Projeto e implementação de um gerador automático
de serviços web a partir de diagramas de classes

Domenico Schettini Filho

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Domenico Schettini Filho

Projeto e implementação de um gerador automático de serviços web a partir de diagramas de classes

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Profa. Dra. Rosana Teresinha Vaccare Braga

USP – São Carlos
Maio de 2016

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

F327p Filho, Domenico Schettini
Projeto e implementação de um gerador automático
de serviços web a partir de diagramas de classes
/ Domenico Schettini Filho; orientadora Rosana
Teresinha Vaccare Braga. - São Carlos - SP, 2016.
185 p.

Dissertação (Mestrado - Programa de Pós-Graduação
em Ciências de Computação e Matemática Computacional)
- Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2016.

1. Serviços. 2. Reúso. 3. Unified Modeling
Language. 4. Geradores de Código. 5. Model-Driven
Development e Service Oriented Architecture. I.
Braga, Rosana Teresinha Vaccare, orient. II. Título.

Domenico Schettini Filho

Design and implementation of an automatic generator of
web services from class diagrams

Master dissertation submitted to the Instituto de
Ciências Matemáticas e de Computação – ICMC-
USP, in partial fulfillment of the requirements for the
degree of the Master Program in Computer Science
and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Profa. Dra. Rosana Teresinha Vaccare Braga

USP – São Carlos
May 2016

Dedico esse trabalho a minha família e amigos.

AGRADECIMENTOS

Eu agradeço ao apoio de toda minha família e amigos, em especial a minha mãe Marli Attie Schettini, a meu pai Domenico Schettini, a meu irmão Lucas Schettini e a meus amigos e colegas de mestrado: Karen, Thiago, Daniel e Iohan. Além de agradecimento muito especial a minha querida orientadora Prof. Dr. Rosana Teresinha Vaccare Braga, pela paciência e dedicação no desenvolvimento desse trabalho.

*“O homen é do tamanho do seu sonho”
(Fernando Pessoa)*

RESUMO

SCHETTINI FILHO, D.. **Projeto e implementação de um gerador automático de serviços web a partir de diagramas de classes**. 2016. 185 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

A indústria de desenvolvimento de software está em constante evolução e em busca de novos desafios, como por exemplo: novas tecnologias, linguagens de programação e estratégias para aumentar a produtividade no desenvolvimento. Com o passar do tempo, essa evolução desejada tem se concretizado, em especial com o surgimento de conceitos como *Service Oriented Architecture* (SOA) e *Model-Driven Development* (MDD). Nesse contexto, ganha importância a reutilização de software, por exemplo por meio do uso de serviços que facilitam a interoperabilidade entre diferentes sistemas. Ambientes de desenvolvimento de software, em especial ambientes de reuso de software, poderiam se adequar à arquitetura SOA para facilitar a disponibilização de serviços a outros ambientes, promovendo sua integração e aumentando a possibilidade de reuso. Após um levantamento bibliográfico sobre ambientes integrados de reuso que utilizassem serviços para permitir o compartilhamento de recursos, percebeu-se que os trabalhos nessa direção são poucos. Ao mesmo tempo, observou-se que para integração de sistemas por meio de serviços é necessário criar camadas de acesso às entidades dos sistemas que compartilharão informações. A implementação desses serviços de forma manual é repetitiva e sujeita a erros, principalmente por programadores que não estão familiarizados com SOA. Por outro lado, diagramas de classes em UML (*Unified Modeling Language*) são de amplo conhecimento por parte de desenvolvedores e estão em um nível de abstração que é suficientemente detalhado para permitir a derivação de serviços de acesso às suas classes. Desta forma, para atender uma demanda crescente de sistemas que precisam oferecer serviços para possibilitar sua integração com outros sistemas, este mestrado tem por objetivo promover a geração automatizada de código de serviços por meio de transformações de modelo para texto, em que o modelo de entrada é um diagrama de classes derivado da UML e os serviços gerados cobrem operações básicas do tipo CRUD. O gerador resultante foi validado por meio de duas provas de conceito e mostrou-se adequado para cumprir os objetivos estabelecidos, visto que o desenvolvedor trabalha com um modelo em alto nível de abstração, não precisando conhecer os detalhes inerentes à implementação dos serviços.

Palavras-chave: Serviços, Reuso, Unified Modeling Language, Geradores de Código, Model-Driven Development e Service Oriented Architecture.

ABSTRACT

SCHETTINI FILHO, D.. **Projeto e implementação de um gerador automático de serviços web a partir de diagramas de classes**. 2016. 185 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

The software development industry is constantly evolving and looking for challenges, such as: new technologies, programming languages and strategies to increase developers productivity. During the course of time, this required evolution has been concretized, especially with the emergency of concepts such as *Service Oriented Architecture* (SOA) and *Model-Driven Development* (MDD). In this context, software reuse gains importance, for example through the use of services that ease the interoperability of different systems. Software development environments, in special reuse environments, could be adapted to the SOA architecture to ease the availability of services to other environments, thus promoting its integration and possibility of reuse. After a literature review about integrated reuse environments that use services to allow resources sharing, it has been observed that there are only a few works in this direction. At the same time, it was observed that for system integration through services it is required to create an access layer for the system entities that share information. The manual implementation of these services is repetitive and error prone, especially for programmers who are not familiar with SOA. On the other hand, UML (*Unified Modeling Language*) class diagrams are widely known by developers and are at a level of abstraction that is sufficiently detailed to allow the derivation of access services to their classes. Thus, to meet a growing demand for systems that need to provide services to enable its integration with other systems, this master's dissertation aims to promote the automated generation of service code through transformations from model to text, where the input model is a class diagram derived from UML, and the generated services cover the basic CRUD operations. The resulting generator has been validated through two proofs of concepts and was adequate to meet the stated objectives, as the developer works with a model in a high abstraction level, and does not need to know the details related with the service implementation.

Key-words: Services, Software Reuse, Reuse, Unified Modeling Language, Code Generator, Model-Driven Development and Service Oriented Architecture.

LISTA DE ILUSTRAÇÕES

Figura 1 – Modelo de SOA em função de seus elementos	33
Figura 2 – Visão lógica do modelo de arquitetura <i>Web services</i>	37
Figura 3 – Modelo arquitetural da AIREs.	44
Figura 4 – <i>Status</i> da Revisão ao final da primeira seleção.	53
Figura 5 – Cenário da distribuição do número de estudos em cada base de pesquisa. . .	54
Figura 6 – Service-Oriented Asset Repository Open Architecture.	56
Figura 7 – Asset Repository Sharing Information Model.	57
Figura 8 – Basic Architecture of a Congregation Engine.	59
Figura 9 – Descrição da arquitetura IPCASCI para composição de processos de negócio (serviços).	61
Figura 10 – Estágios da arquitetura IPCASCI para obtenção do serviço final do proces- sado de descoberta e composição.	62
Figura 11 – Visão alternativa do processo de obtenção da solução final para o serviço pela arquitetura IPCASCI.	63
Figura 12 – Arquitetura do CBD <i>workbench</i>	64
Figura 13 – Modelo arquitetural do CBD <i>workbench</i> adaptado para serviços.	65
Figura 14 – Transformações de códigos executadas pelo modelo.	66
Figura 15 – Ferramentas de Mineração de Serviço.	68
Figura 16 – Visão da definição de interface.	68
Figura 17 – Parte do Metamodelo WSDL 2.0 proposto.	74
Figura 18 – Principais tecnologias adotadas no trabalho.	78
Figura 19 – Diagrama UML utilizado na prova de conceito.	84
Figura 20 – Diagrama <i>Ecore</i> baseado no diagrama de classes da prova de conceito . . .	85
Figura 21 – Metamodelo <i>Ecore</i>	85
Figura 22 – Representação do funcionamento do gerador.	87
Figura 23 – Modelo UML usado no diagrama de classes UML.	109
Figura 24 – Projeto <i>Web</i> dinâmico.	118
Figura 25 – Convertendo projeto <i>Web</i> dinâmico para projeto <i>Maven</i>	119
Figura 26 – Estruturação do Projeto Aires_Server.	119
Figura 27 – Diagrama de classes desenvolvido na segunda prova de conceito.	124
Figura 28 – Modelo hierárquico baseado no diagrama de classes proposto para a segunda prova de conceito.	125
Figura 29 – XMLs de requisição e resposta da operação createPessoa.	126

Figura 30 – XMLs de requisição e resposta da operação retrieveAllPessoa.	127
Figura 31 – Conteúdo da tabela Pessoa apresentado no ambiente <i>MySQL Workbench</i> . . .	128
Figura 32 – Instalação do Acceleo via Marketplace.	141
Figura 33 – Instalação do WTP.	142
Figura 34 – Configuração do Apache CXF no Eclipse.	143
Figura 35 – Configuração do Servidor para publicação dos serviços.	144
Figura 36 – Configuração da plataforma de execução de serviços.	145
Figura 37 – Criação de um novo projeto Acceleo primeira tela.	146
Figura 38 – Criação de um novo projeto Acceleo segunda tela.	147
Figura 39 – Criação de um novo projeto Acceleo terceira tela.	148
Figura 40 – Arquivo mtl após a criação do projeto Acceleo.	149
Figura 41 – Exemplo de Runtime Configuration para um projeto Acceleo.	149
Figura 42 – Conteúdo da pasta src do projeto Acceleo.	150
Figura 43 – Pasta META-INF.	150
Figura 44 – Pasta bin adicionada ao classpath do arquivo META-INF.MF.	151
Figura 45 – Criando um projeto Papyrus.	152
Figura 46 – Definição do tipo de Diagrama.	153
Figura 47 – Nomeando o projeto.	153
Figura 48 – Definição do tipo de diagrama UML.	154
Figura 49 – Dynamic Web Project.	155
Figura 50 – Convertendo Dynamic Web Project em um Projeto Maven.	156
Figura 51 – Criando uma source folder.	156
Figura 52 – Criando a pasta wsdl.	157
Figura 53 – Copiando o arquivo persistence.xml.	158
Figura 54 – Copiando os arquivos Java.	159
Figura 55 – Maven Run Configurations.	160

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Template para geração dos <i>Beans</i>	88
Código-fonte 2 – Template para geração dos <i>Imports</i>	89
Código-fonte 3 – Template para geração dos atributos	90
Código-fonte 4 – Template para geração das operações	92
Código-fonte 5 – Template para geração dos arquivos de configuração e dos serviços .	94
Código-fonte 6 – Template para geração do arquivo persistence.xml	95
Código-fonte 7 – Template para geração do pom.xml	97
Código-fonte 8 – Template para geração do web.xml	101
Código-fonte 9 – Template para geração das classes de manipulação de dados (<i>DAO</i>) .	102
Código-fonte 10 – Template para geração da classe de utilidades	105
Código-fonte 11 – Template para geração das classes que implementam os serviços . . .	106
Código-fonte 12 – Código fonte da classe <i>Asset</i>	109
Código-fonte 13 – persistence.xml	112
Código-fonte 14 – Código fonte da classe de acesso a dados do <i>Asset</i>	113
Código-fonte 15 – Código fonte da classe que implementa as operações de CRUD no <i>Asset</i>	115
Código-fonte 16 – Código fonte cliente	118
Código-fonte 17 – Código para execução dos serviços publicados	151
Código-fonte 18 – Código de geração das entidades <i>beans</i>	161
Código-fonte 19 – Código de geração do servidor.	166

LISTA DE TABELAS

Tabela 1 – Bases de Publicação Seleccionadas	47
Tabela 2 – Números de estudos retornados por base de dados de publicação	50
Tabela 3 – Parte dos artigos seleccionados pela primeira seleção	54
Tabela 4 – Estudos seleccionados ao final da revisão bibliográfica.	73
Tabela 5 – Subconjunto dos artigos obtidos pela primeira seleção.	183

LISTA DE ABREVIATURAS E SIGLAS

ACM		<i>Association for Computing Machinery</i>
AIRES	...	<i>an Architecture to Improve software REuSe</i>
API		<i>Application Program Interface</i>
ARSIM	...	<i>Asset Repository Sharing Information Model</i>
ASSS		<i>Asset Sharing Service Subsystem</i>
AWSDC	..	<i>Algorithm for web service discovery e composition</i>
BPEL		<i>Business Process Execution Language</i>
CASE		<i>Computer-Aided Software Engineering</i>
CBD		<i>Component-based development</i>
CIM		<i>Computation-Independent Model</i>
CRUD		<i>Create, Read, Update e Delete</i>
CSV		<i>Comma Separated Values</i>
HTTP		<i>Hipertext Transfer Protocol</i>
IDL		<i>Interface Description Language</i>
IEEE		<i>Institute of Electrical and Electronics Engineers</i>
IPCASCI	.	<i>Intelligent business Processes Composition based on multi-Agent systems, Semantics and Cloud Integration</i>
ISV		<i>Independent Software Vendor</i>
MDA		<i>Model-Driven Architecture</i>
MDD		<i>Model Driven Development</i>
MDE		<i>Model Driven Engineering</i>
MOF		<i>Meta-Object Facility</i>
MVCASE		<i>Multi Views CASE</i>
OMG		<i>Object Management Group</i>
PICO		<i>Population, Intervention, Comparison e Outcomes</i>
PIM		<i>Platform Independent Model</i>
PSM		<i>Platform Specific Model</i>
RAS		<i>Reusable Assets Specification</i>
REST		<i>REpresentational State Transfer</i>
SARMS	..	<i>Software Asset Repository Management System</i>
SGML	...	<i>Standard Generalized Markup Language</i>

SOA *Service-Oriented Architecture*
SOAP *Simple Object Access Protocol*
Start *State of the Art through Systematic Review*
UDDI *Universal, Description, Discovery*
UML *Unified Modeling Language*
URI *Uniform Resource Identifier*
W3C *World Wide Web Consortium*
WSDL ... *Web Services Description Language*
WTP *Eclipse Web Tools Platform*
XMI *XML Metadata Interchange*
XML *Extensible Markup Language*

SUMÁRIO

1	INTRODUÇÃO	27
1.1	Contextualização	27
1.2	Motivação	28
1.3	Objetivos	29
1.4	Organização da Dissertação	30
2	CONCEITOS BÁSICOS	31
2.1	Introdução	31
2.2	Arquitetura Orientada a Serviços	32
2.2.1	Web Services	36
2.2.2	REST	38
2.3	Model Driven Development	38
2.3.1	Model-Driven Architecture - MDA	39
2.3.2	Geração de Código	41
2.4	AIRES	42
2.5	Considerações finais	44
3	REVISÃO DA LITERATURA	45
3.1	Revisão sistemática	45
3.1.1	Planejamento	45
I	Objetivos de Pesquisa	45
II	Questões de Pesquisa	46
III	Fontes de Seleção	47
IV	Definição dos estudos: Critérios de Exclusão e de Inclusão	48
V	Procedimentos adotados para Seleção dos Estudos	49
VI	Número de estudos retornados após a execução da string de busca nas bases de dados	50
3.1.2	Condução	50
I	Primeira Seleção	50
II	Segunda Seleção	54
III	Resumo do estudo primário Service-Oriented Lightweight Software Asset Repository Open Framework - P5	55
IV	Resumo do artigo Reuse-Oriented Deployment of Software Components - P3	58

V	<i>Resumo do Artigo Intelligent business processes composition based on multi-agent systems - P2</i>	59
VI	<i>Resumo do artigo An Architecture Model for Dynamically Converting Components into Web Services - P1</i>	62
VII	<i>Resumo do Artigo Reusing Existing Object-oriented Code as Web-Services in a SOA - P4</i>	66
3.1.3	<i>Extração dos Dados</i>	68
I	<i>Questão 1 : Quais são os estudos primários existentes relacionados com o desenvolvimento de software, componentes ou ambientes de reúso orientado a serviços?</i>	69
II	<i>Questão 2 : Quais as tecnologias desenvolvidas e adotadas para o desenvolvimento de software, componentes ou ambientes de reúso orientados a serviço?</i>	69
III	<i>Questão 1.1 : Quais os tipos de arquitetura de software aplicada no desenvolvimento de ambientes de reúso orientados a serviço?</i> .	70
IV	<i>Questão 1.2 : Quais as principais dificuldades apresentadas no desenvolvimento de software, componentes, ou ambientes de reúso orientados a serviço?</i>	70
V	<i>Questão 1.3 : Quais as principais vantagens apresentadas pelo desenvolvimento de software, componentes e ambientes orientados a serviço?</i>	70
3.2	<i>Revisão bibliográfica derivada da revisão sistemática</i>	71
3.2.1	<i>Motivação da Revisão Bibliográfica</i>	71
3.2.2	<i>Objetivo da Revisão Bibliográfica</i>	71
3.2.3	<i>Planejamento da Revisão Bibliográfica</i>	71
3.2.4	<i>Condução da Revisão Bibliográfica</i>	72
I	<i>Primeira seleção</i>	72
II	<i>Segunda seleção</i>	73
III	<i>Resumo do estudo primário Top-Down Approach for Web Services Development</i>	73
IV	<i>Resumo do estudo primário Engineering Adaptable Service Oriented Systems: A Model Driven Approach</i>	74
V	<i>Resumo do estudo primário An EMF-based Toolkit for Creation of Domain-specific Data Services</i>	76
3.2.5	<i>Conclusão sobre a Revisão Bibliográfica</i>	77
4	<i>GERADORES DE SERVIÇOS CRUD COM BASE EM MODELOS UML</i>	79
4.1	<i>Considerações iniciais</i>	79
4.2	<i>Visão Geral do projeto</i>	80
4.2.1	<i>Motivação do projeto</i>	80

4.2.2	<i>Objetivo do projeto</i>	81
4.2.3	<i>Solução proposta</i>	81
4.3	Requisitos do gerador proposto	82
4.4	Estrutura e implementação do gerador	86
4.5	Prova de conceito	108
4.6	Avaliação do gerador	122
4.7	Considerações finais	125
5	CONCLUSÕES	129
5.1	Contribuições	129
5.2	Limitações e Opções de projeto	130
5.3	Trabalhos Futuros	131
REFERÊNCIAS		133
APÊNDICE A	TUTORIAL DO PROJETO AIRES_SERVER	139
A.1	Ferramentas Utilizadas	139
A.2	Criação dos projetos para geradores	141
A.2.1	<i>Criação de um diagrama de classes no Papyrus</i>	144
A.3	Criação do projeto servidor	144
A.4	Criação do projeto cliente	147
APÊNDICE B	LISTA DE CÓDIGOS GERADOS PELOS TEMPLATES BEAN_UMLGEN E SERVER_CRUD_UMLGEN	161
APÊNDICE C	ARTIGOS SELECIONADOS DURANTE A PRIMEIRA SELEÇÃO DA REVISÃO SISTEMÁTICA	183

INTRODUÇÃO

1.1 Contextualização

A indústria de desenvolvimento de software está constantemente evoluindo e procurando por novos desafios, como por exemplo: novas tecnologias, linguagens de programação e ferramentas que aumentam o poder, a eficiência e a segurança dos sistemas, bem como a produtividade dos desenvolvedores. Neste contexto, é importante construir novos modelos, arquiteturas e tecnologias, para introduzir novas ferramentas da maneira mais simples e econômica possível (CORIA; CASTELLANOS-GARZÓN; CORCHADO, 2013).

Com o passar do tempo, novas tecnologias têm sido desenvolvidas para superar os problemas enfrentados pelos desenvolvedores, problemas que são conhecidos como a crise do software (HONGMIN; JIN; JINGZHOU, 2010). Nesta crise destacam-se sistemas de software legados complexos, que necessitam de uma atualização para tecnologias mais modernas, porém essa atualização pode levar a um custo elevado e demandar muito tempo de desenvolvimento. Outra forma de resolver a crise do software é adotando a tecnologia de reuso de software que mais adequa-se à realidade em questão. Segundo Yao e Etzkorn (2004), o reuso de software é uma forma de aproveitar-se pedaços (módulos ou modelos) de software utilizados anteriormente em outros projetos em um novo projeto de desenvolvimento de software. O reuso vem sendo objetivo desde que ficou provado que sua utilização diminui os custos de produção, aperfeiçoa a qualidade de software e aumenta a produtividade dos desenvolvedores.

Dentre as abordagens que propagam a reutilização de software, a arquitetura orientada a serviços e o desenvolvimento dirigido por modelos possuem certo destaque. A arquitetura orientada a serviços (SOA - do inglês *Service-Oriented Architecture*) ajuda a integrar diferentes ferramentas, desde que elas sejam projetadas levando-se em consideração os seus princípios. Segundo Ribeiro *et al.* (2010), SOA é um modelo arquitetural que oferece suporte ao *design*, desenvolvimento, implantação e gerenciamento de sistemas, onde serviços fornecem funcionalidade

reutilizável de negócios.

Já o desenvolvimento de software dirigido (ou baseado) em modelos (MDD - do inglês *Model-Driven Development*), ajuda a aumentar o grau de abstração no processo de criação de software, ou seja, mais pessoas podem participar do desenvolvimento, pois com o aumento da abstração a demanda por especialistas não tende a ser mais uma exigência. Segundo [Bellini, Prado e Zaina \(2010\)](#), MDD é uma abordagem de desenvolvimento de software que recomenda a utilização de modelos para vários níveis de abstração, sendo que os principais artefatos do processo são os modelos. MDA (do inglês *Model Driven Architecture*) é caracterizado por ser a formalização dos conceitos implementados por MDD. MDA é parte de uma tendência da indústria de software, que sugere a adição cuidadosa de níveis de abstração no desenvolvimento de software ([MESERVY; FENSTERMACHER, 2005](#)).

Baseado em uma revisão sistemática realizada durante este trabalho, pôde-se constatar que apesar da existência de trabalhos que utilizem o conceito de SOA na reutilização de software, a sua aplicação ainda é um campo aberto para pesquisa e desenvolvimento de novos trabalhos e estudos. Além disso, trata-se de um modelo arquitetural amplamente conhecido e aplicado em trabalhos pelo mundo todo.

Esses dois conceitos (SOA e MDD) são ainda muito atuais e com campo aberto para sua utilização, com o objetivo de aumentar a adoção de técnicas de reúso de software. Neste contexto, este trabalho considera esses conceitos para fundamentar a proposta apresentada.

1.2 Motivação

Nesta seção é descrita a motivação deste trabalho. Basicamente existiram três motivações principais: primeiramente a AIRES e posteriormente os dois levantamentos bibliográficos realizados.

A AIRES (do inglês *an Architecture to Improve software REuSe*) é um modelo arquitetural proposto por [Braga et al. \(2016\)](#), cujo objetivo principal é prover o reúso de software em larga escala, utilizando-se de serviços para isso. Ela destina-se a organizações distribuídas em diferentes localizações e com diferentes necessidades em termos de reúso: desde o reúso de processos até diferentes níveis de reúso de software, como documentos, modelos, código, casos de teste, etc.

Para prover o reúso em larga escala conforme abordado anteriormente, a AIRES baseou-se em SOA para poder atingir o reúso e a interoperabilidade entre seus componentes (ou artefatos). Em virtude dessa necessidade inicial, uma revisão sistemática foi proposta, pelo autor deste texto, com intuito de descobrir estudos relacionados com a abordagem de desenvolvimento de ambientes de reúso baseados em serviço. Nesta revisão sistemática foram encontrados poucos trabalhos que de fato propunham a reutilização de software por meio da proposição de

arquiteturas orientadas a serviço (LEE; KIM; YANG, 2004; ATKINSON; BOSTAN; FINK, 2009; HONGMIN; JIN; JINGZHOU, 2010; CORIA; CASTELLANOS-GARZÓN; CORCHADO, 2013).

Um novo levantamento bibliográfico foi realizado objetivando encontrar trabalhos que propusessem a geração automática de serviços a partir de modelos de dados abstratos. Esse levantamento foi motivado pela AIRES, onde a implementação de serviços era uma necessidade latente, pois a arquitetura proposta baseia-se no conceito de SOA e pelos estudos encontrados durante a revisão sistemática desenvolvida, foram poucos os trabalhos que propunham uma arquitetura de reuso de software baseada em serviços, onde esses eram implementados de forma automatizada.

Assim, após o desenvolvimento desse novo levantamento bibliográfico, notou-se a existência de poucos trabalhos que propunham a geração automática de serviços a partir de modelos abstratos (KENZI *et al.*, 2009; BENDER; BOZIC; KONDOV, 2014; BELLINI; PRADO; ZAINA, 2010). Notou-se também durante esse levantamento e durante a elaboração da AIRES que as principais operações demandadas pelos serviços a serem implementados eram do tipo CRUD (do inglês *Create, Read, Update e Delete*). No entanto, entre os trabalhos encontrados haviam somente dois trabalhos que propunham a criação de serviços e suas operações de maneira completa (KENZI *et al.*, 2009; BENDER; BOZIC; KONDOV, 2014). No trabalho realizado por Bender, Bozic e Kondov (2014) foi proposto a geração de serviços e operações CRUD, porém esse trabalho propôs a criação de um novo metamodelo, enquanto que o trabalho aqui proposto utiliza como modelo o diagrama de classes comum e não expandido. Portanto os trabalhos encontrados que propunham a criação de serviços de forma completa, ou seja, implementação das operações e não somente suas assinaturas foram poucos. Dessa forma o objetivo deste trabalho se justifica, pois existe campo ainda a ser explorado na geração de serviços, que é um trabalho que quando feito manualmente pode ser repetitivo e sujeito a erros.

1.3 Objetivos

Este trabalho tem por objetivo o desenvolvimento de um gerador capaz de implementar serviços com operações do tipo CRUD, a partir de um modelo (diagrama de classes). Esse diagrama de classes não apresenta nenhuma modificação em sua definição, tratando-se de um diagrama de classes convencional. Esse gerador deve ajudar na implementação de serviços visando dar acesso a diferentes informações de cada camada de um sistema, cuja arquitetura é baseada em serviços. Um exemplo de aplicação para esse gerador seria a AIRES (BRAGA *et al.*, 2016), em que a camada de serviços de acesso a seus principais componentes poderiam ser gerada por meio do gerador aqui proposto. Outras ferramentas baseadas em SOA também poderiam usufruir desse gerador, bastando para isso fornecer o diagrama de classes contendo as entidades para as quais deseja-se implementar serviços.

Como foi mencionado na [seção 1.2](#), as operações mais comuns na implementação de um sistema baseado em serviços são as do tipo CRUD. Assim, com o desenvolvimento desse gerador, os serviços necessários para a implementação de um sistema interoperável, modular e flexível seria possível. Com isso, novos módulos poderiam ser adicionados, entidades externas poderiam consumir os serviços publicados e, assim, aumentaria-se o reúso de software.

1.4 Organização da Dissertação

Esta dissertação está organizada da seguinte forma: [Capítulo 1](#) - introduz a motivação, objetivo e contextualiza o texto; [Capítulo 2](#) - aborda os principais conceitos utilizados para desenvolvimento do trabalho proposto; [Capítulo 3](#) - ilustra todo o levantamento bibliográfico realizado; [Capítulo 4](#) - mostra como o trabalho foi executado; e [Capítulo 5](#) - relata as principais contribuições, pontos negativos e abre o trabalho desenvolvido para perspectivas futuras.

CONCEITOS BÁSICOS

Neste capítulo são abordados os principais conceitos e definições adotados durante toda a dissertação. Temas como, Arquitetura Orientada a Serviços (do inglês *Service Oriented Architecture* - SOA), *Web Services* e *REpresentational State Transfer* (REST), *Unified Modeling Language* (UML) e *Model Driven Development* (MDD), são detalhados e discutidos durante este capítulo.

A estrutura empregada neste capítulo é a seguinte: [seção 2.1](#) - ilustra o contexto histórico de forma breve e introduz o reuso de software; [seção 2.2](#) - apresenta os principais conceitos sobre arquitetura orientada a serviços; [subseção 2.2.1](#) - define e contextualiza *web services*; [subseção 2.2.2](#) - define e contextualiza a *REST*; [seção 2.3](#) - define e contextualiza *MDD*; [seção 2.4](#) - define a arquitetura AIRES e mostra sua motivação e público alvo; e [seção 2.5](#) - tece as considerações finais do capítulo.

2.1 Introdução

Com o passar do tempo, as empresas de desenvolvimento de software perceberam a necessidade de produzir software a um menor custo e tempo e com maior qualidade. Um período crucial para a percepção que o desenvolvimento de software estava estagnando foi a ocorrência da "Crise do software", nos anos 70.

Algumas características comuns entre os diferentes projetos existentes puderam ser constatadas durante a crise do software, dentre elas destacavam-se:

- Projetos que ultrapassam o orçamento;
- Projetos com longo tempo de desenvolvimento;
- software de baixa qualidade e eficiência;

As metodologias de desenvolvimento de software evoluíram para amenizar essa crise e aumentar a eficiência e a produtividade durante o desenvolvimento de sistemas. Nesse contexto surgiram as linguagens orientadas a objetos, desenvolvimento baseados em componentes e uso de serviços, entre outros, que possuíam como ideia central comum a adoção do reúso de artefatos, como por exemplo código-fonte, componentes, modelos, etc.

O reúso de software refere-se a uma forma de aproveitar módulos, partes ou modelos de software utilizados em projetos anteriores em um novo projeto de desenvolvimento de software. O reúso vem sendo um objetivo desde que ficou provado que diminui os custos de produção de software e aperfeiçoa sua qualidade, bem como a produtividade do desenvolvedor (YAO; ETZKORN, 2004).

O aumento da complexidade dos sistemas desenvolvidos com a orientação a objetos tem causado aumento nos custos, incentivando os esforços para adoção de tecnologias orientadas a serviço. O uso dessas tecnologias traz uma série de vantagens em relação as demais tecnologias de desenvolvimentos de software, por exemplo (SNEED; VERHOEF; SNEED, 2013):

- A interface do serviço era transparente para o usuário, ou seja, ele sabe como utilizar o serviço, porém não possui conhecimento de sua implementação interna;
- Possível reutilização de código legado sem modificá-lo;
- Permite compartilhamento de operações e métodos entre aplicações internas e externas.

Além das vantagens citadas anteriormente, o desenvolvimento de software baseado em SOA ou serviços possui uma característica marcante, a possibilidade de compartilhamento e de integração com diferentes artefatos de software. Existem diversos repositórios de artefatos com grande potencial de reúso, mas que ficam isolados como ilhas em um oceano. De fato, esses repositórios são geralmente específicos para servir a determinadas necessidades de uma organização e de difícil integração com outros repositórios existentes. O desenvolvimento baseado em serviços pode fazer com que esses repositórios de artefatos tenham seu potencial de reúso mais utilizado, ou seja, as operações oferecidas pelos artefatos existentes no repositório podem ser ofertadas e compartilhadas pela Internet por meio de serviços.

2.2 Arquitetura Orientada a Serviços

Nesta seção, os principais conceitos e características de arquitetura orientada a serviços são discutidos.

Segundo Ribeiro *et al.* (2010), *Service-Oriented Architecture* (SOA) é um modelo de arquitetura que suporta o *design*, desenvolvimento, implantação e gerenciamento de sistemas, onde serviços fornecem funcionalidade reutilizável de negócios.

SOA para [Bellini, Prado e Zaina \(2010\)](#) é a infraestrutura de uma aplicação, onde os negócios são construídos a partir de funções individuais e processos, chamados serviços. SOA apoia-se em certas políticas, especificações e uma série de padrões que permitem a construção de processos. Ainda segundo [Bellini, Prado e Zaina \(2010\)](#) em SOA, um serviço deve ter sua interface bem definida e deve ser executado de forma independente de outros serviços.

Para [Krafzig, Banke e Slama \(2005\)](#), arquitetura orientada a serviços é uma arquitetura e software baseada em conceitos chaves como: *frontend* de aplicação, serviço, repositório de serviços, e barramento de serviços. Considerando que serviço consiste de um contrato, uma ou mais interfaces, e uma implementação.

Os elementos chave mencionados anteriormente podem ser vistos na [Figura 1](#).

Figura 1 – Modelo de SOA em função de seus elementos

Fonte: [Krafzig, Banke e Slama \(2005\)](#).

Os elementos chaves definidos por [Krafzig, Banke e Slama \(2005\)](#), podem ser definidos da seguinte maneira:

- **Frontend de aplicação:** representa o ator principal de uma SOA. Ele inicia e controla todas as atividades dos sistemas. Existem tipos diferentes de *frontend* de aplicação. O tipo mais comum é a interface gráfica de usuário, como aplicações Web ou clientes com interface de usuário, que interagem diretamente com os usuários finais. Entretanto o *frontend* de aplicação não possui a necessidade de interagir diretamente com os usuários

finais, programas *batch* e processos podem invocar suas funcionalidades periodicamente ou como resultado de eventos específicos (KRAFZIG; BANKE; SLAMA, 2005);

- **Serviços:** São componentes de software que tipicamente encapsulam conceitos de negócio de alto nível. A [Figura 1](#) mostra como um serviço é formado por seus componentes. Os componentes de serviço são: (KRAFZIG; BANKE; SLAMA, 2005)
 - **Contrato:** O contrato fornece uma especificação de propósito informal, funcionalidade, restrições e usos do serviço. A forma da especificação do serviço pode variar, dependendo do tipo do serviço. Um elemento não mandatório do contrato é a definição de interface formal baseada em linguagens como *Interface Description Language* (IDL) ou *Web Services Description Language* (WSDL). Contudo mesmo não sendo mandatório, a definição formal de interface de serviço adiciona um benefício significativo ao desenvolvimento do serviço, pois provê abstração e independência de tecnologia, incluindo linguagens de programação protocolos de rede e ambientes de tempo real;
 - **Interface:** A funcionalidade de um serviço é exposta por sua interface. Contudo a descrição da interface é parte do contrato, e a implementação em código da interface é deixada a cargo do cliente;
 - **Implementação:** A implementação do serviço fornece a lógica de negócio requerida e os dados apropriados. A implementação de serviço consiste em um ou mais artefatos como programas, dados de configuração e base de dados;
 - **Lógica de negócio:** A lógica de negócio é encapsulada pelo serviço e é parte de sua implementação. Ela torna-se acessível através das interfaces de serviços;
 - **Dados:** Os serviços podem conter dados. Em particular tratando-se de serviços centrados a dados.
- **Repositório de Serviços:** O repositório de serviços fornecem facilidades para descoberta de serviços e mantém todas informações de uso dos serviços, particularmente se estes serviços necessitam ser descobertos fora do escopo temporal e funcional do projeto que os criaram. O repositório de serviços podem conter informações adicionais do tipo, localização física do serviço, informação sobre o fornecedor, contato de pessoas, taxas de uso, restrições técnicas e etc (KRAFZIG; BANKE; SLAMA, 2005);
- **Barramento de Serviços:** O barramento de serviços conecta todos os participantes de uma SOA, serviços e *frontend de aplicações*, entre si (KRAFZIG; BANKE; SLAMA, 2005).

Segundo Earl (2005), os princípios básicos de orientação a serviços são:

- **Fraco acoplamento:** os serviços são desenvolvidos de modo a não dependerem entre si e não dependerem de interações com outras aplicações, ou seja, um serviço por si só, consegue existir e ser autosuficiente sem a necessidade de interação com outras entidades, serviços ou aplicações. Além disso, permite que o serviço evolua seu design e lógica independente de sua implementação, e que garanta a interoperabilidade básica com consumidores que utilizam suas capacidades;
- **Contrato de serviço:** os serviços propõem suas capacidades e seu propósito por meio de um documento chamado contrato. O contrato talvez seja o componente mais importante da orientação a serviços, porque ele, requer que considerações específicas sejam levadas em contas ao implementar-se a interface de um serviço e ao analisar-se a natureza do conteúdo que será publicado. Os contratos de serviço devem ser otimizados para que os pontos de acesso sejam consistentes e confiáveis;
- **Autonomia:** serviços que possuem controle total sobre a lógica que encapsulam e um alto grau de controle sobre seu ambiente e recursos;
- **Abstração:** a lógica de um serviço é escondida do mundo externo, ou seja, ela se torna abstrata a aplicações externas. Esse princípio ao ocultar o maior número possível de detalhes de um serviço, também mantém um acoplamento fraco;
- **Reusabilidade:** a lógica é dividida entre os serviços, com a intenção de promover o reúso;
- **Composição:** coleções de serviços que podem ser coordenados e montados para formarem serviços mais complexos. A capacidade de compor serviços de maneira eficaz é um requisito importante para atingir alguns dos objetivos fundamentais da computação orientada a serviços;
- **Sem estado (do inglês *Statelessness*):** serviços que diminuem a retenção de informação específica para uma atividade. O gerenciamento excessivo de informações de estado podem comprometer a disponibilidade de um serviço e coibir seu potencial uso. Desta maneira, os serviços são, projetados para manterem informações de estado quando estas forem necessárias;
- **Descoberta de serviços (do inglês *Discoverability*):** serviços são designados para serem externamente descritivos para serem encontrados e acessados pelo mecanismo de descoberta disponível.

