Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

Departamento de Engenharia de Computação e Automação Industrial

Composição Automática de Serviços Web Semânticos: Uma Abordagem com Times Assíncronos e Operadores Genéticos

Autor: Neil Paiva Tizzo

Orientador: Prof. Dr. Eleri Cardozo

Coorientador: Prof. Dr. Juan Manuel Adán Coello

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para a obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: **Engenharia de Computação**.

Banca Examinadora

Prof. Dr. Eleri Cardozo (presidente) — DCA/FEEC/UNICAMP

Profa. Dra. Maria da Graça Campos Pimentel — ICMC/USP

Prof. Dr. Aqueo Kamada — Centro de Tecnologia da Informação Renato Archer (CTI).

Profa. Dra. Maria Beatriz Felgar de Toledo — IC/UNICAMP

Prof. Dr. Romis Ribeiro de Faissol Attux — DCA/FEEC/UNICAMP

Campinas – SP

16/04/2012

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Tizzo, Neil Paiva
T546c Composição automática de serviços web semânticos: uma abordagem com times assíncronos e operadores genéticos / Neil Paiva Tizzo. --Campinas, SP: [s.n.], 2012.

Orientador: Eleri Cardozo,
Cooriendador: Juan Manuel Adán Coello.
Tese de Doutorado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Algoritmos genéticos. 2. Serviços na web - Semântica. 3. Web semântica. I. Cardozo, Eleri. II. Adán Coello, Juan Manuel. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Título em Inglês: Automatic composition of semantic web services: an approach with asynchronous teams and genetic operators
Palavras-chave em Inglês: Genetic algorithms, Services web - Semantic, Semantic web
Área de concentração: Engenharia de Computação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: Maria da Graça Campos Pimentel, Aqueo Kamada, Maria Beatriz Felgar de Toledo, Romis Ribeiro de Faissol Attux
Data da defesa: 16-04-2012
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidato: Neil Paiva Tizzo

Data da Defesa: 16 de abril de 2012

Título da Tese: "Composição Automática de Serviços Web Semânticos: Uma Abordagem com Times Assíncronos e Operadores Genéticos"

Prof. Dr. Eleri Cardozo (Presidente): _____

Prof. Dr. Aqueo Kamada: _____

Profa. Dra. Maria Beatriz Felgar de Toledo: _____

Profa. Dra. Maria da Graça Campos Pimentel: _____

Prof. Dr. Romis Ribeiro de Faissol Attux: _____

Resumo

A automação da composição de serviços Web é, na visão do autor, um dos problemas mais importantes da área de serviços Web. Além de outras características, destaca-se que somente a composição automática é capaz de lidar com ambientes mutáveis onde os serviços são permanentemente inseridos, removidos e modificados. Os métodos existentes para realizar a composição automática de serviços apresentam várias limitações. Alguns tratam de um número muito restrito de fluxos de controles e outros não consideram a marcação semântica dos serviços. Em adição, em muitos casos não há avaliações quantitativas do desempenho dos métodos. Desta forma, o objetivo desta tese é propor um método para realizar a composição automática de serviços Web semânticos que considera os cinco tipos básico de fluxo de controle identificados pela Workflow Management Coalition, a saber: sequencial, separação paralela, sincronização, escolha-exclusiva e união simples; bem como para o fluxo de controle em laço, considerado um fluxo do tipo estrutural. As regras que descrevem a composição entre os serviços são híbridas, baseadas em semântica e em técnicas de recuperação de informação. Os serviços são descritos em OWL-S, uma ontologia descrita em OWL que permite descrever semanticamente os atributos IOPE (parâmetros de entrada, de saída, pré-requisitos e efeitos) de um serviço, mas somente os parâmetros de entrada e saída foram levados em consideração neste trabalho. Para validar a abordagem foi implementado um protótipo que utilizou times assíncronos (A-Teams) com agentes baseados em algoritmos genéticos para realizar a composição segundo os padrões de fluxo sequencial, paralelo e sincronização. A avaliação experimental do algoritmo de composição foi realizada utilizando uma coleção de serviços Web semânticos pública composta de mais de 1000 descrições de serviços. As avaliações de desempenho, em vários cenários típicos, medidas em relação ao tempo de resposta médio e à quantidade de vezes em que a função de avaliação (função *fitness*) é calculada são igualmente apresentadas. Para os casos mais simples de composição, o algoritmo conseguiu reduzir o tempo de resposta em relação a uma busca cega em aproximadamente 97%. Esta redução aumenta à medida que a complexidade da composição também aumenta.

