propriedades, selecione para a propriedade *Parametertype* o valor “Integer”.

3. <u>Executar as Transformações Modelo para Código</u>

Dica Importante: sempre salve todas as alterações no modelo antes de executar as transformações, de forma que essas alterações sejam refletidas no código gerado.

5. Clique com o botão direito sobre o modelo do serviço e selecione a opção *Run As > Run Configurations*.
6. Na caixa do lado esquerdo da janela que se abre, dê um clique duplo sobre a opção *JET Transformation*. Do lado direito aparecerão às opções para execução de transformações.
7. No campo *Transformation Input*, clique sobre o botão *Browse...* e selecione o arquivo do modelo do serviço (*android_model.emergencymetamodel*).
8. No campo *ID* selecione a transformação que deseja aplicar sobre o modelo (*br.ufscar.dc.helth.generatorAndroid2*) e em seguida pressione o botão *Run*. A transformação será executada, sendo o código gerado dentro do projeto AndroExpA (conforme definido na propriedade “Project” do elemento *Model*).

A classe *OccurrenceResource* implementada deverá ficar semelhante à mostrada na Figura F.4.

```
public class OccurrenceResource {
    public String listOccurrences() {
        DefaultHttpClient httpclient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentOccurrence/"
                        + "listOccurrences");
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler responseHandler = new BasicResponseHandler();
        try {
            return httpclient.execute(httpGet, responseHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
    public String listVictimsToOccurrence(Integer idOccurrence) {
        DefaultHttpClient httpclient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentOccurrence/"
                        + "listVictimsToOccurrence" + idOccurrence);
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler responseHandler = new BasicResponseHandler();
        try {
            return httpclient.execute(httpGet, responseHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Figura F.4 - Código da Classe OccurrenceResource

No pacote *br.ufscar.dc.resources* abra a classe NurseResource, na linha:

```
HttpGet httpGet = new HttpGet("");
```

a. Adicione a URL do serviço criado no projeto wsNurse;

4. <u>Testar a aplicação móvel com o emulador do Android</u>

1. Após a implementação da aplicação, teste-a no emulador do Android. Para isto, clique com o botão direito sobre a pasta raiz do projeto AndroExpA e escolha a opção *Run As… > Android Application.*

    a. Caso ainda não exista um dispositivo emulador configurado, o Eclipse emitirá um alerta para a criação de um novo *Android Virtual Device (ADV)*. Neste caso clique em Yes.

    b. Na janela que se abre, clique em “New” para criar um novo ADV. Em seguida dê um nome qualquer para este dispositivo. No campo “Target” selecione “Android 2.2 – APILevel8”. Mantenha as demais configurações como estão e clique no botão “Create ADV”.

    c. Feche todas as janelas referentes à criação do ADV. Em seguida, clique novamente com o botão direito sobre a pasta raiz do projeto AndroExpA e escolha a opção *Run As… > Android Application.* Desta vez, o Eclipse executará a aplicação no ADV recém-criado.

2. Teste a aplicação, verifique se a mesma está funcionando corretamente, trazendo a lista de ocorrências, vítimas e enfermeiras(os).

Apêndice G

MATERIAL DE APOIO DA APLICAÇÃO A – GRUPO ABORDAGEM TRADICIONAL

I - DESENVOLVIMENTO DO SERVIÇO

7. <u>Importar o projeto parcial do Web Service no Eclipse</u>

Importe o projeto *wsNurse* que acompanha este material. Siga os seguintes passos:

7. No IDE Eclipse vá no menu *File* e escolha a opção *Import*...
8. Na janela que se abre, vá à categoria *General* e selecione a opção *Existing Projects Into Workspace*. Pressione o botão *next*.
9. Na próxima janela, clique no botão *Browse* e selecione a pasta raiz do projeto *wsAmbulance* salvo em seu computador.
10. Nessa mesma janela selecione a opção *Copy Projects Into Workspace*.
11. Finalmente, pressione o botão *Finish*. O IDE Eclipse importará o projeto Web de nome *wsAmbulance* ao *workspace* atual.
12. Corrija quaisquer erros de importação de bibliotecas. Clique com o botão direito sobre a pasta raiz do projeto. Selecione a opção Build *Path > Configure Build Path...* Na janela que se abre remova as bibliotecas conflitantes e adicione novamente (Bibliotecas com possíveis erros: *Apache TomCat, MySQL*, biblioteca do *RESTful*).

A estrutura do projeto importado será como a **Figura G.1** abaixo:

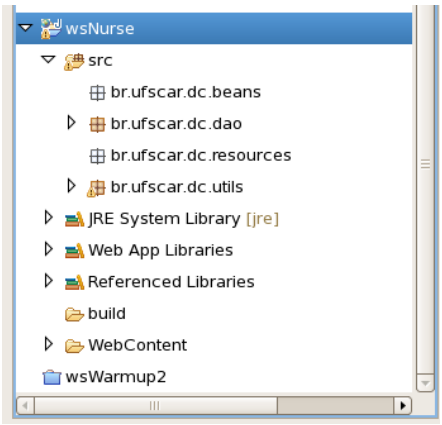

Figura G.1 - Estrutura do projeto importado

8. <u>Criar o banco de dados acessado pelo Serviço</u>

Antes de iniciar a implementação do serviço, crie o banco de dados que será utilizado:

4. Na plataforma Windows, para acessar o terminal do MySQL e criar o banco, deve-se ir em Iniciar *> Programas > MySQL > MySQL Server 5.x > MySQL Command Line Client*.

5. Com o terminal aberto, entre com o seguinte comando: ***mysql –u [usuário_root] –p [senha_usuário-root]***

6. Em seguida, cole o *script* SQL que se encontra no arquivo Nurse_BD.sql que acompanha este material.

As regras de negócio para manipulação do banco de dados já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.dao* do projeto importado. As funcionalidades para manipulação e conversão de objetos JSON e XML encontram-se prontas no pacote *br.ufscar.dc.utils.*

Na classe ***FactoryConnection*** do pacote *br.ufscar.dc.dao*, altere as variáveis ***USER*** e ***PASSWORD***, atribuindo, respectivamente, os dados de usuário e senha locais para acesso ao banco de dados. <u>Exemplo</u>: USER = “root”; PASSWORD=”123456”;.

9. <u>Implementar o Serviço</u>

1. No pacote *br.ufscar.dc.beans* crie a classe *Nurse*.
2. Após a criação da classe, adicione as propriedades *id* (int), *name* (String), *age* (int), *gender* (String) e *dateBirth* (java.util.Date)*.
3. Crie os métodos *get* e *set* para cada uma das propriedades.
4. Em seguida, no pacote *br.ufscar.dc.resources* crie a classe *NurseResource* que representa o serviço associado à entidade Nurse.
5. Na classe *NurseResource* adicione as anotações do RESTful necessárias *(@Path, @Consumes, @Produces*, etc), bem como adicione o método (nomeie-o como *listNurses*) que recuperará a lista de enfermeiras registradas no banco de dados. Para nomear as anotações, utilize os seguintes valores:

Classe *NurseResource*:

```
@Path(“contentNurse”)
@Produces(“{application/json}”)
@Consumes(“{application/json}”)
```

Método *listNurses*:

```
@GET
@Path(“listNurses”)
```

A classe implementada deverá ficar semelhante à mostrada na Figura G.2.

**Ao criar os métodos e atributos indicados neste tutorial, assegure-se de obedecer ao padrão utilizado na nomenclatura adotada, tais como letras maiúsculas, minúsculas, etc.*

10. <u>Testar o Serviço</u>

3. Após a implementação do serviço, teste-o no browser. Para isto, clique com o botão direito sobre a pasta raiz do projeto wsNurse e escolha a opção *Run As… > Run on Server.*

4. O browser será aberto. Teste a operação para listar as enfermeiras registradas no banco de dados, digitando a seguinte URL:
   http://localhost:8080/wsNurse/resources/contentNurse/listNurses

```
import java.sql.ResultSet;
@Path("/contentNurse")
@Consumes( { "application/json" })
@Produces( { "application/json" })
public class NurseResource {

    @GET
    @Path("/listNurses")
    public Response listNurses() {
        NurseDAO nurseDAO = new NurseDAO();
        ResultSet rs = nurseDAO.listNurses();
        List<Nurse> listNurses = new ArrayList<Nurse>();
        try {
            while (rs.next()) {
                Nurse n = new Nurse();
                n.setId(rs.getInt(1));
                n.setName(rs.getString(2));
                n.setAge(rs.getInt(3));
                n.setGender(rs.getString(4));
                n.setDateBirth(rs.getDate(5));
                listNurses.add(n);
            }
            return Response.ok(
                    XMLParser.parseObjectToJSON("listNurses", listNurses))
                    .build();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return Response.ok(XMLParser.parseObjectToJSON("listNurses", null))
                .build();
    }
}
```

Figura G.2 - Código da Classe NurseResource

II - DESENVOLVIMENTO DA APLICAÇÃO MÓVEL

5. <u>Importar o projeto parcial da aplicação móvel no Eclipse</u>

3. Importe o projeto *AndroExpA* que acompanha este material. Siga os mesmos passos do passo "Importar o projeto parcial do Web Service no Eclipse" apresentado anteriormente.

4. Corrija quaisquer erros de importação de bibliotecas referentes à SDK do Android. Vá ao menu *Window > Preferences*. Na janela que se abre, selecione na parte esquerda a opção "Android". Caso o campo SDK Location esteja vazio, clique no botão "Browse..." e indique a localização da pasta "android-sdk" que se encontra no diretório do Eclipse. Após selecionar a pasta, clique em OK.

   A estrutura do projeto importado será como a **Figura G.3** abaixo:

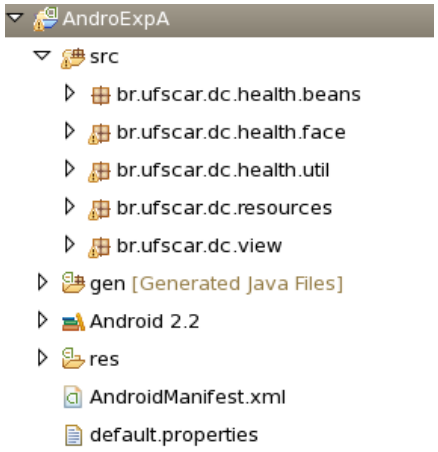

Figura G.3 - Estrutura do projeto Android importado

As interfaces gráficas da aplicação móvel já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.health.face* e *br.ufscar.dc.view* do projeto importado. As funcionalidades para formatação de objetos de dados do tipo Date encontram-se prontas no pacote *br.ufscar.dc.health.util.*

6. Implementar os beans

1. No pacote *br.ufscar.dc.health.beans* crie os *beans Occurrence* e *Victm* associadas à **Aplicação A.** Note que neste pacote já existe o *bean* referente à entidade *Nurse*.
2. Após a criação dos beans, adicione as propriedades*:
    a. Occurrence: id (int), description (String), date (java.util.Date), motive (String), timearrive (java.util.Time), e timeleave (java.util.Time).
    b. Victim: id (int), name (String), age (int), e anammesis (String).
3. Crie os métodos *get* e *set* para cada uma das propriedades.

**Ao criar os métodos e atributos indicados neste tutorial, assegure-se de obedecer ao padrão utilizado na nomenclatura adotada, tais como letras maiúsculas, minúsculas, etc.*

4. Implementar as classes que chamam o serviço

1. No pacote *br.ufscar.dc.resources* crie a classe *OccurrenceResource* que consome o serviço;
    a. Crie o método *listOccurrences*, o qual consome o serviço do Domínio Healthcare já implementado. Para acesso ao serviço, utilize a seguinte URL: http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentOccurrence/listOccurrences .
    b. Crie o método *listVictimToOccurrence*, o qual consome o serviço do Domínio Healthcare já implementado. Para acesso ao serviço, utilize a seguinte URL: http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentOccurrence/ listVictimsToOccurrence/idOccurrence .

A classe implementada deverá ficar semelhante à mostrada na Figura G.4.