A tecnologia mais empregada com arquitetura orientada a serviços é *Web services*. Esta tecnologia de tão utilizada em conjunto com SOA, chega a ser utilizada para designar sistemas desenvolvidos com arquitetura SOA. Essa tecnologia será amplamente abordada na próxima seção.

2.2.1 Web Services

Segundo Chukmol, Benharkat e Amghar (2010), *Web services* são componentes de software autônomos que podem ser descritos, publicados, descobertos e invocados utilizando um conjunto de tecnologias conhecidas: *Simple Object Access Protocol* (SOAP), WSDL (do inglês *Web Services Description Language*) e *Universal, Description, Discovery* (UDDI) e *Integration*. Elas são os pilares de diversas aplicações baseadas em arquiteturas orientadas a serviço devido a sua natureza de plataforma independente (de linguagem de programação e sistema operacional).

Os serviços não são somente encontrados em repositórios de registros implementados por UDDI, eles também podem ser encontrados através da Internet. Portais *Web* baseados em serviços não são mais recursos escassos, eles permitem que usuários procurem por *Web services* pela Internet (CHUKMOL; BENHARKAT; AMGHAR, 2010).

Segundo Bellini, Prado e Zaina (2010) *Web Services* é uma tecnologia que suporta interoperabilidade de *Web sites* e serviços de integração de aplicações, através de aplicações de rede permitindo a adoção do paradigma SOA. Os *Web Services* são definidos através da publicação de seu contrato em um servidor.

A seguir serão descritas as principais tecnologias utilizadas por *Web Services*:

- **Extensible Markup Language (XML):** é uma criação da *World Wide Web Consortium* (W3C)¹ e derivada do popular padrão *Standard Generalized Markup Language* (SGML). Este padrão é uma meta-linguagem amplamente utilizada para permitir que organizações adicionem inteligência para um documento cru, ou em estágio inicial (EARL, 2005);
- **SOAP:** é um protocolo baseado em XML para troca de informações entre computadores. Entretanto, SOAP pode ser usado em uma variedade de sistemas de mensagens, e pode ser entregue por uma variedade de protocolos de transporte, o principal é o *Hipertext Transfer Protocol* (HTTP)² (do inglês *Hypertext Transfer Protocol*) (CERAMI, 2002). Segundo Cho e McGregor (2005), SOAP fornece um canal padrão de comunicação e é um mecanismo baseado em XML para troca de mensagens;
- **WSDL:** representa frequentemente a camada de descrição de serviço dentro de uma pilha de protocolo *Web service*. Em poucas palavras, WSDL é uma gramática XML para especificar interfaces públicas de um *Web service*. Esta interface pública pode incluir informações sobre todas as funções públicas disponíveis, informações de tipo de dados para todas mensagens XML, vinculação sobre protocolo de transporte específico para ser utilizado, e endereçamento de informações para localizar um serviço específico (CERAMI, 2002). Para Bellini, Prado e Zaina (2010) WSDL é um tipo de formato XML utilizado para

¹ *World Wide Web Consortium* (W3C) é uma comunidade internacional que desenvolve padrões abertos para garantir o crescimento a longo prazo da Web, retirado de <http://www.w3.org/>

² *HTTP* é um protocolo de aplicação para distribuição, colaboração de sistemas de informação hipermídia, retirado de http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

descrição de *Web Services* como uma coleção de *endpoints* ou portas de acesso capazes de permitir a troca de mensagens;

- **UDDI:** é utilizado para registrar e localizar *Web services*. Provê um diretório de serviços tipo páginas amarela (CHO; MCGREGOR, 2005).

A figura 2 apresenta uma visão lógica da arquitetura de *Web services*, com as tecnologias fundamentais envolvidas na implementação de serviços (CHO; MCGREGOR, 2005).

Figura 2 – Visão lógica do modelo de arquitetura *Web services*

Fonte: Cho e McGregor (2005).

Web services baseados em SOAP possuem transparência e independência de protocolo. Usando SOAP uma mesma mensagem, de mesmo formato pode ser transportada por vários sistemas de *middleware* sob diferentes protocolos de transporte. Além disso, utilizando-se a linguagem WSDL para descrição de interfaces de serviços pode-se abstrair os detalhes dos protocolos de comunicação e serialização, bem como a plataforma de implementação (PAUTASSO; ZIMMERMANN; LEYMANN, 2008).

Os contratos WSDL fornecem uma descrição processável por máquina da sintaxe e estrutura das mensagens de resposta e de requisição, definindo assim um caminho flexível de evolução para o serviço (PAUTASSO; ZIMMERMANN; LEYMANN, 2008).

Segundo Pautasso, Zimmermann e Leymann (2008), *Web services* baseados em SOAP são empregados em cenários de integração de aplicações empresariais com longo tempo de vida, com avançados requisitos de qualidade de software.

2.2.2 REST

REST (do inglês *REpresentational State Transfer*) foi originalmente criado como uma tecnologia para construção de sistemas de hipermídia de larga escala. O termo REST é utilizado com muita frequência com o protocolo HTTP, pois os princípios deste estilo arquitetural foram utilizados para justificar a escalabilidade do protocolo HTTP 1.0 (PAUTASSO; ZIMMERMANN; LEYMANN, 2008).

REST é baseado em quatro princípios:

- Identificação de recurso através de *Uniform Resource Identifier* (URI): Um *Web service* RESTful expõem um conjunto de recursos que identificam os alvos de interação com seus clientes. Esses recursos são identificados pelo URI;
- Interface uniforme: Recursos são consumidos utilizando-se um conjunto de operações *Create, Read, Update e Delete* (CRUD): *PUT, GET, POST e DELETE*;
- Mensagens auto-descritivas: Os recursos são desacoplados de sua representação e portanto podem ser acessados em uma grande variedade de formatos, como por exemplo HTML, XML, texto simples, PDF, JPEG e etc. Os metadados sobre estes recursos estão disponíveis e são utilizados, por exemplo para controle de cache, detecção de transmissão de erros, etc;
- Interações de estado através de *hyperlinks*: Toda interação com um recurso não possui estado. Interações de estado são baseadas no conceito explícito de transição de estado. Existem diversas técnicas para mudar de estado, por exemplo modificação de uma URI já existente, *cookies*, e etc.

Web services RESTful são conhecidos por serem de simples implementação, porque utilizam-se de padrões bem estabelecidos, como HTTP, XML, URI e etc., e toda infraestrutura necessária tornou-se difundida (PAUTASSO; ZIMMERMANN; LEYMANN, 2008).

Segundo Pautasso, Zimmermann e Leymann (2008), *Web services* RESTful são adequados para aplicações onde as seguintes características não são essenciais: transações, confiabilidade, e segurança em nível de mensagens. Ainda segundo Pautasso, Zimmermann e Leymann (2008), *Web services* RESTful são melhores que *Web services* baseados em SOAP em relação a flexibilidade e controle, porém requerem codificação de baixo nível.

2.3 Model Driven Development

Nesta seção são abordados conceitos e definições relativos a geração de código e desenvolvimento de software orientado a modelos.

Sistemas de software tem evoluído para sistemas cada vez mais complexos, visando atender a crescente demanda de mercado por mudanças e visando integrar fatores externos, por

exemplo outras aplicações. Com isso esses sistemas tem tornado-se cada vez mais imprevisíveis e para atender essa demanda está tornando-se difícil (QAFMOLLA; NGUYEN, 2010).

Model Driven Engineering (MDE) segundo Qafmolla e Nguyen (2010), surge como uma opção para atender a constante demanda por sistemas complexos, pois sua adoção permite aumento de produtividade a partir da maximização de compatibilidade entre sistemas, simplificando o processo de design e promovendo a comunicação entre indivíduos e equipes de trabalho existentes no desenvolvimento do sistema.

Um modelo pode ser considerado uma representação simplificada de um aspecto do mundo. Para o desenvolvimento de sistemas complexos muitos aspectos devem ser considerados desde comportamentos arquiteturais e dinâmicos, funcionalidades e até as interfaces de usuário (QAFMOLLA; NGUYEN, 2010).

Segundo Pagan, Cuadrado e Molina (2011), um modelo é uma instância de um meta-modelo, que define as meta-classes e os relacionamentos que os elementos do modelo podem possuir entre si. Ainda segundo Pagan, Cuadrado e Molina (2011), um modelo pode ser interpretado como um grafo rotulado direcionado, onde os nós representam um objeto (elemento do modelo) e as arestas representam os relacionamentos entre os objetos. Esses relacionamentos podem ser do tipo contém e não-contém.

Um meta-modelo é uma abstração que destaca as propriedades do modelo. O modelo sujeita-se ao meta-modelo da mesma forma que um programa se sujeita a linguagens de programação, na qual é escrito. *Model Driven Engineering* (MDE) é uma metodologia de desenvolvimento de software que foca na criação de modelos ou em abstrações de algo que descreva os elementos de um sistema (QAFMOLLA; NGUYEN, 2010).

Modelos permitem abstrair o mundo real a partir de conceitos e estes podem ser mapeados através de artefatos de engenharia de software. De tal forma, Desenvolvimento Dirigido a Modelos (*Model-Driven Development*) é uma metodologia de desenvolvimento de software desenvolvida sob o preceito de utilizar modelos gráficos (como UML) ou textuais (como *XML Metadata Interchange* (XMI) na geração de software, sem a necessidade de codificação de baixo nível (PASTOR; MOLINA, 2007).

Para permitir a adoção dos conceitos de desenvolvimento propostos por MDD, ferramentas, tecnologias e modelos arquiteturais foram desenvolvidos. Dentre os modelos arquiteturais destaca-se a *Model-Driven Architecture* ou MDA, que é descrita em mais detalhes na seção subseção 2.3.1.

2.3.1 Model-Driven Architecture - MDA

Em 2001, o *Object Management Group* (OMG) lançou a iniciativa conhecida por *Model-Driven Architecture* (MDA) com a ambição de trocar o foco do desenvolvimento de software da escrita de código fonte para o desenvolvimento orientado a modelos. MDA é parte de uma

grande tendência da indústria de software, tendência essa que sugere a adição cuidadosa de níveis de abstração em hardware subjacente. As linguagens de alto nível substituíram as linguagens de montagem (do inglês *assembly*), bibliotecas e *frameworks* substituíram trechos isolados de código, padrões de design substituíram códigos específicos de projeto, com isso vemos que abstração de software está acontecendo de maneira gradativa durante a evolução do processo de desenvolvimento (MESERVY; FENSTERMACHER, 2005).

Segundo Qafmolla e Nguyen (2010), MDA pode ser definida como um *framework*, que define uma arquitetura para modelos. Baseados nesta arquitetura, modelos representam parte da funcionalidade do sistema, estrutura e/ou comportamento. Além do mais, MDA define duas estruturas principais de modelagem, nomeadas *Platform Independent Model* (PIM) e *Platform Specific Model* (PSM). PIM fornece uma especificação formal da estrutura e funcionalidade de um dado sistema, deixando de lado as particularidades técnicas sobre o sistema. PSM, lida portanto com as particularidades técnicas do sistema, ou seja, com as preocupações específicas sobre a implementação do sistema. Ainda segundo Qafmolla e Nguyen (2010), essa abordagem de desenvolvimento orientado a modelos é muito importante e benéfica por diversos motivos. Dentre esses motivos destacam-se:

- Facilidade de reproduzir implementações do mesmo modelo em diferentes plataformas, quando está se referindo a mesma estrutura inicial (modelo) e sem uma semântica específica de plataforma;
- Validação do modelo também é facilitada, pois os conceitos específicos de plataforma são embarcados na implementação, ou seja, menos características a serem validadas.

Segundo Meservy e Fenstermacher (2005), a especificação MDA utiliza o termo plataforma livremente para sugerir não somente um sistema operacional específico, mas também uma plataforma baseada em linguagem de programação, como *Java* e *Python*, e até práticas comuns de desenvolvimento, como criação de métodos acessores para os atributos das classes.

MDA faz uma clara distinção entre um modelo de negócio, o *Computation-Independent Model* (CIM), o modelo de negócio em um contexto de tecnologia específica (PIM), e um modelo que está ligado ao negócio e usa código específico da plataforma (PSM) (MESERVY; FENSTERMACHER, 2005). Sabendo disto o ciclo de vida de um software desenvolvido utilizando-se MDA é:

- Capturar os requisitos em um CIM, onde CIM é o modelo de mais alto nível utilizado para representar o problema sem envolver conceitos de implementação;
- Criar um PIM;
- Transformar um PIM em um ou mais PSMs, adicionando regras específicas de plataforma;

- Transformar PSM em código fonte; e
- Implantar o sistema em um ambiente específico.

2.3.2 Geração de Código

Nesta seção é descrito o conceito de geração de código e duas técnicas para geração de código. A geração de código é diretamente relacionada a MDD, pois as duas abordagens visam como produto final o código gerado de forma automatizada. A geração de código abordada nessa seção é especificamente relacionada a geração de serviços.

Segundo [Krafzig, Banke e Slama \(2005\)](#), o gerador de código encapsula todo o conhecimento técnico em relação a distribuição e invocação de serviços. Com isso, os desenvolvedores podem focar nos aspectos de negócio e definições de interfaces, enquanto que a geração de código suporta os detalhes e problemas com as especificações técnicas dos serviços.

A geração de código desacopla os aspectos funcionais dos aspectos técnicos. A geração de código também oferece suporte a diferentes linguagens de programação, a vários protocolos de rede e de aplicação, e por fim a diferentes sistemas operacionais. A geração de código pode ser utilizada na geração do lado cliente e do lado servidor de uma aplicação, proporcionando benefícios em toda cadeia produtiva do desenvolvimento de software. Além dessas características a geração de código proporciona aumento na qualidade técnica do código, pois ela gera-o a partir de um modelo ou WSDL bem definido ([KRAFZIG; BANKE; SLAMA, 2005](#)).

Segundo [Krafzig, Banke e Slama \(2005\)](#), a geração de código é utilizada de duas formas: abordagem *Top-Down* e abordagem *Bottom-Up*, descritas a seguir:

- Abordagem *Top-Down*: abordagem de desenvolvimento que parte do uso de uma linguagem de definição de interface, por exemplo WSDL, para o processo de desenvolvimento de software. O uso da linguagem de definição de interface permite o desacoplamento de equipes de desenvolvimento, simplificando assim a coordenação de desenvolvedores de serviços e seus clientes;
- Abordagem *Bottom-Up*: geração de código a partir da implementação de *Application Program Interface* (API) de baixo nível. Essa técnica de geração de código é útil para transformar aplicações legadas em serviços, poupando tempo de desenvolvimento pelo reaproveitamento de código legado.

Segundo [Krafzig, Banke e Slama \(2005\)](#) a abordagem de desenvolvimento *Top-Down* é a melhor dentre as duas abordagens anteriormente descritas pelos seguintes motivos:

- Atende aos requisitos dos clientes;
- São desenvolvidas com alto nível de granularidade;

- Fornecem potencial para uso tardio (posterior);
- Podem ser implementadas de forma a garantir a escalabilidade e a integridade do sistema;
- São independentes de implementação subjacente; e
- Fornecem especificações de níveis de serviço apropriadas.

Mesmo diante dessas vantagens, a abordagem *Bottom-up* apresenta outros benefícios:

- O desenvolvedor do serviço pode usar o ambiente de desenvolvimento preferido, juntamente com ferramentas de modelagem sem necessitar utilizar outras ferramentas externas para o desenvolvimento do serviço;
- Rápida geração de código em virtude do acoplamento do gerador de código acoplado ao mesmo ambiente de desenvolvimento do código fonte;
- Facilidade de implementação dos serviços a partir da escolha de uma linguagem de programação conhecida do desenvolvedor. Com isso, o desenvolvedor não necessita aprender uma nova linguagem para transformação de modelos, o que acelera o processo de desenvolvimento.

2.4 AIRES

Técnicas de reúso são utilizadas para aumentar a produtividade e a qualidade no desenvolvimento de processos de software, pois os desenvolvedores possuem a vantagem de utilizarem ativos reutilizáveis para produzir novos programas. É muito importante que o suporte ao reúso aconteça desde o princípio do projeto, por exemplo incluindo a reutilização de processos, e promovendo o reúso de artefatos entre diversas fases do processo de desenvolvimento que ocorrem antes da implementação, e finalmente com o reaproveitamento de código e casos de teste. Essa tarefa não é trivial, pois existe uma grande variedade de processos e artefatos, juntamente com diversos tipos de plataformas e ferramentas de desenvolvimento, para que a integração entre esses objetos ocorra (BRAGA *et al.*, 2016).

Atualmente, os engenheiros de software possuem um grande número de técnicas e ferramentas para permitir a reutilização de software, porém elas são geralmente focadas em formas específicas de reúso. Além disso, segundo (BRAGA *et al.*, 2016), uma revisão sistemática executada por seu grupo de pesquisa constatou que as ferramentas de engenharia de software são muito específicas, por exemplo ferramentas para linha de produto de software, repositórios de componentes ou código, e os desenvolvedores de software costumam empregar mais de uma maneira de reúso durante o processo de desenvolvimento. Eles realizam isso sem suporte unificado, o que pode levar a dificuldades para promover a integração entre os diferentes artefatos

e ferramentas utilizados no desenvolvimento. Para resolver esses problemas, prpuseram uma nova arquitetura conhecida por *an Architecture to Improve software REuSe* (AIRES).

O principal objetivo da AIRES é o de promover o reuso de software em larga escala por meio se serviços, tendo como alvo organizações que são distribuídas em diferentes lugares e com diferentes necessidades de reuso, por exemplo desde reuso de processo até o reuso de códigos e documentos.

Na [Figura 3 \(BRAGA *et al.*, 2016\)](#) é ilustrado o modelo arquitetural desenvolvido pela AIRES. Como pode ser observado, é uma arquitetura que foi desenvolvida em camadas, que são descritas a seguir:

1. Clientes: Essa camada fornece uma interface gráfica de usuário para o gerenciamento de ativos da AIRES e outras aplicações. Dentre as outras aplicações, pode-se destacar *AIRES Manager Frontend* que visa fornecer uma aplicação Web para que os desenvolvedores não necessitem usar suas próprias ferramentas de reuso para poder começar a utilizar a AIRES;
2. Serviços públicos: Podem ser utilizados para acessar a camada de aplicação, também podem ser utilizados para invocar a camada de clientes por aplicações customizadas. Além disso, esses tipos de serviço podem invocar os serviços privados para ter acesso a funcionalidades mais básicas definidas na camada *Core*;
3. Aplicações: os módulos *Software Product Line Manager* e *Certification Manager* são exemplos de aplicações que podem existir nesta camada da AIRES. Essa camada irá disponibilizar alguns módulos de aplicações internos da AIRES, que poderão ser ampliados ou mesmo modificados.;
4. Serviços privados: camada que oferece serviços de acesso à camada *Core* da AIRES. Eles podem ser invocados a partir da camada de aplicação. Esses serviços podem também ser invocados pelos serviços públicos permitindo assim que a camada de clientes ou qualquer aplicação externa seja capaz de interagir com a camada *Core* da AIRES;
5. *Core*: Essa camada serve como sustentação para as outras camadas. Ela representa os módulos centrais, com módulos para incluir a identidade única dos objetos (*UUIds Manager*), módulos para gerenciar os ativos reutilizáveis (*Assets Manager*) e um módulo para promover a leitura de interfaces e exportar ativos como modelos Reusable Assets Specification (RAS), sendo que esse último módulo ainda permite a transformação de modelos simples (providos pela AIRES) para modelos RAS.

Para persistir os objetos relacionados a usuários e à autorização de acesso, foi desenvolvida a base de dados *AIRES Core* e para persistir os ativos em si, outra base de dados foi proposta *Assets Repository*.

Figura 3 – Modelo arquitetural da AIRES.

Fonte: Braga *et al.* (2016).

2.5 Considerações finais

Neste capítulo foram estabelecidos os principais conceitos necessários a compreensão desta dissertação. Conceitos como SOA, *Web Services*, MDD e tecnologias que auxiliam na implementação de serviços foram detalhadas e com isso a integração desses conceitos auxilia na proposição e execução deste trabalho. A AIRES, que foi uma das motivações deste trabalho, também foi brevemente descrita. O trabalho foi direcionado para construção de um gerador de código, tendo como objetivo a automatização e praticidade no desenvolvimento e publicação de serviços. Com isso, outras ferramentas como repositórios de artefatos de software podem ser beneficiadas ao publicar seus artefatos por meio de operações implementadas pelos serviços. O gerador proposto utiliza-se de ferramentas como: *Tomcat*, *Maven* e *Acceleo*.

- *Tomcat*: servidor de código aberto desenvolvido pela *Apache Software Foundation*;
- *Maven*: ferramenta de automação da compilação de software usada principalmente com *Java*. Descreve como o software deve ser compilado e suas dependências;
- *Acceleo*: é uma implementação desenvolvida pelo OMG para a linguagem *Meta-Object Facility* (MOF).

REVISÃO DA LITERATURA

Este capítulo descreve todo processo de revisão da literatura realizada para o projeto. Nesta dissertação observou-se a necessidade de execução de levantamentos bibliográficos, tratando-se de uma revisão sistemática e uma revisão bibliográfica. Durante a realização da primeira revisão sistemática foi observada a necessidade da automatização da geração de código para a implementação de serviços. Com isso, uma nova revisão bibliográfica foi proposta e efetuada. Desta forma, este capítulo está organizado da seguinte forma: [seção 3.1](#) - Revisão sistemática e [seção 3.2](#) - Revisão bibliográfica derivada da revisão sistemática.

3.1 Revisão sistemática

Nesta seção é descrito todo o processo de desenvolvimento de uma revisão sistemática e seus resultados. As etapas do desenvolvimento são: planejamento ([subseção 3.1.1](#)), condução ([subseção 3.1.2](#)) e extração dos dados ([subseção 3.1.3](#)).

3.1.1 Planejamento

Nesta fase, é estabelecido o protocolo da revisão sistemática. Para isto, foi especificado: (i) objetivos da pesquisa; (ii) questões da pesquisa, alcance e especificidade; (iii) fontes para critérios de seleção; (iv) definição dos estudos; e (v) procedimentos para seleção dos estudos.

1 Objetivos de Pesquisa

O objetivo desta revisão sistemática é identificar estudos primários que reportam o desenvolvimento de sistemas ou subsistemas de software baseados em serviços. Além do mais, esta revisão sistemática tem o objetivo de obter uma visão das metodologias e tecnologias que estão sendo utilizadas para desenvolver os sistemas mencionados anteriormente.

II Questões de Pesquisa

Com o objetivo de encontrar possivelmente todos os estudos primários e sumarizar as evidências sobre o uso de serviços no desenvolvimento de softwares em ambientes de reúso, as seguintes questões de pesquisa foram estabelecidas:

- **Questão 1:** Quais são os estudos primários existentes relacionados com o desenvolvimento de software, componentes ou ambientes de reúso orientado a serviços?
- **Questão 2:** Quais são as tecnologias desenvolvidas e adotadas para o desenvolvimento de software, componentes ou ambientes de reúso orientados a serviço?
- **Questão 1.1:** Quais tipos de arquitetura de software são aplicadas no desenvolvimento de ambientes de reúso orientados a serviço?
- **Questão 1.2:** Quais são as principais dificuldades apresentadas no desenvolvimento de software, componentes ou ambientes de reúso orientados a serviços?
- **Questão 1.3:** Quais são as principais vantagens apresentadas pelo desenvolvimento de software, componentes ou ambientes de reúso orientados a serviço?

Uma questão de pesquisa bem formulada é geralmente composta e analisada de diferentes pontos de vista, que definem o alcance e a especificidade. Portanto, é utilizado o *Population, Intervention, Comparison* e *Outcomes* (PICO) para a revisão sistemática. População identifica o grupo populacional observado pela intervenção; Intervenção refere-se ao o que será observado no contexto da revisão sistemática; Comparação define o que é comparado no contexto da revisão sistemática; finalmente, os resultados da intervenção são expressados pelos êxitos.

- **População:** Pesquisadores e desenvolvedores que utilizam serviços no desenvolvimento de software, componentes e ambientes de reúso, com intuito de aumentar a reutilização destes para diferentes objetivos;
- **Intervenção:** A intervenção desta revisão sistemática é representada pela utilização de serviços no desenvolvimento de software e em ambientes de reúso;
- **Comparação:** Não é aplicável; e
- **Resultados:** Uma visão sobre os principais estudos propostos na literatura que listam o uso de serviços no desenvolvimento de software em ambientes de reúso. Também é esperado que se identifique os domínios onde estes serviços tem sido implementados, assim como métodos, requisitos e tecnologias que apoiam seu desenvolvimento.

III Fontes de Seleção

Para estabelecer uma estratégia de busca, foram definidos: critérios para seleção das fontes; listagem das fontes de pesquisa; estudos de linguagem; e palavras chave e seus termos relacionados.

Critérios para seleção das fontes de pesquisa: Três tipos de critérios foram considerados nesta revisão sistemática:

1. Publicação nas base de dados bibliográficos pesquisadas: permite estabelecer quais bases de publicações serão utilizadas para encontrar os estudos primários, adota-se o seguinte critério segundo [Dieste, Griman e Juristo \(2009\)](#): atualização de conteúdo (publicações são regularmente atualizadas); disponibilidade (texto completo dos artigos disponíveis); qualidade dos resultados (precisão dos resultados obtidos pela pesquisa); e versatilidade para exportação (um mecanismo para exportar os resultados da pesquisa se disponível);
2. Trabalhos relacionados: também foram considerados aqueles estudos citados como trabalhos relacionados nos estudos primários relevantes encontrados na bases de publicação; e
3. Especialista: estudos sugeridos por especialistas nas áreas Orientação a Serviços e Reúso de Software foram conduzidos. Embora a indicação de estudos por um especialista possa ser considerada como um viés, adotou-se esta fonte não permitindo assim a perda de evidências importantes.

Lista de fontes: as bases de publicação selecionadas para a revisão sistemática são mostradas na [Tabela 1](#). De acordo com [Dyba, Dingsoyr e Hanssen \(2007\)](#), essas base de dados são suficientes para conduzir revisões sistemáticas no contexto de Engenharia de Software. Além do mais, a Scopus foi adicionada, pois é considerada uma base de dados que contém muitos artigos e citações.

Tabela 1 – Bases de Publicação Selecionadas

Fonte	Localização
ACM Digital Library	dl.acm.org
Compendex	www.engineeringvillage.com
IEEE Xplore	www.ieeexplore.ieee.org
ScienceDirect	www.sciencedirect.com
Scopus	www.scopus.com
Springer	www.springerlink.com
Web of Science	www.isiknowledge.com

Linguagem dos estudos: Somente estudos primários escritos em inglês e português foram considerados nesta revisão sistemática. O inglês foi adotado porque a maioria das pesquisas em Ciência da Computação têm sido publicadas nesta língua e o português foi adotado para que trabalhos importantes escritos nesta língua não sejam excluídos; **Palavras chave:** As principais palavras chaves foram "Service Oriented" e "Reuse Environment", com os seguintes termos relacionados:

- **Service Oriented:** "Service based", "Service-based", "Service Orientation", "SOA", "Service Oriented Architecture", "SaaS", "Web Service" e "Web Services"; e
- **Reuse Environment:** "Reuse Architecture", "Reuse Tool", "Reuse Repository", "Component Reuse" e "Software Reuse".

Vale a pena mencionar que vários acrônimos não foram considerados como termos relacionados. Esta decisão foi baseada no fato de que eles podem referir-se a outros termos não relacionados causando um aumento considerável do número de estudos primários a serem avaliados.

String de busca: Aplicou-se o operador booleano **OR** para ligar os termos principais e seus termos relacionados. Todos estes termos foram combinados usando o operador booleano **AND**. Desta maneira, a *string* final de busca ficou:

("Reuse Environment" OR "Reuse Architecture" OR "Reuse Tool" OR "Reuse Repository" OR "Component Reuse" OR "Software Reuse") AND ("Service Oriented" OR "Service Based" OR "Service-based" OR "Service Orientation" OR SOA OR SaaS OR "Web Service" OR "Web Services")

IV Definição dos estudos: Critérios de Exclusão e de Inclusão

Uma outra importante atividade do planejamento da revisão sistemática é a definição dos critérios de inclusão (CI) e exclusão (CE). Estes critérios tornam possível a inclusão de estudos primários que são relevantes para responder as questões de pesquisa e a exclusão dos estudos que não as respondam. Portanto, os critérios de inclusão definidos para revisão sistemática são:

- **CI1:** Estudos primários que propõem ou relatam o desenvolvimento de repositórios (ambientes) de software (módulos, componentes, serviços e etc) que utilizem serviços para tornarem seu conteúdo de software reutilizável;
- **CI2:** Os estudos primários propõem (relatam) processos, métodos, técnicas ou tecnologias que utilizam serviços para tornar um ambiente, software ou componente reutilizável;

- **CI3:** Os estudos primários não devem ser tutoriais, resumos de artigos, *proceedings* e workshops; e
- **CI4:** Os estudos primários devem estar disponíveis no formato completo.

Os critérios de exclusão são:

- **CE1:** Estudos primários que não propõem (ou relatam) o desenvolvimento de reuso de software, ambiente ou componente não baseadas em serviços;
- **CE2:** Estudos primários propõem contribuições para áreas não relativas ao reuso de ambientes (ou componentes) e serviços;
- **CE3:** O estudo primário é uma versão resumida de um estudo mais completo do mesmo tema e dos mesmos autores;
- **CE4:** O estudo primário não se encontra disponível; e
- **CE5:** Estudos primários disponíveis em línguas diferentes de inglês e português.

V Procedimentos adotados para Seleção dos Estudos

Na revisão sistemática, a seleção e avaliação dos estudos primários foi executada segundo os seguintes passos:

- Primeira seleção: inicialmente, a *string* de busca foi customizada e aplicada em cada base de dados de publicação, previamente listadas na [Tabela 1](#). Para isto, o título, resumo (ou em inglês *abstract*) e as palavras chaves de todos os estudos primários disponíveis foram considerados. Como resultado, um conjunto de estudos primários possivelmente relacionados ao tópico de pesquisa foram obtidos. Portanto, o estudo foi selecionado como relevante ou não. As seções de introdução e conclusão de cada estudo primário foram utilizadas, se necessário, para melhor compreensão do objetivo do estudo. Após esta análise, os estudos selecionados foram lidos integralmente, para próxima etapa do processo de seleção;
- Segunda seleção: Neste passo, cada estudo primário selecionado na fase anterior foi lido completamente e analisado novamente de acordo com os critérios de inclusão e exclusão. Em adição a esse processo, os trabalhos relacionados citados por estes estudos foram avaliados com a aplicação dos critérios de inclusão e exclusão tentando cobrir toda área de estudo. Esta etapa de seleção adicional poderia ser uma grande fonte de evidências, desde que os estudos incluídos frequentemente apresentassem trabalhos relacionados na mesma área de pesquisa. Se a decisão sobre a inclusão ou exclusão do estudo não for clara, este estudo pode ser incluído e analisado posteriormente por algum especialista na área, o que

não foi necessário nesta revisão. Se mesmo assim desacordos acontecerem, discussões posteriores devem ser conduzidas de modo a esclarecer a adição ou não do estudo; e

- Extração dos dados e método de síntese: para extrair os dados pode-se construir tabelas relacionadas a cada questão de pesquisa e usar gráficos para mostrar a aplicação dos critérios de inclusão e exclusão nos estudos primários.

VI Número de estudos retornados após a execução da string de busca nas bases de dados

Nesta seção é apresentado o número de estudos retornados por base de dados de publicação utilizada (elas foram mencionadas em [subseção III](#)) pela [Tabela 1](#). A lista dos números de artigos encontrados se encontra na [Tabela 2](#).

Tabela 2 – Números de estudos retornados por base de dados de publicação

Base de Dados	Total de estudos	Data de busca
ACM	10	05/11/2013
IEEE	111	05/11/2013
Science Direct	9	05/11/2013
Scopus	172	05/11/2013
Web of Science	79	05/11/2013

3.1.2 Condução

A revisão sistemática foi conduzida por somente uma pessoa, um estudante de mestrado, durante o período de outubro a dezembro de 2013.

I Primeira Seleção

Esta seção descreve em detalhes o processo de condução. Ao todo foram obtidos 381 estudos primários, porém, posteriormente como é descrito na etapa de extração, um trabalho foi manualmente incluído na seleção, o que não altera a contagem final de artigos.

Como o mesmo estudo primário pode estar incluído mais de uma *engine* de busca, antes de iniciar sua leitura esses estudos repetidos foram excluídos. Ao todo 117 estudos primários foram excluídos. Para suportar todo processo de revisão sistemática a ferramenta *State of the Art through Systematic Review (Start)*¹ foi utilizada, ela é uma ferramenta de código aberto desenvolvida por alunos da Universidade Federal de São Carlos. Foi utilizado em conjunto com a *Start*, uma outra ferramenta para auxiliar na manutenção das referências bibliográficas, a *Jabref*². Os detalhes da condução em cada base de dados é descrito a seguir.

Biblioteca Digital ACM:

¹ Pode ser encontrada em <http://lapes.dc.ufscar.br/tools/start-tool>

² Gerenciador de referências de código aberto, que pode ser encontrado em <http://jabref.sourceforge.net/>

A busca na base de dados *Association for Computing Machinery* (ACM) foi executada no dia 5 de novembro de 2013, onde dez estudos foram obtidos. A *string* de busca adaptada para esta base de dados foi:

String ACM:

((Title:(*"Reuse Environment"* or *"Reuse Architecture"* or *"Reuse Tool"* or *"Reuse Repository"* or *"Component Reuse"* or *"Software Reuse"*) and (*"Service Oriented"* or *"Service Based"* or *"Service-based"* or *"Service Orientation"* or *SOA* or *SaaS* or *"Web Service"* or *"Web Services"*))) or (Abstract:(*"Reuse Environment"* or *"Reuse Architecture"* or *"Reuse Tool"* or *"Reuse Repository"* or *"Component Reuse"* or *"Software Reuse"*) and (*"Service Oriented"* or *"Service Based"* or *"Service-based"* or *"Service Orientation"* or *SOA* or *SaaS* or *"Web Service"* or *"Web Services"*))) or (Keywords: (*"Reuse Environment"* or *"Reuse Architecture"* or *"Reuse Tool"* or *"Reuse Repository"* or *"Component Reuse"* or *"Software Reuse"*) and (*"Service Oriented"* or *"Service Based"* or *"Service-based"* or *"Service Orientation"* or *SOA* or *SaaS* or *"Web Service"* or *"Web Services"*))))

O maior problema enfrentado na utilização da base ACM foi que, ela permite a exportação de somente um estudo por vez.

IEEE Xplore: Para a base de dados *Institute of Electrical and Electronics Engineers* (IEEE) não foi necessária a definição da *string* para cada campo de pesquisa (por exemplo título, resumo e palavras chave como feito para ACM). Nesta base de dados foi necessário somente definir a *string* de busca e aplicá-la na busca avançada, pois a *engine* de busca, procura automaticamente por todos os campos desejados, bastando para tanto selecioná-los. A *string* definida para essa base de dados foi:

String IEEE:

(*"Reuse Environment"* OR *"Reuse Architecture"* OR *"Reuse Tool"* OR *"Reuse Repository"* OR *"Component Reuse"* OR *"Software Reuse"*) AND (*"Service Oriented"* OR *"Service Based"* OR *"Service-based"* OR *"Service Orientation"* OR *SOA* OR *SaaS* OR *"Web Service"* OR *"Web Services"*)

Aplicando-se essa *string* de busca na base de dados da IEEE no dia 5 de dezembro de 2013 obteve-se 111 estudos primários.

Science Direct: A busca na base de dados Science Direct foi executada no dia 5 de novembro de 2013, retornando 9 estudos primários. A *string* de busca utilizada está detalhada a seguir:

String Science Direct:

TITLE-ABSTR-KEY(("Reuse Environment" OR "Reuse Architecture" OR "Reuse Tool" OR "Reuse Repository" OR "Component Reuse" OR "Software Reuse") AND ("Service Oriented" OR "Service Based" OR "Service-based" OR "Service Orientation" OR SOA OR SaaS OR "Web Service" OR "Web Services"))

Scopus: No dia 5 de novembro de 2013 foi executada uma busca na base de dados Scopus e como resultado 172 estudos primários foram encontrados. Esta foi a base de publicação que mais retornou estudos. Portanto é importante destacar que a *engine* de busca da Scopus indexa estudos primários de muitas outras bases de publicação, por exemplo IEEE. Por outro lado, esta característica pode ser perigosa pois pode ocorrer a duplicidade de muitos artigos, visto que outras base de dados foram utilizadas. Mesmo assim essas outras base de publicação foram consideradas, pois a Scopus pode não indexar todos os estudos primários destas outras base de dados.