Palavras-chave: Algoritmos genéticos, Serviços na web - Semântica, Web semântica.

Abstract

The automation of the composition of Web services is, in the view of the author, one of the most important problems in the area of Web services. Beyond other characteristics, only the automatic composition can deal with a changing environment where the services are permanently inserted, removed, and modified. Existing methods performing the automatic service composition have several limitations. Some deal with a very limited number of control flow patterns, while others do not consider the semantic markup of services. In addition, in many cases there is no quantitative evaluation of the method's performance. In such a way, the objective of this thesis is to propose a method to perform the automatic composition of semantic Web services considering the five basic types of control flow identified by the Workflow Management Coalition, namely: sequential, parallel split, synchronization, exclusive choice and simple merge; and for loop control flow, classified as a structural control flow pattern. The rules that describe the composition of the service are hybrid: based in semantics and in information retrieval techniques. Services are described in OWL-S, an ontology described in OWL that allows the semantically description of the IOPE attributes (input, output, prerequisite and effect) of a service, but only the input and output parameters were taken into consideration in this work. A prototype was implemented to validate the proposed rules. An asynchronous Team (A-Team) algorithm with genetic agents was used to carry out the composition according to the sequential, parallel and synchronization control flows. The experimental evaluation of the composition algorithm employed a public collection of semantic Web services composed of more than 1000 descriptions of services. An experimental performance evaluation showed that, for simple composition cases, the algorithm reduced the average response time in approximately 97%, when compared to blind search. This reduction increases as the composition complexity increases.

Keywords: Genetic algorithms, Services web - Semantic, Semantic web.

Com muito amor, à minha
esposa, Evaine, e aos meus filhos,
Lucca e Enzo.

Agradecimentos

Ao Prof. Dr. Manuel de Jesus Mendes, pela primeira orientação. Minha profunda admiração à sua trajetória científica.

Ao Prof. Dr. Eleri Cardozo, que prontamente aceitou orientar-me quando o Prof. Dr. Manuel de Jesus Mendes se aposentou. Uma gentileza da qual jamais me esquecerei.

Ao Prof. Dr. Juan Manuel Adán Coello, pela coorientação, pela dedicação e apoio recebidos durante todos estes anos. Meu muito obrigado.

Ao Gerd Schürmann e ao Prof. Dr. Dr. h.c. Radu Popesku-Zeletin, pela orientação durante o período em que estive na Alemanha.

À CAPES e ao DAAD, pela bolsa de estudo no Fraunhofer Institute for Open Communication System (FOKUS).

À PUC Minas, pelo apoio financeiro através do seu Programa Permanente de Capacitação Docente (PPCD).

À FAPESP e à FAPEMIG, pelo apoio financeiro recebido para a publicação de artigos.

Aos colegas do Departamento de Ciência da Computação da PUC Minas campus de Poços de Caldas: João, Sônia, Fabiano, Luiz, Luciana, Will, Udo, Boca, Faria, Claudio, Márcio e Iran, pela solidariedade durante esta jornada.