```
public class OccurrenceResource {
    public String listOccurrences() {
        DefaultHttpClient httpclient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentOccurrence/"
                        + "listOccurrences");
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler responseHandler = new BasicResponseHandler();
        try {
            return httpclient.execute(httpGet, responseHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
    public String listVictimsToOccurrence(Integer idOccurrence) {
        DefaultHttpClient httpclient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentOccurrence/"
                        + "listVictimsToOccurrence" + idOccurrence);
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler responseHandler = new BasicResponseHandler();
        try {
            return httpclient.execute(httpGet, responseHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Figura G.4 - Código da Classe OccurrenceResource

5. <u>Testar a aplicação móvel com o emulador do Android</u>

3. Após a implementação da aplicação, teste-a no emulador do Android. Para isto, clique com o botão direito sobre a pasta raiz do projeto AndroExpA e escolha a opção *Run As… > Android Application.*

    a. Caso ainda não exista um dispositivo emulador configurado, o Eclipse emitirá um alerta para a criação de um novo *Android Virtual Device (ADV)*. Neste caso clique em Yes.

    b. Na janela que se abre, clique em “New” para criar um novo ADV. Em seguida dê um nome qualquer para este dispositivo. No campo “Target” selecione “Android 2.2 – APILevel8”. Mantenha as demais configurações como estão e clique no botão “Create ADV”.

    c. Feche todas as janelas referentes à criação do ADV. Em seguida, clique novamente com o botão direito sobre a pasta raiz do projeto AndroExpA e escolha a opção *Run As… > Android Application.* Desta vez, o Eclipse executará a aplicação no ADV recém-criado.

4. Teste a aplicação, verifique se a mesma está funcionando corretamente, trazendo a lista de ocorrências, vítimas e enfermeiras(os).

Apêndice H

DESCRIÇÃO DA APLICAÇÃO B

A *Aplicação B* é uma aplicação que permite a **visualização de pacientes** previamente cadastrados na base de dados da clínica médica. O médico, ao efetuar a consulta ao paciente, realiza o registro do histórico dessa consulta, armazenando-o na base de dados. O médico, responsável pela monitoração do paciente, muitas vezes necessita buscar o histórico do paciente em situação de emergência médica, de forma que possa se planejar apropriadamente durante um atendimento de emergência e chegar ao local da ocorrência com toda a instrumentação necessária devidamente preparada para socorrer a vítima. Contudo, na maioria das vezes, conforme a gravidade da situação, a fim de assegurar o sucesso do atendimento, o médico necessita partir para o atendimento tão logo fique sabendo do local da emergência sem mesmo aguardar o término da chamada recebida pelo operador de uma central de atendimento. Logo, a necessidade vigente é a disponibilização dos detalhes dos pacientes por meio de dispositivos móveis aos médicos, de maneira que estes possam se preparar adequadamente durante o percurso. Esses médicos, munidos de um dispositivo móvel, devem visualizar a ***lista com o nome dos pacientes***. Além de conhecerem os detalhes de um paciente vitimado, algumas vezes esses médicos necessitam de informações a respeito do histórico do paciente que está registrado em sua clínica médica, de forma a efetuar uma análise e acompanhamento do paciente durante a evolução de suas doenças. Assim, esses médicos, munidos de um dispositivo móvel, devem também ser capazes de acessar uma ***lista dos históricos de atendimentos do paciente***. O histórico do paciente contém o identificador, a data, a descrição(ões) do(s) sintomas e tratamento(s) efetuados. Com isso, os médicos poderão visualizar e buscar, por meio de um

dispositivo móvel, de qualquer lugar e a qualquer momento, informações sobre os pacientes, de forma a auxiliar na prestação dos serviços de atendimento de emergência.

Apêndice I

MATERIAL DE APOIO DA APLICAÇÃO B – GRUPO ABORDAGEM DE REÚSO COM DSM E SOA

I - DESENVOLVIMENTO DO SERVIÇO

11. <u>Importar o projeto parcial do Web Service no Eclipse</u>

Importe o projeto *wsNurse* que acompanha este material. Siga os seguintes passos:

13. No IDE Eclipse vá no menu *File* e escolha a opção *Import*...
14. Na janela que se abre, vá à categoria *General* e selecione a opção *Existing Projects Into Workspace*. Pressione o botão *next*.
15. Na próxima janela, clique no botão *Browse* e selecione a pasta raiz do projeto *wsAmbulance* salvo em seu computador.
16. Nessa mesma janela selecione a opção *Copy Projects Into Workspace*.
17. Finalmente, pressione o botão *Finish*. O IDE Eclipse importará o projeto Web de nome *wsAmbulance* ao *workspace* atual.
18. Corrija quaisquer erros de importação de bibliotecas. Clique com o botão direito sobre a pasta raiz do projeto. Selecione a opção Build *Path > Configure Build Path...* Na janela que se abre remova as bibliotecas conflitantes e adicione novamente (Bibliotecas com possíveis erros: *Apache TomCat, MySQL*, biblioteca do *RESTful*).

A estrutura do projeto importado será como a **Figura I.1** a seguir:

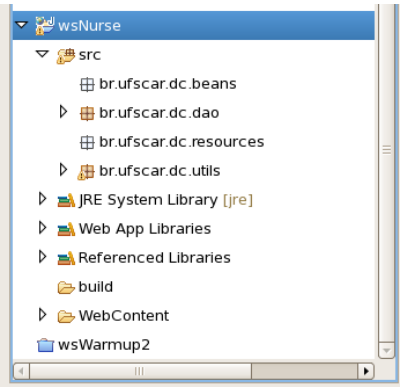

Figura I.1 - Estrutura do projeto importado

12. Criar o banco de dados acessado pelo Serviço

Antes de iniciar a implementação do serviço, crie o banco de dados que será utilizado:

7. Na plataforma Windows, para acessar o terminal do MySQL e criar o banco, deve-se ir em Iniciar > *Programas* > *MySQL* > *MySQL Server 5.x* > *MySQL Command Line Client*.

8. Com o terminal aberto, entre com o seguinte comando: ***mysql –u [usuário_root] –p [senha_usuário-root]***

9. Em seguida, cole o *script* SQL que se encontra no arquivo Nurse_BD.sql que acompanha este material.

As regras de negócio para manipulação do banco de dados já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.dao* do projeto importado. As funcionalidades para manipulação e conversão de objetos JSON e XML encontram-se prontas no pacote *br.ufscar.dc.utils.*

Na classe ***FactoryConnection*** do pacote *br.ufscar.dc.dao*, altere as variáveis ***USER*** e ***PASSWORD***, atribuindo, respectivamente, os dados de usuário e senha locais para acesso ao banco de dados. Exemplo: USER = “root”; PASSWORD=”123456”;.

13. Modelar o Serviço

Para criar o modelo da aplicação, utilize o *plugin* editor de modelos construído na etapa de Engenharia de Domínio da abordagem Reúso com DSM e SOA:

11. Na pasta raiz do projeto *wsNurse* crie uma pasta chamada “*Modelo*” (clique com o botão direito sobre a pasta raiz, opção *New* > *Folder*)

12. Clique com o botão direito sobre a pasta *Modelo* recém-criada e selecione a opção *New* > *Other...*

13. Na janela que se abre, selecione a opção *EmergencyMetamodel Model* da categoria *Example EMF Model Creation Wizards*. Em seguida clique em *Next*.

14. Na próxima janela, nomeie o modelo com o nome *ws_model.emergencymetamodel* Clique em *Next*.

15. Na janela seguinte escolha a opção **Model** para o campo *Model Object*. Em seguida pressione o botão *Finish*. O modelo será criado, tendo como nó raiz o elemento *Model*.

    a. Clique com o botão direito do mouse sobre o elemento *Model* do modelo criado e selecione a opção “Show properties view”. Na caixa de propriedades do elemento *Model*, atribua o nome *wsNurse* para a propriedade “*Project”,* que indica a que projeto o modelo está vinculado.

b. Clique novamente com o botão direito sobre o elemento *Model*, escolha a opção New Child. Em seguida selecione o elemento *Object* que aparece no menu popup que se abre. Será criado um novo elemento *Object* no modelo, filho do elemento *Model*.

c. Agora, clique sobre o elemento Object e adicione o elemento *Nurse*, atribua os seguintes valores:

   i. Package: br.ufscar.dc.beans;

   ii. Stereotype: bean.

d. Adicione um elemento *Attribute* dentro do elemento *Nurse*. Na barra de propriedades, atribua os seguintes valores:

   i. Attribute: age.

e. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “Integer”.

f. Adicione um elemento *Attribute* dentro do elemento *Nurse*. Na barra de propriedades, atribua os seguintes valores:

   i. Attribute: gender.

g. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “String”.

h. Adicione um elemento *Attribute* dentro do elemento *Nurse*. Na barra de propriedades, atribua os seguintes valores:

   i. Attribute: dateBirth.

i. Adicione um elemento Attribute Type dentro do elemento Attribute. Na barra de propriedades, selecione o valor “Java.sql.Date”.

j. Agora, adicione um elemento *Service* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:

   i. Name: NurseResource;

   ii. Package: *br.ufscar.dc.resources;*

   iii. *Path: contentNurse*

   iv. *Protocol: Rest.*

k. Adicione um elemento *Service Input* ao elemento *Service*. Dentro do elemento *Service Input* recém-criado, adicione um elemento *Service Data*. Na propriedade “Mimetype” deste último, selecione o valor “application/json”.

l. Adicione um elemento *Service Output* ao elemento *Service*. Dentro do elemento *Service Output* recém-criado, adicione um elemento *Service Data*. Na propriedade “Mimetype” deste último, selecione o valor “application/json”.

m. Adicione um elemento *Method* Service ao elemento *Service*. Preencha as propriedades deste elemento da seguinte forma:

   i. Method: listNurses;

   ii. Path: listNurses;

   iii. Type: retrieve.

n. Adicione um elemento Return Method ao elemento Method Service. Na propriedade "Return" deste último selecione o valor "Response".