A *string* de busca da Scopus é definida a seguir:

String Scopus:

TITLE-ABSTR-KEY(("Reuse Environment" OR "Reuse Architecture" OR "Reuse Tool" OR "Reuse Repository" OR "Component Reuse" OR "Software Reuse") AND ("Service Oriented" OR "Service Based" OR "Service-based" OR "Service Orientation" OR SOA OR SaaS OR "Web Service" OR "Web Services"))

Web of Science: A busca nesta base de dados assim como nas demais foi realizada no dia 5 de novembro de 2013 e retornou 79 estudos primários. A *string* de busca aplicada sob esta base de publicações é definida a seguir:

String Web of Science:

TI= ("Reuse Environment" OR "Reuse Architecture" OR "Reuse Tool" OR "Reuse Repository" OR "Component Reuse" OR "Software Reuse") AND ("Service Oriented" OR "Service Based" OR "Service-based" OR "Service Orientation" OR SOA OR SaaS OR "Web Service" OR "Web Services")) OR TS= ("Reuse Environment" OR "Reuse Architecture" OR "Reuse Tool" OR "Reuse Repository" OR "Component Reuse" OR "Software Reuse") AND ("Service Oriented" OR "Service Based" OR "Service-based" OR "Service Orientation" OR SOA OR SaaS OR "Web Service" OR "Web Services"))

Nesta primeira fase de seleção dos estudos primários somente foram lidos os títulos, os resumos e as palavras chaves de todos os artigos para selecionar os mais adequados para leitura completa.

Nesta etapa, somente alguns critérios de inclusão e exclusão foram utilizados. Dentre os critérios de inclusão e exclusão utilizados tem-se: CI1, CI2, CE1 e CE2, detalhados na seção 3.1.1 e [subseção IV](#). Os demais critérios necessitam que os artigos sejam lidos por completo para serem aplicados, portanto eles foram executados na etapa de extração dos dados.

A próxima [Figura 4](#) sumariza o número de estudos selecionados, rejeitados e duplicados classificados por essa primeira etapa de seleção. Ao todo 53(14%) foram selecionados, 211(55%) foram rejeitados, enquanto que 117(31%) referem-se a estudos primários duplicados.

Figura 4 – *Status* da Revisão ao final da primeira seleção.

Fonte: Elaborada pelo autor.

Na [Figura 5](#) é ilustrada a distribuição dos artigos pelas diferentes bases de dados utilizadas nesta revisão. A distribuição dos artigos é a seguinte: 172(45%) Scopus, 111(29%) IEEE, 79(21%) Web of Science, 10(3%) ACM e 9(2%) Science Direct. A partir da definição da distribuição dos artigos é clara a maior contribuição das bases de dados Scopus, IEEE e Web of Science para o número de estudos primários encontrados, porém essas mesmas bases de dados tendem a possuir muitos estudos duplicados entre si, e de fato este comportamento foi comprovado.

Figura 5 – Cenário da distribuição do número de estudos em cada base de pesquisa.

Fonte: Elaborada pelo autor.

II Segunda Seleção

Nesta fase os estudos primários selecionados na etapa anterior foram lidos completamente e reclassificados em selecionados ou rejeitados. Os 53 artigos selecionados para leitura completa na etapa anterior foram lidos integralmente e os critérios de exclusão e inclusão que não foram aplicados na etapa anterior, são agora aplicados. Com a aplicação destes critérios e com a leitura dos estudos primários de forma completa chegou-se a um número final de cinco artigos selecionados. Uma lista parcial com 27 estudos mais representativos desta revisão, que foram importantes para obter informações para este trabalho de mestrado, pode ser encontrada no Apêndice C.

A [Tabela 3](#) ilustra alguns dos trabalhos escolhidos pela primeira seleção. Os cinco artigos selecionados são apresentados na [Tabela 3](#) com identificadores P1 a P5. Os artigos P1, P2, P3 e P5 foram escolhidos por apresentarem a proposição de novas arquiteturas de forma a ampliar ou promover o reúso de software. Essas arquiteturas propostas baseiam-se na idéia de repositório de artefatos (ou componentes) para sua implementação, apenas o trabalho P2 utiliza o conceito de nuvem, ao invés de um repositório de artefatos. Três dos artigos escolhidos (P1, P2 e P5) promoveram junto de uma nova arquitetura, o uso de serviços para permitir e ampliar o reúso de software. Finalmente, o estudo P4 foi escolhido por apresentar a reutilização de código legado através da sua publicação por meio de serviços, utilizando um repositório de serviços para seu desenvolvimento.

Tabela 3 – Parte dos artigos selecionados pela primeira seleção

Princípio da Tabela 3		
Identificador	Título	Autor(es)
P1	An architecture model for dynamically converting components into Web services	(LEE; KIM; YANG, 2004)

Continuação da Tabela 3		
Identificador	Título	Autor(es)
P2	Intelligent business processes composition based on multi-agent systems	(CORIA ; CASTELLANOS-GARZÓN ; CORCHADO , 2013)
P3	Reuse-Oriented Deployment of Software Components: Congregation in Service-Oriented Development	(ATKINSON ; BOSTAN ; FINK , 2009)
P4	Reusing existing object-oriented code as web services in a SOA	(SNEED ; VERHOEF ; SNEED , 2013)
P5	Service-oriented lightweight software asset repository open framework	(HONGMIN ; JIN ; JINGZHOU , 2010)
P6	Client-Cloud Computing: A Framework for Service Integration	(CHANG ; KAO ; LO , 2012)
P7	Contextualized User-Centric Multimedia Delivery System for Next Generation Networks	(SIMOES ; MAGEDANZ , 2011)
P8	Registry and Discovery of Services with Variability Based on 2-Level UDDI	(YEOM ; PARK ; MOON , 2011)
P9	Service oriented programming: A new paradigm of software reuse	(SILLITTI ; VERNAZZA ; SUCCI , 2002)
P10	The study on software reuse based on service composition and dependency-aware service oriented architecture	(LIU , 2012)
P11	Towards a semantic-based approach for software reusable component classification and retrieval	(YAO ; ETZKORN , 2004)
Final da Tabela 3		

Os demais trabalhos apresentados na [Tabela 3](#) destacam-se por propor melhorias nos métodos de busca, registro ou composição de serviços em repositório de reuso e possuem os seguintes identificadores: P6, P7, P8, P9, P10 e P11. Apesar disso, esses trabalhos não foram selecionados pois fogem as questões e ao objetivo da revisão sistemática.

Nas próximas seções, os trabalhos P1 a P5 são resumidos, com o intuito de destacar suas características mais importantes, as quais ajudaram na elaboração e proposição do novo levantamento bibliográfico [seção 3.2](#).

III Resumo do estudo primário Service-Oriented Lightweight Software Asset Repository Open Framework - P5

[HongMin, Jin e JingZhou \(2010\)](#) propuseram um repositório de ativos baseado em serviços de arquitetura aberta *Software Asset Repository Management System* (SARMS). O

SARMS é um conjunto de sistemas de software que controlam a criação, manutenção e o uso do repositório. O subsistema compartilhado de ativos de serviço (*Asset Sharing Service Subsystem* (ASSS)) é essencial ao SARMS, e é responsável por fornecer o compartilhamento de serviços de interoperabilidade para ferramentas e repositórios de terceiros, como por exemplo ferramentas de análise, ferramentas de design, ferramentas de implementação, repositórios de ativos de empresas, e etc. ASSS fornece quatro categorias de compartilhamento de serviços, são elas: Gerenciamento de Serviços do Usuário (do inglês *User Management Service*) responsável por manutenções de usuários, Gerenciamento de Serviços de Ativos (do inglês *Asset Management Service*) responsável por compartilhar a identificação e a organização dos ativos, Gerenciamento de Serviços de Autorização (do inglês *Authorization Management Service*) responsável por garantir as permissões de acesso dos usuários compartilhados, e por último temos o Serviço de Meta Dados (do inglês *Meta Data Service*) responsável por prover meta-informação do repositório (HONGMIN; JIN; JINGZHOU, 2010). Na Figura 6 abaixo pode-se observar o SARMS proposto por HongMin, Jin e JingZhou (2010):

Figura 6 – Service-Oriented Asset Repository Open Architecture.

Fonte: HongMin, Jin e JingZhou (2010).

O modelo de informação compartilhado de repositório de ativos (*Asset Repository Sharing Information Model* (ARSIM)) define o conjunto mínimo de entidades de informação sobre compartilhamento de repositórios de ativos e relacionamento entre eles, o que fornece uma base uniforme para compartilhamento de operações de serviço de ativos (HONGMIN; JIN; JINGZHOU, 2010). Dentre elas, duas se destacam:

- Modelo de informação orientado a objeto (do inglês *Object-Oriented Information Model*): O modelo de informação consiste em quatorze classes, onde cada uma possui um conjunto de atributos, operações e relacionamentos. As seis classes núcleo do modelo são: *Software-*

Resource, *ResourceURL*, *User*, *UserGroup*, *PrivilegedItem* e *Authorization*. Este modelo é ilustrado pela na [Figura 7](#);

- Gerenciamento de permissões em repositórios abertos (do inglês *Permission Management in Open Repository*): ARSIM alinha os usuários e grupos baseados em permissões usando a classe *UserGroup* em conjunto com a classe *User*. A classe *Authorization* é usada para modelar que permissões de ativos específicos são identificados pela instância *ResourceURL* e são permitidos por algum objeto *UserGroup*. ARSIM possui seis classes de permissões para permitir um controle fino das interoperações, essas classes são: *SearchPrivilege*, onde os usuários podem somente recuperar os ativos reutilizáveis; *RestrictReadPrivilege*, usuários podem somente ler de forma resumida as informações dos ativos; *ReadPrivilege*, onde usuários podem ler todas as informações relativas ao ativos; *DownloadPrivilege*, os usuários podem adquirir os ativos; *UpdatePrivilege*, usuários podem atualizar os ativos; e por fim *UploadPrivilege*, em que usuários podem fazer o *upload* dos recursos pertencentes aos ativos.

Figura 7 – Asset Repository Sharing Information Model.

Fonte: [HongMin, Jin e JingZhou \(2010\)](#).

[HongMin, Jin e JingZhou \(2010\)](#), desenvolveram interfaces programáticas de serviço baseadas no modelo ARSIM, para fornecer uma série de operações que podem ser acessadas por ferramentas de terceiros para comunicarem-se com o repositório de ativos compartilhado. Essas interfaces são: *Service Interface Protocols* (ou Interface de Protocolos de Serviço), onde SOA fornece mecanismos embutidos para facilitar o fraco acoplamento entre serviços e outros sistemas. Desacoplados de sistemas de terceiros em relação a localização, a protocolos e a duração, as interfaces de programação de repositórios são fornecidas como webservices e registradas como um registro de serviço para fácil descoberta e ligação com repositórios remotos e ferramentas

através de SOAP, WSDL e XML , como protocolos subjacentes; e por último *IService Interface Lists* (ou Interface de Listas de Serviço).

IV Resumo do artigo *Reuse-Oriented Deployment of Software Components* - P3

Atkinson, Bostan e Fink (2009) introduziram um novo conceito referido como congregação (do inglês *Congregation*). Congregação, segundo eles, é um conjunto de ativos de software reutilizáveis e configuráveis coletados com o propósito de construir aplicações distribuídas orientadas a serviço.

As operações de um objeto tipicamente compartilham variáveis de dados, que captam o estado corrente do objeto. Os autores se referem a esse termo como um tipo de estado conhecido, estado de coesão (ou de aderência). Aplicando este conceito ao paradigma de orientação a serviço, observa-se que serviços possuem seu estado de coesão revelados por uma ou mais de suas operações, que são objetos no sentido de orientação a serviços (ATKINSON; BOSTAN; FINK, 2009).

Atkinson, Bostan e Fink (2009) propuseram uma arquitetura chamada de *engine* de congregação. Essa *engine* contém alguns componentes lógicos, como: *Populator*, *Registry and Repository*, *Deployment Engine* e *Exporter*. Essa *engine* pode ser vista na Figura 8.

Uma vez que os ativos para uma certa aplicação ou para um certo processo de negócios tenham sido obtidos, seja por meio de uma busca ou por um processo de desenvolvimento normal, o *Populator* é trazido à tona. O *Populator* é conectado a uma *engine* de busca de componentes como Merobase, por exemplo, que pode ser usada para encontrar ativos de software via interface *web* usando *queries* especiais de busca ou em um ambiente de desenvolvimento integrado (como Eclipse) rodando um *plug-in* como Code Conjurer. Em ambos os casos a API *Populator* é usada para adicionar e configurar os ativos (ATKINSON; BOSTAN; FINK, 2009). Quando os ativos de software são adicionados em forma de código fonte, as fontes dos ativos e suas propriedades precisam ser configuradas e armazenadas no *Registry and Repository* para posterior aplicação e recuperação de informações de segurança (ATKINSON; BOSTAN; FINK, 2009). Se um ativo de uma congregação deve ser implantado para tornar-se acessível para aplicações ou processos de negócio, um engenheiro humano necessita acessar a interface de usuário da *Congregation Engine*. O congregante (engenheiro humano ator principal da congregação) necessita tornar claro, quais ambientes de hospedagem estão disponíveis para apoiar a implantação dos ativos de software para diferentes plataformas de aplicações servidoras. O componente *Deployment Engine*, introduz uma interface de *plugins*, suas interfaces, um modelo genérico de implantação. A *Deployment Engine* tem como objetivo principal tornar-se mais extensível para novas plataformas alvo, a fim de fornecer um modelo unificado de implantação de ambientes de hospedagem. E assim quando o engenheiro humano desejar implantar um ativo de congregação ele especifica quais métodos de ativos os clientes devem ser capazes de usar, bem como a plataforma alvo. Com isso *Deployment Engine* toma todas as ações necessárias para automatizar a implantação do ativo de congregação

na plataforma, incluindo modificações no ativo (ATKINSON; BOSTAN; FINK, 2009).

Quando os ativos de congregação necessitam ser acessados por uma aplicação ou processo de negócios sob desenvolvimento em tempo de projeto, o componente *Explorer* da *engine* da congregação é responsável por entregar o código cliente adequado para a plataforma cliente escolhida em forma de um *proxy* para apoiar o uso do ativo de uma maneira específica. Com uma arquitetura de plug-ins similar à adotada pela *Deployment Engine*, o *Exporter* usa os *plugins* para criar os artefatos do lado cliente na plataforma e linguagem requerida (ATKINSON; BOSTAN; FINK, 2009).

Figura 8 – Basic Architecture of a Congregation Engine.

Fonte: Atkinson, Bostan e Fink (2009).

Resumindo a arquitetura proposta por Atkinson, Bostan e Fink (2009), traz à tona uma plataforma para submissão de ativos de software e de sua recuperação e utilização por aplicações externas, respeitando as características do sistema anteriormente descrito.

V Resumo do Artigo Intelligent business processes composition based on multi-agent systems - P2

Coria, Castellanos-Garzón e Corchado (2013) introduziram uma nova arquitetura denominada *Intelligent business Processes Composition based on multi-Agent systems, Semantics and Cloud Integration* (IPCASCI), que tem por objetivo compor *web services*, para obter novos serviços de maneira automática iniciando pelos requisitos enviados pelos usuários. Este processo

foi executado de tal forma que a construção automática de serviços foi eficiente e possui um comportamento inteligente. Comportamento inteligente compreende a habilidade do sistema em processar os requisitos de entrada introduzidos pelo usuário, de tal forma a executar os seguintes passos: a) análise dos dados de entrada; b) encontrar os *web services* que atendem aos requisitos de entrada; e c) executar a composição automática dos *web services* para seus processos de negócio correspondentes. Portanto, como resultado final, novos serviços foram implementados pelos requisitos fornecidos, que foram alcançados por um processo automático de composição e descoberta.

Os diferentes componentes que integram a arquitetura IPCASCI são ilustrados pela Figura 9. Os itens contidos nesta figura são descritos a seguir (CORIA; CASTELLANOS-GARZÓN; CORCHADO, 2013):

- Sistema de Nuvem (do inglês *Cloud System*): Plataforma de computação em nuvem que apoia a arquitetura proposta. A plataforma fornece um ambiente de execução e armazenamento de dados;
- *Web services*: Os serviços existentes na arquitetura são usados pelo processo para composição de novos *web services*;
- Registro UDDI (do inglês *Universal Description Discovery and Integration*): Sistema universal de registro, onde os serviços utilizados são armazenados. Isto permite um acesso aberto aos serviços armazenados na arquitetura. Desta forma, os *web services* implementados pela arquitetura podem ser reutilizados independentemente da proposta dos autores;
- Sistema multi-agente baseado em organizações virtuais: Este sistema implementa funcionalidades que executam a descoberta e composição de serviços: a) Sistema de análise: analisa o conteúdo semântico introduzido pelo usuário, para estruturá-lo em itens computacionais; b) Sistema de busca: descoberta de serviços que armazenam a semântica e as restrições sintáticas fornecidas pelo usuário; e c) Sistema de composição: uma vez determinados os serviços e seus relacionamentos, que armazenam os requisitos semânticos do usuário, a composição do serviço é aplicada sob esses serviços para obtenção de um novo serviço, que satisfaça os requisitos do usuário;
- Ontologias: ontologias distintas modelam o conhecimento semântico que pode ser incluído nos serviços;
- Arquivo *Business Process Execution Language* (BPEL): armazena a composição de serviços que preencham os requisitos apresentados pelo usuário para obtenção da solução desejada.

Figura 9 – Descrição da arquitetura IPCASCI para composição de processos de negócio (serviços).

Fonte: Coria, Castellanos-Garzón e Corchado (2013).

Os passos seguidos pela arquitetura IPCASCI podem ser vistos no algoritmo apresentado na Figura 10. O detalhamento do algoritmo de descoberta e composição de serviços (*Algorithm for web service discovery e composition (AWSDC)*) é fornecida a seguir (CORIA; CASTELLANOS-GARZÓN; CORCHADO, 2013):

- O usuário introduz os requisitos do novo serviço a ser construído por meio de um sistema assistido. Esse sistema permite ao usuário definir uma lista de termos (em forma de módulos) representando os requisitos do novo serviço. A lista de termos é delimitada pelo repositório de serviços do sistema da nuvem. O resultado deste processo é um conjunto de módulos relacionados (sistema de análise) de tal maneira que o registro de cada módulo tenha as seguintes informações: a) Nome; b) Entrada; c) Saída; d) Processo executado pelo módulo (conceito semântico que ele representa); e) Domínio de conceito semântico representado pelo módulo (domínio ontológico); f) Pré-condições de execução; e g) Relacionamento com outros módulos;
- Para cada módulo do passo acima uma busca automática dos serviços, que incluem os requisitos do módulo, é executada. Além disso, a busca é realizada de maneira sintática e semântica;
- Das informações extraídas do passo acima, um diagrama BPD (do inglês *Business Process Diagram*) reduzido é construído através do diagrama padrão BPMN (do inglês *Business Process Management Notation*), no qual cada serviço é representado como uma atividade e também mostrando as interações existentes entre os serviços;

- O diagrama BPD é exibido pelo software assistente do usuário, pois podem existir vários serviços que tenham implementado uma atividade simples. Neste caso, o usuário pode selecionar um serviço diferente do apresentado como padrão pelo diagrama, ou seja, ele pode usar o serviço que quiser, pois todos implementam a mesma atividade;
- Ontologias: ontologias distintas modelam o conhecimento semântico que pode ser incluído nos serviços;
- Do diagrama BPD, uma composição BPEL é executada de tal forma que o serviço especificado pelo usuário é construído para poder ser posteriormente invocado como mais um serviço da plataforma.

A [Figura 11](#) ilustra o processo descrito no algoritmo anteriormente descrito.

Figura 10 – Estágios da arquitetura IPCASCI para obtenção do serviço final do processado de descoberta e composição.

Fonte: [Coria, Castellanos-Garzón e Corchado \(2013\)](#).

VI Resumo do artigo *An Architecture Model for Dynamically Converting Components into Web Services* - P1

Um componente é uma entidade lógica que serve a um propósito especial. Componentes são auto descritivos. A interface de um componente descreve detalhes suficientes para fazer com que seja possível sua utilização pelos usuários ([LEE; KIM; YANG, 2004](#)).

[Lee, Kim e Yang \(2004\)](#) adaptaram o modelo *Component-based development* (CBD) *workbench*, para permitir o uso de serviços no reúso dos componentes. Este *workbench* é um modelo de arquitetura para desenvolvimento baseado em componentes. Ele engloba desenvolvimento, entrada de dados, busca e recuperação de componentes homogêneos ou heterogêneos. O

Figura 11 – Visão alternativa do processo de obtenção da solução final para o serviço pela arquitetura IPCASCI.

Fonte: Coria, Castellanos-Garzón e Corchado (2013).

objetivo principal do *workbench* é atingir a interoperabilidade entre componentes por meio de um padrão comum (LEE; KIM; YANG, 2004).

O modelo de arquitetura do modelo CBD utilizado para adaptação pode ser visto na Figura 12. Este modelo de repositório (CBD *workbench*) consiste de três subsistemas maiores (LEE; KIM; YANG, 2004):

- Subsistema fornecedor (*Provider Subsystem*);
- Subsistema consumidor (*Consumer Subsystem*);
- Repositório de componentes (*Component Repository*).

O subsistema fornecedor é responsável por aceitar componentes de qualquer tipo para o repositório. O fornecedor é geralmente um desenvolvedor, um vendedor independente de software (*Independent Software Vendor* (ISV)) ou uma empresa que desenvolve e fornece componentes funcionais, homogêneos ou heterogêneos que podem ser recuperados do repositório de componentes para desenvolvimento de novas aplicações (LEE; KIM; YANG, 2004).

O subsistema consumidor representa o usuário que está interessado em utilizar componentes em sua aplicação. Esse subsistema busca por um componente específico baseado nos requisitos fornecidos (LEE; KIM; YANG, 2004).

Finalizando o modelo CBD *workbench*, têm-se finalmente, o Repositório de Componentes. Esse repositório é composto de um diretório servidor, onde os componentes são armazenados e um banco de dados é utilizado para armazenar os atributos dos componentes (LEE; KIM; YANG, 2004).

Figura 12 – Arquitetura do CBD *workbench*.

Fonte: Lee, Kim e Yang (2004).

O CBD *workbench* foi estendido para integrar as funcionalidades da tecnologia de web services. Esse novo modelo arquitetural pode ser observado pela na [Figura 13](#). O modelo arquitetural proposto, permite a interoperabilidade expondo componentes escritos em Java ou C++ como serviços. Como a tecnologia de *web services* segue um padrão aberto da internet, os componentes que são convertidos em serviços são tornados interoperáveis facilmente (LEE; KIM; YANG, 2004).

Os consumidores buscam por um componente baseado nas características desejadas e na funcionalidade possuída pelo componente. O serviço de *web service* principal procura no repositório por componentes baseado no critério de busca inserido pelo consumidor. Quando este critério coincide com o encontrado, um *web service* é dinamicamente gerado para o componente selecionado e os detalhes do *web service* são retornados para o cliente. Já quando o critério de pesquisa não coincide com os de nenhum componente, então a busca é feita agora nos registros UDDI e se houver um serviço que tenha as funcionalidades desejadas pelo consumidor, então seus detalhes são repassados ao cliente. Assim funciona a execução no modelo proposto na [Figura 13](#) (LEE; KIM; YANG, 2004).

Foi proposto por Lee, Kim e Yang (2004) a conversão de componentes escritos em Java

Figura 13 – Modelo arquitetural do CBD workbench adaptado para serviços.

Fonte: Lee, Kim e Yang (2004).

e C++ na forma de serviços. Para implantar componentes Java como serviços, foram utilizadas as funcionalidades do Apache SOAP. Um WSDL foi criado a partir da ferramenta java2wsdl disponível no Apache AXIS (LEE; KIM; YANG, 2004).

Enquanto que para conversão de componentes em C++, a ferramenta escolhida foi o *toolkit* gSOAP. Esse *toolkit* é usado para criar *web services* a partir de componentes escritos em C/C++ (LEE; KIM; YANG, 2004).

Segundo Lee, Kim e Yang (2004) os componentes escritos em Java e C++ foram convertidos com sucesso para *web services*.

VII Resumo do Artigo *Reusing Existing Object-oriented Code as Web-Services in a SOA - P4*

Em um trabalho anterior, [Sneed, Schedl e Sneed \(2012\)](#), promoveram o empacotamento de software legado escrito em COBOL. Para esse empacotamento o código passou por um processo de reengenharia para que possíveis pontos de entrada pudessem ser descobertos ([SNEED; VERHOEF; SNEED, 2013](#)).

No caso de códigos orientados a objetos, os pontos de entrada já estão definidos antecipadamente. Eles são identificados como sendo métodos públicos que tem uma interface do tipo API, que pode ser invocada externamente com os requeridos tipos de parâmetros de entrada ([SNEED; VERHOEF; SNEED, 2013](#)).

Neste novo trabalho, [Sneed, Verhoef e Sneed \(2013\)](#) propõem empacotar códigos escritos nas linguagens C++ e Java por meio da utilização de serviços, para tornar esses códigos passíveis de execução remota. A abordagem proposta neste novo trabalho envolve três transformações de linguagem: a) na primeira transformação as interfaces de métodos são convertidas para uma tabela relacional; b) na segunda transformação a interface WSDL é gerada a partir da tabela relacional; e c) e na terceira transformação o processo de negócios BPEL, que consome o serviço, e um *script* de teste, são gerados a partir da interface de serviço WSDL ([SNEED; VERHOEF; SNEED, 2013](#)). Esses passos podem ser observados na [Figura 14](#).

Figura 14 – Transformações de códigos executadas pelo modelo.

Fonte: [Sneed, Verhoef e Sneed \(2013\)](#).

O processo de transformação das interfaces de métodos das linguagens C++, C# e Java para interfaces de operação WSDL é realizado de maneira simples. Primeiramente, somente os métodos públicos são considerados para que a transformação seja efetuada. Depois, os nomes dos métodos das interfaces das linguagens orientadas a objeto se tornam os nomes das operações, assim como os nomes e tipos dos argumentos dos métodos tornam-se parâmetros de entrada em WSDL. Porém, existem dois problemas que podem dificultar essa conversão, são eles: a) conversão dos resultados de retorno, visto que dentro de uma mesma interface de método pode existir mais de um retorno do mesmo tipo, porém com valores diferentes. Os retornos do método

devem ser recolhidos e convertidos em um conjunto alternativo de parâmetros de saída WSDL; e b) o segundo problema acontece quando existem métodos polimórficos, ou seja, métodos que possuem o mesmo nome e podem aparecer em muitas classes. Portanto, para diferenciar esses métodos é necessário qualificar o nome das operações WSDL correspondente a eles. Essa qualificação do nome é feita por exemplo adicionando-se o nome da classe que contém o método no nome da operação. Normalmente o nome da classe é colocado anteriormente ao nome da operação, e ambos são separados por um ponto (SNEED; VERHOEF; SNEED, 2013).

Quatro ferramentas diferentes foram utilizadas para minerar o código orientado a objetos e extrair os serviços utilizados na arquitetura SOA. São eles (SNEED; VERHOEF; SNEED, 2013):

- SoftRedoc: analisa o código orientado a objetos identificando os métodos públicos e interfaces, para convertê-los em uma tabela de casos e operações com seus parâmetros e tipos de retorno. Essa ferramenta cria uma tabela *Comma Separated Values* (CSV) para cada componente;
- WSDLGen: processa as tabelas CSV para criar uma definição de interface de serviço WSDL para cada componente, incluindo declaração de dados, operações, mensagens e parâmetros do componente;
- SoftReuse: processa a interface de definição WSDL para criar um repositório de interfaces para inspeção e edição de elementos das interfaces de serviço. Para operar isso a ferramenta possui uma interface gráfica de usuário, onde uma tabela relacional contendo serviços, operações, mensagens e parâmetros é exibida para manipulação. Essa ferramenta pode ser vista na Figura 16;
- BPELGen: executa uma busca no repositório de interfaces, para encontrar uma interface específica e gera a definição de processo BPEL para este serviço, bem como um *script* de teste para testar o processo.

Todo o processo e de transformação, bem como de linguagem em conjunto com as ferramentas de mineração são apresentadas na Figura 15.

O método desenvolvido por Sneed, Verhoef e Sneed (2013) foi testado tanto em aplicações Java como DotNet. Segundo Sneed, Verhoef e Sneed (2013), os testes do método proposto aplicado as duas aplicações complexas (uma em Java e outra utilizando a plataforma DotNet) mostraram-se um sucesso, em virtude da criação dos serviços para estas aplicações, permitindo assim uma maior interação com o mundo exterior.

I Questão 1 : Quais são os estudos primários existentes relacionados com o desenvolvimento de software, componentes ou ambientes de reuso orientado a serviços?

Como descrito anteriormente alguns artigos propuseram melhorias na busca, registro e composição de serviços, entretanto a maioria dos estudos primários encontrados referiam-se a outros temas muito divergentes da proposta procurada por esta revisão sistemática.

Os artigos selecionados ao final da fase de seleção foram muito interessantes e puderam ser utilizados como fonte de inspiração para nova revisão bibliográfica proposta. Essa revisão é descrita na [seção 3.2](#). O estudo primário P5 propõe o desenvolvimento de arquitetura orientada a serviços a partir de uma arquitetura existente, chamada de CBD (*Component-based development*) *workbench*. Esse trabalho responde a questão 1 pois propõem a utilização de um ambiente integrado de ativos, onde os serviços podem ser reutilizados desde que se encaixem em uma das quatro categorias de compartilhamento declaradas no ambiente integrado proposto. Essas categorias se parecem com módulos que classificam os serviços por áreas de operação, ou seja, os módulos podem tornar-se especialistas no atendimento de certos tipos de demandas.

O estudo primário P2 propõe uma arquitetura orientada a serviços em nuvem, onde os serviços são armazenados em um repositório UDDI e assim podem ser reutilizados por outras aplicações ou serviços. Este trabalho responde com sucesso a questão de pesquisa aqui apresentada, assim como os trabalhos P1 e P5.

O trabalho P4 é o mais diferente de todos, pois além deste trabalho basear-se no reaproveitamento de código legado (C++ e Java), a partir da criação de interfaces de serviço armazenadas em um repositório, enquanto que os outros trabalhos propõem arquiteturas novas ou extensão de algumas existentes para promover o reuso.

II Questão 2 : Quais as tecnologias desenvolvidas e adotadas para o desenvolvimento de software, componentes ou ambientes de reuso orientados a serviço?

Os trabalhos propostos apresentaram definições de novas arquiteturas de software para promover o reuso tanto de componentes, código fontes e ativos de software. Os trabalhos que propuseram novas arquiteturas foram P1, P2, P3 e P5. Desses, dois se destacaram mais: P2 por se tratar da proposição de uma arquitetura de serviços baseada em computação em nuvem, ou seja, um tema muito atual e recorrente e P5 porque o modelo proposto é uma referência importante de como integrar módulos diferentes dentro de um mesmo repositório, permitindo a composição de novos serviços ou mesmo comunicação com aplicações externas, o que é similar ao que o projeto propõem.

Dentre as principais tecnologias adotadas pelos trabalhos destacam-se: XML (para realização de comunicação entre aplicações escritas em diferentes linguagens), WSDL (para descrição do serviço) e UDDI (registro de serviços).

III Questão 1.1 : Quais os tipos de arquitetura de software aplicada no desenvolvimento de ambientes de reuso orientados a serviço?

A maioria dos estudos selecionados utiliza a arquitetura SOA (*Service-Oriented Architecture*) para implementar suas propostas. Somente um estudo não propôs uma arquitetura, o P4. Porém, os que propuseram uma nova arquitetura apenas dois destacam-se, P2 e P5. P2 propôs uma arquitetura denominada IPCASCI (*Intelligent business Processes Composition based on multi-Agent systems, Semantics and Cloud Integration*) e P5 propôs um repositório de ativos de software baseados em serviços de arquitetura aberta ou SARMS (*Software Asset Repository Managment System*). Outro trabalho, o P3, propôs uma arquitetura, denominada congregação.

IV Questão 1.2 : Quais as principais dificuldades apresentadas no desenvolvimento de software, componentes, ou ambientes de reuso orientados a serviço?

Ao realizar a leitura completa dos artigos selecionados, bem como dos rejeitados, foram encontrados na maior parte das vezes traços similares de dificuldades de implementação de serviços para promover reutilização de software. Dentre essas dificuldades algumas podem ser destacadas:

- Demanda de profissionais que dominem as tecnologias utilizadas na implantação e desenvolvimento de ambientes ou repositórios orientados a serviços. Tecnologias como WSDL, XML, UDDI e SOAP;
- Necessidade de conhecimento em reengenharia. Dependendo do tipo do projeto, talvez, seja necessário uma reengenharia no código ou componente legado a fim de determinação de possíveis pontos de variação, onde serviços e aplicações externas podem estabelecer comunicação com o software reusado.

V Questão 1.3 : Quais as principais vantagens apresentadas pelo desenvolvimento de software, componentes e ambientes orientados a serviço?

Existem diversas vantagens na utilização de serviços para promoção do reuso de software em um ambiente ou repositório de software. Dentre essas vantagens pode-se destacar:

- Aumento da confiabilidade do sistema, pois como as novas aplicações são ao menos em parte constituídas de softwares legados já testados e confiáveis, automaticamente esses novos sistemas já tem um grau de confiabilidade maior que os desenvolvidos do zero;
- Aumento da produtividade, pois ao menos parte das aplicações surgem de softwares legados a carga de trabalho é reduzida;
- Diminuição do custo e do tempo de desenvolvimento pelo mesmo motivo explicado no item acima.

3.2 Revisão bibliográfica derivada da revisão sistemática

Nesta seção são descritos a motivação para realização de uma nova revisão bibliográfica e também a metodologia adotada nesta revisão.

3.2.1 *Motivação da Revisão Bibliográfica*

Durante o desenvolvimento da revisão sistemática abordada na [seção 3.1](#), pode-se observar a existência de poucos trabalhos que propõem a geração de serviços automaticamente. O único trabalho encontrado que propôs a geração de serviços foi o de [Sneed, Verhoef e Sneed \(2013\)](#). Este trabalho propunha a geração de serviços através de código legado, ou seja, o serviço era gerado a partir de uma implementação existente, portanto apresentando baixo nível de abstração. Além disso, a AIRES³ serviu também de motivação para a realização da nova revisão bibliográfica, pois como ela é uma arquitetura baseada em SOA, a geração automática de serviços passou a ser um tema importante de pesquisa para permitir sua implementação.

3.2.2 *Objetivo da Revisão Bibliográfica*

Considerando a motivação da revisão bibliográfica, identificou-se que era necessário a realização de um novo levantamento de estudos primários que tivessem por objetivo a geração automática de serviços a partir de modelos abstratos, como por exemplo diagrama de classes UML. Nota-se ainda pela revisão sistemática e pela busca de métodos de geração de serviços a partir de modelos, que as principais operações que os serviços necessitam implementar são as operações de CRUD⁴. Tendo por base essas suas motivações chega-se a conclusão que uma nova revisão bibliográfica que permita encontrar estudos primários que proponham geração de serviços com operações do tipo CRUD a partir de modelo abstratos é um bom campo para pesquisa, e portanto uma nova revisão bibliográfica foi realizada.

3.2.3 *Planejamento da Revisão Bibliográfica*

Tendo em vista a motivação desenvolvida na [subseção 3.2.1](#) uma nova revisão bibliográfica foi proposta. Essa revisão baseia-se nos conceitos de revisão sistemática apresentados anteriormente, porém não aplica todos eles no seu desenvolvimento por uma limitação de tempo do autor do texto. Dentre os conceitos empregados têm-se a definição da *string* de busca, aplicação dela em base de dados bibliográficas e a execução da primeira e segunda seleção na [subseção 3.2.4](#).

³ AIRES: arquitetura para desenvolvimento de software focada no reúso de artefatos, foi apresentada em um artigo submetido em uma conferência internacional, e está em análise.

⁴ CRUD : Este termo é normalmente relacionado como um acrônimo para *Create, Read, Update e Delete*, que são as quatro operações básicas utilizadas em base de dados relacionais ou em interfaces para criação, consulta, atualização e destruição de dados, respectivamente. Fonte: <https://pt.wikipedia.org/wiki/CRUD>

A *string* de busca foi calibrada em duas bases de dados bibliográficas (IEEE e Scopus). Essa *string* de busca foi formatada após a leitura de artigos oriundos da revisão sistemática abordada anteriormente e a leitura de artigos específicos sobre geração de código através de modelos abstratos. Com isso chegou-se a *string* apresentada abaixo:

String de busca:

("Code Generation" OR "Code generation" OR "code generation" OR "code-generation") AND ("Web services" OR "Web-services" OR "Web Services") AND ("MDD" OR "Model Driven Development") AND ("UML" OR "Unified Modeling Language")

Os termos chave da *string* de busca apresentada anteriormente são: *Code Generation*, *Web services*, *Model Driven Development* e *UML*. Essa *string* de busca foi calibrada principalmente na base de dados bibliográficos IEEE.