Pesquisa desenvolvida com o auxílio do CENAPAD-SP.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

T - Terminologia da ontologia utilizada pelas regras de casamento e de composição de serviços especificada em OWL-DL $SHOIN(D)$

CT^T - Hierarquia de subsunção de conceitos de T

C^A - Expressão lógica dependente da variável A

$f(C^A)$ - Função de avaliação da expressão lógica C^A

S – Serviço

AbS – Serviço abstrato

AtS – Serviço atômico

CpS - Serviço composto

FrS – Serviço fronteira

JsS – Serviço jusante

MtS – Serviço montante

$CcS = \{CcS_1, CcS_2, \dots, CcS_\mu\}$ - Repositório de serviços concretos

CcS_i - Serviço concreto i

$OP^S = \{op_1^S, op_2^S, \dots, op_\gamma^S\}$ - Conjunto de operações do serviço S

op_i^S - Operação i do serviço S

$IN^S = \{in_1^S, in_2^S, \dots, in_\alpha^S\}$ - Conjunto de parâmetros de entrada do serviço S

in_i^S - Parâmetro de entrada i do serviço S

$IN^{op_i^S} = \left\{in_1^{op_i^S}, in_2^{op_i^S}, \dots, in_\delta^{op_i^S}\right\}$ - Conjunto de parâmetros de entrada da operação i do serviço S

$in_j^{op_i^S}$ - Parâmetros de entrada j da operação i do serviço S

$OUT^S = \{out_1^S, out_2^S, \dots, out_\beta^S\}$ - Conjunto de parâmetros de saída do serviço S

out_i^S - Parâmetro de saída i do serviço S

$OUT^{op_i^S} = \left\{out_1^{op_i^S}, out_2^{op_i^S}, \dots, out_\varepsilon^{op_i^S}\right\}$ - Conjunto de parâmetros de saída da operação i do serviço S

$out_j^{op_i^S}$ - Parâmetros de saída j da operação i do serviço S

pv_i^{AbS} - Operação polivalente i do serviço abstrato AbS

$POS^S = \{pos_1^S, pos_2^S, \cdots, pos_\lambda^S\}$ - Conjunto de pós-condições do serviço S

pos_i^S – pós-condição do serviço S

$PRE^S = \{pre_1^S, pre_2^S, \cdots, pre_\theta^S\}$ - Conjunto de pré-condições do serviço S

pre_i^S – pré-condição do serviço S

α - Quantidade total dos parâmetros de entrada do serviço S

β - Quantidade total dos parâmetros de saída do serviço S

γ - Quantidade total de operações do serviço S

δ - Quantidade total dos parâmetros de entrada da operação op_i^S

ε - Quantidade total dos parâmetros de saída da operação op_i^S

θ - Quantidade de pré-condições do serviço S

λ – Quantidade de pós-condições do serviço S

$SIM(A, B)$ - Similaridade entre os conceitos A e B

$SIM^{E}(A, B)$ - Similaridade do tipo exata entre os conceitos A e B

$SIM^{AD}(A, B)$ - Similaridade do tipo ascendência direta entre os conceitos A e B

$SIM^{AI}(A, B)$ - Similaridade do tipo ascendência indireta entre os conceitos A e B

$SIM^{DD}(A, B)$ – Similaridade do tipo descendência direta entre os conceitos A e B

$SIM^{DI}(A, B)$ – Similaridade do tipo descendência indireta entre os conceitos A e B

$SIM^{S}(A, B)$ - Similaridade sintática entre os conceitos A e B

$\dot{-}$ - Diferença de conceitos em CT^{T}

$\dot{+}$ - Soma de conceitos em CT^{T}

$Exclusivo\ (A, B)$ - Conjunto dos conceitos de A e de B que não possuem similaridade entre si

$CME(A, B)$ - Conjunto formado pelos conceitos mais específicos de A e de B que possuem similaridade entre si