14. <u>Executar as Transformações Modelo para Código</u>

Dica Importante: sempre salve todas as alterações no modelo antes de executar as transformações, de forma que essas alterações sejam refletidas no código gerado.

9. Clique com o botão direito sobre o modelo do serviço e selecione a opção *Run As > Run Configurations*.
10. Na caixa do lado esquerdo da janela que se abre, dê um clique duplo sobre a opção *JET Transformation*. Do lado direito aparecerão as opções para execução de transformações.
11. No campo *Transformation Input*, clique sobre o botão *Browse...* e selecione o arquivo do modelo do serviço.
12. No campo *ID* selecione a transformação que deseja aplicar sobre o modelo (*br.ufscar.dc.helth.codegeneratorJEEWS*) e em seguida pressione o botão *Run*. A transformação será executada, sendo o código gerado dentro do projeto *wsNurse* (conforme definido na propriedade "Project" do elemento Model).

15. <u>Implementar o Serviço</u>

2. A classe implementada deverá ficar semelhante à mostrada na Figura I.2.

**Ao criar os métodos e atributos indicados neste tutorial, assegure-se de obedecer ao padrão utilizado na nomenclatura adotada, tais como letras maiúsculas, minúsculas, etc.*

```
import java.sql.ResultSet;
@Path("/contentNurse")
@Consumes( { "application/json" })
@Produces( { "application/json" })
public class NurseResource {

    @GET
    @Path("/listNurses")
    public Response listNurses() {
        NurseDAO nurseDAO = new NurseDAO();
        ResultSet rs = nurseDAO.listNurses();
        List<Nurse> listNurses = new ArrayList<Nurse>();
        try {
            while (rs.next()) {
                Nurse n = new Nurse();
                n.setId(rs.getInt(1));
                n.setName(rs.getString(2));
                n.setAge(rs.getInt(3));
                n.setGender(rs.getString(4));
                n.setDateBirth(rs.getDate(5));
                listNurses.add(n);
            }
            return Response.ok(
                    XMLParser.parseObjectToJSON("listNurses", listNurses))
                    .build();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return Response.ok(XMLParser.parseObjectToJSON("listNurses", null))
                .build();
    }
}
```

Figura I.2 - Código da Classe NurseResource

16. <u>Testar o Serviço</u>

5. Após a implementação do serviço, teste-o no browser. Para isto, clique com o botão direito sobre a pasta raiz do projeto wsNurse e escolha a opção *Run As… > Run on Server.*

6. O browser será aberto. Teste a operação para listar as enfermeiras registradas no banco de dados, digitando a seguinte URL:
   http://localhost:8080/wsNurse/resources/contentNurse/listNurses

II - DESENVOLVIMENTO DA APLICAÇÃO MÓVEL

7. Importar o projeto parcial da aplicação móvel no Eclipse

5. Importe o projeto *AndroExpB* que acompanha este material. Siga os mesmos passos do passo "Importar o projeto parcial do Web Service no Eclipse" apresentado anteriormente.
6. Corrija quaisquer erros de importação de bibliotecas referentes à SDK do Android. Vá ao menu *Window > Preferences*. Na janela que se abre, selecione na parte esquerda a opção "Android". Caso o campo SDK Location esteja vazio, clique no botão "Browse..." e indique a localização da pasta "android-sdk" que se encontra no diretório do Eclipse. Após selecionar a pasta, clique em OK.

   A estrutura do projeto importado será como a **Figura I.3** a seguir:

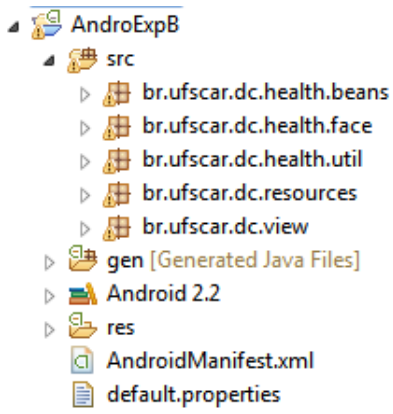

Figura I.3 - Estrutura do projeto Android importado

As interfaces gráficas da aplicação móvel já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.health.face* e *br.ufscar.dc.view* do projeto importado. As funcionalidades para formatação de objetos de dados do tipo Date encontram-se prontas no pacote *br.ufscar.dc.health.util.*

8. Modelar o Serviço

Para criar o modelo da aplicação, utilize o *plugin* editor de modelos construído na etapa de Engenharia de Domínio da abordagem Reúso com DSM e SOA:

16. Na pasta raiz do projeto *AndroExpB* crie uma pasta chamada "*Modelo*" (clique com o botão direito sobre a pasta raiz, opção *New > Folder*)
17. Clique com o botão direito sobre a pasta *Modelo* recém-criada e selecione a opção *New > Other...*
18. Na janela que se abre, selecione a opção *EmergencyMetamodel Model* da categoria *Example EMF Model Creation Wizards*. Em seguida clique em *Next*.
19. Na próxima janela, nomeie o modelo com o nome *android_model.emergencymetamodel* Clique em *Next*.
20. Na janela seguinte escolha a opção **Model** para o campo *Model Object*. Em seguida pressione o botão *Finish*. O modelo será criado, tendo como nó raiz o elemento *Model*.

a. Clique com o botão direito do mouse sobre o elemento *Model* do modelo criado e selecione a opção “Show properties view”. Na caixa de propriedades do elemento *Model*, atribua o nome *AndroExpB* para a propriedade “*Project”,* que indica a que projeto o modelo está vinculado.

b. Clique novamente com o botão direito sobre o elemento *Model*, escolha a opção New Child. Em seguida selecione o elemento *Object* que aparece no menu popup que se abre. Será criado um novo elemento *Object* no modelo, filho do elemento *Model*.

c. Adicione um elemento *Patient* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:

    i. Package: br.ufscar.dc.health.beans;

    ii. Stereotype: bean.

d. Adicione um elemento *Attribute* dentro do elemento *Patient*. Na barra de propriedades, atribua os seguintes valores:

    i. Attribute: dataBirth.

e. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “Java.Util.Date”.

f. Adicione um elemento *Attribute* dentro do elemento *Class*. Na barra de propriedades, atribua os seguintes valores:

    i. Attribute: sex.

g. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “String”.

h. Adicione um elemento *History* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:

    i. Package: br.ufscar.dc.health.beans;

    ii. Stereotype: bean.

i. Adicione um elemento *Attribute* dentro do elemento *History*. Na barra de propriedades, atribua os seguintes valores:

    i. Attribute: treatment.

j. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “String”.

k. Clique novamente com o botão direito sobre o elemento *Model*, escolha a opção New Child. Em seguida selecione o elemento *Reuse Set* que aparece no menu popup que se abre. Será criado um novo elemento chamado *Reuse Set* no modelo, filho do elemento *Model*.

    i. O elemento Reuse Set contém os serviços juntamente com seus métodos do Domínio Healthcare para reúso pela aplicações.

l. Adicione um elemento *Class* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:

    i. Name: PatientResource;

    ii. Package: br.ufscar.dc.resources;

    iii. Stereotype: reuse.

m. Adicione um elemento *Method* dentro do element *Class* recém criado. Na barra de propriedades, atribua os seguintes valores:
    i. Has Operation Service: Listpatients listPatients;
    ii. Has Service Reuse: Patientresources PatientResource;
    iii. Method: listPatients;
    iv. Stereotype: reuse.

n. Adicione um elemento *Return Type* dentro do elemnto Method recém criado. Na barra de propriedades, selecione para a propriedade *ReturnType* o valor "String".

o. Adicione um elemento *Method* dentro do element *Class*. Na barra de propriedades, atribua os seguintes valores:
    i. Has Operation Service: Listpatienthistory listPatientAllHistory;
    ii. Has Service Reuse: Patientresources PatientResource;
    iii. Method: listPatientAllHistory;
    iv. Stereotype: reuse.

p. Adicione um elemento *Return Type* dentro do elemnto Method recém criado. Na barra de propriedades, selecione para a propriedade *ReturnType* o valor "String".

q. Adicione um elemento *Parameter* dentro do elemnto Method recém criado. Na barra de propriedades, atribua o seguinte valore:
    i. Parameter: idPatient

r. Adicione um elemento *Parameter Type* dentro do elemnto *Parameter* recém criado. Na barra de propriedades, selecione para a propriedade *Parametertype* o valor "Integer".

9. <u>Executar as Transformações Modelo para Código</u>

Dica Importante: sempre salve todas as alterações no modelo antes de executar as transformações, de forma que essas alterações sejam refletidas no código gerado.

13. Clique com o botão direito sobre o modelo do serviço e selecione a opção *Run As > Run Configurations*.
14. Na caixa do lado esquerdo da janela que se abre, dê um clique duplo sobre a opção *JET Transformation*. Do lado direito aparecerão às opções para execução de transformações.
15. No campo *Transformation Input*, clique sobre o botão *Browse...* e selecione o arquivo do modelo do serviço (*android_model.emergencymetamodel*).
16. No campo *ID* selecione a transformação que deseja aplicar sobre o modelo (*br.ufscar.dc.helth.generatorAndroid2*) e em seguida pressione o botão *Run*. A transformação será executada, sendo o código gerado dentro do projeto AndroExpB (conforme definido na propriedade "Project" do elemento *Model*).