Após sua calibração ela foi executada em duas bases de dados bibliográficos: IEEE e Scopus. Essas duas bases de dados foram adotadas pelo alto número de artigos relevantes retornados em pesquisas deste tipo. Na busca realizada na IEEE foram encontrados 190 estudos primários, já na Scopus, 8 estudos foram encontrados. Os estudos primários encontrados na Scopus existiam também na IEEE e por esse motivo essa base de dados foi deixada de lado e a etapa de seleção foi efetuada somente nos artigos retornados pela IEEE.

3.2.4 Condução da Revisão Bibliográfica

Nesta etapa é descrito como a seleção dos artigos encontrados na IEEE foi realizada e resumos dos artigos selecionados apresentados.

I Primeira seleção

Nessa etapa do processo de condução da revisão bibliográfica os artigos são selecionados com base nos objetivos descritos na [subseção 3.2.2](#). Os objetivos são enumerados a seguir:

1. Geração automática de código através de modelos abstratos, como por exemplo diagramas UML;
2. Estudos primários que apresentem implementações de operações de CRUD para serviços.

Os estudos primários considerados aptos a serem selecionados atenderam ao menos um dos objetivos anteriormente descritos. Os estudos que fugiram a algum dos objetivos foram automaticamente excluídos.

Após as definições dos critérios de seleção a serem aplicados na revisão bibliográfica, todos os títulos e resumos dos 190 estudos primários, encontrados na IEEE, foram lidos e

submetidos a aplicação dos critérios. Ao final da aplicação dos critérios de seleção chegou-se ao número de 30 artigos para prosseguir para segunda seleção.

II Segunda seleção

Nesta etapa 30 artigos selecionados durante a última fase foram lidos por completo e os critérios descritos na [subseção I](#) foram novamente aplicados. Durante essa fase notou-se que poucos trabalhos encontrados propunham a geração completa de código, a partir de modelos abstratos, ou seja, poucos estudos primários propunham a geração de serviços com a implementação de suas operações de maneira completa, em sua maioria os estudos propunham a geração de interfaces de serviços ou somente a geração de seus contratos. Como somente interessava a essa pesquisa os estudos primários que propusessem a geração completa de serviços e suas operações, a maioria foi excluída ao final dessa seleção. No total foram selecionados somente três artigos e eles são listados na [Tabela 4](#).

Tabela 4 – Estudos selecionados ao final da revisão bibliográfica.

Título	Ano de Publicação
Top-Down Approach for Web Services Development	2010
Engineering Adaptable Service Oriented Systems: A Model Driven Approach	2009
An EMF-based Toolkit for Creation of Domain-specific Data Services	2014

Dois dos três estudos primários listados na [Tabela 4](#) realizam a geração de códigos de serviços a partir de um novo metamodelo originado a partir da UML ([KENZI *et al.*, 2009](#); [BENDER; BOZIC; KONDOV, 2014](#)). O trabalho de [Bellini, Prado e Zaina \(2010\)](#) apesar de gerar apenas o esqueleto do serviço, ou seja, os métodos são gerados sem corpo e possuindo somente suas assinaturas, é importante a ponto de ser selecionado, pois apresenta o desenvolvimento de serviços a partir de um modelo abstrato (diagrama de classes UML customizado) e atende assim ao menos a um dos critérios de seleção.

III Resumo do estudo primário *Top-Down Approach for Web Services Development*

[Bellini, Prado e Zaina \(2010\)](#) propuseram um trabalho que explora o desenvolvimento de software baseado em DSM (do inglês *Domain-Specific Modeling*) para elaboração de um metamodelo WSDL 2.0. Esse modelo foi proposto para ser utilizado a partir ferramentas *Computer-Aided Software Engineering* (CASE), para poder auxiliar na reutilização de metamodelos e na construção de novos modelos abstratos. A ferramenta CASE utilizada foi o *Multi Views CASE* (MVCASE).

Como entrada para o processo de geração de serviços, era fornecido um diagrama de classes, onde os engenheiros de software identificavam *Web services* pela utilização de estereótipos, por exemplo «**Service**». Baseados nesses protótipos o metamodelo WSDL 2.0

proposto era reutilizado na geração de um novo metamodelo mais específico, pois agora este se adequa ao diagrama de classes de entrada. Portanto, esse novo metamodelo WSDL 2.0 gerado era então a saída do MVCASE, junto dos esqueletos dos serviços (BELLINI; PRADO; ZAINA, 2010).

Figura 17 – Parte do Metamodelo WSDL 2.0 proposto.

Fonte: Bellini, Prado e Zaina (2010).

Na Figura 17 destacam-se alguns elementos, detalhados a seguir:

- **Description**: representa o nome do contrato disponível;
- **Interface**: forma e representa a interface do serviço;
- **Operation**: descreve as operações compostas pela interface;
- **Input e Output**: são parâmetros que compõem a aplicação.

IV Resumo do estudo primário *Engineering Adaptable Service Oriented Systems: A Model Driven Approach*

Segundo Kenzi *et al.* (2009), *Web services* e de maneira geral SOC (do inglês *Service Oriented Computing*) é um paradigma computacional para o desenvolvimento de sistemas de

informação de fraco acoplamento, forte interoperabilidade, flexível e ágil. SOC é baseado no elemento conhecido por serviço, que potencialmente interage com diversos tipos de clientes, necessidades e requisitos.

Para atender essa demanda, [Kenzi et al. \(2009\)](#) propuseram uma abordagem de engenharia dirigida a modelos para o desenvolvimento de SOS (do inglês *Service Oriented Systems*). A abordagem foi demonstrada através do CMS (do inglês *Course Management System*). O CMS permite a estudantes matricularem-se em cursos à distância e acessar toda documentação, exercícios, provas, além de ser possível estabelecer comunicação com o professor, por exemplo através de fóruns e email.

A abordagem proposta por [Kenzi et al. \(2009\)](#) estende o modelo UML, ou seja, baseia-se na noção de perfil do modelo UML, permitindo assim a definição de novos elementos e a partir disso definindo um novo metamodelo. Além disso, para superar os desafios de adaptabilidade, especialmente com a emergência do paradigma computacional orientado a contexto o conceito de serviços multivisão foi aplicado.

O conceito de visão (do inglês *View*) é largamente utilizado em campos computacionais como DBMS (do inglês *Data Base Management System*), *workflow*, *Web services* e etc. No caso do trabalho proposto por [Kenzi et al. \(2009\)](#), o conceito de visão significa a separação funcional de preocupações, destacando o que é especificamente esperado de um ator.

Baseado no conceito de visão [Kenzi et al. \(2009\)](#), propuseram o conceito de serviços multivisão, com a modelagem em um primeiro momento realizada através de um diagrama de classes, que destaca as necessidades e requisitos de usuário e os requisitos desde o princípio do ciclo de desenvolvimento orientado a serviços.

Para permitir o desenvolvimento deste novo conceito e também permitir a adaptabilidade, [Kenzi et al. \(2009\)](#) desenvolveram duas ferramentas VSoaML e VSoaMLTool, permitindo assim, a geração de código.

VSoaML⁵ permite a modelagem e a especificação de sistemas orientados a serviço adaptáveis. Eles definem uma série de estereótipos, suas associações, restrições e semântica. Em outras palavras, definem uma linguagem específica de domínio fornecendo todas as capacidades necessárias para o desenvolvimento de uma solução ([KENZI et al., 2009](#)).

VSoaMLTool é uma ferramenta associada com o metamodelo VSoaML. Essa ferramenta permite a edição de modelos, geração automática de código a partir destes modelos, além de implantar os descritores dos serviços multivisão seguindo a arquitetura referência para *Web service* ([KENZI et al., 2009](#)).

Para testar o metamodelo e a ferramenta proposta, [Kenzi et al. \(2009\)](#) propuseram um estudo de caso, considerando uma visão reduzida do sistema CMS. Como conclusão este estudo

⁵ VSoaML: é um metamodelo estendido a partir da UML, permitindo com sua adoção um maior nível de abstração.

de caso foi um sucesso, permitindo a automatização da geração de código e a adaptabilidade necessária aos diferentes tipos de serviços e operações demandados pelos atores.

V Resumo do estudo primário *An EMF-based Toolkit for Creation of Domain-specific Data Services*

A complexidade de aplicações para simulações e análise de dados em ciência e engenharia tem aumentado dramaticamente ao longo do tempo. Essas aplicações são frequentemente desenvolvidas na forma de um fluxo genérico, combinando muitos componentes de software, originados de diversos domínios de aplicação. O desenvolvimento de aplicações compostas requerem um esforço grande de coordenação de especialistas de domínio ou o aprendizado de códigos fonte de programas de domínio específico por muitos pesquisadores. Por outro lado, ambientes existentes para simulação e análise de dados fornecidos pelos centros computacionais são muito complexos para não especialistas. Simulações computacionais e análise de dados são tipicamente planejadas e executadas por cientistas e engenheiros que são normalmente especialistas no campo da computação e da engenharia de software [Bender, Bozic e Kondov \(2014\)](#);

[Bender, Bozic e Kondov \(2014\)](#) propuseram uma solução genérica para modelagem e gerenciamento de dados, em fluxo científico, de uma maneira simples e uniforme. Em outras palavras, os autores propuseram um *framework* para gerenciamento dirigido a modelo de dados em aplicações científicas e de engenharia.

A arquitetura desenvolvida por [Bender, Bozic e Kondov \(2014\)](#) no *framework* proposto pode ser subdividida em três partes: metamodelo, gerador de serviço e serviço de acesso a dados.

- **Metamodelo:** [Bender, Bozic e Kondov \(2014\)](#) optaram pelo desenvolvimento de um novo metamodelo utilizando *Ecore* para criar um metamodelo simples e centrado a dados, em oposição a UML. O objetivo da criação de um novo metamodelo foi que ele pudesse ser compreendido por pessoas não especialistas, ou seja, sem o conhecimento técnico necessário para compreensão;
- **Gerador de serviços:** editor gráfico para criar um modelo de domínio baseado no metamodelo proposto. Adicionalmente, os usuários tiveram que definir uma série de propriedades que são necessárias para garantir acesso ao repositório de dados final. Essas propriedades que tipicamente contém URL de conexão e credenciais de segurança;
- **Serviço de acesso a dados:** para suportar um grande número de aplicações e fazer com que esta parte do *framework* fosse acessível via *Web*, o paradigma arquitetural SOA foi adotado, fazendo uso de *Web services* e padrões abertos de tecnologia. O principal componente do *framework* é o *kernel*.

O *kernel* é constituído de três partes: camada de persistência, camada de representação e camada de recursos, descritas a seguir:

- **Camada de persistência:** lida com o mapeamento do objeto de dados e entidades de armazenamento. [Bender, Bozic e Kondov \(2014\)](#) optaram por adotar um tipo de armazenamento de dados que é flexível, ou seja, o usuário torna-se capaz de selecionar o melhor tipo de camada de persistência (bando de dados) que lhe atenda;
- **Camada de representação:** camada responsável por serializar ou deserializar objeto de dados para formatos de transporte, como por exemplo XML e JSON;
- **Camada de recursos:** camada que define as interfaces de serviços, agindo como uma camada de controle, mapeando os serviços para operações CRUD da camada de persistência.

As principais tecnologias adotadas por [Bender, Bozic e Kondov \(2014\)](#) são apresentadas na [Figura 18](#) e detalhadas a seguir:

- **Modelagem:** pode-se adotar o editor EMF para criação do metamodelo ou o GMF (do inglês *Graphical Modeling Framework*), que permite o desenvolvimento de editores de diagramas baseados em modelos Ecore, sem a necessidade de codificação;
- **Transformação de modelo para texto:** foi escolhido o Acceleo⁶ por seguir o padrão MOF (do inglês *Model to text transformation Language*) da OMG;
- **API do *kernel*:** foram escolhidos as tecnologias JDO, JAXB e JAX-RS. JDO porque suporta o mapeamento de objetos Java para vários tipos de armazenamento (modelos relacionais, baseados em documentos e baseados em *web*) a partir de sua implementação de referência *DataNucleus*. JAXB serve para converter os objetos Java em XML, possibilitando assim troca de mensagens. JAX-RS foi escolhida por ser a especificação padrão para *Web services* do tipo RESTful em Java.

3.2.5 Conclusão sobre a Revisão Bibliográfica

Após a execução da revisão bibliográfica pode-se constatar algumas características relacionadas à pesquisa, que são destacadas a seguir:

- Existência de poucos trabalhos que propõem a geração de serviços a partir de modelos abstratos, mais especificamente UML;

⁶ Acceleo: linguagem que permite a transformação de metamodelos em texto, ou seja, permite uma transformação do tipo model-to-text.

Figura 18 – Principais tecnologias adotadas no trabalho.

Fonte: [Bender, Bozic e Kondov \(2014\)](#).

- Existência de poucos trabalhos que propõem a geração de serviços com operações CRUD, tanto que somente foi encontrado o trabalho proposto por [Bender, Bozic e Kondov \(2014\)](#);
- A maior parte dos trabalhos encontrados propunha a geração de WSDL e XSD ou apenas a geração de interfaces para serviços;
- Poucos estudos primários encontrados propunham a criação de serviços com a implementação CRUD com apenas o diagrama de classes como modelo de entrada, sendo que nenhum estudo, se propôs a implementar a geração de serviços, a partir de um diagrama de classes puro sem modificações.

A partir das características descritas anteriormente e dos objetivos apresentados nesta revisão bibliográfica, notou-se a oportunidade da proposição de um trabalho que tem como objetivo a geração de serviços e suas operações de CRUD, a partir de um diagrama de classes não estendido (ou customizado).

GERADORES DE SERVIÇOS CRUD COM BASE EM MODELOS UML

Neste capítulo é descrita a implementação do trabalho proposto, ou seja, é detalhado o projeto proposto para o desenvolvimento de um gerador e sua implementação, bem como é apresentada uma prova de conceito realizada pelo autor e uma segunda prova de conceito realizada por um aluno de mestrado como meio de validação desse gerador. O capítulo está estruturado da seguinte forma: [seção 4.1](#) - considerações iniciais sobre o capítulo; [seção 4.2](#) - visão geral do projeto implementado; [seção 4.3](#) - descrevem-se as principais características do gerador; [seção 4.4](#) - são descritos os aspectos técnicos do gerador (estrutura e implementação); [seção 4.5](#) - é descrita uma prova de conceito realizada pelo autor desta dissertação como forma de avaliar o gerador, [seção 4.6](#) - é descrita uma segunda prova de conceito para avaliar o gerador, realizada por um aluno de mestrado; e [seção 4.7](#) - são apresentadas as conclusões finais do capítulo.

4.1 Considerações iniciais

Após o levantamento bibliográfico realizado, verificou-se que geradores de serviço com ênfase em operações do tipo CRUD são muito limitados, por exemplo não geram a lógica dos serviços. Alguns dos trabalhos encontrados utilizam modelos UML como entrada para geração de serviço, por exemplo [Kenzi *et al.* \(2009\)](#), [Gronmo *et al.* \(2004\)](#) e [Bellini, Prado e Zaina \(2010\)](#). Nesses trabalhos, os serviços são prioritariamente representados por diagrama de classes, onde são identificados majoritariamente pela utilização de estereótipos. Já o trabalho proposto neste texto difere-se por não utilizar a notação de estereótipos na identificação de serviços e na proposição da criação de operações de CRUD para todas as entidades presentes no modelo. Além disso, este trabalho propõem a criação de entidades do tipo *Bean*, que são entidades que representam as classes no banco de dados.

4.2 Visão Geral do projeto

Nesta seção, o projeto proposto é descrito em um nível mais alto, para que a compreensão sobre o desenvolvimento do projeto ocorra de forma mais natural e que os conceitos empregados sejam melhor compreendidos. Primeiramente, a motivação e o objetivo do projeto são revisitados ([subseção 4.2.1](#) e [subseção 4.2.2](#)). Em seguida, a solução proposta é apresentada em um nível mais alto de abstração ([subseção 4.2.3](#)).

4.2.1 Motivação do projeto

Este projeto tem por base algumas motivações descritas a seguir:

- A AIRES, introduzida na Seção 2.4, é um modelo arquitetural proposto por [Braga et al. \(2016\)](#), cujo objetivo principal é prover o reúso de software em larga escala, utilizando-se de serviços para isso. No projeto da AIRES são necessárias camadas de serviços para permitir a interoperabilidade tanto entre módulos da própria AIRES quanto com aplicações externas que interagem com ela. Prevê-se a inclusão de outros módulos futuramente, portanto é desejável facilitar a implementação de serviços que provêm acesso a esses módulos;
- Em uma revisão sistemática inicial foram investigados estudos relacionados com a abordagem de desenvolvimento de ambientes de reúso baseados em serviço. Essa revisão trouxe como resultado poucos estudos primários ([LEE; KIM; YANG, 2004](#); [ATKINSON; BOSTAN; FINK, 2009](#); [HONGMIN; JIN; JINGZHOU, 2010](#); [CORIA; CASTELLANOS-GARZÓN; CORCHADO, 2013](#));
- Um novo levantamento bibliográfico foi proposto motivado pelos poucos estudos encontrados anteriormente e pelas necessidades da AIRES. Esse levantamento tinha por objetivo encontrar estudos que propusessem a geração automática de serviços a partir de modelos abstratos (por exemplo diagrama de classes). Como resultado desse levantamento, foram encontrados poucos trabalhos que cumpriam esse objetivo ([KENZI et al., 2009](#); [BENDER; BOZIC; KONDOV, 2014](#); [BELLINI; PRADO; ZAINA, 2010](#));
- Dois dos estudos selecionados ([KENZI et al., 2009](#); [BENDER; BOZIC; KONDOV, 2014](#)) propõem a geração de código de serviços a partir de modelos UML customizados. Já o trabalho de [Bellini, Prado e Zaina \(2010\)](#) apresenta o desenvolvimento de um novo modelo de contrato (WSDL), porém gera apenas interfaces de serviços.

Como foram encontrados poucos estudos primários que propusessem a geração automatizada de código, em especial de serviços que implementam operações CRUD, decidiu-se por propor um gerador que resolvesse tal problema sem que o desenvolvedor tenha que conhecer detalhes técnicos do funcionamento dos serviços. Considera-se que diagramas de classes da

UML são largamente conhecidos e provêm um nível de abstração bastante alto. Desta forma, o desenvolvedor pode fornecer o diagrama de classes contendo as entidades para as quais deseja obter serviços CRUD, e tem como saída os serviços prontos para serem invocados.

4.2.2 *Objetivo do projeto*

Este projeto teve por objetivo permitir a geração de código a partir de modelos abstratos, para auxiliar no desenvolvimento e evolução da AIRES, e para propor uma opção de desenvolvimento automatizado de serviços para desenvolvedores ou pesquisadores que não possuam conhecimentos específicos para a criação e publicação de serviços.

4.2.3 *Solução proposta*

Para atender a motivação descrita na [subseção 4.2.1](#), um projeto de desenvolvimento de um gerador foi elaborado. Esse projeto possui algumas características importantes para que a motivação fosse atendida, que são:

- Adoção de um modelo abstrato como entrada para as transformações que o gerador fosse realizar. A opção por um modelo abstrato deu-se pela facilidade de entendimento e construção por pesquisadores, desenvolvedores ou engenheiros de software que não possuem conhecimentos específicos sobre serviços, porém tem a necessidade de implementar operações básicas de serviço (CRUD);
- Automatização da construção do projeto que implementa e executa os serviços. Esse projeto é conhecido como projeto servidor, por ser executado em uma localização remota, executar as operações demandadas e seguir a clássica arquitetura cliente/servidor. Esse aspecto do projeto é importante para manter a abrangência do público alvo descrito no item anterior;
- Automatização da construção de um projeto cliente para tornar possível a execução de testes de maneira simples e rápida. Essa característica tem a intenção de implementar o lado servidor da arquitetura cliente/servidor mencionada no item anterior. Esse aspecto do projeto tem importância similar à descrita no interior, ou seja, permitir a existência de um público alvo que possua conhecimento difuso entre si;
- Adoção de tecnologias estabelecidas no mercado e que tivessem algum suporte da comunidade de desenvolvedores. Essas características são necessárias para garantir a continuidade do projeto visando trabalhos futuros e possíveis reutilizações do projeto visando sua melhoria, facilidade de modificação e de manutenção.

As características destacadas anteriormente visam permitir que o público alvo seja constituído por pessoas heterogêneas quanto ao grau de conhecimento, e com isso que o projeto alcance abrangência de utilização.

Para permitir que as características descritas anteriormente fossem atendidas, algumas definições foram tomadas. Dentre as quais pode-se destacar:

- Adoção do diagrama de classes da UML como modelo de entrada;
- Existência de dois templates para geração de forma autônoma do projeto cliente e do projeto servidor;
- Adoção de alguma ferramenta de gerenciador de projetos que permita a construção dos projetos cliente e servidor de maneira automatizada;
- Escolha por uma linguagem de programação para ser utilizada na implementação do produto final, que fosse consolidada em termos de uso e suporte da comunidade de desenvolvedores;
- Escolha por uma tecnologia de implementação de serviços que fosse consolidada em termos de uso e suporte da comunidade de desenvolvedores;
- Definição por uma tecnologia que permita a persistência de dados em diferentes tipos de banco de dados.

As definições anteriormente descritas foram adotadas no projeto e são descritas com maior rigor técnico nas próximas seções.

4.3 Requisitos do gerador proposto

Nesta seção, apresentam-se os principais requisitos do gerador proposto, justificando as opções de projeto que foram feitas para delimitar o escopo de forma apropriada, para permitir a implementação no contexto desta dissertação de mestrado.

Conforme mencionado na Seção 4.2.1, o gerador tem como um dos objetivos facilitar o desenvolvimento e evolução da AIRES. Nesse sentido, pode-se identificar alguns requisitos advindos desse objetivo, que são:

1. Como a AIRES tem sua arquitetura baseada na estrutura da RAS [OMG \(2005\)](#), para uma maior padronização na estrutura de representação dos *assets* e facilidade de integração com outras ferramentas de reúso, o gerador deve ser capaz de gerar serviços CRUD para modelos semelhantes ao da RAS;

2. Na AIRES, os *Assets* devem ser disponibilizados por meio de serviços, portanto serviços referentes a operações CRUD devem ser gerados para facilitar o acesso a cada entidade da RAS;
3. Módulos que irão compor a AIRES devem se comunicar por meio de serviços, portanto os serviços devem estar disponíveis para acesso por esses módulos;
4. Módulos da AIRES devem se comunicar com aplicações externas por meio de serviços, portanto os serviços também devem estar disponíveis para acesso externo.

Conforme já mencionado, o trabalho proposto adotou serviços referentes a operações do tipo CRUD (*Create*, *Read*, *Update* e *Delete*). Essas operações devem atender a ferramentas que necessitam integrar diferentes módulos e aplicações a partir de modelos abstratos, como a *UML*.

As operações do tipo CRUD, são descritas a seguir:

- *Create*: conjunto de operação que realizam a inserção de dados;
- *Delete*: conjunto de operações que lidam com a remoção de dados;
- *Read*: conjunto de operações que recuperam dados, ou seja, pesquisam dados baseados em algum critério de busca;
- *Update*: conjunto de operações definidas para atualização de dados existentes.

Outros requisitos do gerador proposto podem ser caracterizados como decisões de projeto, para tornar viável sua implementação no tempo disponível neste projeto de mestrado. Essas opções de projeto são descritas a seguir:

- Dados de entrada: como dado de entrada o gerador aceita somente modelos UML, mais especificamente diagrama de classes, para a implementação das operações CRUD;
- Tipos de Modelos UML: o gerador aceita alguns modelos de diagrama de classes UML, feitos no Papyrus¹, Ecore² e XMI (*XML Metadata Interchange*);
- Atributos devem ser nomeados, ou seja, além de possuírem seu tipo especificado no diagrama de classes, os atributos necessitam ser nomeados;
- Cada entidade (classe) deve ter somente um atributo que possua o sufixo "_id". Se os atributos da classe não apresentarem esse sufixo ou o nome da classe o gerador o inclui automaticamente na forma de um atributo do tipo inteiro. Esse sufixo indica que uma

¹ Plugin para IDE Eclipse. Este plugin contém um ambiente de edição para modelos UML baseados em EMF *Eclipse Modeling Framework*, ou seja, esta ferramenta possui editores de diagramas para linguagens de modelagem baseadas em EMF, como por exemplo *UML 2* e *SysML*.

² Plugin para edição de metamodelos para a IDE Eclipse.

determinada classe do diagrama possui um atributo do tipo *Identifier*, ou seja, esse atributo é a chave primária que identifica cada instância de objeto da classe;

- A recuperação da informação no banco de dados ocorre de maneira ansiosa ou "*eager*", ou seja, se existe uma entidade composta por outras, quando ela for acessada as entidades que a compõem serão todas recuperadas de uma vez.
- Banco de dados padrão: o banco de dados adotado como padrão para persistir as classes do modelo no banco foi o **MySQL**. Porém, foi previsto no gerador que o tipo do banco de dados possa ser passado como parâmetro no diagrama de classes como uma **EAnnotation**³. Esse fato é apresentado na Figura 20;
- Padrão de troca de mensagens adotado: *SOAP* (definido no Capítulo 2), foi o padrão adotado no gerador;
- Ferramenta de compilação e construção de projeto adotada: *Maven* (definido no Capítulo 2);
- Regra de negócio: implementada na forma de *DAO*, ou seja, na forma de classes de acesso a dados que trabalham diretamente com o banco de dados.

Na Figura 19 é ilustrado o diagrama de classes usado na prova de conceito do gerador. Esse diagrama respeita as limitações descritas anteriormente.

Figura 19 – Diagrama UML utilizado na prova de conceito.

Fonte: Elaborada pelo autor.

Na Figura 20, pode-se observar em destaque as estruturas *EAnnotation* que contêm as informações sobre o tipo de banco de dados a ser adotado. Nota-se, também que as estruturas

³ *EAnnotation* é um tipo de anotação definido para modelos *Ecore*

Figura 20 – Diagrama *Ecore* baseado no diagrama de classes da prova de conceito

Fonte: Elaborada pelo autor.

EAnnotation devem estar sobre a raiz do modelo, ou seja, abaixo da estrutura *Model*, para que a geração do código ocorra.

A Figura 21 ilustra o metamodelo *Ecore*, nela pode-se observar a presença da estrutura *EAnnotation*.

Figura 21 – Metamodelo *Ecore*

Fonte: Jörges e Steffen (2011).

Quanto ao processo de desenvolvimento do gerador proposto, inicialmente foi feita a implementação manual de serviços *CRUD* e o desenvolvimento de materiais de apoio para outros estudantes que necessitavam iniciar-se no mundo de desenvolvimento de software orientado a serviços. Durante o desenvolvimento dos materiais de apoio e da implementação que serviu de base para esses materiais, notou-se a existência de necessidades comuns como: 1 - operações de *CRUD* para manipulação dos dados; 2 - camada de acesso a dados; e 3 - arquivos de configuração necessários a publicação dos serviços.

Foi então realizada a abstração com base no código desses serviços, observando os trechos de código em comum que poderiam ser gerados através de transformações de modelo para texto. As principais características identificadas dos trechos de código foram as seguintes:

- Nome dos atributos para composição de *getters* e *setters*;
- Trechos de códigos comuns as operações de *CRUD*. Em todas essas operações era necessário a abertura de uma transação e o encerramento dela, além de algumas delas necessitarem de *commits* (por exemplo as operações de criação e atualização de dados);
- Necessidade de criação de uma camada de acesso a dados (*DAOs*) contendo as operações de *CRUD*;
- Necessidade de criação da camada de serviços que por meio dos *DAOs* executariam as operações de *CRUD*.

Esse foi o início do processo de abstração, que posteriormente resultou nos templates a serem utilizados na geração.

4.4 Estrutura e implementação do gerador

Nesta seção são apresentados os principais aspectos técnicos referentes à estrutura do gerador, bem como as ferramentas utilizadas em sua implementação.

Na [Figura 22](#) é representado o funcionamento do gerador proposto. Essa figura apresenta os dois templates que constituem o gerador de maneira destacada: 1 - *Bean_UMLGen* e 2 - *Server_CRUD_UMLGen*. Esses templates são conhecidos como templates principais e podem conter outros templates sendo executados em função de sua execução. Eles são chamados de templates principais pois iniciam e normalmente concentram o início da execução dos demais templates. O template *Bean_UMLGen* gera as entidades que serão persistidas no banco de dados e as classes cliente que irão simular a execução dos serviços. O template *Server_CRUD_UMLGen* é responsável pela geração de todos os arquivos de configuração necessários para publicação do serviço, por exemplo: *pom.xml*, *cxflbeans.xml* e *web.xml*, além da geração dos *DAOs* e das classes que implementam o serviço.

Para implementação dos templates foi utilizado o *Eclipse Mars Modeling*⁴ e *Acceleo*. Para publicação dos serviços foi utilizado o *Maven*, *Hibernate*, *Eclipse Web Tools Platform* (WTP) e o servidor para publicação *Apache Tomcat*. Mais detalhes sobre as ferramentas e versões utilizadas podem ser encontradas no [Apêndice A](#).

As escolhas por essas ferramentas e tecnologias são justificadas a seguir:

- *Acceleo*: foi escolhido por ser de fácil compreensão e por realizar a transformação de modelo para texto de forma simples e eficiente;

⁴ Disponível em: <<http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/mars1>>

Figura 22 – Representação do funcionamento do gerador.

Fonte: Elaborada pelo autor.

- **Java:** foi escolhida por ser uma linguagem portátil, de farta documentação disponível e pela integração muito próxima às demais ferramentas escolhidas, por exemplo *Hibernate* e *Maven*;
- ***Hibernate*:** *framework* para mapeamento objeto relacional em linguagem Java. Este *framework* apresenta a possibilidade de mapear a partir do domínio orientado a objetos até as tradicionais base de dados relacionais. Isto foi de suma importância para o projeto, visto que facilita a persistência de objetos Java, os Java Beans;
- ***Eclipse Mars*:** era a versão mais atual da IDE quando o gerador começou a ser implementado. O Eclipse foi escolhido também por dar um grande suporte a ferramentas de desenvolvimento orientado a modelos, como *Acceleo* e *Papyrus*;
- ***Maven*:** ferramenta utilizada para a construção automatizada do ambiente de desenvolvimento. *Maven* como as demais ferramentas é uma ferramenta de grande utilização por empresas e pela academia, visando principalmente o *build* automático de projetos, onde

essa ferramenta é responsável pelo *download* de todas as dependências do projeto e pela geração do contrato (WSDL) e do *schema* (XSD);

- *Tomcat*: foi escolhido pela fácil integração com o ambiente de desenvolvimento (*Eclipse*) e pela necessidade de um servidor para publicação do serviço.

O template *Bean_UMLGen* recebe como modelo de entrada um diagrama de classes (UML) e produz tanto as entidades (*Bean*) que serão persistidas no banco de dados, bem como as classes cliente que irão executar os serviços. A geração das entidades é feita utilizando-se esse template escrito em *Acceleo* para geração das classes, porém três templates auxiliam este template principal, um para geração das importações necessárias, um para geração dos atributos e outro para geração dos métodos *get* e *set*.

O [Código-fonte 1](#) apresenta o template para geração das classes *bean*:

Código-fonte 1: Template para geração dos *Beans*

```

1 [template public generateClass(m : Model)]
2
3 [for (c : Class | m.eContents(Class)) ]
4   [comment Check if the class has more than one attribute id/]
5   [if ((c.attribute->select(at : Property | at.name.endsWith('_id')
        or at.name.endsWith('_Id') or at.name.endsWith('_id') or at.name.
        endsWith('_ID'))->size()<2))]
6     [file (c.name + '.java', false, 'UTF-8')]
7     [if ( (c.eAnnotations->size()>0) and (c.eAnnotations->select(
        ant : EAnnotation | ant.source.contains('package') and ant.source.
        contains('bean') and (not(ant.source.contains(';')))->size()>0 )
        )]
8 [c.eAnnotations->select(ant : EAnnotation | ant.source.contains('
        package') and ant.source.contains('bean') and (not(ant.source.
        contains(';')))) ).source.toString() + ';'*/]
9     [else]
10 package rosana.labes.icmc.usp.br;
11     [/if]
12
13
14 [generateImports(m, c)/]
15
16 @Entity
17 public class [c.name/] {
18
19     [generateAttributes(m, c)/]
20
21     public [c.name/](){}
22

```

```
23     [generateOperations(c) /]
24
25 }
26
27     [/file]
28 [/if]
29
30
31 [/for]
32
33 [/template]
34
35 [/template]
```

No [Código-fonte 1](#) pode-se observar, na linha 3, um laço no modelo de entrada (diagrama de classes UML), pois é necessário percorrer todas as classes do modelo. Na linha 5 é possível observar-se que é executada uma comparação que determina a geração do arquivo ".java" que implementará o *bean*, essa comparação verifica se dentre os atributos da classe existe no máximo um que contenha o sufixo "_id" (a função desse sufixo foi explicada na [seção 4.3](#)), ou seja, o arquivo será gerado pelo template se e somente se a classe não possuir mais de um atributo com sufixo "_id". Além disso, pode-se observar que três outros templates aparecem dentro da iteração: *generateImports*, *generateAttributes* e *generateOperations*. Cada um desses templates tem a seguinte função:

- *generateImports*: Esse *template* é responsável por gerenciar as dependências da classe a ser gerada;
- *generateAttributes*: *Template* responsável por gerenciar os atributos da classe a ser gerada;
- *generateOperations*: *Template* responsável por gerar os métodos *set* e *get* das classes.

Os templates citados anteriormente são listados nos seguintes códigos-fonte: [Código-fonte 2](#), [Código-fonte 3](#) e [Código-fonte 4](#).