$S_i \circledcirc S_j$ – Composição em laço entre os serviços S_i e S_j

$S_i \prec S_j$ – Composição sequencial entre os serviços S_i e S_j

$S_i \parallel S_j$ – Composição paralela entre os serviços S_i e S_j

$S_i \oplus S_j$ – Composição escolha exclusiva entre os serviços S_i e S_j

$\circledcirc$ - Laço

$\prec$ - Precedência

$\|$ - Paralelo

$\oplus$ - Escolha exclusiva

$\leftarrow$ - Atribuição

Lista de Siglas e Abreviações

AG - Algoritmo Genético

API – Application Program Interface

A-Teams – Asynchronous Teams

CENAPAD - Centro Nacional de Processamento de Alto Desempenho

CORBA - Common Object Request Broker Architecture

CoRE - Component Runtime Engine

CoSMoS - Component Service Model with Semantics

CSDL - Composite Service Definition Language

CSSL - Composite Service Specification Language

DAML-S - DARPA Agent Markup Language for Service

DCE - Distributed Computing Environment

DCOM - Distributed Component Object Model

DP - Desvio padrão

Esecsy - Evolutionary Service Composition System

FTP - File Transfer Protocol

HTN - Hierarchical Task Network

IA - Inteligência Artificial

IOPE – Input, Output, Preconditions and Effects

IRS-II - Internet Reasoning Service

Java RMI - Java Remote Method Invocation

MDA - Model Driven Architecture

MDP - Markov Decision Process

MOF - MetaObject Facility

ms - milisegundos

OASIS - Organization for the Advancement of Structured Information Standards

OMG – Object Management Group

OWL - Web Ontology Language

OWL-S – Semantic Markup for Web Service

PDDL - Planning Domain Definition Language

PPM - Polymorphic Process Model

QoS - Quality of Service

RDF - Resource Description Framework

RI – Recuperação de Informação

RPC - Remote Procedure Call

SAWSDL - Semantic Annotations for WSDL

SeGSeC - Semantic Graph based Service Composition

SOA - Service Oriented Architecture

SWSF - Semantic Web Service Framework

SWSO - Semantic Web Services Ontology

TRM - Tempo de Resposta Médio

UDDI - Universal Description, Discovery and Integration

UML - Unified Modeling Language

URL - Uniform Resource Locator

W3C - World Wide Web Consortium

WfMC - Workflow Management Coalition

WS-BPEL - Web Services Business Process Execution Language

WSDL - Web Service Description Language

WSDL-S - Web Service Semantics

WSESL - Web Service Execution Specification Language

WSML - Web Service Modeling Language

WSMO - Web Service Modeling Ontology

XML - Extensible Markup Language

Trabalhos Publicados pelo Autor

Artigos Publicados em Revistas

1. TIZZO, N. P.; ADÁN-COELLO, J. M.; CARDOZO, E. "**Dynamic Service Matching in WS-BPEL with SPARQL**". International Transactions on Systems Science and Applications, v. 7, 2011. ISSN 1751-1461.

Trabalhos publicados em anais de eventos (completo)

2. TIZZO, N. P.; ADÁN-COELLO, J. M.; CARDOZO, E. "**Automatic Composition of Semantic Web Services Using A-Teams with Genetic Agents**". Proceedings of the 2011 IEEE Congress on Evolutionary Computation. New Orleans, USA, 5-8 June 2011, p. 370–377.

3. TIZZO, N. P.; ADÁN-COELLO, J. M.; CARDOZO, E. "**Improving Dynamic Web Service Selection in WS-BPEL Using SPARQL**". Proceedings 2011 IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC 2011), Anchorage, AK, USA, 9-12 October 2011. 864 - 871. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6083760.

4. FLUEGGE, M.; SANTOS, I. J. G.; TIZZO, N. P.; MADEIRA, E. "**Challenges and Techniques in the Road to Dynamically Compose Web Services".** In: 6th International Conference on Web Engineering, 2006, Palo Alto, California, USA. Proceedings of the 6th international conference on Web engineering. New York, NY, USA: ACM Press, 2006. p.40 - 47

5. MENDES, M. J.; TIZZO, N. P.; BORELLI, J. R. **Federated Web Services Coordination Platform.** In: The IFIP Conference on E-Commerce, E-Business and E-Government, 2003, Guarujá. The IFIP Conference on E-Commerce, E-Business and E-Government. , 2003.