A classe *PatientResource* implementada deverá ficar semelhante à mostrada na Figura I.4.

```
public class PatientResource {
    public PatientResource() {
    }

    public String listPatients() {
        DefaultHttpClient httpClient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentPatient/"
                        + "listPatients");
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler respondeHandler = new BasicResponseHandler();
        try {
            return httpClient.execute(httpGet, respondeHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return "";
    }

    public String listPatientAllHistory(int idPatient) {
        DefaultHttpClient httpClient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentPatient/"
                        + "listPatientAllHistory/" + idPatient);
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler respondeHandler = new BasicResponseHandler();
        try {
            return httpClient.execute(httpGet, respondeHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return "";
    }
}
```

Figura I.4 - Código da Classe PatientResource

No pacote *br.ufscar.dc.resources* abra a classe NurseResource, na linha:

```
HttpGet httpGet = new HttpGet("");
```

b. Adicione a URL do serviço criado no projeto wsNurse;

10. <u>Testar a aplicação móvel com o emulador do Android</u>

5. Após a implementação da aplicação, teste-a no emulador do Android. Para isto, clique com o botão direito sobre a pasta raiz do projeto AndroExpB e escolha a opção *Run As… > Android Application.*

   a. Caso ainda não exista um dispositivo emulador configurado, o Eclipse emitirá um alerta para a criação de um novo *Android Virtual Device (ADV)*. Neste caso clique em Yes.

   b. Na janela que se abre, clique em “New” para criar um novo ADV. Em seguida dê um nome qualquer para este dispositivo. No campo “Target” selecione “Android 2.2 – APILevel8”. Mantenha as demais configurações como estão e clique no botão “Create ADV”.

   c. Feche todas as janelas referentes à criação do ADV. Em seguida, clique novamente com o botão direito sobre a pasta raiz do projeto AndroExpB e escolha a opção *Run As… > Android Application.* Desta vez, o Eclipse executará a aplicação no ADV recém-criado.

Teste a aplicação, verifique se a mesma está funcionando corretamente, trazendo a lista de pacientes, histórico do paciente e enfermeiras(os).

Apêndice J

MATERIAL DE APOIO DA APLICAÇÃO B – GRUPO TRADICIONAL

I - DESENVOLVIMENTO DO SERVIÇO

17. <u>Importar o projeto parcial do Web Service no Eclipse</u>

Importe o projeto *wsNurse* que acompanha este material. Siga os seguintes passos:

19. No IDE Eclipse vá no menu *File* e escolha a opção *Import*...
20. Na janela que se abre, vá à categoria *General* e selecione a opção *Existing Projects Into Workspace*. Pressione o botão *next*.
21. Na próxima janela, clique no botão *Browse* e selecione a pasta raiz do projeto *wsAmbulance* salvo em seu computador.
22. Nessa mesma janela selecione a opção *Copy Projects Into Workspace*.
23. Finalmente, pressione o botão *Finish*. O IDE Eclipse importará o projeto Web de nome *wsAmbulance* ao *workspace* atual.
24. Corrija quaisquer erros de importação de bibliotecas. Clique com o botão direito sobre a pasta raiz do projeto. Selecione a opção Build *Path > Configure Build Path...* Na janela que se abre remova as bibliotecas conflitantes e adicione novamente (Bibliotecas com possíveis erros: *Apache TomCat, MySQL*, biblioteca do *RESTful*).

A estrutura do projeto importado será como a **Figura J.1** abaixo:

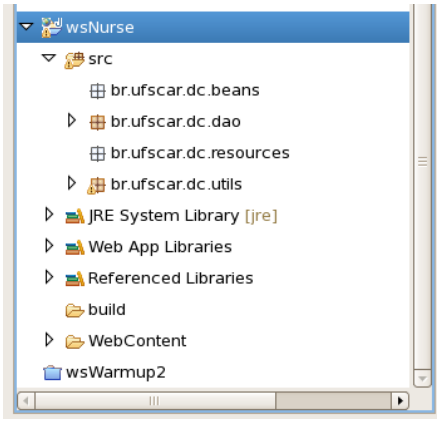

Figura J.1 - Estrutura do projeto importado

18. <u>Criar o banco de dados acessado pelo Serviço</u>

Antes de iniciar a implementação do serviço, crie o banco de dados que será utilizado:

10. Na plataforma Windows, para acessar o terminal do MySQL e criar o banco, deve-se ir em Iniciar *> Programas > MySQL > MySQL Server 5.x > MySQL Command Line Client*.
11. Com o terminal aberto, entre com o seguinte comando: ***mysql –u [usuário_root] –p [senha_usuário-root]***
12. Em seguida, cole o *script* SQL que se encontra no arquivo Nurse_BD.sql que acompanha este material.

As regras de negócio para manipulação do banco de dados já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.dao* do projeto importado. As funcionalidades para manipulação e conversão de objetos JSON e XML encontram-se prontas no pacote *br.ufscar.dc.utils.*

Na classe ***FactoryConnection*** do pacote *br.ufscar.dc.dao*, altere as variáveis ***USER*** e ***PASSWORD***, atribuindo, respectivamente, os dados de usuário e senha locais para acesso ao banco de dados. <u>Exemplo</u>: USER = “root”; PASSWORD=”123456”;.

19. <u>Implementar o Serviço</u>

6. No pacote *br.ufscar.dc.beans* crie a classe *Nurse*.
7. Após a criação da classe, adicione as propriedades *id* (int), *name* (String), *age* (int), *gender* (String) e *dateBirth* (java.util.Date)*.
8. Crie os métodos *get* e *set* para cada uma das propriedades.
9. Em seguida, no pacote *br.ufscar.dc.resources* crie a classe *NurseResource* que representa o serviço associado à entidade Nurse.
10. Na classe *NurseResource* adicione as anotações do RESTful necessárias *(@Path, @Consumes, @Produces*, etc), bem como adicione o método (nomeie-o como *listNurses*) que recuperará a lista de enfermeiras registradas no banco de dados. Para nomear as anotações, utilize os seguintes valores:

    Classe *NurseResource*:

    @Path(“contentNurse”)

    @Produces(“{application/json}”)

    @Consumes(“{application/json}”)

    Método *listNurses*:

    @GET

    @Path(“listNurses”)

    A classe implementada deverá ficar semelhante à mostrada na Figura J.2.

**Ao criar os métodos e atributos indicados neste tutorial, assegure-se de obedecer ao padrão utilizado na nomenclatura adotada, tais como letras maiúsculas, minúsculas, etc.*

20. <u>Testar o Serviço</u>

7. Após a implementação do serviço, teste-o no browser. Para isto, clique com o botão direito sobre a pasta raiz do projeto wsNurse e escolha a opção *Run As… > Run on Server.*

8. O browser será aberto. Teste a operação para listar as enfermeiras registradas no banco de dados, digitando a seguinte URL:
   http://localhost:8080/wsNurse/resources/contentNurse/listNurses

```
import java.sql.ResultSet;
@Path("/contentNurse")
@Consumes( { "application/json" })
@Produces( { "application/json" })
public class NurseResource {

    @GET
    @Path("/listNurses")
    public Response listNurses() {
        NurseDAO nurseDAO = new NurseDAO();
        ResultSet rs = nurseDAO.listNurses();
        List<Nurse> listNurses = new ArrayList<Nurse>();
        try {
            while (rs.next()) {
                Nurse n = new Nurse();
                n.setId(rs.getInt(1));
                n.setName(rs.getString(2));
                n.setAge(rs.getInt(3));
                n.setGender(rs.getString(4));
                n.setDateBirth(rs.getDate(5));
                listNurses.add(n);
            }
            return Response.ok(
                    XMLParser.parseObjectToJSON("listNurses", listNurses))
                    .build();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return Response.ok(XMLParser.parseObjectToJSON("listNurses", null))
                .build();
    }
}
```

Figura J.2 - Código da Classe NurseResource

II - DESENVOLVIMENTO DA APLICAÇÃO MÓVEL

11. Importar o projeto parcial da aplicação móvel no Eclipse

7. Importe o projeto *AndroExpB* que acompanha este material. Siga os mesmos passos do passo "Importar o projeto parcial do Web Service no Eclipse" apresentado anteriormente.

8. Corrija quaisquer erros de importação de bibliotecas referentes à SDK do Android. Vá ao menu *Window > Preferences*. Na janela que se abre, selecione na parte esquerda a opção "Android". Caso o campo SDK Location esteja vazio, clique no botão "Browse..." e indique a localização da pasta "android-sdk" que se encontra no diretório do Eclipse. Após selecionar a pasta, clique em OK.

   A estrutura do projeto importado será como a **Figura J.3** abaixo:

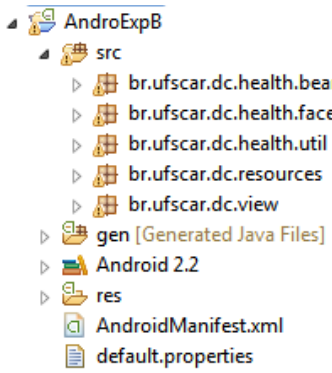

Figura J.3 - Estrutura do projeto Android importado

As interfaces gráficas da aplicação móvel já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.health.face* e *br.ufscar.dc.view* do projeto importado. As funcionalidades para formatação de objetos de dados do tipo Date encontram-se prontas no pacote *br.ufscar.dc.health.util.*

12. Implementar os beans

6. No pacote *br.ufscar.dc.health.beans* crie os *beans Patient* e *History* associadas à **Aplicação B.** Note que neste pacote já existe o *bean* referente à entidade *Nurse*.
7. Após a criação dos beans, adicione as propriedades*:
    a. Patient: id (int), name (String), dateBirth (java.util.Date), bloodtype (String), sex (String), e allergy (String).
    b. History: id (int), description (String), dateHistory (java.util.Date), e treatment (String).
8. Crie os métodos *get* e *set* para cada uma das propriedades.

**Ao criar os métodos e atributos indicados neste tutorial, assegure-se de obedecer ao padrão utilizado na nomenclatura adotada, tais como letras maiúsculas, minúsculas, etc.*

9. Implementar as classes que chamam o serviço

2. No pacote *br.ufscar.dc.resources* crie a classe *PatientResource* que consome o serviço;
    a. Crie o método *listPatients*, o qual consome o serviço do Domínio Healthcare já implementado. Para acesso ao serviço, utilize a seguinte URL: http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentPatient/listPatients .
    b. Crie o método *listVictimToOccurrence*, o qual consome o serviço do Domínio Healthcare já implementado. Para acesso ao serviço, utilize a seguinte URL: http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentPatient/listPatientAllHistory/idPatient .