Código-fonte 2: Template para geração dos *Imports*

```
1 [template public generateImports(m: Model, c : Class)]
2   [if (c.attribute->exists(a: Property | a.upper <> 1 and a.upper <>
    0))]
3 import java.util.List;
4 import java.util.ArrayList;
5   [/if]
6 import javax.persistence.Entity;
```

```

7 [if ( (c.attribute->select(at : Property | at.name.endsWith('Id') or
    at.name.endsWith('id') or at.name.endsWith('ID') or at.name.
    endsWith('iD'))->size()<2) and (c.attribute->select(at : Property
    | at.name.endsWith('Id') or at.name.endsWith('id') or at.name.
    endsWith('ID') or at.name.endsWith('iD'))->size()>0))]
8 import javax.persistence.Id;
9 import javax.persistence.GeneratedValue;
10 [/if]
11 [if (c.attribute->exists(s : Property | checkAttribute(m, s)))]
12 import javax.persistence.FetchType;
13 [/if]
14 [for (a : Property | c.attribute)]
15     [if ((a.lower = 1) and (a.upper = 1) )]
16         [if (checkAttribute(m, a))]
17 import javax.persistence.OneToOne;
18         [/if]
19     [elseif ( (((a.lower = 1) or (a.lower = 0)) and (a.upper <> 1)) )]
20         [if (checkAttribute(m, a))]
21 import javax.persistence.OneToMany;
22         [/if]
23     [elseif ( ((a.lower <> 1) and (a.lower <> 0) and ((a.upper = 1) or
        (a.upper = 0))) )]
24         [if (checkAttribute(m, a))]
25 import javax.persistence.ManyToOne;
26         [/if]
27     [elseif ( ((a.lower <> 1) and (a.lower <> 0) and (a.upper <> 1)) )]
28         [if (checkAttribute(m, a))]
29 import javax.persistence.ManyToMany;
30         [/if]
31     [/if]
32 [/for]
33 [/template]

```

Código-fonte 3: Template para geração dos atributos

```

1 [template public generateAttributes(m : Model, c : Class)]
2
3     [if (c.attribute->select(at : Property | at.name.endsWith('_id')
        or at.name.endsWith('_Id') or at.name.endsWith('_iD') or at.name.
        endsWith('_ID'))->size()<1)]
4     @Id
5     @GeneratedValue(strategy=GenerationType.AUTO)
6     private Integer id;
7     [/if]
8     [for (attribute : Property | c.attribute)]
9         [comment Here the multiplicity of the classes are generated /]
10

```

```

11     [if (attribute.name.endsWith('Id') or attribute.name.endsWith('id
    ') or attribute.name.endsWith('ID') or attribute.name.endsWith('iD
    '))]
12     @Id
13     @GeneratedValue(strategy=GenerationType.AUTO)
14     private [attribute.type.name/] [if (checkIfAttributeHasNoName(
    attribute))][attribute.type.name.toLowerCaseFirst()/][else][attribute.
    name.toLowerCaseFirst()/][if];
15     [elseif ( (attribute.lower = 1) and (attribute.upper = 1))]
16     [if (checkAttribute(m, attribute))]
17     @OneToOne(fetch = FetchType.EAGER)
18     [/if]
19     private [attribute.type.name/] [if (checkIfAttributeHasNoName(
    attribute))][attribute.type.name.toLowerCaseFirst()/][else][attribute.
    name.toLowerCaseFirst()/][if];
20     [elseif ( ((attribute.lower = 1) or (attribute.lower = 0)) and ((
    attribute.upper <> 1)))]
21     [if (checkAttribute(m, attribute))]
22     @OneToOne(fetch = FetchType.EAGER)
23     [/if]
24     private List<[attribute.type.name/]> [if (checkIfAttributeHasNoName
    (attribute))][attribute.type.name.toLowerCaseFirst()/][else][attribute
    .name.toLowerCaseFirst()/][if] = new ArrayList<[attribute.type.name
    /]>();
25     [elseif ( (attribute.lower <> 1) and (attribute.lower <> 0) and
    ((attribute.upper = 1) or (attribute.upper = 0)) )]
26     [if (checkAttribute(m, attribute))]
27     @ManyToOne(fetch = FetchType.EAGER)
28     [/if]
29     private [attribute.type.name/] [if (checkIfAttributeHasNoName(
    attribute))][attribute.type.name.toLowerCaseFirst()/][else][attribute.
    name.toLowerCaseFirst()/][if];
30     [elseif ( (attribute.lower <> 1) and (attribute.lower <> 0) and (
    attribute.upper <> 1))]
31     [if (checkAttribute(m, attribute))]
32     @ManyToOne(fetch = FetchType.EAGER)
33     [/if]
34     private List<[attribute.type.name/]> [if (checkIfAttributeHasNoName
    (attribute))][attribute.type.name.toLowerCaseFirst()/][else][attribute
    .name.toLowerCaseFirst()/][if] = new ArrayList<[attribute.type.name
    /]>();
35     [else]
36     private [attribute.type.name/] [if (checkIfAttributeHasNoName(
    attribute))][attribute.type.name.toLowerCaseFirst()/][else][attribute.
    name.toLowerCaseFirst()/][if];
37     [/if]
38     [/for]

```

```
39 [/template]
```

Código-fonte 4: Template para geração das operações

```

1 [template public generateOperations(c : Class)]
2
3   [for (attribute : Property | c.attribute) separator ('\n')]
4
5     public [if (((attribute.lower = 1) or (attribute.lower = 0)) and
6       ((attribute.upper <> 1)))] List<[attribute.type.name/]>[else][
7       attribute.type.name/] [/if] get[if (checkIfAttributeHasNoName(
8       attribute))][attribute.type.name.toUpperFirst()]/[else][attribute.
9       name.toUpperFirst()]/[/if]() {
10
11       return this.[if (checkIfAttributeHasNoName(attribute))][
12       attribute.type.name.toLowerFirst()]/[else][attribute.name.
13       toLowerFirst()]/[/if];
14
15     }
16   [/for]
17
18   [for (attribute : Property | c.eAllContents(Property)) separator
19     ('\n')]
20
21     [if (not(checkIfHasID(attribute)))]
22     public void set[if (checkIfAttributeHasNoName(attribute))][
23     attribute.type.name.toUpperFirst()]/[else][attribute.name.
24     toUpperFirst()]/[/if]([if (((attribute.lower = 1) or (attribute.
25     lower = 0)) and ((attribute.upper <> 1)))] List<[attribute.type.
26     name/]>[else][attribute.type.name/] [/if] [if (
27     checkIfAttributeHasNoName(attribute))][attribute.type.name.
28     toLowerFirst()]/[else][attribute.name.toLowerFirst()]/[/if]){
29
30       this.[if (checkIfAttributeHasNoName(attribute))][attribute.type
31       .name.toLowerFirst()]/[else][attribute.name.toLowerFirst()]/[/if]
32       = [if (checkIfAttributeHasNoName(attribute))][attribute.type.name.
33       toLowerFirst()]/[else][attribute.name.toLowerFirst()]/[/if];
34
35     }
36   [/if]
37 [/for]
38 [/template]
```

No [Código-fonte 2](#) deve-se observar os seguintes pontos: 1 - linha 7 é utilizado uma estrutura de decisão para determinar se a importação de *ID* e *GeneratedValue* deve ser realizada, essa estrutura de decisão verifica se existe no máximo um atributo com o nome que contenha o sufixo "_id"; 2 - linha 11 é utilizado uma estrutura de decisão para determinar se o tipo do atributo é definido dentro do diagrama de classes *UML*, ou seja, se esse atributo for uma classe do diagrama a recuperação dos dados irá usar o *FetchType* do tipo *Eager* e portanto precisa de

importação; finalmente 3 - entre as linhas 14 a 32 é realizada a verificação com relação aos tipo de relacionamentos existentes na classe, para que as anotações que definem a cardinalidade sejam importadas corretamente.

O [Código-fonte 4](#) possui dois laços: o primeiro entre as linhas 3 e 8 para geração dos métodos *get* e o segundo entre as linhas 10 e 17 para geração dos métodos *set*. Dentro dos laços existem estruturas de decisão que verificam se os atributos possuem nome, não possuindo nome o tipo do atributo é transformado no nome do atributo, porém com a primeira letra no formato minúsculo para caracterizar a definição de uma variável.

Dentre os códigos-fonte apresentados anteriormente ([Código-fonte 2](#), [Código-fonte 3](#) e [Código-fonte 4](#)) o que mais necessita ser destacado é o [Código-fonte 3](#), pois nesse template é definido o mapeamento que será utilizado para persistir o objeto no banco de dados, ou seja, nesse template é definido como o *Hibernate* irá persistir as entidades geradas pelo [Código-fonte 1](#). Sobre este template pode-se observar alguns pontos:

- Anotação *@Id*: é definida em dois momentos, primeiramente nas linhas de 3 a 6, é verificado se a classe possui algum atributo nomeado com o sufixo "_id" e não possuindo esse atributo ele é criado pelo template. Esta medida foi necessária para que todas as classes tenham ao menos uma chave primária para recuperação dos dados. Em um segundo momento, a anotação *@Id* é declarada nas linhas 12 a 14. Nessas linhas nota-se a utilização da *Query*⁵. Ela é utilizada para verificar se o atributo contém um nome, se o atributo não possuir um nome seu nome passa a ser o nome do seu tipo com a primeira letra definida em letra minúscula;
- Anotação *@OneToOne*: é definido entre as linhas 15 a 19. É importante destacar duas verificações feitas entre essas linhas, primeiramente os limites superior e inferior do atributo são comparados a 1, ou seja, a cardinalidade do atributo é verificada para determinar se ele deve receber a anotação *@OneToOne* e a segunda verificação é aquela que acontece na linha 16 e é checado se o atributo pertence ao modelo, ou seja, na *Query checkAttribute* é verificado se o atributo (*attribute*) é uma classe definida no modelo "m";
- Anotação *@OneToMany*: é definido entre as linhas 20 e 24. Acontecem as mesmas verificações destacadas no item anterior, porém a cardinalidade muda e agora como é um relacionamento de "um para muitos" é necessário um tipo de dados complexo para definição do atributo (no caso foi adotado um tipo de dados complexo do *Java* a *Collection List*);
- Anotação *@ManyToOne*: é definido entre as linhas 25 e 29. Acontecem as mesmas verificações destacadas no item anterior, porém a cardinalidade é diferente;

⁵ Query: é um tipo de método que é definido dentro do código *Acceleio* para ser utilizado repetidas vezes, como um método definido em linguagens de programação usual

- Anotação *@ManyToMany*: é definido entre as linhas 30 e 35. Acontecem as mesmas verificações destacadas no item anterior, porém a cardinalidade é diferente e o relacionamento é de “muitos para muitos” é necessário um tipo de dados complexo para definição do atributo (no caso foi adotado um tipo de dados complexo do *Java* a *Collection List*).

Como mencionado anteriormente, o template *Server_CRUD_UMLGen*, gera as implementações dos serviços baseados no diagrama de classes, arquivos de configuração necessários à publicação dos serviços em um repositório ou servidor como o *Tomcat* e as classes que manipulam o acesso aos dados, conhecidas como DAO (do inglês *Data Access Object*). O template *Server_CRUD_UMLGen* é mais complexo e maior que o gerador de entidades de dados e portanto será detalhado aos poucos.

O template principal *Server_CRUD_UMLGen* gera os arquivos de maneira individualizada e controla o fluxo de geração dos arquivos. O [Código-fonte 5](#) (template principal) executa outros oito templates, cada template é responsável pela geração de um arquivo ou grupo de arquivos em específico. Serão detalhados apenas cinco templates com mais detalhes, são eles: [Código-fonte 6](#), [Código-fonte 7](#), [Código-fonte 8](#), [Código-fonte 9](#) e [Código-fonte 11](#).

A seguir é ilustrado o template principal *Server_CRUD_UMLGen*, [Código-fonte 5](#):

Código-fonte 5: Template para geração dos arquivos de configuração e dos serviços

```

1 [template public generateElement(m : Model)]
2 [comment @main/]
3
4 [generatePOM(m)/]
5 [generateBeansXml(m)/]
6 [generateWebXml(m)/]
7 [generateHibernateXml(m)/]
8 [generatePersistenceXml(m)/]
9 [generateUtil()/]
10
11
12 [for (c : Class | m.eContents(Class))]
13
14
15     [generateDAO(c)/]
16     [generateService(c)/]
17
18 [/for]
19
20 [/template]
```

Observa-se no [Código-fonte 5](#) que os demais templates são executados a partir dele e que existe um modelo de entrada *m* do tipo *Model*. O modelo de entrada continua sendo o diagrama

de classes UML utilizado no gerador Bean_UMLGen. Esse template gera os seguinte arquivos: **cxfrbeans.xml**, **web.xml**, **pom.xml**, **persistence.xml**, **classes DAO** e *classes que implementam serviços*. Esses arquivos posteriormente poderão ser utilizados no desenvolvimento de um projeto servidor, onde em conjunto com as entidades de dados geradas pelo Bean_UMLGen, poderão ser utilizadas por outras aplicações.

O **Código-fonte 6** apresentado a seguir é responsável pela geração do arquivo de configuração **persistence.xml**, nesse arquivos dados como qual o banco de dados é utilizado na aplicação são definidos. Esse tem relação direta com as anotações do modelo *Ecore* apresentadas na [seção 4.3](#), pois a partir dessas anotações dados do banco de dados podem ser passados como autoreferência através do modelo UML.

Código-fonte 6: Template para geração do arquivo persistence.xml

```

1 [template public generatePersistenceXml (m : Model)]
2 [file ('persistence' + '.xml', false, 'UTF-8')]
3 <persistence xmlns="http://java.sun.com/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://
6     java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
7   version="2.0">
8   <persistence-unit name="entityManager">
9     <provider>org.hibernate.ejb.HibernatePersistence</provider>
10    [comment Here the entities will be added to the persistence.xml/]
11    <!-- Annotated entity classes -->
12    [for (c : Class | m.eContents(Class))]
13    <class>
14      [if ( (c.eAnnotations->size()>0) and (c.eAnnotations->select(ant
15        : EAnnotation | ant.source.contains('package') and ant.source.
16        contains('bean') and (not(ant.source.contains(';'))))>size()>0 )
17      )]
18      [c.eAnnotations->select(ant : EAnnotation | ant.source.contains('
19        package') and ant.source.contains('bean') and (not(ant.source.
20        contains(';')))).source.substring(9).toString() + '.']/[c.name/][
21      else]rosana.labes.icmc.usp.br.[c.name/][[/if]</class>
22    [/for]
23    <properties>
24      [if (m.eAnnotations->size()>0)]
25      [for (at : EAnnotation | m.eAnnotations)]
26      [if (at->select(ant : EAnnotation | ant.source.contains('Driver
27        '))>size() = 1)]
28        <property name="hibernate.connection.driver_class" value="[at->
29          any(ant : EAnnotation | ant.source.contains('Driver')).source.
30          toString()]" />
31      [elseif (at->select(ant : EAnnotation | ant.source.contains('
32        jdbc:'))>size() = 1)]

```

```

22     <property name="hibernate.connection.driver_class" value="[at->
any(ant : EAnnotation | ant.source.contains('jdbc:')).source.
toString()]" />
23     [elseif (at->select(ant : EAnnotation | ant.source.contains('
user'))->size() = 1)]
24     <property name="hibernate.connection.driver_class" value="[at
->any(ant : EAnnotation | ant.source.contains('user')).source.
toString().substring(6)]" />
25     [elseif (at->select(ant : EAnnotation | ant.source.contains('
password'))->size() = 1)]
26     <property name="hibernate.connection.driver_class" value="[at->
any(ant : EAnnotation | ant.source.contains('password')).source.
toString().substring(10)]" />
27     [elseif (at->select(ant : EAnnotation | ant.source.contains('
dialect'))->size() = 1)]
28     <!-- SQL dialect -->
29     <property name="hibernate.dialect" value="[at->any(ant :
EAnnotation | ant.source.contains('dialect')).source.toString()]"
/>
30     [/if]
31     [/for]
32     [else]
33     <property name="hibernate.connection.driver_class" value="com.
mysql.jdbc.Driver" />
34     <property name="hibernate.connection.url" value="jdbc:mysql://
localhost:3306/aires" />
35     <property name="hibernate.connection.username" value="root" />
36     <property name="hibernate.connection.password" value="root" />
37     <!-- SQL dialect -->
38     <property name="hibernate.dialect" value="org.hibernate.
dialect.MySQLDialect" />
39     [/if]
40     [comment Here the hibernate will always update de database and
if it is not creates will create from scratch /]
41     <property name="hibernate.hbm2ddl.auto" value="create" />
42     </properties>
43 </persistence-unit>
44 </persistence>
45 [/file]
46 [/template]

```

Algumas considerações importantes a respeito do [Código-fonte 6](#) são realizadas a seguir:

1. Entre as linhas 11 e 15 existe um laço que percorre todo o modelo "m"(esse modelo se trata do diagrama de classes *uml* de entrada do template) para adicionar as entidades que existem no modelo e serão persistidas pelo *Hibernate*. Observa-se que o nome da classe é

autoreferenciado entre as *tags* "class".

2. Entre as linhas 17 e 31 é verificado se existem anotações que definem o banco de dados a ser adotado. Primeiramente é verificado na linha 17 se existe alguma anotação no modelo "m". Após essa verificação, entre as linhas 18 e 31 é realizado um laço entre as anotações do modelo e anotação por anotação é verificado a existência dos dados necessários a definição do banco de dados adotado. Para que o banco de dados seja adicionado corretamente é obrigatório que o modelo possua cinco anotações: 1 - anotação com sub-texto *Driver*, que define o driver *jdbc* utilizado para conexões com o banco de dados; 2 - anotação com sub-texto *jdbc:*, que define a url de conexão com o banco de dados; 3 - anotação com sub-texto *user*, que define o nome de usuário utilizado na conexão com o banco de dados; 4 - anotação com sub-texto *password*, que define a senha adotada na conexão com o banco de dados; e por fim 5 - anotação com sub-texto *dialect*, que define o dialeto que o *Hibernate* terá de adotar, ou seja, define o tipo de linguagem *SQL* a ser adotada.
3. Na linha 41 é definido que o banco deve ser sempre criado, ou seja, ao executar pela primeira vez alguma das operações de *CRUD* o banco de dados será criado.

O [Código-fonte 7](#) gera o arquivo de configuração para um projeto do tipo *Maven*. Esse arquivo denomina-se *pom.xml* e toma conta de todas as dependências do projeto, ou seja, adquire todas as dependências necessárias a execução do projeto por meio dos repositórios *Maven* e também executa transformações se necessário, por exemplo gerando o contrato do serviço a partir das classes *Java* definidas.

Código-fonte 7: Template para geração do *pom.xml*

```

1 [template public generatePOM (m : Model)]
2 [file ('pom' + '.xml', false, 'UTF-8')]
3 <?xml version="1.0" encoding="UTF-8"?>
4 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
   www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
   apache.org/maven-v4_0_0.xsd">
6   <modelVersion>4.0.0</modelVersion>
7   <!-- Maven Identification for Server Project -->
8   [if ( (m.eAnnotations.source.contains('GroupID:')->size()==1) and (m.
   .eAnnotations.source.contains('ArtifactID:')->size()==1) and (m.
   eAnnotations.source.contains('Packaging:')->size()==1) and (m.
   eAnnotations.source.contains('Version:')->size()==1) )]
9     [for (at : EAnnotation | m.eAnnotations)]
10      [if (at->select(ant : EAnnotation | ant.source.contains('
   GroupID:'))->size() = 1)]
11 <groupId>[at->any(ant : EAnnotation | ant.source.contains('GroupID
   :'))).source.toString().substring(10)/]</groupId>

```

```

12     [elseif (at->select(ant : EAnnotation | ant.source.contains('
    ArtifactID:'))->size() = 1)]
13 <artifactId>[at->any(ant : EAnnotation | ant.source.contains('
    ArtifactID:')).source.toString().substring(13)/]</artifactId>
14 <name>[at->any(ant : EAnnotation | ant.source.contains('ArtifactID
    :')).source.toString().substring(13)/]</name>
15     [elseif (at->select(ant : EAnnotation | ant.source.contains('
    Packaging:'))->size() = 1)]
16 <packaging>[at->any(ant : EAnnotation | ant.source.contains('
    Packaging:')).source.toString().substring(12)/]</packaging>
17     [elseif (at->select(ant : EAnnotation | ant.source.contains('
    Version: '))->size() = 1)]
18 <version>[at->any(ant : EAnnotation | ant.source.contains('Version:
    ')).source.toString().substring(10)/]</version>
19     [/if]
20     [/for]
21 [else]
22 <groupId>AIRES_Server</groupId>
23 <artifactId>AIRES_Server</artifactId>
24 <version>0.0.1-SNAPSHOT</version>
25 <packaging>war</packaging>
26 [/if]
27
28 <build>
29     <sourceDirectory>src</sourceDirectory>
30     <plugins>
31         <plugin>
32             <artifactId>maven-compiler-plugin</artifactId>
33             <version>3.3</version>
34             <configuration>
35                 <source>1.8</source>
36                 <target>1.8</target>
37             </configuration>
38         </plugin>
39         <plugin>
40             <artifactId>maven-war-plugin</artifactId>
41             <version>2.6</version>
42             <configuration>
43                 <warSourceDirectory>WebContent</warSourceDirectory>
44                 <failOnMissingWebXml>>false</failOnMissingWebXml>
45             </configuration>
46         </plugin>
47
48         <!-- plugin 3- apache cxf codegen wsdl2java goal -->
49         <plugin>
50             <groupId>org.apache.cxf</groupId>
51             <artifactId>cxf-java2ws-plugin</artifactId>

```

```

52     <version>${cxf.version}</version>
53     <dependencies>
54         <dependency>
55             <groupId>org.apache.cxf</groupId>
56             <artifactId>cxf-rt-frontend-jaxws</artifactId>
57             <version>${cxf.version}</version>
58         </dependency>
59         <dependency>
60             <groupId>org.apache.cxf</groupId>
61             <artifactId>cxf-rt-frontend-simple</artifactId>
62             <version>${cxf.version}</version>
63         </dependency>
64     </dependencies>
65     <executions>
66         [comment for loop to iterate through services /]
67         [for (c : Class | m.eContents(Class))]
68         <execution>
69             <id>[c.name/] -exec</id>
70             <phase>process-classes</phase>
71             <configuration>
72                 <className>rosana.labes.icmc.usp.br.[c.name + '
ServiceImpl']/</className>
73                 <serviceName>[c.name + 'Service']/</serviceName>
74             </configuration>
75             <address>http://localhost:8095/[c.name + '
Service']/</address>
76             <outputFile>${basedir}/src/main/resources/wsdl
/[c.name + 'Service']/wsdl</outputFile>
77             <argline>-createxsdimports</argline>
78             <genWsd1>true</genWsd1>
79             <verbose>true</verbose>
80             <skip>false</skip>
81             <genWrapperbean>false</genWrapperbean>
82             <genClient>false</genClient>
83             <genServer>false</genServer>
84         </configuration>
85     </goals>
86     <goal>java2ws</goal>
87 </goals>
88 </execution>
89 [/for]
90 </executions>
91 </plugin>
92 </plugins>
93 </build>
94 <properties>

```

```

95     <cxf.version>3.1.2</cxf.version>
96     <spring.version>4.1.0.RELEASE</spring.version>
97     <hibernate.version>4.0.1.Final</hibernate.version>
98     <org.slf4j.version>1.7.5</org.slf4j.version>
99     <mysql.version>5.1.32</mysql.version>
100    <cxf.scope>compile</cxf.scope>
101        <compileSource>1.7</compileSource>
102        <maven.compiler.target>1.7</maven.compiler.target>
103        <maven.compiler.source>1.7</maven.compiler.source>
104        <project.build.sourceEncoding>UTF-8</project.build.
    sourceEncoding>
105 </properties>
106
107 <dependencies>
108     <!-- apache cxf jax-ws 3.1.2 -->
109     <dependency>
110         <groupId>org.apache.cxf</groupId>
111         <artifactId>cxf-rt-frontend-jaxws</artifactId>
112         <version>${cxf.version}</version>
113         <scope>${cxf.scope}</scope>
114     </dependency>
115     <dependency>
116         <groupId>org.apache.cxf</groupId>
117         <artifactId>cxf-rt-transport-http</artifactId>
118         <version>${cxf.version}</version>
119         <scope>${cxf.scope}</scope>
120     </dependency>
121
122     <!-- Spring -->
123     <!-- spring framework 4.1.0 -->
124     <dependency>
125         <groupId>org.springframework</groupId>
126         <artifactId>spring-web</artifactId>
127         <version>${spring.version}</version>
128         <scope>compile</scope>
129     </dependency>
130     ....
131 </project>
132 [/file]
133 [/template]

```

O [Código-fonte 7](#) tem alguns pontos importantes a serem destacados:

- Seção *build*: linhas 28 a 92, nessa parte do arquivo são definidas as características de compilação e de geração dos contratos e *schema* dos serviços. Entre as linhas 31 a 38 é definido qual será o compilador utilizado pelo *Maven*, sua versão e as versões de seu *target*

e *source*. Entre as linhas 49 a 64 é definido o plugin (*wsdl2java*⁶) que realiza a geração do contrato e do *schema* dos serviços e enquanto que entre as linhas 65 a 92 é definido por meio de um laço quais as classes de serviço que terão os arquivos descritos anteriormente gerados. Ainda entre as linhas 65 a 92 é definida a localização dos arquivos gerados pelo plugin *wsdl2java*, a localização pode ser identificada pela tag "*outputFile*";

- Seção *properties*: seção definida entre as linhas 94 a 105, define as versões dos arquivos de dependência que o *Maven* irá adquirir de seu repositório de forma que o projeto servidor possa ser construído e executado com sucesso;
- Seção *dependencies*: seção definida entre as linhas 107 a 129, identifica quais são as dependências do projeto servidor, ou seja, quais arquivos serão adquiridos do repositório do *Maven*.

A seguir é listado mais um template que gera um arquivo de configuração necessário ao projeto servidor, **Código-fonte 8**:

Código-fonte 8: Template para geração do web.xml

```

1 [template public generateWebXml(m : Model)]
2 [file ('web.xml', false, 'UTF-8')]
3 <?xml version="1.0" encoding="UTF-8"?>
4 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns
   = "http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://
   java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
   app_3_0.xsd" version="3.0">
5   <display-name>AIRES_Server</display-name>
6   <servlet>
7     <description>Apache CXF Endpoint</description>
8     <display-name>cxfr</display-name>
9     <servlet-name>cxfr</servlet-name>
10    <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</
      servlet-class>
11    <load-on-startup>1</load-on-startup>
12  </servlet>
13  <servlet-mapping>
14    <servlet-name>cxfr</servlet-name>
15    <url-pattern>/services/*</url-pattern>
16  </servlet-mapping>
17  <session-config>
18    <session-timeout>60</session-timeout>
19  </session-config>
20  <context-param>
21    <param-name>contextConfigLocation</param-name>

```

⁶ *wsdl2java*: é um plugin que é disponibilizado com o *Apache CXF*

```

22     <param-value>WEB-INF/cxf-beans.xml</param-value>
23 </context-param>
24 <listener>
25     <listener-class>org.springframework.web.context.
        ContextLoaderListener</listener-class>
26 </listener>
27 </web-app>
28 [/file]
29 [/template]

```

O **Código-fonte 8** define alguns aspectos importantes do projeto servidor:

- URL padrão: é a URL básica para acesso aos serviços publicados pelo projeto servidor a ser construído. Esse padrão foi definido pela *tag* "url-pattern";
- *Servlet*: Classe responsável por receber as requisições e enviar as respostas à entidade solicitante, ou seja, é uma classes contêiner que define como as requisições e as respostas irão acontecer. Nesse template foi adotado o *Servlet* padrão do *Apache CXF*, *CXFServlet*.

O **Código-fonte 9** define as classes de manipulação de dados (*DAO*). Essas classes executam as regras de negócio dos serviços, ou seja, implementam as operações de *CRUD* junto ao banco de dados. Este é definido pelo arquivo de configuração *persistence.xml*.

Código-fonte 9: Template para geração das classes de manipulação de dados (*DAO*)

```

1 [template public generateDAO (c : Class) ]
2 [file (c.name + 'DAO.java', false, 'UTF-8')]
3 package rosana.labes.icmc.usp.br;
4
5 import rosana.labes.icmc.usp.br.HibernateUtil;
6 import javax.persistence.EntityManager;
7 import javax.persistence.Query;
8 import org.hibernate.HibernateException;
9 import java.util.List;
10 import rosana.labes.icmc.usp.br.[c.name/];
11
12 public class [c.name/]DAO {
13
14     public void create[c.name.toUpperFirst()/] ([c.name/] [c.name.
        toLowerFirst()/]){
15
16         EntityManager em = HibernateUtil.getEntityManagerFactory().
            createEntityManager();
17
18         try {

```



```

19         em.getTransaction().begin();
20         em.persist([c.name.toLowerFirst()/]);
21         em.getTransaction().commit();
22         System.out.println("Records created successfully");
23     } catch (HibernateException e) {
24         e.printStackTrace();
25     }
26
27 }
28
29 public List<[c.name/]> retrieveAll[c.name.toUpperFirst()/]() {
30
31     EntityManager em = HibernateUtil.getEntityManagerFactory().
createEntityManager();
32     List<[c.name.toUpperFirst()/]> [c.name.toLowerFirst() + '1' /]
= null;
33     try {
34         em.getTransaction().begin();
35         Query query = em.createQuery("FROM [c.name.toUpperFirst()/]")
;
36         [c.name.toLowerFirst() + '1' /] = (List<[c.name/]>) query.
getResultList();
37         System.out.println("Records retrieved successfully");
38     } catch (HibernateException e) {
39
40         e.printStackTrace();
41     }
42
43     return [c.name.toLowerFirst() + '1;' /]
44 }
45
46 [if (( (c.attribute->select(at : Property | at.name.endsWith('id
' ) or at.name.endsWith('Id') or at.name.endsWith('iD') or at.name.
endsWith('ID'))->size()<2) and (c.attribute->select(at : Property
| at.name.endsWith('id') or at.name.endsWith('Id') or at.name.
endsWith('iD') or at.name.endsWith('ID'))->size()>0) ))]
47 public [c.name/] retrieve[c.name.toUpperFirst()/](int id) {
48
49     EntityManager em = HibernateUtil.getEntityManagerFactory().
createEntityManager();
50     [c.name/] [c.name.toLowerFirst() + '1' /] = null;
51     try {
52         [c.name.toLowerFirst() + '1' /] = ([c.name/])em.find([c.
name + '.class' /], id);
53         System.out.println("Records retrieved successfully");
54     } catch (HibernateException e) {
55

```

```

56         e.printStackTrace();
57     }
58
59     return [c.name.toLowerFirst() + '1;']/]
60 }
61 [/if]
62
63 [if (( (c.attribute->select(at : Property | at.name.endsWith('id
        ') or at.name.endsWith('Id') or at.name.endsWith('id') or at.name.
        endsWith('ID'))->size()<2) and (c.attribute->select(at : Property
        | at.name.endsWith('id') or at.name.endsWith('Id') or at.name.
        endsWith('id') or at.name.endsWith('ID'))->size()>0) ))]
64     public [c.name/] delete[c.name.toUpperFirst()/]([c.name/] [c.
        name.toLowerFirst()/]) {
65
66         EntityManager em = HibernateUtil.getEntityManagerFactory().
        createEntityManager();
67
68         [c.name/] [c.name.toLowerFirst() + '1' /] = null;
69         try {
70             em.getTransaction().begin();
71             [comment Here we can considere thar exists one attribute with
        name id, because of the entity template pre-condition /]
72             [c.name.toLowerFirst() + '1' /] = ([c.name/])em.find([c
        .name + '.class' /], [c.name.toLowerFirst() + '.get']/][c.attribute
        ->select(att : Property | att.name.contains('id')).name.
        toUpperFirst() + '()' /]);
73             em.remove([c.name.toLowerFirst() + '1' /]);
74             em.getTransaction().commit();
75             System.out.println("[c.name/] records deleted!");
76
77         } catch (HibernateException e) {
78
79             e.printStackTrace();
80
81         }
82
83     return [c.name.toLowerFirst() + '1' /];
84
85 }
86 [/if]
87
88     public void update[c.name.toUpperFirst()/] ([c.name/] [c.name.
        toLowerFirst()/]) {
89
90         EntityManager em = HibernateUtil.getEntityManagerFactory().
        createEntityManager();

```

```
91
92         try {
93             em.getTransaction().begin();
94             em.merge([c.name.toLowerFirst()/]);
95             em.getTransaction().commit();
96             System.out.println("[c.name.toUpperFirst()/] records
updated!");
97         } catch (HibernateException e) {
98
99             e.printStackTrace();
100         }
101     }
102
103 }
104 [/file]
105 [/template]
```

Em relação ao [Código-fonte 9](#) alguns aspectos importantes em termos de implementação a serem considerados:

1. Utilização do pacote *javax.persistence*: definido nas linhas 6 e 7, a utilização deste pacote permite uma maior longevidade das operações *CRUD*, pois o pacote *javax.persistence* apresenta interfaces que são adotadas pela comunidade, por exemplo a interface *EnvityManager* pode ser utilizada sem preocupação com relação ao suporte dado pela comunidade, pois quaisquer modificações em sua implementação serão transparentes para o usuário, que a utiliza, sem precisar se importar com os detalhes;
2. Utilização de uma classe de utilidades: a classe é gerada pelo [Código-fonte 10](#), e ela serve para criação da classe *EntityManager* por meio da interface *EntityManagerFactory*. Com essa classe de utilidades somente uma instância de *EntityManagerFactory* é criada, permitindo assim melhor gerenciamento de recursos. A classe de utilidades gerada pelo [Código-fonte 10](#) foi utilizada no [Código-fonte 9](#) nas seguintes linhas: 16, 31, 49, 66 e 90.

Código-fonte 10: Template para geração da classe de utilidades

```
1 [template public generateUtil (ct : OclAny)]
2 [file ('HibernateUtil' + '.java', false, 'UTF-8')]
3 package rosana.labes.icmc.usp.br;
4
5 import javax.persistence.EntityManagerFactory;
6 import javax.persistence.Persistence;
7 import org.hibernate.SessionFactory;
8
9 public class HibernateUtil {
```

```

10     private static final EntityManagerFactory factory;
11     static {
12         try {
13
14             factory = Persistence.createEntityManagerFactory("
entityManager");
15         } catch (Throwable ex) {
16             System.err.println("EntityManagerFactory creation failed"
+ ex);
17             throw new ExceptionInInitializerError(ex);
18         }
19     }
20
21     public static EntityManagerFactory getEntityManagerFactory() {
22         return factory;
23     }
24 }
25 [/file]
26 [/template]

```

Finalizando essa seção é apresentado o [Código-fonte 11](#):

Código-fonte 11: Template para geração das classes que implementam os serviços

```

1 [template public generateService (c : Class) ? (c.allFeatures()->
    filter(Property)->select(at : Property | at.name.contains('id') or
    at.name.contains('Id') or at.name.contains('iD') or at.name.
    contains('ID'))->size() <=1) ]
2 [file (c.name + 'ServiceImpl.java', false, 'UTF-8')]
3 package rosana.labes.icmc.usp.br;
4
5 import java.util.List;
6
7 import javax.jws.WebService;
8 import javax.jws.soap.SOAPBinding;
9 import javax.jws.soap.SOAPBinding.Style;
10 import javax.jws.soap.SOAPBinding.Use;
11 import rosana.labes.icmc.usp.br.[c.name + 'DAO'];
12
13 @WebService(targetNamespace = "http://rosana.labes.icmc.usp.br/",
    portName = "[c.name/]Port", serviceName = "[c.name/]Service")
14 @SOAPBinding(style = Style.DOCUMENT, use = Use.LITERAL)
15 public class [c.name/]ServiceImpl {
16
17     private [c.name + 'DAO'] [c.name.toLowerFirst() + 'DAO'];
18
19     public [c.name/]ServiceImpl([c.name + 'DAO'] [c.name.

```

```

        toLowerFirst() + 'DAO' /]) {
20         this.[c.name.toLowerFirst() + 'DAO' /]=[c.name.toLowerFirst() +
        'DAO' /];
21     }
22
23     public [c.name/]ServiceImpl(){
24         this.[c.name.toLowerFirst() + 'DAO' /] = new [c.name + 'DAO' /]()
        ;
25     }
26
27     public void create[c.name.toUpperFirst() /] ([c.name/] [c.name.
        toLowerFirst() /]) {
28         this.[c.name.toLowerFirst() + 'DAO' /].create[c.name.
        toUpperFirst() /]([c.name.toLowerFirst() /]);
29     }
30
31     public List<[c.name /]> retrieveAll[c.name.toUpperFirst() /]() {
32         return ['this.'+ c.name.toLowerFirst() + 'DAO' /].retrieveAll[c.
        name.toUpperFirst() /]();
33     }
34
35     [if (( (c.attribute->select(at : Property | at.name.endsWith('id
        ') or at.name.endsWith('Id') or at.name.endsWith('iD') or at.name.
        endsWith('ID'))->size()<2) and (c.attribute->select(at : Property
        | at.name.endsWith('id') or at.name.endsWith('Id') or at.name.
        endsWith('iD') or at.name.endsWith('ID'))->size()>0) ))]
36     public [c.name/] retrieve[c.name.toUpperFirst() /](int id) {
37         return ['this.'+ c.name.toLowerFirst() + 'DAO' /].retrieve[c.
        name.toUpperFirst() /](id);
38     }
39     [/if]
40
41     [if (( (c.attribute->select(at : Property | at.name.endsWith('id
        ') or at.name.endsWith('Id') or at.name.endsWith('iD') or at.name.
        endsWith('ID'))->size()<2) and (c.attribute->select(at : Property
        | at.name.endsWith('id') or at.name.endsWith('Id') or at.name.
        endsWith('iD') or at.name.endsWith('ID'))->size()>0) ))]
42     public [c.name/] delete[c.name.toUpperFirst() /]([c.name/] [c.name
        .toLowerFirst() /]) {
43         return ['this.'+ c.name.toLowerFirst() + 'DAO' /].delete[c.
        name.toUpperFirst() /]([c.name.toLowerFirst() /]);
44     }
45     [/if]
46
47     public void update[c.name.toUpperFirst() /] ([c.name/] [c.name.
        toLowerFirst() /]) {
48         ['this.'+ c.name.toLowerFirst() + 'DAO' /].update[c.name.

```

```
        toUpperFirst() /] ([c.name.toLowerFirst() /]);  
49      }  
50  
51    }  
52  
53 [/file]  
54 [/template]
```

No [Código-fonte 11](#) são definidos o protocolo de troca de mensagens dos serviços (*SOAP*) e quais operações os serviços irão fornecer. O protocolo de troca de mensagens *SOAP* foi definido na linha 14, enquanto que as características do contrato, por exemplo *targetNameSpace*, *portName* e nome do serviço, foram definidos na linha 13. Ambas as definições ocorreram por meio de anotações e autoreflexão⁷. Também, pode-se observar no [Código-fonte 11](#) que os dados trocados entre os serviços utilizaram a lógica de negócio desenvolvida nas classes de acesso a dados para persistirem as requisições.

4.5 Prova de conceito

Nesta seção é descrita a prova de conceito promovida pelo autor do texto, com o objetivo de testar a abordagem proposta e descrita nas seções: [seção 4.3](#) e [seção 4.4](#).