A classe implementada deverá ficar semelhante à mostrada na Figura J.4.

```
public class PatientResource {
    public PatientResource() {
    }

    public String listPatients() {
        DefaultHttpClient httpClient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentPatient/"
                        + "listPatients");
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler respondeHandler = new BasicResponseHandler();
        try {
            return httpClient.execute(httpGet, respondeHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return "";
    }

    public String listPatientAllHistory(int idPatient) {
        DefaultHttpClient httpClient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/contentPatient/"
                        + "listPatientAllHistory/" + idPatient);
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler respondeHandler = new BasicResponseHandler();
        try {
            return httpClient.execute(httpGet, respondeHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return "";
    }
}
```

Figura J.4 - Código da Classe PatientResource

10. <u>Testar a aplicação móvel com o emulador do Android</u>

6. Após a implementação da aplicação, teste-a no emulador do Android. Para isto, clique com o botão direito sobre a pasta raiz do projeto AndroExpB e escolha a opção *Run As… > Android Application.*

    a. Caso ainda não exista um dispositivo emulador configurado, o Eclipse emitirá um alerta para a criação de um novo *Android Virtual Device (ADV)*. Neste caso clique em Yes.

    b. Na janela que se abre, clique em "New" para criar um novo ADV. Em seguida dê um nome qualquer para este dispositivo. No campo "Target" selecione "Android 2.2 – APILevel8". Mantenha as demais configurações como estão e clique no botão "Create ADV".

    c. Feche todas as janelas referentes à criação do ADV. Em seguida, clique novamente com o botão direito sobre a pasta raiz do projeto AndroExpB e escolha a opção *Run As… > Android Application.* Desta vez, o Eclipse executará a aplicação no ADV recém-criado.

7. Teste a aplicação, verifique se a mesma está funcionando corretamente, trazendo a lista de pacientes, histórico do paciente e enfermeiras(os).

Apêndice K

DESCRIÇÃO DA APLICAÇÃO C

A *Aplicação C* é uma aplicação que permite a **visualização dos dados sobre uma determinada doença**, previamente cadastrada na base de dados da unidade de atendimentos de emergência de uma determinada cidade. Os funcionários da unidade de atendimentos de emergência de uma determinada cidade, responsáveis pela execução do atendimento, muitas vezes necessitam conhecer os detalhes das ocorrências atuais enquanto dirigem-se para o atendimento, de forma que possam se planejar apropriadamente durante o caminho e chegar ao local da emergência com toda a instrumentação necessária preparada para socorrer a vítima. Contudo, na maioria das vezes, conforme a gravidade da situação, a fim de assegurar o sucesso do atendimento, esses funcionários necessitam partir para o atendimento tão logo fiquem sabendo do local da emergência sem mesmo aguardar o término da chamada recebida pelo operador. Dentre as necessidades vigentes, tem-se a disponibilização de uma ***lista de doenças*** e seus detalhes, de maneira que os funcionários possam se preparar adequadamente durante o percurso para socorrer o paciente vitimado. Além de conhecerem os detalhes das doenças, esses funcionários também necessitam de informações mais específicas a respeito dos sintomas associados às doenças, de forma que as medicações e procedimentos a serem prestados durante o socorro da vítima não interfiram na evolução de seu tratamento. Assim, outra funcionalidade necessária refere-se à apresentação de uma ***lista de sintomas associados à doença*** visualizada. Cada doença contém seu identificador (CID), sua descrição, a indicação do tratamento. Cada sintoma compreende seu identificador e a descrição do mesmo. Com isso, os funcionários poderão visualizar, por meio de um dispositivo móvel, de qualquer lugar e a qualquer momento, informações sobre doenças e seus sintomas, de forma a auxiliar na prestação dos serviços atendimento de emergência.

Apêndice L

MATERIAL DE APOIO DA APLICAÇÃO C – GRUPO ABORDAGEM DE REÚSO COM DSM E SOA

I - DESENVOLVIMENTO DO SERVIÇO

21. <u>Importar o projeto parcial do Web Service no Eclipse</u>

Importe o projeto *wsNurse* que acompanha este material. Siga os seguintes passos:

25. No IDE Eclipse vá no menu *File* e escolha a opção *Import*...
26. Na janela que se abre, vá à categoria *General* e selecione a opção *Existing Projects Into Workspace*. Pressione o botão *next*.
27. Na próxima janela, clique no botão *Browse* e selecione a pasta raiz do projeto *wsAmbulance* salvo em seu computador.
28. Nessa mesma janela selecione a opção *Copy Projects Into Workspace*.
29. Finalmente, pressione o botão *Finish*. O IDE Eclipse importará o projeto Web de nome *wsAmbulance* ao *workspace* atual.
30. Corrija quaisquer erros de importação de bibliotecas. Clique com o botão direito sobre a pasta raiz do projeto. Selecione a opção Build *Path > Configure Build Path...* Na janela que se abre remova as bibliotecas conflitantes e adicione novamente (Bibliotecas com possíveis erros: *Apache TomCat, MySQL*, biblioteca do *RESTful*).

A estrutura do projeto importado será como a **Figura L.1** a seguir:

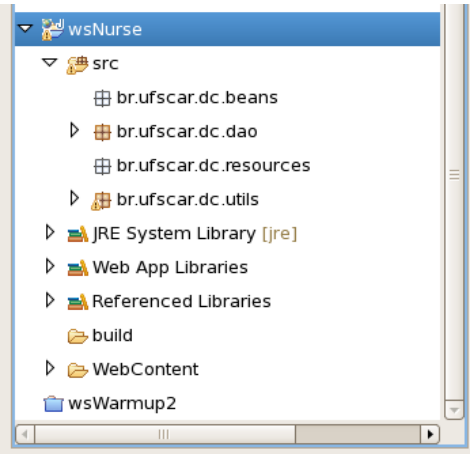

Figura L.1 - Estrutura do projeto importado

22. Criar o banco de dados acessado pelo Serviço

Antes de iniciar a implementação do serviço, crie o banco de dados que será utilizado:

13. Na plataforma Windows, para acessar o terminal do MySQL e criar o banco, deve-se ir em Iniciar > *Programas* > *MySQL* > *MySQL Server 5.x* > *MySQL Command Line Client*.
14. Com o terminal aberto, entre com o seguinte comando: ***mysql –u [usuário_root] –p [senha_usuário-root]***
15. Em seguida, cole o *script* SQL que se encontra no arquivo Nurse_BD.sql que acompanha este material.

As regras de negócio para manipulação do banco de dados já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.dao* do projeto importado. As funcionalidades para manipulação e conversão de objetos JSON e XML encontram-se prontas no pacote *br.ufscar.dc.utils.*

Na classe ***FactoryConnection*** do pacote *br.ufscar.dc.dao*, altere as variáveis ***USER*** e ***PASSWORD***, atribuindo, respectivamente, os dados de usuário e senha locais para acesso ao banco de dados. Exemplo: USER = “root”; PASSWORD=”123456”;.

23. Modelar o Serviço

Para criar o modelo da aplicação, utilize o *plugin* editor de modelos construído na etapa de Engenharia de Domínio da abordagem Reúso com DSM e SOA:

21. Na pasta raiz do projeto *wsNurse* crie uma pasta chamada “*Modelo*” (clique com o botão direito sobre a pasta raiz, opção *New* > *Folder*)
22. Clique com o botão direito sobre a pasta *Modelo* recém-criada e selecione a opção *New* > *Other...*
23. Na janela que se abre, selecione a opção *EmergencyMetamodel Model* da categoria *Example EMF Model Creation Wizards*. Em seguida clique em *Next*.
24. Na próxima janela, nomeie o modelo com o nome *ws_model.emergencymetamodel* Clique em *Next*.
25. Na janela seguinte escolha a opção **Model** para o campo *Model Object*. Em seguida pressione o botão *Finish*. O modelo será criado, tendo como nó raiz o elemento *Model*.
    a. Clique com o botão direito do mouse sobre o elemento *Model* do modelo criado e selecione a opção “Show properties view”. Na caixa de propriedades do elemento *Model*, atribua o nome *wsNurse* para a propriedade “*Project”,* que indica a que projeto o modelo está vinculado.

b. Clique novamente com o botão direito sobre o elemento *Model*, escolha a opção New Child. Em seguida selecione o elemento *Object* que aparece no menu popup que se abre. Será criado um novo elemento *Object* no modelo, filho do elemento *Model*.

c. Agora, clique sobre o elemento Object e adicione o elemento *Nurse*, atribua os seguintes valores:

   i. Package: br.ufscar.dc.beans;

   ii. Stereotype: bean.

d. Adicione um elemento *Attribute* dentro do elemento *Nurse*. Na barra de propriedades, atribua os seguintes valores:

   i. Attribute: age.

e. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “Integer”.

f. Adicione um elemento *Attribute* dentro do elemento *Nurse*. Na barra de propriedades, atribua os seguintes valores:

   i. Attribute: gender.

g. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor “String”.

h. Adicione um elemento *Attribute* dentro do elemento *Nurse*. Na barra de propriedades, atribua os seguintes valores:

   i. Attribute: dateBirth.

i. Adicione um elemento Attribute Type dentro do elemento Attribute. Na barra de propriedades, selecione o valor “Java.sql.Date”.

j. Agora, adicione um elemento *Service* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:

   i. Name: NurseResource;

   ii. Package: *br.ufscar.dc.resources;*

   iii. *Path: contentNurse*

   iv. *Protocol: Rest.*

k. Adicione um elemento *Service Input* ao elemento *Service*. Dentro do elemento *Service Input* recém-criado, adicione um elemento *Service Data*. Na propriedade “Mimetype” deste último, selecione o valor “application/json”.

l. Adicione um elemento *Service Output* ao elemento *Service*. Dentro do elemento *Service Output* recém-criado, adicione um elemento *Service Data*. Na propriedade “Mimetype” deste último, selecione o valor “application/json”.

m. Adicione um elemento *Method* Service ao elemento *Service*. Preencha as propriedades deste elemento da seguinte forma:

   i. Method: listNurses;

   ii. Path: listNurses;

   iii. Type: retrieve.

n. Adicione um elemento Return Method ao elemento Method Service. Na propriedade "Return" deste último selecione o valor "Response".

24. <u>Executar as Transformações Modelo para Código</u>

Dica Importante: sempre salve todas as alterações no modelo antes de executar as transformações, de forma que essas alterações sejam refletidas no código gerado.

17. Clique com o botão direito sobre o modelo do serviço e selecione a opção *Run As > Run Configurations*.
18. Na caixa do lado esquerdo da janela que se abre, dê um clique duplo sobre a opção *JET Transformation*. Do lado direito aparecerão as opções para execução de transformações.
19. No campo *Transformation Input*, clique sobre o botão *Browse...* e selecione o arquivo do modelo do serviço.
20. No campo *ID* selecione a transformação que deseja aplicar sobre o modelo (*br.ufscar.dc.helth.codegeneratorJEEWS*) e em seguida pressione o botão *Run*. A transformação será executada, sendo o código gerado dentro do projeto *wsNurse* (conforme definido na propriedade "Project" do elemento Model).