Para a prova de conceito foi considerado um diagrama de classes que contivesse poucas entidades para facilitar a depuração dos testes, além de facilitar a extrapolação dos testes, ou seja, adicionar-se elementos como *EAnnotation* nas classes, criar novos tipos de dados dentro do próprio diagrama e etc. Visando facilitar a implementação e a integração com as ferramentas utilizadas no desenvolvimento do gerador, o diagrama de classes foi gerado no *Papyrus*, que é um *plugin* para o *Eclipse* que permite edição de metamodelos (como diagramas UML) de forma simples e funcional.

O diagrama de classes usados na prova de conceito foi adaptado do modelo conceitual da RAS([OMG, 2005](#)) e é apresentado na [Figura 23](#).

Na [Figura 23](#) pode-se observar a presença de três classes: *Asset*, *Solution* e *Artifact*. Destas três classes apenas *Solution* não possui o atributo do tipo "id", ou seja, um atributo que seja o identificador único para um objeto desta classe. Para que este diagrama de classes fosse passível de testes foram necessárias algumas modificações em sua estrutura e serão enumerados a seguir:

1. Atributos com o nome id tiveram seus tipos trocados de *String* para *int* e tiveram seus nome alterados para "nome_da_classe_id", pois o gerador procura no modelo pelo sufixo "_id" no nome do atributo. O tipo de dado foi alterado para facilitar a propriedade de auto-

⁷ Autoreflexão: propriedade da linguagem *Java* e ocorre também por meio da utilização de anotações

Figura 23 – Modelo UML usado no diagrama de classes UML.

Fonte: Adaptada de [OMG \(2005\)](#).

incremento presente no *Hibernate* para chaves primárias. Essa propriedade pode ser vista nos *beans* gerados, como no [Código-fonte 12](#), pelo uso da anotação `@GeneratedValue`;

2. Foi adicionado um atributo `id` na classe *solution*, visando facilitar a recuperação de objetos desse tipo pelo *Hibernate*. Esse atributo é adicionado pelo próprio gerador *Bean_UMLGen* as classes que não possuem nenhum atributo com o nome contendo o sufixo `"_id"`.

Com as modificações realizadas anteriormente foi possível executar os testes de geração e posterior execução dos serviços. Primeiramente, o template *Bean_UMLGen* foi executado e gerou todas as classes de entidades de dados necessárias a execução do projeto. No [Código-fonte 12](#) pode-se observar como a classe *Asset* foi desenvolvida.

Código-fonte 12: Código fonte da classe *Asset*

```

1 package rosana.labes.icmc.usp.br;
2
3 import javax.persistence.Entity;
4 import javax.persistence.Id;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import javax.persistence.FetchType;
8 import javax.persistence.OneToOne;
9

```

```
10
11 @Entity
12 public class Asset {
13
14
15
16
17     @OneToOne(fetch = FetchType.EAGER)
18     private Solution solution;
19
20     private String name;
21
22     @Id
23     @GeneratedValue(strategy=GenerationType.AUTO)
24     private Integer asset_id;
25
26     private String date;
27
28     private String state;
29
30     private String version;
31
32     private String access_rights;
33
34     private String short_description;
35
36
37     public Asset(){
38
39
40         public Solution getSolution(){
41             return this.solution;
42         }
43
44         public String getName(){
45             return this.name;
46         }
47
48         public Integer getId(){
49             return this.id;
50         }
51
52         public String getDate(){
53             return this.date;
54         }
55
56         public String getState(){
```

```
57         return this.state;
58     }
59
60     public String getVersion(){
61         return this.version;
62     }
63
64     public String getAccess_rights(){
65         return this.access_rights;
66     }
67
68     public String getShort_description(){
69         return this.short_description;
70     }
71
72     public void setSolution(Solution solution){
73         this.solution = solution;
74     }
75
76     public void setName(String name){
77         this.name = name;
78     }
79
80     public void setDate(String date){
81         this.date = date;
82     }
83
84     public void setState(String state){
85         this.state = state;
86     }
87
88     public void setVersion(String version){
89         this.version = version;
90     }
91
92     public void setAccess_rights(String access_rights){
93         this.access_rights = access_rights;
94     }
95
96     public void setShort_description(String short_description){
97         this.short_description = short_description;
98     }
99
100 }
```

No [Código-fonte 12](#) pode-se observar:

1. Linha 11: a declaração da anotação `@Entity`, com essa anotação a classe transforma-se em uma entidade de dados para a JPA (*Java Persistence API*), ou seja, a partir de agora essa classe pode ser persistida no banco de dados no formato de uma tabela, onde seus atributos seriam as colunas. É importante ser lembrado que o *Hibernate* cuida para que as classes sejam persistidas;
2. Linha 17: é identificado a utilização da anotação `@OneToOne`, essa anotação é gerada em virtude do relacionamento existente entre a classe *Asset* e a classe *Solution*, conforme pode-se observar na [Figura 23](#);
3. Linhas 22 e 23: nestas linhas são definidos o atributo que será o identificador unitário da entidade *Asset* e o tipo de estratégia de geração dos valores para esse campo. A anotação `@Id` identifica que esse atributo faz parte ou é a chave primária dessa entidade, enquanto que a anotação `@GeneratedValue` permite a definição do tipo de estratégia de geração de valores para esse campo (identificada pelo atributo *strategy*). A estratégia adotada foi de geração automática, ou seja, o banco de dados incrementará o valor do atributo a cada inserção da base de dados.

Após a geração das entidades de dados foi executado o template `Server_CRUD_UMLGen` para criação dos arquivos de configuração, das classes de manipulação de dados (*DAOs*) e das classes que implementam os serviços. No [Código-fonte 13](#) pode-se observar a definição do banco de dados e algumas de suas características já explicadas na [seção 4.4](#).

Código-fonte 13: persistence.xml

```

1 <persistence xmlns="http://java.sun.com/xml/ns/persistence"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://
   java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
4   version="2.0">
5   <persistence-unit name="entityManager">
6     <provider>org.hibernate.ejb.HibernatePersistence</provider>
7     <!-- Annotated entity classes -->
8     <class>
9   rosana.labes.icmc.usp.br.Artifact</class>
10    <class>
11  rosana.labes.icmc.usp.br.Solution</class>
12    <class>
13  rosana.labes.icmc.usp.br.Asset</class>
14    <!-- Excluding unlisted classes -->
15    <exclude-unlisted-classes>false</exclude-unlisted-classes>
16    <properties>
17      <property name = "javax.persistence.schema-generation.database.
   action" value="none"/>

```

```

18     <property name="hibernate.connection.driver_class" value="com.
mysql.jdbc.Driver" />
19     <property name="hibernate.connection.url" value="jdbc:mysql://
localhost:3306/jwt" />
20     <property name="hibernate.connection.username" value="root" />
21     <property name="hibernate.connection.password" value="root" />
22     <!-- SQL dialect -->
23     <property name="hibernate.dialect" value="org.hibernate.
dialect.MySQLDialect" />
24     <property name="hibernate.hbm2ddl.auto" value="create" />
25 </properties>
26 </persistence-unit>
27 </persistence>

```

Considerações a respeito do [Código-fonte 13](#):

1. Linha 6: é definida a classe que irá persistir os dados ao banco de dados, classe *Hibernate-Persistence*;
2. 8 a 13: são definidas as entidades de dados que podem ser persistidas no banco de dados;
3. Linhas 16 a 25: são definidos detalhes sobre o banco de dados adotado, por exemplo qual a *driver* utilizado para conexão e se o banco será criado ao se realizar a primeira consulta (linha 24).

A seguir são listados, os códigos-fonte [Código-fonte 14](#) e [Código-fonte 15](#), que demonstram as implementações respectivamente da classe de acesso a dados para *Asset* e o serviço que utiliza-se dessa classe de acesso a dados para realizar as operações de *CRUD* no *Asset*.

Código-fonte 14: Código fonte da classe de acesso a dados do *Asset*

```

1 package rosana.labes.icmc.usp.br;
2
3 import rosana.labes.icmc.usp.br.HibernateUtil;
4 import javax.persistence.EntityManager;
5 import javax.persistence.Query;
6 import org.hibernate.HibernateException;
7 import java.util.List;
8 import rosana.labes.icmc.usp.br.Asset;
9
10 public class AssetDAO {
11
12     public void createAsset (Asset asset){
13
14         EntityManager em = HibernateUtil.getEntityManagerFactory().
createEntityManager();

```

```
15
16     try {
17         em.getTransaction().begin();
18         em.persist(asset);
19         em.getTransaction().commit();
20         System.out.println("Records created successfully");
21     } catch (HibernateException e) {
22         e.printStackTrace();
23     }
24
25 }
26
27 public List<Asset> retrieveAllAsset() {
28
29     EntityManager em = HibernateUtil.getEntityManagerFactory().
createEntityManager();
30     List<Asset> asset1 = null;
31     try {
32         em.getTransaction().begin();
33         Query query = em.createQuery("FROM Asset");
34         asset1 = (List<Asset>) query.getResultList();
35         System.out.println("Records retrieved successfully");
36     } catch (HibernateException e) {
37
38         e.printStackTrace();
39     }
40
41     return asset1;
42 }
43
44 public Asset retrieveAsset(int id) {
45
46     EntityManager em = HibernateUtil.getEntityManagerFactory().
createEntityManager();
47     Asset asset1 = null;
48     try {
49         asset1 = (Asset)em.find(Asset.class, id);
50         System.out.println("Records retrieved successfully");
51     } catch (HibernateException e) {
52
53         e.printStackTrace();
54     }
55
56     return asset1;
57 }
58
59 public Asset deleteAsset(Asset asset) {
```

```
60
61     EntityManager em = HibernateUtil.getEntityManagerFactory().
        createEntityManager();
62
63     Asset asset1 = null;
64     try {
65         em.getTransaction().begin();
66         asset1 = (Asset)em.find(Asset.class, asset.getId());
67         em.remove(asset1);
68         em.getTransaction().commit();
69         System.out.println("Asset records deleted!");
70
71     } catch (HibernateException e) {
72
73         e.printStackTrace();
74
75     }
76
77     return asset1;
78
79 }
80
81 public void updateAsset (Asset asset) {
82
83     EntityManager em = HibernateUtil.getEntityManagerFactory().
        createEntityManager();
84
85     try {
86         em.getTransaction().begin();
87         em.persist(asset);
88         em.getTransaction().commit();
89         System.out.println("Asset records updated!");
90     } catch (HibernateException e) {
91
92         e.printStackTrace();
93     }
94 }
95
96 }
```

Código-fonte 15: Código fonte da classe que implementa as operações de CRUD no *Asset*

```
1 package rosana.labes.icmc.usp.br;
2
3 import java.util.List;
4
5 import javax.jws.WebService;
```

```
6 import javax.jws.soap.SOAPBinding;
7 import javax.jws.soap.SOAPBinding.Style;
8 import javax.jws.soap.SOAPBinding.Use;
9 import rosana.labes.icmc.usp.br.Asset;
10 import rosana.labes.icmc.usp.br.AssetDAO;
11
12 @WebService(targetNamespace = "http://rosana.labes.icmc.usp.br/",
13             portName = "AssetPort", serviceName = "AssetService")
14 @SOAPBinding(style = Style.DOCUMENT, use = Use.LITERAL)
15 public class AssetServiceImpl {
16     private AssetDAO assetDAO;
17
18     public AssetServiceImpl(AssetDAO assetDAO) {
19         this.assetDAO = assetDAO;
20     }
21
22     public AssetServiceImpl() {
23         this.assetDAO = new AssetDAO();
24     }
25
26     public void createAsset(Asset asset) {
27         this.assetDAO.createAsset(asset);
28     }
29
30     public List<Asset> retrieveAllAsset() {
31         return this.assetDAO.retrieveAllAsset();
32     }
33
34     public Asset retrieveAsset(int id) {
35         return this.assetDAO.retrieveAsset(id);
36     }
37
38     public Asset deleteAsset(Asset asset) {
39         return this.assetDAO.deleteAsset(asset);
40     }
41
42     public void updateAsset(Asset asset) {
43         this.assetDAO.updateAsset(asset);
44     }
45
46 }
```

Após a geração dos códigos, um projeto denominado `AIRES_Server`, foi construído para que os testes fossem realizados. O passo a passo de como esse projeto foi construído será apresentado no [Apêndice B](#). Os códigos gerados no projeto utilizado para a prova de conceito

podem ser vistos no [Apêndice A](#).

Apesar dos códigos e do passo a passo sobre a construção do projeto servidor AIRES_Server estarem detalhados nos apêndices (Apêndice A e Apêndice B), a seguir é feito um resumo dos principais passos para que o projeto possa ser executado com sucesso na *IDE* de desenvolvimento *Eclipse*, utilizando-se o servidor *Tomcat*.

1. Cria-se um projeto *Web* dinâmico no *Eclipse* como na [Figura 24](#);
2. Converte-se o projeto criado no passo anterior para um projeto do tipo *Maven*, como na [Figura 25](#);
3. Cria-se duas pastas fonte (do inglês, *main source*): 1 - `src/main/java` e 2 - `src/main/resources`. Esse passo é ilustrado pela [Figura 26](#);
4. Cria-se o pacote "rosana.labes.icmc.usp.br" e copie todos os arquivos ".java" gerados para esse pacote;
5. Copia-se o arquivo de configuração "pom.xml" para a pasta raiz do projeto no *Eclipse*;
6. Dentro da pasta `WebContent` cria-se a pasta `wsdl`;
7. Na pasta `wsdl` criada no passo anterior copia-se todos os contratos e ".xsd" criados;
8. Copia-se para pasta `WEB-INF` os seguintes arquivos de configuração : 1 - `cxfr-beans.xml` e 2 - `web.xml`;
9. Finalizando, cria-se a pasta `META-INF` dentro da pasta "`src/main/resources`" e copia-se o arquivo `persistence.xml` para ela.

Ao executar os passos descritos anteriormente o projeto AIRES_Server estará preparado para ser publicado. Para esta prova de conceito, escolheu-se o servidor *Tomcat* para publicar o projeto AIRES_Server.

Após a criação do projeto é necessária a criação de um projeto cliente para poder testar os serviços disponibilizados no servidor *Tomcat*. O projeto cliente pode ser criado como um projeto java comum na *IDE* de desenvolvimento *Eclipse*. Para que o projeto localize corretamente os serviços publicados por meio do projeto AIRES_Server criado anteriormente é necessário que os *Stubs* do cliente sejam criados a partir dos contratos (arquivos com a extensão ".wsdl") e dos *schemas* (arquivos com extensão ".xsd"). Segundo [Pulier e Taylor \(2006\)](#), é a parte do software que fica localizado no lado do cliente (considerando-se uma arquitetura cliente servidor), e

Figura 24 – Projeto Web dinâmico.

Fonte: Elaborada pelo autor.

encapsula todas as funcionalidades técnicas, como por exemplo protocolo de comunicação, aspectos da dissertação e realizar o *marshalling*⁸ e *unmarshalling*⁹ dos dados.

No **Código-fonte 16** pode-se observar como a aplicação cliente foi implementada. Os *stubs* foram gerados a partir dos contratos e *schemas*, utilizando-se um aplicativo que vem junto do Apache CXF, o *wsdl2java*. Ao executar esse aplicativo para todos os contratos de serviço gerados a partir do template *Server_CRUD_UMLGen* os *stubs* do cliente puderam ser criados e foram utilizados nos testes.

Código-fonte 16: Código fonte cliente

```
1 package com.main.test;
2
3 import java.util.List;
4
5 import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;
6
7 import rosana.labes.icmc.usp.br.Artifact;
8 import rosana.labes.icmc.usp.br.Asset;
```

⁸ *Marshalling*: é o processo de transformar a representação em memória de um objeto em um formato de dados, que permita o armazenamento e a transmissão dos dados, por exemplo em *bytes*. Fonte: [https://en.wikipedia.org/wiki/Marshalling_\(computer_science\)](https://en.wikipedia.org/wiki/Marshalling_(computer_science))

⁹ *Unmarshalling*: é conceito oposto ao de *Marshalling*, ou seja, é o processo que permite transformar a representação dos dados em algo que seja armazenado na memória do software

Figura 25 – Convertendo projeto Web dinâmico para projeto Maven.

Fonte: Elaborada pelo autor.

Figura 26 – Estruturação do Projeto Aires_Server.

Fonte: Elaborada pelo autor.

```

9 import rosana.labes.icmc.usp.br.AssetServiceImpl;
10 import rosana.labes.icmc.usp.br.Solution;
11
12 public class TestMain {
13
14     public static void main(String[] args) {
15         JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
16

```

```

17     // Use the URL defined in the soap address portion of the WSDL
18     factory.setAddress("http://localhost:8095/AIRES_Server/services/
AssetPort");
19
20     // Utilize the class which was auto-generated by Apache CXF
wsdl2java
21     factory.setServiceClass(AssetServiceImpl.class);
22
23     Object client = factory.create();
24
25     try {
26
27         // Call the Web Service to perform an operation
28         List<Asset> lasset = null;
29         Asset asset = null;
30         asset = new Asset();
31         asset.setAccessRights("public");
32         //asset.setId(34);
33         asset.setDate("21/12/2014");
34         asset.setName("Modulo de segurança");
35         asset.setShortDescription("short_description");
36         Solution sol = new Solution();
37         sol.setAsset(new Asset());
38         //sol.setAsset(asset);
39         //asset.setSolution(sol);
40         Artifact art = new Artifact();
41         art.setAccessRights("public");
42         art.setDigestName("any");
43         art.setDigestValue("10");
44         art.setName("modulo_de_segurança");
45         art.setReference("none");
46         art.setType("n/a");
47         art.setVersion("0.1");
48         asset.setState("SP");
49         asset.setVersion("0.1");
50         //((AssetServiceImpl)client).createAsset(asset);
51         /*
52         lasset = ((AssetServiceImpl)client).retrieveAllAsset();
53
54         for(int i =0; i<lasset.size(); i++){
55             System.out.println("Asset" + "Asset Date: " + lasset.get(i).
getDate() + " Asset Access Rights: " + lasset.get(i).
getAccessRights() + " Asset Name: " + lasset.get(i).getName());
56         }
57         //System.out.println(response);
58         */
59         Asset as = ((AssetServiceImpl)client).retrieveAsset(1);

```

```
60      System.out.println("Asset" + " Asset Date: " + as.getDate() + "
      Asset Access Rights: " + as.getAccessRights() + " Asset Name: " +
      as.getName());
61  } catch (SecurityException e) {
62
63      e.printStackTrace();
64
65  } catch (IllegalArgumentException e) {
66
67      e.printStackTrace();
68
69  }
70
71  }
72
73 }
```

No [Código-fonte 16](#) pode-se observar o uso dos *stubs* por todo o código. Detalhando melhor os *stubs* clientes observa-se:

- Linha 18: é definido a *url* de acesso ao serviço, no caso serviço para acesso ao *Asset*. Pode-se observar ainda que ao definir a *url* de acesso necessita-se colocar o *portName*, este foi definido na implementação do serviço no [Código-fonte 15](#). A *url* é formada da seguinte maneira: endereço-do-servidor:porta-de-acesso/Nome-do-Projeto-Servidor/padrão-de-url/*PortName*, onde "padrão-de-url" é definido no arquivo de configuração web.xml pela tag "*url-pattern*";
- Linha 21: a classe que implementa o serviço é definida;
- Linha 23: objeto cliente, é criada utilizando-se a classe *factory* do pacote *JaxWsProxyFactoryBean*. Esse objeto criado irá executar as operações definidas pela classe definida no passo anterior;
- Linhas 27 a 49: um objeto do tipo *Asset* é criado, além de objetos do tipo *Solution* e *Artifact*;
- Linha 59: ilustra a execução de uma operação do serviço definido anteriormente, neste caso para recuperar o *Asset* de *id* igual a um.

Sem os *stubs*, a criação de uma classe de teste como a ilustrada pelo [Código-fonte 16](#) não seria possível. As linhas de código detalhadas anteriormente utilizam-se destes *stubs*, criados a partir dos contratos do serviço. Para execução do projeto cliente é necessário adicionar os *stubs* ao projeto. Esses passos estão mais detalhados no [Apêndice A](#).

A prova de conceito mostrou-se um sucesso, pois todos os serviços puderam ser executados corretamente, ou seja, as operações de *CRUD* definidas para acessar as classes *Asset*, *Artifact* e *Solution* puderam ser testadas com êxito.

4.6 Avaliação do gerador

Nesta seção é abordada a avaliação proposta em forma de uma nova prova de conceito, desta vez executada por um aluno de mestrado, integrante do mesmo grupo de pesquisa do autor. O aluno não apresentava conhecimento prévio de SOA ou qualquer tipo de conceito e tecnologia a respeito de serviços, constituindo-se assim de um candidato ideal para a prova de conceito proposta. O objetivo foi atestar se o gerador proposto era passível de ser utilizado por pessoas com pouco conhecimento a respeito das tecnologias relacionadas ao desenvolvimento e implantação de serviços.

Para o desenvolvimento da nova prova de conceito, foi desenvolvido um tutorial com os passos necessários para execução, implementação e implantação de serviços de forma automatizada. Esse tutorial, foi posteriormente revisado, visando sanar as dificuldades apresentadas pelo estudante que executou sua primeira versão, e é apresentado no [Apêndice A](#). Este documento (tutorial) desenvolvido visa facilitar a compreensão e a utilização do gerador, de forma que a pessoa que o utilizar tem de ser capaz de implantar e utilizar os serviços e as operações de *CRUD* geradas. Por falta de tempo o aluno enviou ao autor todos os serviços gerados, para que o mesmo pudesse testar os serviços. Portanto parte da execução do projeto foi executada pelo autor da dissertação. Essa parte é referente a execução das operações que constituem o serviço, mesmo assim o aluno acompanhou o processo de execução.

Antes de partir para o detalhamento das dificuldades encontradas pelo aluno no tutorial e das características positivas é detalhado a seguir de forma resumida como o tutorial opera:

- Primeiramente conforme descrito na [seção 4.4](#) foram desenvolvidos dois templates (Bean-UMLGen e Server_CRUD_UMLGen) e para que eles possam funcionar é necessário a criação de dois projetos Acceleo no *Eclipse*;
- Após a geração do código, por meio dos projetos descritos no item anterior, é necessário que seja criado um projeto do tipo *Web Dinâmico* no *Eclipse*. Depois de sua criação o projeto é convertido para um projeto do tipo *Maven*. Finalmente, após a estruturação do projeto convertido com os arquivos gerados, pelos projetos do item anterior, ele pode ser publicado em um servidor de aplicação (como *Tomcat*) para testes. Esse projeto é denominado de projeto servidor.

A seguir listam-se as principais dificuldades encontradas pelo aluno de mestrado durante a execução do tutorial:

1. Segundo o aluno que executou a prova de conceito, o tutorial apresenta ausência de algumas pequenas informações, porém importantes para o prosseguimento da implantação dos templates e publicação dos serviços. Essas informações são em grande parte relacionadas a configurações do ambiente de desenvolvimento;
2. Dificuldades em utilizar modelos UML não gerados no formato XMI. Esse formato bem como o formato ".uml" derivado da criação de um diagrama de classes pelo *Papyrus*, foram utilizados com sucesso na prova de conceito. Porém, a utilização de modelos UML que não utilizem esses formatos, ou seja, que sejam convertidos para esse formato mostrou-se problemático, por exemplo no caso da utilização da ferramenta *Astah*, o diagrama de classes pode ser convertido para XMI, porém apresentou problemas ao utilizar-se junto dos templates;
3. Alguns dos passos do tutorial em que o aluno apresentou maiores dificuldades, foram os passos relativos a criação dos dois projetos Acceleo para geração do código a partir do diagrama de classes. A dificuldade abordada pelo aluno relativa aos projetos Acceleo, foi o fato de ter de criar um projeto do zero e ter de sobrescrever o conteúdo do gerador (arquivo ".mtl") para gerar os códigos necessários.

Além dessas dificuldades, foi requisitado ao aluno que utilizou o tutorial quais os pontos positivos encontrados na participação dessa prova de conceito. Os pontos positivos estão listados a seguir:

1. A execução da prova de conceito mostrou-se interessante e aplicável a qualquer desenvolvedor que necessite da implementação de operações do tipo CRUD para no desenvolvimento de serviços;
2. A utilização do diagrama de classes como modelo de entrada para o a geração de código mostrou-se promissora, pois o aluno não tinha conhecimentos de orientação a serviços e conseguiu utilizar o tutorial, com algumas dificuldades conforme anteriormente descrito;
3. A adoção da linguagem *Java* para geração de serviços, mostrou-se adequada, pois existem muitas ferramentas que auxiliam na automatização dos serviços. Ferramentas e tecnologias como: *Apache CXF*, *Hibernate* e *Spring*. O *Apache CXF* por exemplo apresenta o programa *wsdl2java* que permite a criação de *stubs* clientes para promover a criação do projeto cliente que testa os serviços implantados.

Na [Figura 27](#) pode-se observar o diagrama de classes proposto para a prova de conceito. Esse diagrama foi desenvolvido no *plugin Papyrus* do *Eclipse*. Esse *plugin* é de fácil usabilidade e entendimento. Ele possui dois tipos de editores, um gráfico e outro que utiliza uma estrutura hierárquica, similar a uma árvore de diretórios em um sistema operacional. Os editores do *Papyrus* são ilustrados respectivamente pela [Figura 27](#) e pela [Figura 28](#).

Figura 27 – Diagrama de classes desenvolvido na segunda prova de conceito.

Fonte: Elaborada pelo autor.

Durante a segunda prova de conceito foram utilizadas duas formas de testes:

- SoapUI: ferramenta que permite a criação e execução de maneira rápida e fácil de testes funcionais automatizados, de regressão e de conformidade. Essa ferramenta fornece uma cobertura completa para testes e suporta tecnologias e protocolos padrões utilizados na indústria;
- Cliente construído a partir de *stubs* de cliente: projeto criado para executar a chamada dos serviços publicados através do projeto servidor.

Na Figura 29 é ilustrado o processo de criação de uma pessoa através do preenchimento do XML de requisição de SOAP, nesta figura pode-se observar a resposta da requisição, novamente em XML. Na Figura 30 é demonstrada a requisição de recuperação de todas as pessoas cadastradas e sua resposta. Por fim, na Figura 31 são listadas as entradas existentes na tabela pessoa. O conteúdo da Figura 31 foi extraída da ferramenta *MySQL Workbench*, ela permite a consulta ao banco de dados criado pelo projeto servidor. Lembrando que o projeto servidor cria o banco de dados na primeira operação evocada por uma das formas descritas anteriormente.

Figura 28 – Modelo hierárquico baseado no diagrama de classes proposto para a segunda prova de conceito.

Fonte: Elaborada pelo autor.

4.7 Considerações finais

Nesta seção foram apresentadas as considerações finais sobre o gerador de serviços proposto, bem como as considerações a respeito das provas de conceito executadas.

Conforme descrito em detalhes nas seções anteriores, o gerador desenvolvido tem por objetivo facilitar o reúso de software por meio de geração automatizada de serviços, onde as operações são do tipo CRUD e o modelo de entrada é um diagrama de classes UML. Para testar esse gerador foram propostas duas provas de conceito, a primeira executada pelo autor do texto e a segunda por um aluno com conhecimento restrito quando ao mundo de desenvolvimento de software por meio de serviços. Baseado nisso, concluiu-se que o gerador conseguiu implementar de fato o objetivo desejado, porém algumas melhorias ainda necessitam ser realizadas e também ficaram abertas algumas possibilidades para trabalhos futuros que são detalhadas no [Capítulo 5](#).

Figura 29 – XMLs de requisição e resposta da operação createPessoa.

Fonte: Elaborada pelo autor.

Figura 30 – XMLs de requisição e resposta da operação retrieveAllPessoa.

The screenshot shows a web browser window with the address bar displaying `http://localhost:8095/Aires_lohan_Server/services/PessoaPort`. The browser is displaying two XML documents side-by-side, both in 'Raw' view.

Left XML (Request):

```
<?xml version='1.0' encoding='UTF-8'>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <ros:retrieveAllPessoa/>
</soapenv:Envelope>
```

Right XML (Response):

```
<?xml version='1.0' encoding='UTF-8'>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:retrieveAllPessoaResponse xmlns:ns2="http://rosana.labes.icmc.usp.br/">
      <return>
        <bairro>Primavera</bairro>
        <cidade>Barretos</cidade>
        <endereco>Av 45</endereco>
        <nome>Domenico</nome>
      </return>
      <return>
        <bairro>Primavera</bairro>
        <cidade>Barretos</cidade>
        <endereco>Av 43</endereco>
        <nome>Domenico</nome>
      </return>
      <return>
        <bairro>Centro</bairro>
        <cidade>São Carlos</cidade>
        <endereco>Avenida São Carlos</endereco>
        <nome>Rodrigo da Silva</nome>
      </return>
    </ns2:retrieveAllPessoaResponse>
  </soap:Body>
</soap:Envelope>
```

Fonte: Elaborada pelo autor.

Figura 31 – Conteúdo da tabela Pessoa apresentado no ambiente *MySQL Workbench*.

The screenshot shows the MySQL Workbench interface. At the top, a SQL editor contains the query `select * from pessoa;`. Below the editor, the 'Result Grid' tab is active, displaying the results of the query in a table format. The table has five columns: `id`, `bairro`, `cidade`, `endereco`, and `nome`. There are three rows of data. The second row is highlighted in blue.

	id	bairro	cidade	endereco	nome
▶	1	Primavera	Barretos	Av 45	Domenico
	2	Primavera	Barretos	Av 43	Domenico
	3	Centro	São Carlos	Avenida São Carlos	Rodrigo da Silva

Fonte: Elaborada pelo autor.

CONCLUSÕES

5.1 Contribuições

Reutilização de software vem sendo almejada há alguns anos por pesquisadores, cientistas e desenvolvedores, pois visa auxiliá-los com o aumento da qualidade do software, redução do tempo de desenvolvimento e redução dos custos de produção. Algumas técnicas que propõem implementar o conceito de reúso foram utilizadas na proposição deste trabalho, SOA e MDD.

A utilização de SOA e MDD em conjunto mostrou-se muito promissora e de fácil integração. Enquanto SOA promove a interoperabilidade, flexibilidade e reutilização de serviços para desenvolvimento de software, MDD promove a automatização e a abstração de conceitos necessários ao processo de desenvolvimento.

Após estudos e pesquisas em áreas relacionadas como objetivo desta dissertação, foi identificada a relevância de implementar geradores de serviços com operações do tipo CRUD, tendo como modelo de entrada um diagrama de classes UML. As seguintes contribuições foram mapeadas:

- Aumento do nível de abstração para o desenvolvimento de serviços com operações do tipo CRUD, pois os geradores propostos partem de um diagrama de classes, que é um modelo abstrato;
- Automatização do processo de geração do código, responsável por implementar os serviços;
- Diminuição do tempo de desenvolvimento de serviços que necessitem de operações do tipo CRUD;
- Disponibilização de um tutorial sobre como implementar serviços (este documento não possui nenhuma relação com os geradores propostos). Este tutorial tem como público alvo,

desenvolvedores que necessitem implementar serviços para uma determinada tarefa em específico, porém não possuem conhecimento prévio sobre o desenvolvimento de serviços. O tutorial foi utilizado com sucesso por um aluno de graduação, com as características anteriormente citadas;

- Disponibilização de um tutorial sobre como implementar serviços com operações de CRUD a partir dos geradores propostos. Esse processo foi detalhado no [Capítulo 4](#);
- Condução e apresentação de uma revisão sistemática sobre ambientes de reúso de software baseados em serviços;
- Condução e apresentação de um levantamento bibliográfico sobre a geração de serviços a partir de modelos abstratos, por exemplo diagramas de classes UML, utilizando-se de MDD para atingir esse objetivo.

5.2 Limitações e Opções de projeto

Nesta seção são apresentadas as principais limitações ou opções de projeto do trabalho proposto. As opções de projeto estão listadas a seguir:

- Necessidade de um diagrama de classes desenvolvido no formato XMI ou utilizando-se o *Papyrus*;
- Diagrama de classes desenvolvido deve apresentar no máximo um atributo, com a *substring* "id" como sufixo em seu nome;
- Os geradores produzem serviços, operações e classes de acesso a dados (DAOs) somente a partir das classes do diagrama de entrada;
- Os serviços são implementados somente na linguagem *Java* e a escolha do banco de dados é limitada pelo tipo de banco suportado pelo *Hibernate*; e
- O método de recuperação de classes que possuam atributos compostos e estejam no topo da hierarquia de composição é do tipo *Eager*¹, ou seja, esse tipo de recuperação de dados carrega todas as instâncias de dados da hierarquia de composição, movido a uma simples consulta à entidade mais alta da hierarquia.

As limitações do trabalho proposto identificadas foram:

- A validação dos geradores foi feita somente por uma prova de conceito. É possível que com estudos mais aprofundados sejam encontrados outros problemas que demandem solução apropriada;

¹ Eager: formato de recuperação de dados suportado pelo *Hibernate*. Além desse formato, tem-se o *Lazy*.

- A utilização dos geradores demanda um certo tempo para instalar todo o ambiente de software necessário, bem como para entender o ciclo de desenvolvimento para execução da geração dos serviços, o que poderia ser resolvido se fosse desenvolvido algum tipo de ferramenta de auxílio ao uso dos geradores;
- Os geradores implementados podem não atender completamente ao modelo de arquitetura proposto pela AIRES, principalmente por não ter sido realizada validação da lógica que gera os serviços necessários aos relacionamentos. Como a AIRES ainda encontra-se em desenvolvimento, essa validação só será possível depois que ela for terminada.

5.3 Trabalhos Futuros

Nesta seção são detalhadas as oportunidades de trabalhos futuros vislumbrados a partir desta dissertação. São elas:

- Possibilidade de ser desenvolvido um gerador para o lado cliente, que é interessante pois possibilitaria uma forma de facilitar os testes dos serviços implementados pelos geradores propostos;
- Possibilidade de expandir os geradores existentes, por exemplo para gerar serviços e entidades de dados também através dos relacionamentos entre entidades do modelo de entrada;
- Possibilidade de aprimorar os geradores existentes para que aceite mais de um atributo como identificador da entidade;
- Possibilitar que os geradores implementados gerem códigos em mais de um tipo de linguagem e de um tipo de banco de dados;
- Desenvolvimento de um *plugin* para *Eclipse* que englobe um editor UML, para desenvolvimento do diagrama de classes, e integre os geradores propostos neste trabalho de forma a automatizar todo o ciclo de desenvolvimento necessário à implementação dos serviços;
- Realização de estudos de caso ou experimentos controlados para avaliar os geradores com modelos de entrada mais complexos e se necessário modificar os templates para prever outras situações que surgirem;
- Realização de um estudo de caso direcionado à arquitetura proposta pela AIRES, após a melhoria proposta no segundo item, para validá-las e auxiliar na implementação dos serviços da AIRES.

REFERÊNCIAS

ANDERSON, W. What COTS and software reuse teach us about SOA. In: IEEE Tech Council Software Engn; Carnegie Mellon Univ, Software Engn Inst; Natl Res Council Canada; Conseil Natl Rech Canada; European Software Inst. **ICCBSS 2007: Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, Proceedings**. [S.l.], 2007. p. 141–149. ISBN 978-0-7695-2785-7. 6th International Conference on Commercial-off-Shelf (COTS)-Based Software Systems, Banff, CANADA, FEB 26-MAR 02, 2007. Citado na página 185.

ATKINSON, C.; BOSTAN, P.; FINK, F. Reuse-oriented deployment of software components: Congregation in service-oriented development. In: . Porto: [s.n.], 2009. p. 65–72. ISBN 9780769537771. Citado 6 vezes nas páginas 29, 55, 58, 59, 80 e 183.

BELLINI, A.; PRADO, A. D.; ZAINA, L. Top-down approach for web services development. In: **Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on**. [S.l.: s.n.], 2010. p. 426–431. Citado 8 vezes nas páginas 28, 29, 33, 36, 73, 74, 79 e 80.

BENDER, A.; BOZIC, S.; KONDOV, I. An emf-based toolkit for creation of domain-specific data services. In: **Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on**. [S.l.: s.n.], 2014. p. 30–40. Citado 6 vezes nas páginas 29, 73, 76, 77, 78 e 80.

BRAGA, R. T. V.; FELONI, D.; PACINI, K.; FILHO, D. S.; GOTTARDI, T. Aires: an architecture to improve software reuse. In: **The 15th International Conference on Software Reuse**. [S.l.: s.n.], 2016. p. 1–16. Citado 6 vezes nas páginas 28, 29, 42, 43, 44 e 80.