25. <u>Implementar o Serviço</u>

3. A classe implementada deverá ficar semelhante à mostrada na Figura L.2.

**Ao criar os métodos e atributos indicados neste tutorial, assegure-se de obedecer ao padrão utilizado na nomenclatura adotada, tais como letras maiúsculas, minúsculas, etc.*

```
import java.sql.ResultSet;
@Path("/contentNurse")
@Consumes( { "application/json" })
@Produces( { "application/json" })
public class NurseResource {

    @GET
    @Path("/listNurses")
    public Response listNurses() {
        NurseDAO nurseDAO = new NurseDAO();
        ResultSet rs = nurseDAO.listNurses();
        List<Nurse> listNurses = new ArrayList<Nurse>();
        try {
            while (rs.next()) {
                Nurse n = new Nurse();
                n.setId(rs.getInt(1));
                n.setName(rs.getString(2));
                n.setAge(rs.getInt(3));
                n.setGender(rs.getString(4));
                n.setDateBirth(rs.getDate(5));
                listNurses.add(n);
            }
            return Response.ok(
                    XMLParser.parseObjectToJSON("listNurses", listNurses))
                    .build();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return Response.ok(XMLParser.parseObjectToJSON("listNurses", null))
                .build();
    }
}
```

Figura L.2 - Código da Classe NurseResource

26. Testar o Serviço

9. Após a implementação do serviço, teste-o no browser. Para isto, clique com o botão direito sobre a pasta raiz do projeto wsNurse e escolha a opção *Run As… > Run on Server.*

10. O browser será aberto. Teste a operação para listar as enfermeiras registradas no banco de dados, digitando a seguinte URL: http://localhost:8080/wsNurse/resources/contentNurse/listNurses

II - DESENVOLVIMENTO DA APLICAÇÃO MÓVEL

13. Importar o projeto parcial da aplicação móvel no Eclipse

9. Importe o projeto *AndroExpC* que acompanha este material. Siga os mesmos passos do passo "Importar o projeto parcial do Web Service no Eclipse" apresentado anteriormente.

10. Corrija quaisquer erros de importação de bibliotecas referentes à SDK do Android. Vá ao menu *Window > Preferences*. Na janela que se abre, selecione na parte esquerda a opção "Android". Caso o campo SDK Location esteja vazio, clique no botão "Browse…" e indique a localização da pasta "android-sdk" que se encontra no diretório do Eclipse. Após selecionar a pasta, clique em OK.

    A estrutura do projeto importado será como a **Figura L.3** abaixo:

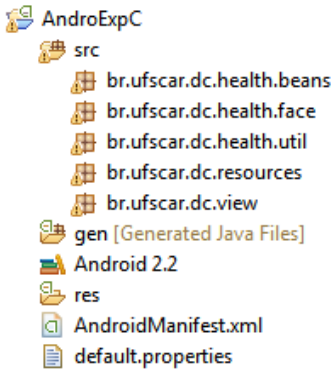

Figura L.3 - Estrutura do projeto Android importado

As interfaces gráficas da aplicação móvel já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.health.face* e *br.ufscar.dc.view* do projeto importado. As funcionalidades para formatação de objetos de dados do tipo Date encontram-se prontas no pacote *br.ufscar.dc.health.util.*

14. Modelar o Serviço

Para criar o modelo da aplicação, utilize o *plugin* editor de modelos construído na etapa de Engenharia de Domínio da abordagem Reúso com DSM e SOA:

26. Na pasta raiz do projeto *AndroExpC* crie uma pasta chamada "*Modelo*" (clique com o botão direito sobre a pasta raiz, opção *New > Folder*)

27. Clique com o botão direito sobre a pasta *Modelo* recém-criada e selecione a opção *New > Other...*

28. Na janela que se abre, selecione a opção *EmergencyMetamodel Model* da categoria *Example EMF Model Creation Wizards*. Em seguida clique em *Next*.

29. Na próxima janela, nomeie o modelo com o nome *android_model.emergencymetamodel* Clique em *Next*.
30. Na janela seguinte escolha a opção **Model** para o campo *Model Object*. Em seguida pressione o botão *Finish*. O modelo será criado, tendo como nó raiz o elemento *Model*.
    a. Clique com o botão direito do mouse sobre o elemento *Model* do modelo criado e selecione a opção "Show properties view". Na caixa de propriedades do elemento *Model*, atribua o nome *AndroExpC* para a propriedade "*Project",* que indica a que projeto o modelo está vinculado.
    b. Clique novamente com o botão direito sobre o elemento *Model*, escolha a opção New Child. Em seguida selecione o elemento *Object* que aparece no menu popup que se abre. Será criado um novo elemento *Object* no modelo, filho do elemento *Model*.
    c. Adicione um elemento *Disease* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:
        i. Package: br.ufscar.dc.health.beans;
        ii. Stereotype: bean.
    d. Adicione um elemento *Attribute* dentro do elemento *Disease*. Na barra de propriedades, atribua os seguintes valores:
        i. Attribute: treatment.
    e. Adicione um elemento *Attribute Type* dentro do elemento *Attribute*. Na barra de propriedades, selecione o valor "String".
    f. Adicione um elemento *Symptom* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:
        i. Package: br.ufscar.dc.health.beans;
        ii. Stereotype: bean.
    g. Clique novamente com o botão direito sobre o elemento *Model*, escolha a opção New Child. Em seguida selecione o elemento *Reuse Set* que aparece no menu popup que se abre. Será criado um novo elemento chamado *Reuse Set* no modelo, filho do elemento *Model*.
        i. O elemento Reuse Set contém os serviços juntamente com seus métodos do Domínio Healthcare para reúso pela aplicações.
    h. Adicione um elemento *Class* dentro do elemento *Object*. Na barra de propriedades, atribua os seguintes valores:
        i. Name: DiseaseResource;
        ii. Package: br.ufscar.dc.resources;
        iii. Stereotype: reuse.
    i. Adicione um elemento *Method* dentro do element *Class* recém criado. Na barra de propriedades, atribua os seguintes valores:
        i. Has Operation Service: Listdiseases listDiseases;
        ii. Has Service Reuse: Lookupdisease LookupDisease;
        iii. Method: listDiseases;

iv. Stereotype: reuse.

j. Adicione um elemento *Return Type* dentro do elemnto Method recém criado. Na barra de propriedades, selecione para a propriedade *ReturnType* o valor “String”.

k. Adicione um elemento *Method* dentro do element *Class*. Na barra de propriedades, atribua os seguintes valores:

i. Has Operation Service: Listsymptomstodisease listSymptomsToDisease;

ii. Has Service Reuse: Lookupdisease LookupDisease;

iii. Method: listSymptomsToDisease;

iv. Stereotype: reuse.

l. Adicione um elemento *Return Type* dentro do elemnto Method recém criado. Na barra de propriedades, selecione para a propriedade *ReturnType* o valor “String”.

m. Adicione um elemento *Parameter* dentro do elemnto Method recém criado. Na barra de propriedades, atribua o seguinte valore:

i. Parameter: idDisease

n. Adicione um elemento *Parameter Type* dentro do elemnto *Parameter* recém criado. Na barra de propriedades, selecione para a propriedade *Parametertype* o valor “Integer”.

15. <u>Executar as Transformações Modelo para Código</u>

Dica Importante: sempre salve todas as alterações no modelo antes de executar as transformações, de forma que essas alterações sejam refletidas no código gerado.

21. Clique com o botão direito sobre o modelo do serviço e selecione a opção *Run As > Run Configurations*.

22. Na caixa do lado esquerdo da janela que se abre, dê um clique duplo sobre a opção *JET Transformation*. Do lado direito aparecerão às opções para execução de transformações.

23. No campo *Transformation Input*, clique sobre o botão *Browse...* e selecione o arquivo do modelo do serviço (*android_model.emergencymetamodel*).

24. No campo *ID* selecione a transformação que deseja aplicar sobre o modelo (*br.ufscar.dc.helth.generatorAndroid2*) e em seguida pressione o botão *Run*. A transformação será executada, sendo o código gerado dentro do projeto AndroExpC (conforme definido na propriedade “Project” do elemento *Model*).

A classe *DiseaseResource* implementada deverá ficar semelhante à mostrada na Figura J.4.

```
public class DiseaseResource {
    public String listDiseases() {
        DefaultHttpClient httpclient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/lookupDisease"
                        + "/listDiseases");
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler responseHandler = new BasicResponseHandler();
        try {
            return httpclient.execute(httpGet, responseHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    public String listSymptomsToDisease(Integer idDisease) {
        DefaultHttpClient httpclient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/lookupDisease"
                        + "/listSymptomsToDisease/" + idDisease);
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler responseHandler = new BasicResponseHandler();
        try {
            return httpclient.execute(httpGet, responseHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Figura L.4 - Código da Classe DiseaseResource

No pacote *br.ufscar.dc.resources* abra a classe NurseResource, na linha:

```
HttpGet httpGet = new HttpGet("");
```

c. Adicione a URL do serviço criado no projeto wsNurse;

16. <u>Testar a aplicação móvel com o emulador do Android</u>

8. Após a implementação da aplicação, teste-a no emulador do Android. Para isto, clique com o botão direito sobre a pasta raiz do projeto AndroExpC e escolha a opção *Run As… > Android Application.*

    a. Caso ainda não exista um dispositivo emulador configurado, o Eclipse emitirá um alerta para a criação de um novo *Android Virtual Device (ADV)*. Neste caso clique em Yes.

    b. Na janela que se abre, clique em “New” para criar um novo ADV. Em seguida dê um nome qualquer para este dispositivo. No campo “Target” selecione “Android 2.2 – APILevel8”. Mantenha as demais configurações como estão e clique no botão “Create ADV”.

    c. Feche todas as janelas referentes à criação do ADV. Em seguida, clique novamente com o botão direito sobre a pasta raiz do projeto AndroExpC e escolha a opção *Run As… > Android Application.* Desta vez, o Eclipse executará a aplicação no ADV recém-criado.

9. Teste a aplicação, verifique se a mesma está funcionando corretamente, trazendo a lista de doenças, lista de sintomas e enfermeiras(os).

Apêndice M

MATERIAL DE APOIO DA APLICAÇÃO C – GRUPO TRADICIONAL

I - DESENVOLVIMENTO DO SERVIÇO

27. <u>Importar o projeto parcial do Web Service no Eclipse</u>

Importe o projeto *wsNurse* que acompanha este material. Siga os seguintes passos:

31. No IDE Eclipse vá no menu *File* e escolha a opção *Import*...
32. Na janela que se abre, vá à categoria *General* e selecione a opção *Existing Projects Into Workspace*. Pressione o botão *next*.
33. Na próxima janela, clique no botão *Browse* e selecione a pasta raiz do projeto *wsAmbulance* salvo em seu computador.
34. Nessa mesma janela selecione a opção *Copy Projects Into Workspace*.
35. Finalmente, pressione o botão *Finish*. O IDE Eclipse importará o projeto Web de nome *wsAmbulance* ao *workspace* atual.
36. Corrija quaisquer erros de importação de bibliotecas. Clique com o botão direito sobre a pasta raiz do projeto. Selecione a opção Build *Path > Configure Build Path...* Na janela que se abre remova as bibliotecas conflitantes e adicione novamente (Bibliotecas com possíveis erros: *Apache TomCat, MySQL*, biblioteca do *RESTful*).

A estrutura do projeto importado será como a **Figura M.1** abaixo:

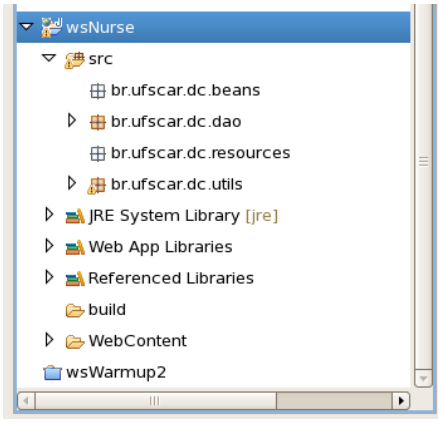

Figura M.1 - Estrutura do projeto importado

28. <u>Criar o banco de dados acessado pelo Serviço</u>

Antes de iniciar a implementação do serviço, crie o banco de dados que será utilizado:

16. Na plataforma Windows, para acessar o terminal do MySQL e criar o banco, deve-se ir em Iniciar *> Programas > MySQL > MySQL Server 5.x > MySQL Command Line Client*.
17. Com o terminal aberto, entre com o seguinte comando: ***mysql –u [usuário_root] –p [senha_usuário-root]***
18. Em seguida, cole o *script* SQL que se encontra no arquivo Nurse_BD.sql que acompanha este material.

As regras de negócio para manipulação do banco de dados já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.dao* do projeto importado*.* As funcionalidades para manipulação e conversão de objetos JSON e XML encontram-se prontas no pacote *br.ufscar.dc.utils.*

Na classe ***FactoryConnection*** do pacote *br.ufscar.dc.dao*, altere as variáveis ***USER*** e ***PASSWORD***, atribuindo, respectivamente, os dados de usuário e senha locais para acesso ao banco de dados. <u>Exemplo</u>: USER = “root”; PASSWORD=”123456”;.

29. <u>Implementar o Serviço</u>

11. No pacote *br.ufscar.dc.beans* crie a classe *Nurse*.
12. Após a criação da classe, adicione as propriedades *id* (int), *name* (String), *age* (int), *gender* (String) e *dateBirth* (java.util.Date)*.
13. Crie os métodos *get* e *set* para cada uma das propriedades.
14. Em seguida, no pacote *br.ufscar.dc.resources* crie a classe *NurseResource* que representa o serviço associado à entidade Nurse.
15. Na classe *NurseResource* adicione as anotações do RESTful necessárias *(@Path, @Consumes, @Produces*, etc), bem como adicione o método (nomeie-o como *listNurses*) que recuperará a lista de enfermeiras registradas no banco de dados. Para nomear as anotações, utilize os seguintes valores:

Classe *NurseResource*:

```
@Path(“contentNurse”)
@Produces(“{application/json}”)
@Consumes(“{application/json}”)
```

Método *listNurses*:

```
@GET
@Path(“listNurses”)
```

A classe implementada deverá ficar semelhante à mostrada na Figura M.2.

**Ao criar os métodos e atributos indicados neste tutorial, assegure-se de obedecer ao padrão utilizado na nomenclatura adotada, tais como letras maiúsculas, minúsculas, etc.*

30. <u>Testar o Serviço</u>

11. Após a implementação do serviço, teste-o no browser. Para isto, clique com o botão direito sobre a pasta raiz do projeto wsNurse e escolha a opção *Run As… > Run on Server.*

12. O browser será aberto. Teste a operação para listar as enfermeiras registradas no banco de dados, digitando a seguinte URL:
http://localhost:8080/wsNurse/resources/contentNurse/listNurses

```
import java.sql.ResultSet;
@Path("/contentNurse")
@Consumes( { "application/json" })
@Produces( { "application/json" })
public class NurseResource {

    @GET
    @Path("/listNurses")
    public Response listNurses() {
        NurseDAO nurseDAO = new NurseDAO();
        ResultSet rs = nurseDAO.listNurses();
        List<Nurse> listNurses = new ArrayList<Nurse>();
        try {
            while (rs.next()) {
                Nurse n = new Nurse();
                n.setId(rs.getInt(1));
                n.setName(rs.getString(2));
                n.setAge(rs.getInt(3));
                n.setGender(rs.getString(4));
                n.setDateBirth(rs.getDate(5));
                listNurses.add(n);
            }
            return Response.ok(
                    XMLParser.parseObjectToJSON("listNurses", listNurses))
                    .build();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return Response.ok(XMLParser.parseObjectToJSON("listNurses", null))
                .build();
    }
}
```

Figura M.2 - Código da Classe NurseResource

II - DESENVOLVIMENTO DA APLICAÇÃO MÓVEL

17. Importar o projeto parcial da aplicação móvel no Eclipse

11. Importe o projeto *AndroExpC* que acompanha este material. Siga os mesmos passos do passo "Importar o projeto parcial do Web Service no Eclipse" apresentado anteriormente.

12. Corrija quaisquer erros de importação de bibliotecas referentes à SDK do Android. Vá ao menu *Window > Preferences*. Na janela que se abre, selecione na parte esquerda a opção "Android". Caso o campo SDK Location esteja vazio, clique no botão "Browse…" e indique a localização da pasta "android-sdk" que se encontra no diretório do Eclipse. Após selecionar a pasta, clique em OK.

A estrutura do projeto importado será como a **Figura M.3** abaixo:

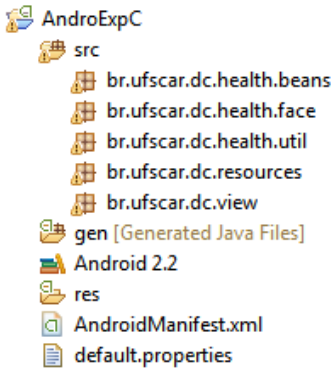

Figura M.3 - Estrutura do projeto Android importado

As interfaces gráficas da aplicação móvel já encontram-se implementadas nos pacotes de código fonte *br.ufscar.dc.health.face* e *br.ufscar.dc.view* do projeto importado. As funcionalidades para formatação de objetos de dados do tipo Date encontram-se prontas no pacote *br.ufscar.dc.health.util.*

18. Implementar os beans

11. No pacote *br.ufscar.dc.health.beans* crie os *beans Disease* e *Symptom* associadas à **Aplicação C.** Note que neste pacote já existe o *bean* referente à entidade *Nurse*.
12. Após a criação dos beans, adicione as propriedades*:
    a. *Disease*: id (int), name (String), cid (String), description (String), e treatment (String).
    b. Symptom: id (int), e description (String).
13. Crie os métodos *get* e *set* para cada uma das propriedades.

**Ao criar os métodos e atributos indicados neste tutorial, assegure-se de obedecer ao padrão utilizado na nomenclatura adotada, tais como letras maiúsculas, minúsculas, etc.*

14. Implementar as classes que chamam o serviço

3. No pacote *br.ufscar.dc.resources* crie a classe *DiseaseResource* que consome o serviço;
    a. Crie o método *listDiseases*, o qual consome o serviço do Domínio Healthcare já implementado. Para acesso ao serviço, utilize a seguinte URL: http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/lookupDisease/listDiseases .
    b. Crie o método *listSymptomsToDisease*, o qual consome o serviço do Domínio Healthcare já implementado. Para acesso ao serviço, utilize a seguinte URL: http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/lookupDisease/listSymptomsToDisease/idDisease .

A classe implementada deverá ficar semelhante à mostrada na Figura M.4.

```
public class DiseaseResource {
    public String listDiseases() {
        DefaultHttpClient httpclient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/lookupDisease"
                        + "/listDiseases");
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler responseHandler = new BasicResponseHandler();
        try {
            return httpclient.execute(httpGet, responseHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    public String listSymptomsToDisease(Integer idDisease) {
        DefaultHttpClient httpclient = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(
                "http://webservice.ges.dc.ufscar.br:8090/wsEmergencyHealthcare/resources/lookupDisease"
                        + "/listSymptomsToDisease/" + idDisease);
        httpGet.setHeader("Accept", "application/json");
        httpGet.setHeader("Content-type", "application/json");
        ResponseHandler responseHandler = new BasicResponseHandler();
        try {
            return httpclient.execute(httpGet, responseHandler);
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Figura M.4 - Código da Classe *DiseaseResource*

15. <u>Testar a aplicação móvel com o emulador do Android</u>

10. Após a implementação da aplicação, teste-a no emulador do Android. Para isto, clique com o botão direito sobre a pasta raiz do projeto AndroExpC e escolha a opção *Run As… > Android Application.*

    a. Caso ainda não exista um dispositivo emulador configurado, o Eclipse emitirá um alerta para a criação de um novo *Android Virtual Device (ADV)*. Neste caso clique em Yes.

    b. Na janela que se abre, clique em "New" para criar um novo ADV. Em seguida dê um nome qualquer para este dispositivo. No campo "Target" selecione "Android 2.2 – APILevel8". Mantenha as demais configurações como estão e clique no botão "Create ADV".

    c. Feche todas as janelas referentes à criação do ADV. Em seguida, clique novamente com o botão direito sobre a pasta raiz do projeto AndroExpC e escolha a opção *Run As… > Android Application.* Desta vez, o Eclipse executará a aplicação no ADV recém-criado.

11. Teste a aplicação, verifique se a mesma está funcionando corretamente, trazendo a lista de doenças, lista de sintomas e enfermeiras(os).