CABALLÉ, S.; XHAFA, F. Clpl: Providing software infrastructure for the systematic and effective construction of complex collaborative learning systems. **Journal of Systems and Software**, v. 83, n. 11, p. 2083 – 2097, 2010. ISSN 0164-1212. Interplay between Usability Evaluation and Software Development. Citado na página 184.

CERAMI, E. **Web Services Essentials**. [S.l.]: O'Reilly, 2002. Citado na página 36.

CHANG, H.-C.; KAO, Y.-M.; LO, C.-C. Client-cloud computing: A framework for service integration. In: **Computer, Consumer and Control (IS3C), 2012 International Symposium on**. [S.l.: s.n.], 2012. p. 646–649. Citado 2 vezes nas páginas 55 e 184.

CHEN, Z.; CHEN, Z.-g.; DENG, X.-h.; CHEN, L.-x. An effective assemble-oriented framework for grid Web service. **JOURNAL OF CENTRAL SOUTH UNIVERSITY OF TECHNOLOGY**, 14, n. 4, p. 557–562, AUG 2007. ISSN 1005-9784. Citado na página 184.

CHO, I.-H.; MCGREGOR, J. D. Component specification for enterprise software development on web services environment. In: **Software Engineering Research, Management and Applications, 2005. Third ACIS International Conference on**. [S.l.: s.n.], 2005. p. 438–445. Citado 3 vezes nas páginas 36, 37 e 184.

CHUKMOL, U.; BENHARKAT, A.-N.; AMGHAR, Y. Towards a user-oriented framework for web service discovery, reuse and evolution. In: . Marrakesh: [s.n.], 2010. p. 415–422. ISBN 9780769539591. Citado 2 vezes nas páginas [36](#) e [184](#).

CORIA, J. A. G.; CASTELLANOS-GARZÓN, J. A.; CORCHADO, J. M. Intelligent business processes composition based on multi-agent systems. **Expert Systems with Applications**, 2013. ISSN 09574174. Cited By (since 1996)0; Article in Press. Citado 10 vezes nas páginas [27](#), [29](#), [55](#), [59](#), [60](#), [61](#), [62](#), [63](#), [80](#) e [183](#).

DIESTE, O.; GRIMAN, A.; JURISTO, N. Developing search strategies for detecting relevant experiments. **Empirical Software Engineering**, v. 14, n. 5, p. 513–539, 2009. ISSN 13823256. Citado na página [47](#).

DYBA, T.; DINGSOYR, T.; HANSEN, G. Applying systematic reviews to diverse study types: An experience report. In: **Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on**. [S.l.: s.n.], 2007. p. 225–234. ISSN 1938-6451. Citado na página [47](#).

EARL, T. **Service-Oriented Architecture: Concepts, Technology, and Design**. [S.l.]: Prentice Hall, 2005. Citado 2 vezes nas páginas [34](#) e [36](#).

EZENWOKE, A.; MISRA, S.; ADIGUN, M. An approach for e-commerce on-demand service-oriented product line development. **Acta Polytechnica Hungarica**, v. 10, n. 2, p. 69–87, 2013. ISSN 17858860. Cited By (since 1996)0. Citado na página [183](#).

GRONMO, R.; SKOGAN, D.; SOLHEIM, I.; OLDEVIK, J. Model-driven web services development. In: **e-Technology, e-Commerce and e-Service, 2004. EEE '04. 2004 IEEE International Conference on**. [S.l.: s.n.], 2004. p. 42–45. Citado na página [79](#).

HASSAN, A.; HOLT, R. A reference architecture for web servers. In: **Reverse Engineering, 2000. Proceedings. Seventh Working Conference on**. [S.l.: s.n.], 2000. p. 150–159. ISSN 1095-1350. Citado na página [184](#).

HONGMIN, R.; JIN, L.; JINGZHOU, Z. Software asset repository open framework supporting customizable faceted classification. In: **Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on**. [S.l.: s.n.], 2010. p. 1–4. Citado 7 vezes nas páginas [27](#), [29](#), [55](#), [56](#), [57](#), [80](#) e [183](#).

JÖRGES, S.; STEFFEN, B. Leveraging service-orientation for combining code generation frameworks. In: **Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on**. [S.l.: s.n.], 2011. p. 198–207. Citado na página [85](#).

KART, F.; MIAO, G.; MOSER, L.; MELLIAR-SMITH, P. A distributed e-healthcare system based on the service oriented architecture. In: **Services Computing, 2007. SCC 2007. IEEE International Conference on**. [S.l.: s.n.], 2007. p. 652–659. Citado na página [183](#).

KENZI, A.; ASRI, B. E.; NASSAR, M.; KRIOUILE, A. Engineering adaptable service oriented systems: A model driven approach. In: **Service-Oriented Computing and Applications (SOCA), 2009 IEEE International Conference on**. [S.l.: s.n.], 2009. p. 1–8. Citado 6 vezes nas páginas [29](#), [73](#), [74](#), [75](#), [79](#) e [80](#).

KOUAMOU, G. E. Building a Service-Oriented ERP from an Open Source Software. In: Boness, K and Fernandes, JM and Hall, JG (Ed.). **2009 FOURTH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING ADVANCES (ICSEA 2009)**. [S.l.], 2009. p. 33–38. ISBN 978-1-4244-4779-4. 4th International Conference on Software Engineering Advances, Porto, PORTUGAL, SEP 20-25, 2009. Citado na página 184.

KRAFZIG, D.; BANKE, K.; SLAMA, D. **Enterprise SOA: Service-Oriented Architecture Best Practices**. [S.l.]: Prentice Hall, 2005. Citado 3 vezes nas páginas 33, 34 e 41.

LEE, R.; KIM, H.-K.; YANG, H. S. An architecture model for dynamically converting components into web services. In: **Software Engineering Conference, 2004. 11th Asia-Pacific**. [S.l.: s.n.], 2004. p. 648–654. ISSN 1530-1362. Citado 8 vezes nas páginas 29, 54, 62, 63, 64, 65, 80 e 183.

LI, M.; YU, B.; QI, M.; ANTONOPOULOS, N. Automatically wrapping legacy software into services: A grid case study. **PEER-TO-PEER NETWORKING AND APPLICATIONS**, 1, n. 2, p. 139–147, JUN 2008. ISSN 1936-6442. Citado na página 184.

LIU, Y. The study on software reuse based on service composition and dependency-aware service oriented architecture. **International Journal of Digital Content Technology and its Applications**, v. 6, n. 1, p. 161–168, 2012. ISSN 19759339. Cited By (since 1996)8. Citado 2 vezes nas páginas 55 e 184.

MESERVY, T.; FENSTERMACHER, K. Transforming software development: an mda road map. **Computer**, v. 38, n. 9, p. 52–58, Sept 2005. ISSN 0018-9162. Citado 2 vezes nas páginas 28 e 40.

MULCAHY, J.; HUANG, S.; VEGHTE, A. Leveraging service-oriented architecture to extend a legacy commerce system. In: **Systems Conference, 2010 4th Annual IEEE**. [S.l.: s.n.], 2010. p. 243–248. Citado na página 184.

OMG. **Reusable Asset Specification**. 2005. Disponível em: <<http://www.omg.org/spec/RAS/2.2/>>. Citado 3 vezes nas páginas 82, 108 e 109.

PAGAN, J. E.; CUADRADO, J. S.; MOLINA, J. G. Morsa: A scalable approach for persisting and accessing large models. In: **Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems**. Berlin, Heidelberg: Springer-Verlag, 2011. (MODELS'11), p. 77–92. ISBN 978-3-642-24484-1. Citado na página 39.

PASATCHA, P.; SUNAT, K. The Educational System Support Services Based on the Service Oriented Architecture. In: **2008 INTERNATIONAL SYMPOSIUM ON COMMUNICATIONS AND INFORMATION TECHNOLOGIES**. [S.l.: s.n.], 2008. p. 215–218. ISBN 978-1-4244-2335-4. International Symposium on Communication and Information Technologies, Vientiane, LAOS, OCT 21-23, 2008. Citado na página 183.

PASTOR, O.; MOLINA, J. C. **Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. ISBN 3540718672. Citado na página 39.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMAN, F. Restful web services vs. "big"web services: Making the right architectural decision. In: . [S.l.: s.n.], 2008. p. 805–814. Cited By (since 1996)137. Citado 2 vezes nas páginas 37 e 38.

PULIER, E.; TAYLOR, H. **Understanding Enterprise SOA**. [S.l.]: Manning, 2006. Citado na página 117.

QAFMOLLA, X.; NGUYEN, V. C. Automation of web services development using model driven techniques. In: **Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on**. [S.l.: s.n.], 2010. v. 3, p. 190–194. Citado 2 vezes nas páginas 39 e 40.

RIBEIRO, H.; MEIRA, S. de L.; ALMEIDA, E. de; LUCREDIO, D.; ALVARO, A.; ALVES, V.; GARCIA, V. An assessment on technologies for implementing core assets in service-oriented product lines. In: **Software Components, Architectures and Reuse (SBCARS), 2010 Fourth Brazilian Symposium on**. [S.l.: s.n.], 2010. p. 90–99. Citado 3 vezes nas páginas 27, 32 e 184.

RODRIGUES, J.; LEITE, A.; DAMASCENO, J.; SILVEIRA, P.; GARCIA, V.; MEIRA, S. An approach to developing multi-tenancy saas using metaprogramming. In: . Sao Paulo, SP: [s.n.], 2012. p. 207–210. ISBN 9781450317061. Citado na página 183.

SILLITTI, A.; VERNAZZA, T.; SUCCI, G. Service oriented programming: A new paradigm of software reuse. In: Gacek, C (Ed.). **SOFTWARE REUSE: METHODS, TECHNIQUES, AND TOOLS, PROCEEDINGS**. [S.l.], 2002. (LECTURE NOTES IN COMPUTER SCIENCE, 2319), p. 269–280. ISBN 3-540-43483-6. ISSN 0302-9743. 7th International Conference on Software Reuse (ICSR-7), AUSTIN, TEXAS, APR 15-19, 2002. Citado 2 vezes nas páginas 55 e 184.

SIMOES, J.; MAGEDANZ, T. Contextualized User-Centric Multimedia Delivery System for Next Generation Networks. **TELECOMMUNICATION SYSTEMS**, 48, n. 3-4, p. 301–316, DEC 2011. ISSN 1018-4864. Citado 2 vezes nas páginas 55 e 184.

SNEED, H.; SCHEDL, S.; SNEED, S. Linking legacy services to the business process model. In: **2012 IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)**. Riva del Garda, Trento: [s.n.], 2012. p. 17–26. ISBN 9781467330015. Citado na página 66.

SNEED, H. M.; VERHOEF, C.; SNEED, S. H. Reusing existing object-oriented code as web services in a soa. In: **Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2013 IEEE 7th International Symposium on the**. [S.l.: s.n.], 2013. p. 31–39. ISSN 2326-6910. Citado 7 vezes nas páginas 32, 55, 66, 67, 68, 71 e 183.

YAO, H.; ETZKORN, L. Towards a semantic-based approach for software reusable component classification and retrieval. In: . Huntsville, AL: [s.n.], 2004. p. 110–115. ISBN 1581138709; 9781581138702. Citado 4 vezes nas páginas 27, 32, 55 e 184.

YEOM, K.; PARK, J.; MOON, M. Registry and discovery of services with variability based on 2-level uddi. In: . Busan: [s.n.], 2011. p. 305–310. ISBN 9780769544298. Citado 2 vezes nas páginas 55 e 184.

YUE, D.; CUI, L. C.; JUN, L. W.; XIANG, F. W. Based on soa architecture and component software reuse architecture research. In: **Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on**. [S.l.: s.n.], 2010. p. 517–520. Citado na página 184.

ZHU, H. Building reusable components with service-oriented architectures. In: Zhang, D and Khoshgoftaar, TM and Shyu, ML (Ed.). **Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration**. [S.l.], 2005. p. 96–101. ISBN 0-7803-9093-8. IEEE International Conference on Information Reuse and Integration (IRI-2005), Las Vegas, NV, AUG 15-17, 2005. Citado na página [184](#).

TUTORIAL DO PROJETO AIRES_SERVER

Neste [Apêndice A](#) é detalhado um passo a passo de como uma pessoa interessada em utilizar os geradores propostos no trabalho pode utilizá-los.

Os geradores desenvolvidos durante a implementação do trabalho possuem o objetivo de facilitar o desenvolvimento de serviços aos desenvolvedores. A forma proposta para facilitar o desenvolvimento de serviços, foi a de geração automática de serviços com operações do tipo CRUD a partir diagrama de classes (modelos abstratos). Conforme mencionado no [Capítulo 3](#), foram encontrados poucos estudos que propusessem a geração automática de serviços através de modelos abstratos. Assim notou-se que esse campo de pesquisa estava aberto a experimentações e proposições de novos trabalhos. Este tutorial visa orientar o público interessado em automatizar a implementação de serviços e suas operações, porém não possuem conhecimento para desenvolver essa tarefa sem algum auxílio.

Este [Apêndice A](#) é estruturado da seguinte forma: [seção A.1](#) - identifica as ferramentas necessárias a implementação do tutorial e também demonstra onde elas encontram-se disponíveis; [seção A.2](#) - ilustra como criar os dois projetos necessários para implementar os geradores; [seção A.3](#) - define como o projeto servidor, onde os serviços são implementados, deve ser criado; e [seção A.4](#) - define como o projeto cliente será criado.

A.1 Ferramentas Utilizadas

Antes de iniciar a criação dos projetos necessários implementação e execução dos serviços a partir dos geradores a preparação do ambiente de execução é necessária. Para que isto seja possível algumas ferramentas são necessárias, são elas:

- Tomcat 7¹;

¹ Tomcat 7 - <https://tomcat.apache.org/download-70.cgi>

- Maven²;
- Eclipse Modeling, com os seguintes plugins instalados: M2E³ (Maven to Eclipse), WTP⁴ (Web Tools Platform) e o Apache CXF⁵;
- MYSQL⁶, para que o MYSQL seja instalado é necessário que o Microsoft Visual Studio e o Python 3.4 estejam instalados antes.

Após adquirir todas as ferramentas listadas anteriormente é necessária a preparação do ambiente de desenvolvimento.

1. Primeiramente é necessário instalar o Eclipse Modeling, versão utilizada foi a *Mars*;
2. Após a instalação do Eclipse Modeling é necessário instalar o plugin do Acceleo via *MarketPlace* (Help->Eclipse Marketplace). Na tela exibida pela [Figura 32](#) execute a opção de instalação; Após a finalização da instalação o Eclipse deverá ser reiniciado;
3. O plugin Papyrus⁷ deve ser instalado via *Marketplace* como o plugin do Acceleo;
4. Com o Eclipse reiniciado é o momento de instalar-se o plugin WTP. Para instala-lo siga até o menu Help->Install New Software e clique em Add ([Figura 33](#)), preencha os campos name com WTP e location com a url onde o plugin pode ser encontrado. Novamente ao final da instalação reinicie o Eclipse;
5. Repita o passo anterior para instalar o M2E, selecionando o *Maven Integration for Eclipse*;
6. Instale o Apache CXF no computador. Após a instalação, abra o Eclipse para efetuar a configuração do Apache CXF no Eclipse (lembrando que o plugin WTP deve estar devidamente instalado para poder executar esse passo). Com o Eclipse aberto acesse a opção Window->Preferences->WebServices->CXF 2.x Preferences e adicione o diretório de instalação do Apache CXF ([Figura 34](#));
7. O servidor para publicação dos serviços deve ser também configurado. Para isso, a opção Window->Preferences->Server->Runtime Environments deve ser acessada e o diretório onde o servidor a ser utilizado deve ser selecionado (neste tutorial foi utilizado o Tomcat 7). Ilustrado pela [figura 35](#);
8. Por último o Web Service também necessita saber onde será publicado e para tanto a seguinte configuração é necessária: Acesse Window->Preferences->Web Services->Server and Runtime e associe ao servidor desejado o Apache CXF instalado ([Figura 36](#)).

² Maven - <https://maven.apache.org/download.cgi>

³ M2E - <http://download.eclipse.org/technology/m2e/releases>

⁴ WTP - <http://download.eclipse.org/releases/mars/>

⁵ CXF - <https://cxf.apache.org/download.html>

⁶ MSQ - <https://www.mysql.com/downloads/>

⁷ Papyrus - plugin que permite o desenvolvimento de modelos abstratos, como diagramas UML

Figura 32 – Instalação do Acceleo via Marketplace.

Fonte: Elaborada pelo autor.

A.2 Criação dos projetos para geradores

Nesta seção é detalhada a criação dos projetos Acceleo que gerarão todo código necessário a implementação e execução dos serviços e também como criar um modelo no Papyrus subseção A.2.1.

É necessária a criação de dois projetos Acceleo, um para cada template fornecido (eles possuem a extensão ".mtl"). Os projetos Acceleo podem se chamar respectivamente Beans_Template e Server_Template, onde o primeiro nome sugerido seria relacionado ao projeto que gereria as entidades do banco de dados e o segundo todos os arquivos necessários a criação do

Figura 33 – Instalação do WTP.

Fonte: Elaborada pelo autor.

projeto servidor. A Figura 37 e a Figura 38 mostram os primeiros passos da criação de um projeto Acceleo. Enquanto que a Figura 39 define o passo mais importante da criação do projeto, que é a definição dos metamodelos utilizados, ou seja, metamodelos que seram aceitos pelo template do gerador. Além disso, nesta tela é definido o tipo de argumento de entrada para o template, neste caso *Model*.

Os metamodelos selecionados são:

1. <http://www.eclipse.org/uml2/5.0.0/UML>;
2. <http://www.eclipse.org/papyrus/0.7.0/SysML>;
3. <http://www.eclipse.org/emf/2002/Ecore>;
4. <http://www.eclipse.org/emf/2003/XMLType>.

Com um dos projetos Acceleo criado é necessário alterar o arquivo *generate.mtl*⁸. Esse arquivo deve ser alterado para conter a tag `[comment @main]` (essa tag também pode ser adicionada durante a criação do projeto ao selecionar a opção *Main template* na Figura 39, marque também a opção *Generate File*). Após essa modificação é necessário copiar todo conteúdo existente abaixo da tag *module* (Figura 40) de um dos templates enviados (*generate Bean.mtl* e *generate_server.mtl*). O conteúdo desses templates são detalhados no Apêndice B.

⁸ *generate.mtl*: arquivo gerado durante a criação do projeto Acceleo, ele localiza-se dentro da pasta *src* e dentro do pacote *commom*

Figura 34 – Configuração do Apache CXF no Eclipse.

Fonte: Elaborada pelo autor.

Finalizando as configurações para execução dos projetos templates criados devem ser criadas. Nesta seção isto é demonstrado, da mesma forma para o outro projeto Acceleo uma outra configuração deve ser criada. Para prosseguir com a criação do ambiente de execução vá em Run->Run Configurations->Acceleo Applications (Figura 41). Na Figura 41 os campos devem ser preenchidos como a seguir:

1. Project: o projeto Acceleo desejado deve ser selecionado;
2. Main class: classe java que interpreta o template (".mtl") deve ser selecionada. Essa classe encontra-se no mesmo pacote que o template (arquivo ".mtl"), ou seja, dentro do pacote *commom*;
3. Model: modelo UML de entrada para o gerador deve ser selecionado;
4. Target: é o local onde os arquivos gerados devem ser copiados. No projeto utilizado como exemplo foi criada uma pasta denominada *generated_code* na raiz do projeto (Figura 42);
5. Runner: a opção escolhida deve ser Acceleo Application, porém para escolher essa opção a pasta bin do projeto deve ser adicionada ao classpath do arquivo MANIFEST.MF (arquivo encontrado dentro da pasta MEETA-INF). Para isso abra-se o arquivo e selecione a opção *Runtime* e adicione a pasta bin ao classpath. Os passos são demonstrados pela Figura 43 e pela Figura 44.

Figura 35 – Configuração do Servidor para publicação dos serviços.

Fonte: Elaborada pelo autor.

A.2.1 Criação de um diagrama de classes no Papyrus

Nesta seção é detalhado como criar um diagrama de classes no plugin Papyrus do Eclipse.

Os passos necessários para criação de um projeto Papyrus são:

1. Cria-se um projeto Papyrus (Figura 45);
2. Escolha a linguagem (UML no caso) (Figura 46);
3. Defina um nome para o projeto (Figura 47);
4. Escolha o tipo de diagrama UML que deseja construir (de classes no caso) e marque a opção "A UML model with basic primitive types" (Figura 48);
5. Por fim abra o arquivo ".di" criado, e construa seu diagrama. Esse diagrama pode ser utilizado como modelo de entrada para os projetos Aceleo criados na seção A.2.

A.3 Criação do projeto servidor

Nesta seção são detalhados os passos necessários à configuração do projeto servidor, onde os serviços gerados à partir dos projetos Aceleo criados na seção A.2 serão publicados.

Passos necessários para criação do projeto servidor são detalhados a seguir:

Figura 36 – Configuração da plataforma de execução de serviços.

Fonte: Elaborada pelo autor.

1. Primeiramente deve-se criar um projeto do tipo *Dynamic Web Project* (Figura 49). File->New->Other->Web->Dynamic Web Project, o nome do projeto pode ser Services_Server;
2. Converta o projeto criado no passo anterior para um projeto do tipo Maven, clique com o botão direito sobre o projeto e selecione a opção Configure->Convert to Maven Project (Figura 50);
3. Verifique se no projeto resultante do passo anterior existem as duas *source folders* a seguir: src/main/java e src/main/resources. Se elas não existirem crie-las clicando-se no projeto com o botão esquerdo e seguindo até a opção New->Other->Source Folder(Dentro de Java). Preencha a opção *Folder name* com o caminho completo das pastas (Figura 51);
4. Se a pasta wsdl não existir dentro de WebContent ela deve ser criada (Figura 52);
5. Criar dentro da pasta src/main/java o pacote "rosana.labes.icmc.usp.br" e dentro da pasta src/main/resources a pasta "META-INF";
6. Copiar os arquivos gerados cxf-beans.xml e web.xml para a pasta WebContent/WEB-INF, e se esse diretório não existir crie-o;
7. Copiar o arquivos gerado persistence.xml para a pasta META-INF (criada no passo 5) e também para a raiz da pasta src/main/resources (Figura 53);
8. Copiar todos os arquivos ".java" gerados nos dois projetos Acceleio implementados na seção A.2 para o pacote criado dentro da pasta src/main/java (Figura 54);

Figura 37 – Criação de um novo projeto Acceleo primeira tela.

Fonte: Elaborada pelo autor.

9. Copiar o arquivo pom.xml para a raiz do projeto;
10. Configurar a execução de run do Maven com os argumentos “clean install -X”. Para isso vá em Run Configurations->Maven Build clique com o botão direito sobre Maven Build e adicione uma nova configuração. Para adicionar uma nova configuração preencha os campos *Name* com o nome da configuração (Aires_Server_Run, por exemplo), *base directory* com o diretório do projeto e por fim *Goals* com os argumentos citados anteriormente [Figura 55](#);
11. Execute a run configuration definida no passo anterior. Ao final da execução copie os arquivos wsdl e xsd gerados (eles foram gerados dentro da source folder “src/main/resources” no pacote wsdl) para a pasta wsdl criada no passo 4;
12. Adicione um servidor de publicação como o Tomcat ao Eclipse, New->Other->Server e selecione o servidor desejado. Após criado o servidor Tomcat irá aparecer na *View Servers* do Eclipse.

Para configurar o Tomcat criado no último passo da listagem anterior são necessários mais alguns passos, mostrados a seguir:

- Siga até a *View Servers* e dê um duplo clique em Tomcat. O *overview* do Tomcat será aberto;

Figura 38 – Criação de um novo projeto Acceleo segunda tela.

Fonte: Elaborada pelo autor.

- Mude a porta padrão do protocolo HTTP para a porta 8095;
- Adicione o projeto criado na listagem anterior na aba *Source*. Para abrir a janela que contém a aba *Source* clique em *Open launch configuration* e siga até a aba.

O projeto criado anteriormente (*Services_Server*) pode ser publicado no ambiente do Tomcat simplesmente arrastando-o até a instância do Tomcat existente na *View Servers*. Após a cópia do projeto para a instância do Tomcat, o mesmo pode ser iniciado e assim o serviço estará publicado. Para testar o acesso ao serviço basta ir em um navegador e digitar na barra de endereços a seguinte URL: `http://localhost:8095/NomeDoProjetoServidor/services`. Se os serviços forem listados a publicação dos mesmos terá tido sucesso.

A.4 Criação do projeto cliente

Nesta seção é detalhada a criação de um projeto cliente para que os serviços publicados na [seção A.3](#) sejam testados. Os passos são detalhados a seguir:

Figura 39 – Criação de um novo projeto Acceleo terceira tela.

Fonte: Elaborada pelo autor.

1. Cria-se um projeto java comum, com o nome desejado;
2. Dentro desse projeto na pasta src devem ser criados dois pacotes ("com.main.test" e "rosana.labes.icmc.usp.br");
3. Adiciona-se a variável de ambiente *Path* do Windows a pasta bin do Apache CXF (se usar outro sistema operacional verificar como adicionar a variável correspondente o caminho);
4. Crie uma pasta para criação dos *stubs* do cliente;
5. Prossigua até o diretório onde encontram-se os arquivos wsdl e xsd gerados do projeto Servidor criado anteriormente execute o seguinte comando: **wsdl2java -fe jaxws21 -autoNameResolution -p nome_do_pacote_desejado -d "Diretorio_destino_dos_stubs"**. Onde nome_do_pacote_desejado é o nome do pacote onde as classes Java referentes aos *stubs* cliente serão gerados e Diretorio_destino_dos_stubs é o caminho completo para a pasta criada no item anterior;

Figura 40 – Arquivo mtl após a criação do projeto Acceleo.

```

[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/papyrus/0.7.0/SysML', 'http:

[template public generateElement(anAllocate : Allocate)]
[comment @main /]
[file (Test.java, false, 'UTF-8')]
    Teste
[/file]

[/template]

```

Fonte: Elaborada pelo autor.

Figura 41 – Exemplo de Runtime Configuration para um projeto Acceleo.

Fonte: Elaborada pelo autor.

6. Copie todas as classes geradas no item anterior para o pacote `rosana.labes.icmc.usp.br` do projeto criado;
7. Crie uma classe com uma função `main` no pacote `com.main.test`. Essa classe deve ter o código similar ao apresentado no [Código-fonte 17](#).

No [Código-fonte 17](#) pode-se observar três pontos principais:

Figura 42 – Conteúdo da pasta src do projeto Acceleo.

Fonte: Elaborada pelo autor.

Figura 43 – Pasta META-INF.

Fonte: Elaborada pelo autor.

1. Criação do objeto `JaxWsProxyFactoryBean`. Esse objeto é o responsável por definir o serviço que será testado no momento;
2. O objeto `JaxWsProxyFactoryBean` factory executa os métodos `setAddress` e `setServiceClass`, para respectivamente identificar onde a URL de publicação do serviço e com a classe que implementa o serviço;
3. Por fim um objeto `client` é criado a partir do objeto `factory` na linha 19. Esse objeto permite a execução das operações implementadas pelo serviço a partir de uma conversão do tipo *casting*, por exemplo `((ContaServiceImpl)client).createConta(conta)`, onde o *casting* é `ContaServiceImpl`.

Com o projeto implementado agora basta executá-lo como uma aplicação Java.

Figura 44 – Pasta bin adicionada ao classpath do arquivo META-INF.MF.

Fonte: Elaborada pelo autor.

Código-fonte 17: Código para execução dos serviços publicados

```

1 package com.main.test;
2
3 import java.util.List;
4 import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;
5 import com.aires.iohan.stubs.Conta;
6 import com.aires.iohan.stubs.ContaServiceImpl;
7
8 public class TestMain {
9
10     public static void main(String[] args) {
11         JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
12
13         // Use the URL defined in the soap address portion of the WSDL
14         factory.setAddress("http://localhost:8095/Aires_Iohan_Server/
15                             services/ContaPort");
16
17         // Utilize the class which was auto-generated by Apache CXF
18         wsdl2java
19         factory.setServiceClass(ContaServiceImpl.class);
20
21         Object client = factory.create();
22
23         try {
24
25             // Call the Web Service to perform an operation
26             List<Conta> lconta = null;
27             Conta conta = null;

```

Figura 45 – Criando um projeto Papyrus.

Fonte: Elaborada pelo autor.

```
26     conta = new Conta();
27     conta.setDigito(77);
28     conta.setNumero(1155);
29     conta.setSaldo(1500.88);
30     ((ContaServiceImpl)client).createConta(conta);
31 } catch (SecurityException e) {
32
33     e.printStackTrace();
34
35 } catch (IllegalArgumentException e) {
36
37     e.printStackTrace();
38
39 }
40
41 }
42
43 }
```

Figura 46 – Definição do tipo de Diagrama.

Fonte: Elaborada pelo autor.

Figura 47 – Nomeando o projeto.

Fonte: Elaborada pelo autor.

Figura 48 – Definição do tipo de diagrama UML.

Initialization information
Select root element name and diagram kind

Root model element name :
RootElement

Select a Diagram Kind:

Diagram name	Name	Quantity
☒ Class Diagram		1
☐ Communication Diagram		

You can load a template:

☒ A UML model with basic primitive types (ModelWithBasicTypes)

Choose a profile to apply

Browse workspace

Fonte: Elaborada pelo autor.

Figura 49 – Dynamic Web Project.

New Dynamic Web Project

Dynamic Web Project

✖ Resource already exists with a different case.

Project name: Aires_Server

Project location

☒ Use default location

Location: E:\Workspace\Aires_Serve Browse...

Target runtime

Apache Tomcat v7.0 New Runtime...

Dynamic web module version

3.0

Configuration

Default Configuration for Apache Tomcat v7.0 Modify...

A good starting point for working with Apache Tomcat v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

EAR project name: EAR New Project...

Working sets

☐ Add project to working sets

Working sets: Select...

? < Back Next > Finish Cancel

Fonte: Elaborada pelo autor.

Figura 50 – Convertendo Dymanic Web Project em um Projeto Maven.

Fonte: Elaborada pelo autor.

Figura 51 – Criando uma source folder.

Fonte: Elaborada pelo autor.

Figura 52 – Criando a pasta wsdl.

Fonte: Elaborada pelo autor.

Figura 53 – Copiando o arquivo persistence.xml.

Fonte: Elaborada pelo autor.

Figura 54 – Copiando os arquivos Java.

Fonte: Elaborada pelo autor.

Figura 55 – Maven Run Configurations.

Fonte: Elaborada pelo autor.

LISTA DE CÓDIGOS GERADOS PELOS TEMPLATES BEAN_UMLGEN E SERVER_CRUD_UMLGEN

Neste [Apêndice B](#) são apresentados os códigos desenvolvidos nos geradores *generate_bean.mtl* e *generate_server.mtl*. O código *generate_bean.mtl* é representado por [Código-fonte 18](#) e *generate_server.mtl* é representado por [Código-fonte 19](#). As explicações sobre os *templates* Acceleo utilizados para implementar os dois geradores listados neste apêndice foram descritas no [Capítulo 4](#). Portanto neste [Apêndice B](#) todos os códigos são listados somente.

Código-fonte 18: Código de geração das entidades *beans*.

```
1 [comment encoding = UTF-8 /]
2 [module generate('http://www.eclipse.org/uml2/5.0.0/UML', 'http://www
  .eclipse.org/papyrus/0.7.0/SysML', 'http://www.eclipse.org/papyrus
  /0.7.0/controlmode', 'http://www.eclipse.org/papyrus/0.7.0/SysML/
  Blocks', 'http://www.eclipse.org/emf/2002/Ecore'')]
3
4
5 [template public generateElement(m : Model)]
6 [comment @main/]
7
8 [generateClass(m)/]
9
10 [/template]
11
12 [comment This templates has a verification in the attributes, the
  model must contains only one attribute with id included in the
  name string /]
13 [comment This template only accept classes with annotations, the
  annotations must contain all kind of strings but only one with the
```

```

        package name for bean entities and this package name must not
        contain ;/]
14 [template public generateClass(m : Model)]
15 [comment ? ((c.allFeatures()->filter(Property)->select(at : Property
    | at.name.contains('id'))->size()<=1) and (c.eAnnotations->select(
    ant : EAnnotation | ant.source.contains('package') and ant.source.
    contains('bean') )->size() >= 1) )/]]
16
17 [for (c : Class | m.eContents(Class)) ]
18   [comment Check if the class has more than one attribute id/]
19   [if ((c.attribute->select(at : Property | at.name.endsWith('_id')
    or at.name.endsWith('_Id') or at.name.endsWith('_iD') or at.name.
    endsWith('_ID'))->size()<2))]
20     [file (c.name + '.java', false, 'UTF-8')]
21       [if ( (c.eAnnotations->size())>0) and (c.eAnnotations->select(
    ant : EAnnotation | ant.source.contains('package') and ant.source.
    contains('bean') and (not(ant.source.contains(';')))->size())>0 )
    )]
22 [c.eAnnotations->select(ant : EAnnotation | ant.source.contains('
    package') and ant.source.contains('bean') and (not(ant.source.
    contains(';')) ) .source.toString() + ';' ;/]]
23   [else]
24 package rosana.labes.icmc.usp.br;
25   [/if]
26
27
28 [generateImports(m, c)/]
29
30 @Entity
31 public class [c.name/] {
32
33   [generateAttributes(m, c)/]
34
35   public [c.name/](){}
36
37   [generateOperations(c)/]
38
39 }
40
41   [/file]
42 [/if]
43
44
45 [/for]
46
47 [/template]
48

```

```

49 [template public generateImports(m: Model, c : Class)]
50   [if (c.attribute->exists(a: Property | a.upper <> 1 and a.upper <>
      0))]
51   import java.util.List;
52   import java.util.ArrayList;
53   [/if]
54   import javax.persistence.Entity;
55   [if ( (c.attribute->select(at : Property | at.name.endsWith('_Id') or
      at.name.endsWith('_id') or at.name.endsWith('_ID') or at.name.
      endsWith('_id'))->size()<2) and (c.attribute->select(at : Property
      | at.name.endsWith('Id') or at.name.endsWith('id') or at.name.
      endsWith('ID') or at.name.endsWith('iD'))->size()>=0))]
56   import javax.persistence.Id;
57   import javax.persistence.GeneratedValue;
58   import javax.persistence.GenerationType;
59   [/if]
60   [comment m.allOwnedElements()->filter(Classifier).attribute->exists(s
      : Property | checkAttribute(m, s))/]
61   [if (c.attribute->exists(s : Property | checkAttribute(m, s)))]
62   import javax.persistence.FetchType;
63   [/if]
64   [for (a : Property | c.attribute)]
65     [if ((a.lower = 1) and (a.upper = 1) )]
66       [if (checkAttribute(m, a))]
67       import javax.persistence.OneToOne;
68       [/if]
69     [elseif ( (((a.lower = 1) or (a.lower = 0)) and (a.upper <> 1)) )]
70       [if (checkAttribute(m, a))]
71       import javax.persistence.OneToMany;
72       [/if]
73     [elseif ( ((a.lower <> 1) and (a.lower <> 0) and ((a.upper = 1) or
      (a.upper = 0))) )]
74       [if (checkAttribute(m, a))]
75       import javax.persistence.ManyToOne;
76       [/if]
77     [elseif ( ((a.lower <> 1) and (a.lower <> 0) and (a.upper <> 1)) )]
78       [if (checkAttribute(m, a))]
79       import javax.persistence.ManyToMany;
80       [/if]
81     [/if]
82   [/for]
83 [/template]
84
85
86 [template public generateAttributes(m : Model, c : Class)]
87
88   [if (c.attribute->select(at : Property | at.name.endsWith('_id') or

```

```

        at.name.endsWith('_Id') or at.name.endsWith('_id') or at.name.
        endsWith('_ID'))->size()<1)]
89  @Id
90  @GeneratedValue(strategy=GenerationType.AUTO)
91  private Integer id;
92  [/if]
93  [for (attribute : Property | c.attribute)]
94      [comment Here the multiplicity of the classes are generated /]
95
96      [if (attribute.name.endsWith('_Id') or attribute.name.endsWith('_
          _id') or attribute.name.endsWith('_ID') or attribute.name.endsWith
          ('_id'))]
97  @Id
98  @GeneratedValue(strategy=GenerationType.AUTO)
99  private [attribute.type.name/] [if (checkIfAttributeHasNoName(
          attribute))][attribute.type.name.toLowerCaseFirst()/][else][attribute.
          name.toLowerCaseFirst()/][if];
100      [elseif ( (attribute.lower = 1) and (attribute.upper = 1))]
101          [if (checkAttribute(m, attribute))]
102  @OneToOne(fetch = FetchType.EAGER)
103      [/if]
104  private [attribute.type.name/] [if (checkIfAttributeHasNoName(
          attribute))][attribute.type.name.toLowerCaseFirst()/][else][attribute.
          name.toLowerCaseFirst()/][if];
105      [elseif ( ((attribute.lower = 1) or (attribute.lower = 0)) and ((
          attribute.upper <> 1)))]
106          [if (checkAttribute(m, attribute))]
107  @OneToMany(fetch = FetchType.EAGER)
108      [/if]
109  private List<[attribute.type.name/]> [if (checkIfAttributeHasNoName
          (attribute))][attribute.type.name.toLowerCaseFirst()/][else][attribute
          .name.toLowerCaseFirst()/][if] = new ArrayList<[attribute.type.name
          /]>();
110      [elseif ( (attribute.lower <> 1) and (attribute.lower <> 0) and
          ((attribute.upper = 1) or (attribute.upper = 0)) )]
111          [if (checkAttribute(m, attribute))]
112  @ManyToOne(fetch = FetchType.EAGER)
113      [/if]
114  private [attribute.type.name/] [if (checkIfAttributeHasNoName(
          attribute))][attribute.type.name.toLowerCaseFirst()/][else][attribute.
          name.toLowerCaseFirst()/][if];
115      [elseif ( (attribute.lower <> 1) and (attribute.lower <> 0) and (
          attribute.upper <> 1))]
116          [if (checkAttribute(m, attribute))]
117  @ManyToMany(fetch = FetchType.EAGER)
118      [/if]
119  private List<[attribute.type.name/]> [if (checkIfAttributeHasNoName

```

```

        (attribute))) [attribute.type.name.toLowerFirst() /] [else] [attribute
        .name.toLowerFirst() /] [/if] = new ArrayList<[attribute.type.name
        /]>();
120     [else]
121 private [attribute.type.name/] [if (checkIfAttributeHasNoName(
        attribute))] [attribute.type.name.toLowerFirst() /] [else] [attribute.
        name.toLowerFirst() /] [/if];
122     [/if]
123 [/for]
124 [/template]
125
126 [template public generateOperations(c : Class)]
127
128 [for (attribute : Property | c.attribute) separator ('\n')]
129
130     public [if (((attribute.lower = 1) or (attribute.lower = 0)) and
        ((attribute.upper <> 1)))] List<[attribute.type.name/]>[else] [
        attribute.type.name/] [/if] get [if (checkIfAttributeHasNoName(
        attribute))] [attribute.type.name.toUpperFirst() /] [else] [attribute.
        name.toUpperFirst() /] [/if]() {
131         return this.[if (checkIfAttributeHasNoName(attribute))] [
        attribute.type.name.toLowerFirst() /] [else] [attribute.name.
        toLowerFirst() /] [/if];
132     }
133 [/for]
134
135 [for (attribute : Property | c.eAllContents(Property)) separator
        ('\n')]
136
137     [if (not(checkIfHasID(attribute)))]
138     public void set [if (checkIfAttributeHasNoName(attribute))] [
        attribute.type.name.toUpperFirst() /] [else] [attribute.name.
        toUpperFirst() /] [/if] ([if (((attribute.lower = 1) or (attribute.
        lower = 0)) and ((attribute.upper <> 1)))] List<[attribute.type.
        name/]>[else] [attribute.type.name/] [/if] [if (
        checkIfAttributeHasNoName(attribute))] [attribute.type.name.
        toLowerFirst() /] [else] [attribute.name.toLowerFirst() /] [/if]) {
139         this.[if (checkIfAttributeHasNoName(attribute))] [attribute.type
        .name.toLowerFirst() /] [else] [attribute.name.toLowerFirst() /] [/if]
        = [if (checkIfAttributeHasNoName(attribute))] [attribute.type.name.
        toLowerFirst() /] [else] [attribute.name.toLowerFirst() /] [/if];
140     }
141     [/if]
142 [/for]
143
144 [/template]
145

```

```

146 [comment This section will have the declarations of some queries.
    This queries help on template generation./]
147 [query public checkAttribute(m : Model, at : Property) : Boolean = m.
    allOwnedElements()->filter(Classifier)->exists(class : Classifier
    | (at.type.name=class.name)) /]
148 [query public checkIfHasID(at : Property) : Boolean = at.name.
    endsWith('_id') or at.name.endsWith('_Id') or at.name.endsWith('_
    _id') or at.name.endsWith('_ID') /]
149 [query public checkIfAttributeHasNoName(at : Property) : Boolean = at
    .name.equalsIgnoreCase('') /]

```

Código-fonte 19: Código de geração do servidor.

```

1 [comment encoding = UTF-8 /]
2 [module generate('http://www.eclipse.org/uml2/5.0.0/UML', 'http://www
    .eclipse.org/papyrus/0.7.0/SysML', 'http://expansionmodel.ecore',
    'http://www.eclipse.org/emf/2002/Ecore', 'http://www.eclipse.org/
    papyrus/0.7.0/SysML/Blocks', 'http://www.eclipse.org/ocl/1.1.0/UML
    ')]
3
4
5 [template public generateElement(m : Model)]
6 [comment @main/]
7
8 [generatePOM(m)/]
9 [generateBeansXml(m)/]
10 [generateWebXml(m)/]
11 [generateHibernateXml(m)/]
12 [generatePersistenceXml(m)/]
13 [generateUtil()/]
14
15
16 [for (c : Class | m.eContents(Class))]
17
18
19     [generateDAO(c)/]
20     [generateService(c)/]
21
22 [/for]
23
24 [/template]
25
26 [template public generateBeansXml(m : Model)]
27 [file ('cxf-beans.xml', false, 'UTF-8')]
28
29 <?xml version="1.0" encoding="UTF-8"?>
30 <beans xmlns="http://www.springframework.org/schema/beans"

```

```

31  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jaxws="
    http://cxf.apache.org/jaxws"
32  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.
    xsd">
33  <!-- Defining the beans -->
34  [for (c : Class | m.eContents(Class))]
35  <jaxws:endpoint xmlns:tns="http://rosana.labes.icmc.usp.br/" id="[c
    .name + 'Service']/"
36      implementor="rosana.labes.icmc.usp.br.[c.name + 'ServiceImpl']/"
    wsdlLocation="wsdl/[c.name + 'Service.wsdl']/"
37      endpointName="tns:[c.name + 'Port']/" serviceName="tns:[c.name +
    'Service']/" address="/[c.name + 'Port']/">
38      <jaxws:features>
39          <bean class="org.apache.cxf.feature.LoggingFeature" />
40      </jaxws:features>
41  </jaxws:endpoint>
42  [/for]
43 </beans>
44 [/file]
45 [/template]
46
47 [template public generateWebXml(m : Model)]
48 [file ('web.xml', false, 'UTF-8')]
49 <?xml version="1.0" encoding="UTF-8"?>
50 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns
    ="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://
    java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
    app_3_0.xsd" version="3.0">
51  <display-name>AIRES_Server</display-name>
52  <servlet>
53      <description>Apache CXF Endpoint</description>
54      <display-name>cxfr</display-name>
55      <servlet-name>cxfr</servlet-name>
56      <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</
    servlet-class>
57      <load-on-startup>1</load-on-startup>
58  </servlet>
59  <servlet-mapping>
60      <servlet-name>cxfr</servlet-name>
61      <url-pattern>/services/*</url-pattern>
62  </servlet-mapping>
63  <session-config>
64      <session-timeout>60</session-timeout>
65  </session-config>
66  <context-param>

```

```

67     <param-name>contextConfigLocation</param-name>
68     <param-value>WEB-INF/cxf-beans.xml</param-value>
69 </context-param>
70 <listener>
71     <listener-class>org.springframework.web.context.
        ContextLoaderListener</listener-class>
72 </listener>
73 </web-app>
74 [/file]
75 [/template]
76
77
78 [template public generatePOM (m : Model) {i : Integer = 1;}]
79 [file ('pom' + '.xml', false, 'UTF-8')]
80 <?xml version="1.0" encoding="UTF-8"?>
81 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
        www.w3.org/2001/XMLSchema-instance"
82     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
        apache.org/maven-v4_0_0.xsd">
83     <modelVersion>4.0.0</modelVersion>
84     <!-- Maven Identification for Server Project -->
85     [if ( (m.eAnnotations.source.contains('GroupID:')->size()==1) and (m.
        eAnnotations.source.contains('ArtifactID:')->size()==1) and (m.
        eAnnotations.source.contains('Packaging:')->size()==1) and (m.
        eAnnotations.source.contains('Version:')->size()==1) )]
86     [for (at : EAnnotation | m.eAnnotations)]
87         [if (at->select(ant : EAnnotation | ant.source.contains('
            GroupID:')->size() = 1)]
88         <groupId>[at->any(ant : EAnnotation | ant.source.contains('GroupID
            :'))).source.toString().substring(10)/]</groupId>
89         [elseif (at->select(ant : EAnnotation | ant.source.contains('
            ArtifactID:')->size() = 1)]
90         <artifactId>[at->any(ant : EAnnotation | ant.source.contains('
            ArtifactID:')).source.toString().substring(13)/]</artifactId>
91         <name>[at->any(ant : EAnnotation | ant.source.contains('ArtifactID
            :'))).source.toString().substring(13)/]</name>
92         [elseif (at->select(ant : EAnnotation | ant.source.contains('
            Packaging:')->size() = 1)]
93         <packaging>[at->any(ant : EAnnotation | ant.source.contains('
            Packaging:')).source.toString().substring(12)/]</packaging>
94         [elseif (at->select(ant : EAnnotation | ant.source.contains('
            Version: ')>size() = 1)]
95         <version>[at->any(ant : EAnnotation | ant.source.contains('Version:
            ')).source.toString().substring(10)/]</version>
96     [/if]
97     [/for]
98 [else]

```

```

99     <groupId>AIRES_Server</groupId>
100    <artifactId>AIRES_Server</artifactId>
101    <version>0.0.1-SNAPSHOT</version>
102    <packaging>war</packaging>
103    [/if]
104
105    <build>
106        <sourceDirectory>src</sourceDirectory>
107        <plugins>
108            <plugin>
109                <artifactId>maven-compiler-plugin</artifactId>
110                <version>3.3</version>
111                <configuration>
112                    <source>1.8</source>
113                    <target>1.8</target>
114                </configuration>
115            </plugin>
116            <plugin>
117                <artifactId>maven-war-plugin</artifactId>
118                <version>2.6</version>
119                <configuration>
120                    <warSourceDirectory>WebContent</warSourceDirectory>
121                    <failOnMissingWebXml>false</failOnMissingWebXml>
122                </configuration>
123            </plugin>
124
125            <!-- plugin 3- apache cxf codegen wsdl2java goal -->
126            <plugin>
127                <groupId>org.apache.cxf</groupId>
128                <artifactId>cxf-java2ws-plugin</artifactId>
129                <version>${cxf.version}</version>
130                <dependencies>
131                    <dependency>
132                        <groupId>org.apache.cxf</groupId>
133                        <artifactId>cxf-rt-frontend-jaxws</artifactId>
134                        <version>${cxf.version}</version>
135                    </dependency>
136                    <dependency>
137                        <groupId>org.apache.cxf</groupId>
138                        <artifactId>cxf-rt-frontend-simple</artifactId>
139                        <version>${cxf.version}</version>
140                    </dependency>
141                </dependencies>
142                <executions>
143                    [comment for loop to iterate through services /]
144                    [for (c : Class | m.eContents(Class))]
145                    <execution>

```

```

146         <id>[c.name/]-exec</id>
147         <phase>process-classes</phase>
148         <configuration>
149             <className>rosana.labes.icmc.usp.br.[c.name + '
ServiceImpl']/></className>
150             <serviceName>[c.name + 'Service']/></serviceName
>
151             <address>http://localhost:8095/[c.name + '
Service']/></address>
152             <outputFile>${basedir}/src/main/resources/wsdl
/[c.name + 'Service']/wsdl</outputFile>
153             <argline>-createxsdimports</argline>
154             <genWsdL>true</genWsdL>
155             <verbose>true</verbose>
156             <skip>false</skip>
157             <genWrapperbean>false</genWrapperbean>
158             <genClient>false</genClient>
159             <genServer>false</genServer>
160         </configuration>
161         <goals>
162             <goal>java2ws</goal>
163         </goals>
164     </execution>
165 </for>
166 </executions>
167 </plugin>
168 </plugins>
169 </build>
170
171 <properties>
172     <cxf.version>3.1.2</cxf.version>
173     <spring.version>4.1.0.RELEASE</spring.version>
174     <hibernate.version>4.0.1.Final</hibernate.version>
175     <org.slf4j.version>1.7.5</org.slf4j.version>
176     <mysql.version>5.1.32</mysql.version>
177     <cxf.scope>compile</cxf.scope>
178     <compileSource>1.7</compileSource>
179     <maven.compiler.target>1.7</maven.compiler.target>
180     <maven.compiler.source>1.7</maven.compiler.source>
181     <project.build.sourceEncoding>UTF-8</project.build.
sourceEncoding>
182 </properties>
183
184 <dependencies>
185     <!-- apache cxf jax-ws 3.1.2 -->
186     <dependency>
187         <groupId>org.apache.cxf</groupId>

```

```

188         <artifactId>cxfrtfrontendjaxws</artifactId>
189         <version>${cxf.version}</version>
190         <scope>${cxf.scope}</scope>
191     </dependency>
192     <dependency>
193         <groupId>org.apache.cxf</groupId>
194         <artifactId>cxfrttransportshttp</artifactId>
195         <version>${cxf.version}</version>
196         <scope>${cxf.scope}</scope>
197     </dependency>
198
199     <!-- Spring -->
200     <!-- spring framework 4.1.0 -->
201     <dependency>
202         <groupId>org.springframework</groupId>
203         <artifactId>spring-web</artifactId>
204         <version>${spring.version}</version>
205         <scope>compile</scope>
206     </dependency>
207
208     <!-- Hibernate -->
209     <dependency>
210         <groupId>org.hibernate</groupId>
211         <artifactId>hibernate-core</artifactId>
212         <version>${hibernate.version}</version>
213     </dependency>
214     <dependency>
215         <groupId>org.hibernate</groupId>
216         <artifactId>hibernate-validator</artifactId>
217         <version>4.2.0.Final</version>
218     </dependency>
219     <dependency>
220         <groupId>org.hibernate.common</groupId>
221         <artifactId>hibernate-commons-annotations</artifactId>
222         <version>${hibernate.version}</version>
223         <classifier>tests</classifier>
224     </dependency>
225     <dependency>
226         <groupId>org.hibernate.javax.persistence</groupId>
227         <artifactId>hibernate-jpa-2.0-api</artifactId>
228         <version>1.0.1.Final</version>
229     </dependency>
230     <dependency>
231         <groupId>org.hibernate</groupId>
232         <artifactId>hibernate-entitymanager</artifactId>
233         <version>${hibernate.version}</version>
234     </dependency>

```

```
235
236 <!-- MySQL -->
237 <dependency>
238     <groupId>mysql</groupId>
239     <artifactId>mysql-connector-java</artifactId>
240     <version>${mysql.version}</version>
241 </dependency>
242
243 <!-- Logging -->
244 <dependency>
245     <groupId>org.slf4j</groupId>
246     <artifactId>slf4j-api</artifactId>
247     <version>${org.slf4j.version}</version>
248 </dependency>
249 <dependency>
250     <groupId>org.slf4j</groupId>
251     <artifactId>jcl-over-slf4j</artifactId>
252     <version>${org.slf4j.version}</version>
253     <scope>runtime</scope>
254 </dependency>
255 <dependency>
256     <groupId>org.slf4j</groupId>
257     <artifactId>slf4j-log4j12</artifactId>
258     <version>${org.slf4j.version}</version>
259     <scope>runtime</scope>
260 </dependency>
261 <dependency>
262     <groupId>log4j</groupId>
263     <artifactId>log4j</artifactId>
264     <version>1.2.15</version>
265     <exclusions>
266         <exclusion>
267             <groupId>javax.mail</groupId>
268             <artifactId>mail</artifactId>
269         </exclusion>
270         <exclusion>
271             <groupId>javax.jms</groupId>
272             <artifactId>jms</artifactId>
273         </exclusion>
274         <exclusion>
275             <groupId>com.sun.jdmk</groupId>
276             <artifactId>jmxtools</artifactId>
277         </exclusion>
278         <exclusion>
279             <groupId>com.sun.jmx</groupId>
280             <artifactId>jmxri</artifactId>
281         </exclusion>
```

```

282         </exclusions>
283         <scope>runtime</scope>
284     </dependency>
285
286 </dependencies>
287
288 </project>
289 [/file]
290 [/template]
291
292
293 [template public generateHibernateXml (m : Model) ]
294 [file ('hibernate' + '.cfg.xml', false, 'UTF-8')]
295 <?xml version="1.0" encoding="utf-8"?>
296 <!DOCTYPE hibernate-configuration PUBLIC
297 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
298 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
299
300 <hibernate-configuration>
301     <session-factory>
302         [comment The connection settings could be found in the
303         EAnnotations of the model, but if they are not found the mysql
304         database settings are used./]
305         <!-- Connection settings -->
306         [if (m.eAnnotations->size()>0)]
307         [for (at : EAnnotation | m.eAnnotations)]
308             [if (at->select(ant : EAnnotation | ant.source.contains('Driver
309             '))->size() = 1)]
310             <property name="hibernate.connection.driver_class">[at->any(ant :
311             EAnnotation | ant.source.contains('Driver')).source.toString()
312             /]</property>
313             [elseif (at->select(ant : EAnnotation | ant.source.contains('
314             jdbc:'))->size() = 1)]
315             <property name="hibernate.connection.driver_class">[at->any(ant :
316             EAnnotation | ant.source.contains('jdbc:')).source.toString()/]</
317             property>
318             [elseif (at->select(ant : EAnnotation | ant.source.contains('
319             user'))->size() = 1)]
320             <property name="hibernate.connection.driver_class">[at->any(ant :
321             EAnnotation | ant.source.contains('user')).source.toString().
322             substring(6)/]</property>
323             [elseif (at->select(ant : EAnnotation | ant.source.contains('
324             password'))->size() = 1)]
325             <property name="hibernate.connection.driver_class">[at->any(ant :
326             EAnnotation | ant.source.contains('password')).source.toString().
327             substring(10)/]</property>

```

```

314     [elseif (at->select(ant : EAnnotation | ant.source.contains('
dialect'))->size() = 1)]
315     <!-- SQL dialect -->
316     <property name="hibernate.dialect">[at->any(ant : EAnnotation |
ant.source.contains('dialect')).source.toString()]/></property>
317     [/if]
318     [/for]
319     [else]
320     <property name="hibernate.connection.driver_class">com.mysql.jdbc
.Driver</property>
321     <property name="hibernate.connection.url">jdbc:mysql://localhost
:3306/jwt</property>
322     <property name="hibernate.connection.username">root</property>
323     <property name="hibernate.connection.password">root</property>
324     <!-- SQL dialect -->
325     <property name="hibernate.dialect">org.hibernate.dialect.
MySQLDialect</property>
326     [/if]
327     <!-- Print executed SQL to stdout -->
328     <property name="show_sql">true</property>
329     <property name="format_sql">true</property>
330     [comment Here the entities mapping will be constructed /]
331     <!-- Annotated entity classes -->
332     [for (c : Class | m.eContents(Class))]
333     <mapping class="
334     [if ( (c.eAnnotations->size())>0) and (c.eAnnotations->select(ant
: EAnnotation | ant.source.contains('package') and ant.source.
contains('bean') and (not(ant.source.contains(';'))))->size()>0 )
)]
335     [c.eAnnotations->select(ant : EAnnotation | ant.source.contains('
package') and ant.source.contains('bean') and (not(ant.source.
contains(';')))).source.substring(9).toString() + '.'/][c.name/]
336     [else]
337     rosana.labes.icmc.usp.br.[c.name/]
338     [/if]"/>
339     [/for]
340     </session-factory>
341 </hibernate-configuration>
342 [/file]
343 [/template]
344
345 [template public generatePersistenceXml (m : Model)]
346 [file ('persistence' + '.xml', false, 'UTF-8')]
347 <persistence xmlns="http://java.sun.com/xml/ns/persistence"
348     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
349     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://
java.sun.com/xml/ns/persistence/persistence_2_0.xsd"

```



```

350 version="2.0">
351 <persistence-unit name="entityManager">
352   <provider>org.hibernate.ejb.HibernatePersistence</provider>
353   [comment Here the entities will be added to the persistence.xml/]
354   <!-- Annotated entity classes -->
355   [for (c : Class | m.eContents(Class))]
356   <class>
357     [if ( (c.eAnnotations->size())>0) and (c.eAnnotations->select(ant
: EAnnotation | ant.source.contains('package') and ant.source.
contains('bean') and (not(ant.source.contains(';')))->size())>0 )
)]
358     [c.eAnnotations->select(ant : EAnnotation | ant.source.contains('
package') and ant.source.contains('bean') and (not(ant.source.
contains(';')))).source.substring(9).toString() + '.'/][c.name/][
else]rosana.labes.icmc.usp.br.[c.name/][[/if]</class>
359   [/for]
360   <!-- Excluding unlisted classes -->
361   <exclude-unlisted-classes>false</exclude-unlisted-classes>
362   <properties>
363     <property name = "javax.persistence.schema-generation.database.
action" value="none"/>
364     [if (m.eAnnotations->size())>0]]
365     [for (at : EAnnotation | m.eAnnotations)]
366     [if (at->select(ant : EAnnotation | ant.source.contains('Driver
'))->size() = 1)]
367     <property name="hibernate.connection.driver_class" value="[at->
any(ant : EAnnotation | ant.source.contains('Driver')).source.
toString()/]" />
368     [elseif (at->select(ant : EAnnotation | ant.source.contains('
jdbc:'))->size() = 1)]
369     <property name="hibernate.connection.driver_class" value="[at->
any(ant : EAnnotation | ant.source.contains('jdbc:')).source.
toString()/]" />
370     [elseif (at->select(ant : EAnnotation | ant.source.contains('
user'))->size() = 1)]
371     <property name="hibernate.connection.driver_class" value ="[at
->any(ant : EAnnotation | ant.source.contains('user')).source.
toString().substring(6)/]" />
372     [elseif (at->select(ant : EAnnotation | ant.source.contains('
password'))->size() = 1)]
373     <property name="hibernate.connection.driver_class" value="[at->
any(ant : EAnnotation | ant.source.contains('password')).source.
toString().substring(10)/]" />
374     [elseif (at->select(ant : EAnnotation | ant.source.contains('
dialect'))->size() = 1)]
375     <!-- SQL dialect -->

```

```

376     <property name="hibernate.dialect" value="[at->any(ant :
EAnnotation | ant.source.contains('dialect')).source.toString()]"
/>
377     [/if]
378     [/for]
379     [else]
380     <property name="hibernate.connection.driver_class" value="com.
mysql.jdbc.Driver" />
381     <property name="hibernate.connection.url" value="jdbc:mysql://
localhost:3306/aires" />
382     <property name="hibernate.connection.username" value="root" />
383     <property name="hibernate.connection.password" value="root" />
384     <!-- SQL dialect -->
385     <property name="hibernate.dialect" value="org.hibernate.
dialect.MySQLDialect" />
386     [/if]
387     [comment Here the hibernate will always update de database and
if it is not creates will create from scratch /]
388     <property name="hibernate.hbm2ddl.auto" value="create" />
389     </properties>
390 </persistence-unit>
391 </persistence>
392 [/file]
393 [/template]
394
395
396
397 [template public generateUtil (ct : OclAny)]
398 [file ('HibernateUtil' + '.java', false, 'UTF-8')]
399 package rosana.labes.icmc.usp.br;
400
401 import javax.persistence.EntityManagerFactory;
402 import javax.persistence.Persistence;
403 import org.hibernate.SessionFactory;
404
405 public class HibernateUtil {
406     private static final EntityManagerFactory factory;
407     static {
408         try {
409             [comment Maybe the persistence-unit name property could be send
to template inside of a EAnnotation. What you think? /]
410             factory = Persistence.createEntityManagerFactory("
entityManager");
411         } catch (Throwable ex) {
412             System.err.println("EntityManagerFactory creation failed"
+ ex);
413             throw new ExceptionInInitializerError(ex);

```

```
414     }
415 }
416
417 public static EntityManagerFactory getEntityManagerFactory() {
418     return factory;
419 }
420 }
421
422
423 [/file]
424 [/template]
425
426
427 [template public generateDAO (c : Class) ]
428 [comment ((c.allFeatures()->filter(Property)->select(at : Property |
    at.name.contains('id'))->size()<=1) and (c.eAnnotations->select(
    ant : EAnnotation | ant.source.contains('package') and ant.source.
    contains('bean') )->size() >= 1)/]
429 [file (c.name + 'DAO.java', false, 'UTF-8')]
430 package rosana.labes.icmc.usp.br;
431
432 import rosana.labes.icmc.usp.br.HibernateUtil;
433 import javax.persistence.EntityManager;
434 import javax.persistence.Query;
435 import org.hibernate.HibernateException;
436 import java.util.List;
437 import rosana.labes.icmc.usp.br.[c.name/];
438
439
440
441
442 public class [c.name/]DAO {
443
444     public void create[c.name.toUpperFirst()/] ([c.name/] [c.name.
    toLowerFirst()/]){
445
446         EntityManager em = HibernateUtil.getEntityManagerFactory().
    createEntityManager();
447
448         try {
449             em.getTransaction().begin();
450                 em.persist([c.name.toLowerFirst()/]);
451                 em.getTransaction().commit();
452             System.out.println("Records created successfully");
453         }catch(HibernateException e){
454             e.printStackTrace();
455         }
```

```

456
457     }
458
459     public List<[c.name/]> retrieveAll([c.name.toUpperFirst()/])() {
460
461         EntityManager em = HibernateUtil.getEntityManagerFactory().
createEntityManager();
462         List<[c.name.toUpperFirst()/]> [c.name.toLowerFirst() + '1' /]
= null;
463         try {
464             em.getTransaction().begin();
465             Query query = em.createQuery("FROM [c.name.toUpperFirst()/]")
;
466             [c.name.toLowerFirst() + '1' /] = (List<[c.name/]>) query.
getResultList();
467             System.out.println("Records retrieved successfully");
468         } catch (HibernateException e) {
469
470             e.printStackTrace();
471         }
472
473         return [c.name.toLowerFirst() + '1; ' /]
474     }
475
476     [if (( (c.attribute->select(at : Property | at.name.endsWith('id
') or at.name.endsWith('Id') or at.name.endsWith('id') or at.name.
endsWith('ID'))->size()<2) and (c.attribute->select(at : Property
| at.name.endsWith('id') or at.name.endsWith('Id') or at.name.
endsWith('id') or at.name.endsWith('ID'))->size()>0) ))]
477     public [c.name/] retrieve([c.name.toUpperFirst()/](int id) {
478
479         EntityManager em = HibernateUtil.getEntityManagerFactory().
createEntityManager();
480         [c.name/] [c.name.toLowerFirst() + '1' /] = null;
481         try {
482             [c.name.toLowerFirst() + '1' /] = ([c.name/])em.find([c.
name + '.class' /], id);
483             System.out.println("Records retrieved successfully");
484         } catch (HibernateException e) {
485
486             e.printStackTrace();
487         }
488
489         return [c.name.toLowerFirst() + '1; ' /]
490     }
491 [/if]
492

```

```

493     [if (( (c.attribute->select(at : Property | at.name.endsWith('id
      ') or at.name.endsWith('Id') or at.name.endsWith('id') or at.name.
      endsWith('ID'))->size()<2) and (c.attribute->select(at : Property
      | at.name.endsWith('id') or at.name.endsWith('Id') or at.name.
      endsWith('ID') or at.name.endsWith('ID'))->size()>0) ))]
494     public [c.name/] delete[c.name.toUpperFirst()/]([c.name/] [c.
      name.toLowerFirst()/]) {
495
496         EntityManager em = HibernateUtil.getEntityManagerFactory().
      createEntityManager();
497
498         [c.name/] [c.name.toLowerFirst() + '1' /] = null;
499         try {
500             em.getTransaction().begin();
501             [comment Here we can considere thar exists one attribute with
      name id, because of the entity template pre-condition /]
502             [c.name.toLowerFirst() + '1' /] = ([c.name/])em.find([c
      .name + '.class' /], [c.name.toLowerFirst() + '.get' /][c.attribute
      ->select(att : Property | att.name.contains('id')).name.
      toUpperFirst() + '()' /]);
503             em.remove([c.name.toLowerFirst() + '1' /]);
504             em.getTransaction().commit();
505             System.out.println("[c.name/] records deleted!");
506
507         } catch (HibernateException e) {
508
509             e.printStackTrace();
510
511         }
512
513     return [c.name.toLowerFirst() + '1' /];
514
515     }
516 [/if]
517
518     public void update[c.name.toUpperFirst()/] ([c.name/] [c.name.
      toLowerFirst()/]) {
519
520         EntityManager em = HibernateUtil.getEntityManagerFactory().
      createEntityManager();
521
522         try {
523             em.getTransaction().begin();
524             em.merge([c.name.toLowerFirst()/]);
525             em.getTransaction().commit();
526             System.out.println("[c.name.toUpperFirst()/] records
      updated!");

```

```
527         } catch (HibernateException e) {
528
529             e.printStackTrace();
530         }
531     }
532
533 }
534
535
536
537
538 [/file]
539 [/template]
540
541 [template public generateService (c : Class) ? (c.allFeatures()->
    filter(Property)->select(at : Property | at.name.contains('id') or
    at.name.contains('Id') or at.name.contains('iD') or at.name.
    contains('ID'))->size() <=1) ]
542 [comment ? ((c.allFeatures()->filter(Property)->select(at : Property
    | at.name.contains('id'))->size() <=1) and (c.eAnnotations->select
    (ant : EAnnotation | ant.source.contains('package') and ant.source
    .contains('bean') )->size() >= 1) ) /]
543
544 [file (c.name + 'ServiceImpl.java', false, 'UTF-8')]
545 package rosana.labes.icmc.usp.br;
546
547 import java.util.List;
548
549 import javax.jws.WebService;
550 import javax.jws.soap.SOAPBinding;
551 import javax.jws.soap.SOAPBinding.Style;
552 import javax.jws.soap.SOAPBinding.Use;
553 import rosana.labes.icmc.usp.br.[c.name + 'DAO']/];
554 //import rosana.labes.icmc.usp.br.[ 'I' + c.name + 'Service']/];
555 //import rosana.labes.icmc.usp.br.[c.name]/];
556
557
558
559 @WebService(targetNamespace = "http://rosana.labes.icmc.usp.br/",
    portName = "[c.name/]Port", serviceName = "[c.name/]Service")
560 @SOAPBinding(style = Style.DOCUMENT, use = Use.LITERAL)
561 public class [c.name/]ServiceImpl {
562
563     private [c.name + 'DAO']/] [c.name.toLowerFirst() + 'DAO']/];
564
565     public [c.name/]ServiceImpl([c.name + 'DAO']/] [c.name.
    toLowerFirst() + 'DAO']/]){
```

```

566         this.[c.name.toLowerFirst() + 'DAO']/]=[c.name.toLowerFirst() +
'DAO']/];
567     }
568
569     public [c.name/]ServiceImpl(){
570         this.[c.name.toLowerFirst() + 'DAO']/ = new [c.name + 'DAO']/();
571     }
572
573     public void create[c.name.toUpperFirst()/] ([c.name/] [c.name.
toLowerFirst()/]){
574         this.[c.name.toLowerFirst() + 'DAO']/.create[c.name.
toUpperFirst()/]([c.name.toLowerFirst()/]);
575     }
576
577     public List<[c.name/]> retrieveAll[c.name.toUpperFirst()/]() {
578         return ['this.'+ c.name.toLowerFirst() + 'DAO']/.retrieveAll[c.
name.toUpperFirst()/]();
579     }
580
581     [if (( (c.attribute->select(at : Property | at.name.endsWith('id
') or at.name.endsWith('Id') or at.name.endsWith('iD') or at.name.
endsWith('ID'))->size()<2) and (c.attribute->select(at : Property
| at.name.endsWith('id') or at.name.endsWith('Id') or at.name.
endsWith('iD') or at.name.endsWith('ID'))->size()>0) ))]
582     public [c.name/] retrieve[c.name.toUpperFirst()/](int id) {
583         return ['this.'+ c.name.toLowerFirst() + 'DAO']/.retrieve[c.
name.toUpperFirst()/](id);
584     }
585     [/if]
586
587     [if (( (c.attribute->select(at : Property | at.name.endsWith('id
') or at.name.endsWith('Id') or at.name.endsWith('iD') or at.name.
endsWith('ID'))->size()<2) and (c.attribute->select(at : Property
| at.name.endsWith('id') or at.name.endsWith('Id') or at.name.
endsWith('iD') or at.name.endsWith('ID'))->size()>0) ))]
588     public [c.name/] delete[c.name.toUpperFirst()/]([c.name/] [c.name
.toLowerFirst()/]) {
589         return ['this.'+ c.name.toLowerFirst() + 'DAO']/.delete[c.
name.toUpperFirst()/]([c.name.toLowerFirst()/]);
590     }
591     [/if]
592
593     public void update[c.name.toUpperFirst()/] ([c.name/] [c.name.
toLowerFirst()/]) {
594         ['this.'+ c.name.toLowerFirst() + 'DAO']/.update[c.name.
toUpperFirst()/]([c.name.toLowerFirst()/]);

```

```
595      }  
596  
597  }  
598  
599  [/file]  
600  
601  [/template]
```

ARTIGOS SELECIONADOS DURANTE A PRIMEIRA SELEÇÃO DA REVISÃO SISTEMÁTICA

Tabela 5 – Subconjunto dos artigos obtidos pela primeira seleção.

Identificador	Título	Autor(es)
P1	An architecture model for dynamically converting components into Web services	(LEE; KIM; YANG, 2004)
P2	Intelligent business processes composition based on multi-agent systems	(CORIA; CASTELLANOS-GARZÓN; CORCHADO, 2013)
P3	Reuse-Oriented Deployment of Software Components: Congregation in Service-Oriented Development	(ATKINSON; BOSTAN; FINK, 2009)
P4	Reusing existing object-oriented code as web services in a SOA	(SNEED; VERHOEF; SNEED, 2013)
P5	Service-oriented lightweight software asset repository open framework	(HONGMIN; JIN; JINGZHOU, 2010)
P6	A distributed e-education system based on the service Oriented Architecture	(PASATCHA; SUNAT, 2008)
P7	A distributed e-healthcare system based on the service oriented architecture	(KART <i>et al.</i> , 2007)
P8	An approach for e-Commerce on-demand service-oriented product line development	(EZENWOKE; MISRA; ADIGUN, 2013)
P9	An approach to developing multi-tenancy SaaS using metaprogramming	(RODRIGUES <i>et al.</i> , 2012)

Continuação da Tabela 5		
Identificador	Título	Autor(es)
P10	An effective assemble-oriented framework for grid Web service	(CHEN <i>et al.</i> , 2007)
P11	An assessment on technologies for implementing core assets in service-oriented product lines	(RIBEIRO <i>et al.</i> , 2010)
P12	A reference architecture for Web servers	(HASSAN; HOLT, 2000)
P13	Automatically wrapping legacy software into services: A grid case study	(LI <i>et al.</i> , 2008)
P14	Based on SOA architecture and component software reuse architecture research	(YUE <i>et al.</i> , 2010)
P15	Building a service-oriented ERP from an open source software	(KOUAMOU, 2009)
P16	Building reusable components with service-oriented architectures	(ZHU, 2005)
P17	Client-Cloud Computing: A Framework for Service Integration	(CHANG; KAO; LO, 2012)
P18	CLPL: Providing software infrastructure for the systematic and effective construction of complex collaborative learning systems	(CABALLÉ; XHAFA, 2010)
P19	Component specification for enterprise software development on Web services environment	(CHO; MCGREGOR, 2005)
P20	Contextualized User-Centric Multimedia Delivery System for Next Generation Networks	(SIMOES; MAGEDANZ, 2011)
P21	Leveraging service-oriented architecture to extend a legacy commerce system	(MULCAHY; HUANG; VEGHTE, 2010)
P22	Registry and Discovery of Services with Variability Based on 2-Level UDDI	(YEOM; PARK; MOON, 2011)
P23	Service oriented programming: A new paradigm of software reuse	(SILLITTI; VERNAZZA; SUCCI, 2002)
P24	The study on software reuse based on service composition and dependency-aware service oriented architecture	(LIU, 2012)
P25	Towards a semantic-based approach for software reusable component classification and retrieval	(YAO; ETZKORN, 2004)
P26	Towards a user-oriented framework for web service discovery, reuse and evolution	(CHUKMOL; BENHARKAT; AMGHAR, 2010)

Continuação da Tabela 5		
Identificador	Título	Autor(es)
P27	What COTS and Software Reuse Teach Us about SOA	(ANDERSON, 2007)