Apêndice N

FORMULÁRIO DE COLETA DE DADOS – GRUPO DA ABORDAGEM DE REÚSO COM DSM E SOA

Grupo nº: ____

Anote nas tabelas abaixo os horários de início e fim de cada atividade realizada.

CONSTRUÇÃO DOS SERVIÇOS PARA REÚSO:

Atividade	Hora início	Hora Fim
Projeto (modelagem, decisões de tecnologias, padrões e frameworks adicionais que o grupo julgar necessário).	___:___h	___:___h
Implementação (codificação do serviço).	___:___h	___:___h
Testes (chamada do serviço no browser).	___:___h	___:___h

CONSTRUÇÃO DA APLICAÇÃO MÓVEL QUE CONSOME O SERVIÇO:

Atividade	Hora início	Hora Fim
Projeto (modelagem, decisões de tecnologias, padrões e frameworks adicionais que o grupo julgar necessário).	___:___h	___:___h
Implementação (codificação da aplicação móvel).	___:___h	___:___h
Testes (teste da aplicação móvel no emulador do Android).	___:___h	___:___h

LINHAS DE CÓDIGO IMPLEMENTADAS:

Anote as quantidades de linha de código geradas automaticamente e manualmente. Considere como linhas de código geradas automaticamente todo código pronto que é criado pelo IDE (exemplo: *geração automática de métodos get e set; criação automatizada de blocos try-catch*), bem como o código que é gerado pelas transformações Modelo-para-Código.

DICA: *Para a contagem do código gerado de forma automática efetue a contagem logo após a aplicação das transformações Modelo para Código. Em seguida, complemente o código do serviço ou da aplicação até sua finalização. Realize novamente a contagem. A diferença entre a contagem atual e a contagem anterior será a quantidade de linhas de código implementadas manualmente.*

⇨ ***CÓDIGO IMPLEMENTADO NA CONSTRUÇÃO DO SERVIÇO:***

Linhas de Código geradas **automaticamente**[*]: _____	[*]Inclua na contagem apenas o código implementado pelo grupo. Não inclua o código já pronto no projeto parcial importado ao seu ambiente de desenvolvimento.
Linhas de Código implementadas **manualmente**[*]: _____	

⇨ ***CÓDIGO IMPLEMENTADO NA CONSTRUÇÃO DA APLICAÇÃO MÓVEL:***

Linhas de Código geradas **automaticamente**[*]: _____	[*]Inclua na contagem apenas o código implementado pelo grupo. Não inclua o código já pronto no projeto parcial importado ao seu ambiente de desenvolvimento.
Linhas de Código implementadas **manualmente**[*]: _____	

Anote os problemas técnicos encontrados durante a execução do experimento, indicando o horário de identificação e resolução do problema, bem como sua breve descrição.

Descrição do Problema	**Hora de Identificação**	**Hora de Resolução**
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h

Apêndice O

FORMULÁRIO DE COLETA DE DADOS – GRUPO DO PROCESSO TRADICIONAL

Grupo nº: ____

Anote nas tabelas abaixo os horários de início e fim de cada atividade realizada.

CONSTRUÇÃO DOS SERVIÇOS PARA REÚSO:

Atividade	Hora início	Hora Fim
Projeto (modelagem, decisões de tecnologias, padrões e frameworks adicionais que o grupo julgar necessário).	___:___h	___:___h
Implementação (codificação do serviço).	___:___h	___:___h
Testes (chamada do serviço no browser).	___:___h	___:___h

CONSTRUÇÃO DA APLICAÇÃO MÓVEL QUE CONSOME O SERVIÇO:

Atividade	Hora início	Hora Fim
Projeto (modelagem, decisões de tecnologias, padrões e frameworks adicionais que o grupo julgar necessário).	___:___h	___:___h
Implementação (codificação da aplicação móvel).	___:___h	___:___h
Testes (teste da aplicação móvel no emulador do Android).	___:___h	___:___h

LINHAS DE CÓDIGO IMPLEMENTADAS:

Anote as quantidades de linha de código geradas automaticamente e manualmente. Considere como linhas de código geradas automaticamente todo código pronto que é criado pelo IDE (exemplo: *geração automática de métodos get e set; criação automatizada de blocos try-catch*) ou por qualquer mecanismo de geração automática de código que possa ter sido empregado pelo grupo.

- ⇨ ***CÓDIGO IMPLEMENTADO NA CONSTRUÇÃO DO SERVIÇO:***

Linhas de Código geradas **automaticamente**[*]: _____	[*]Inclua na contagem apenas o código implementado pelo grupo. Não inclua o código já pronto no projeto parcial importado ao seu ambiente de desenvolvimento.
Linhas de Código implementadas **manualmente**[*]: _____	

⇨ ***CÓDIGO IMPLEMENTADO NA CONSTRUÇÃO DA APLICAÇÃO MÓVEL:***

Linhas de Código geradas **automaticamente**[*]: _____	[*]Inclua na contagem apenas o código implementado pelo grupo. Não inclua o código já pronto no projeto parcial importado ao seu ambiente de desenvolvimento.
Linhas de Código implementadas **manualmente**[*]: _____	

Anote os problemas técnicos encontrados durante a execução do experimento, indicando o horário de identificação e resolução do problema, bem como sua breve descrição.

Descrição do Problema	**Hora de Identificação**	**Hora de Resolução**
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h
	___:___h	___:___h

Apêndice P

FORMULÁRIO DE AVALIAÇÃO – GRUPO DA ABORDAGEM DE REÚSO COM DSM E SOA

Grupo nº: ____ RA: _____

1) Você considera que a aplicação da abordagem Reúso com DSM e SOA auxiliou no desenvolvimento de serviços e aplicações móveis? Justifique.	☐ Sim ☐ Não ☐ Parcialmente

2) Você sentiu dificuldades na realização das atividades da abordagem? Especifique.	☐ Sim ☐ Não ☐ Parcialmente

3) Quais sugestões você teria para melhorar a abordagem?

4) Você acredita que o uso da linguagem específica de domínio usada para a modelagem facilitaram o desenvolvimento de serviços/aplicações móveis para um domínio? Justifique.	☐ Sim ☐ Não ☐ Parcialmente

5)Você considera viável, sob o ponto de vista de esforço e eficiência no desenvolvimento de aplicações móveis, reutilizar componentes de software pré-fabricados? Justifique.	☐ Sim ☐ Não ☐ Parcialmente

Obrigado por sua colaboração!

Apêndice Q

FORMULÁRIO DE AVALIAÇÃO – GRUPO DA ABORDAGEM TRADICIONAL

Grupo nº: ____ RA: _____

1) Você utilizou algum mecanismo e/ou ferramenta para suporte à geração automática dos serviços/aplicações móveis? Esse mecanismo/ferramenta atendeu satisfatoriamente suas necessidades?

2) Você sentiu dificuldades na realização das atividades para o desenvolvimento dos serviços/aplicações móveis? Especifique.	☐ Sim ☐ Não ☐ Parcialmente

3) Você considera que uma abordagem que forneça um roteiro e mecanismos de apoio especificamente voltados para o desenvolvimento de serviços/aplicações móveis facilitaria sua tarefa? Justifique.	☐ Sim ☐ Não ☐ Parcialmente

4) Você acha que é viável, sob o ponto de vista de esforço e eficiência no desenvolvimento de aplicações móveis, construir os serviços e aplicações a partir do zero, ou seja, sem reúso de software? Justifique.	☐ Sim ☐ Não ☐ Parcialmente

Obrigado por sua colaboração!

Anexo A

TABELA PADRÃO DE DISTRIBUIÇÃO DE PROBABILIDADE ESTATÍSTICA *T* DE *STUDENT**

Área contida nas duas caudas laterais (bicaudal) da distribuição *t* de *Student*									
gl/α		Graus de Significância							
		0,9900	0,9500	0,9000	0,1000	0,4400	0,3450	0,0850	0,0150
Graus de Liberdade	1	0,0157	0,0787	0,1584	6,3138	1,2088	1,6610	7,4451	42,4335
	2	0,0141	0,0708	0,1421	2,9200	0,9559	1,2259	3,2073	8,0728
	3	0,0136	0,0681	0,1366	2,3534	0,8879	1,1181	2,5357	5,0473
	4	0,0133	0,0667	0,1338	2,1318	0,8565	1,0697	2,2777	4,0880
	5	0,0132	0,0659	0,1322	2,0150	0,8385	1,0424	2,1429	3,6338
	6	0,0131	0,0654	0,1311	1,9432	0,8268	1,0248	2,0605	3,3723
	7	0,0130	0,0650	0,1303	1,8946	0,8186	1,0125	2,0049	3,2032
	8	0,0129	0,0647	0,1297	1,8595	0,8125	1,0035	1,9650	3,0851
	9	0,0129	0,0645	0,1293	1,8331	0,8079	0,9966	1,9349	2,9982
	10	0,0129	0,0643	0,1289	1,8125	0,8042	0,9911	1,9115	2,9316
	11	0,0128	0,0642	0,1286	1,7959	0,8012	0,9867	1,8927	2,8789
	12	0,0128	0,0640	0,1283	1,7823	0,7987	0,9830	1,8772	2,8363
	13	0,0128	0,0639	0,1281	1,7709	0,7966	0,9799	1,8644	2,8010
	14	0,0128	0,0638	0,1280	1,7613	0,7948	0,9773	1,8535	2,7714
	15	0,0127	0,0638	0,1278	1,7531	0,7933	0,9750	1,8442	2,7462
	16	0,0127	0,0637	0,1277	1,7459	0,7919	0,9731	1,8361	2,7245
	17	0,0127	0,0636	0,1276	1,7396	0,7907	0,9713	1,8290	2,7056
	18	0,0127	0,0636	0,1274	1,7341	0,7897	0,9698	1,8227	2,6889
	19	0,0127	0,0635	0,1274	1,7291	0,7887	0,9684	1,8172	2,6742
	20	0,0127	0,0635	0,1273	1,7247	0,7879	0,9672	1,8122	2,6611
	21	0,0127	0,0635	0,1272	1,7207	0,7871	0,9661	1,8077	2,6493
	22	0,0127	0,0634	0,1271	1,7171	0,7864	0,9651	1,8036	2,6387
	23	0,0127	0,0634	0,1271	1,7139	0,7858	0,9641	1,8000	2,6290
	24	0,0127	0,0634	0,1270	1,7109	0,7852	0,9633	1,7966	2,6203
	25	0,0127	0,0633	0,1269	1,7081	0,7847	0,9625	1,7935	2,6122

**"Student's" Collected Papers (Ed. Egar S. Pearson e J. Wishart, University College, Londres, 1942).*