6. MENDES, M. J.; TIZZO, N. P.; FIGUEIREDO, A.; KAMADA, A.; RODRIGUES, M. A.; MUNIZ, C. R.; DAMASCENO, L. **Federation of Web Services.** In: Jornadas Chilenas de Computacion, 2002. Workshop SDs Copiapó. , 2002.

Capítulos de livros

7. FIGUEIREDO, A.; MENDES, M. J.; KAMADA, A.; BORELLI, J. R.; RODRIGUES, M. A.; DAMASCENO, L.; SOUZA JUNIOR, J. G.; TIZZO, N. P. "**Applying MDA Concepts in an e-Government Platform**". In: Electronic Government. Ed. : Springer Berlin / Heidelberg, 2004, v.3183, p. 260-265.

8. TIZZO, N. P.; BORELLI, J. R.; MENDES, M. J.; DAMASCENO, L.; KAMADA, A.; FIGUEIREDO, A.; RODRIGUES, M. A.; SOUZA JUNIOR, J. G. "**Service Composition Applied to E-Government**". In: Building the E-Service Society ed.: Springer Boston, 2004, v.146, p. 307-326.

Capítulo 1

Introdução

Houve até uma certa ocasião em que tive de carregar uma porção deles, lá da margem do Irtich para um local a uns duzentos metros da muralha da fortaleza, levei nisso dois meses. Feria-me os ombros a corda com que os amarrava; sem falar no peso. Mas gostei, pois a tarefa fortalecia meu corpo; no começo eu só aguentava transportar oito tijolos de cada vez, sendo o peso de oito arráteis. Por fim eu já carregava até mesmo doze ou quinze, fato que me alegrava. Isso de vida de prisão requer força física tanto quanto a moral; do contrário, como suportar vida tão excomungada. E eu tinha que sobreviver a ela...

Dostoiévski, Fiódor. Recordações das casas dos mortos. Edição Saraiva, São Paulo, 1949, p 200.

A necessidade de reutilização de programas computacionais[1] ou de trechos de programas é tema antigo, assim como a necessidade de integrá-los. A evolução da computação pode ser entendida como um movimento dialético entre pressão e alívio. O mercado pressiona os profissionais da área para o desenvolvimento de novas linguagens,

[1] O termo programa computacional, ou simplesmente programa, será utilizado como tradução do termo em inglês software.

ferramentas, modelos, métodos etc. que possam de alguma forma solucionar os problemas existentes. Então novas soluções aparecem, mas, em pouco tempo, o mercado sacia-se; novos problemas surgem e a pressão volta [1]. Este movimento dialético é uma das molas propulsoras de áreas como a Engenharia de Computação e a Ciência da Computação.

Em outros tempos, a reutilização de programas foi conseguida, se não totalmente, pelo menos em parte, com o advento das funções (ou procedimentos), das bibliotecas e mais recentemente com a programação orientada a objetos e componentes, para citar somente alguns exemplos. Por sua vez, *middlewares* como Distributed Computing Environment (DCE) [2], Distributed Component Object Model (DCOM) [3], Common Object Request Broker Architecture (CORBA) [4], Java Remote Method Invocation (Java RMI) [5] e .NET Framework [6] aumentam o grau de abstração da integração, baseada, essencialmente, em Chamada Remota de Procedimentos (RPC[2]) [7]. Entretanto, a cada nova solução abre-se um novo mundo de possibilidades... e a pressão ressurge.

O desenvolvimento de programas tenta se moldar ao contexto atual, que é da universalidade da Internet, da globalização e do senso de urgência cada vez mais prioritário. A Arquitetura Orientada a Serviços (SOA[3]) é uma proposta para a solução dos problemas que surgem diante deste cenário. Ela permite, por exemplo, o rápido desenvolvimento a partir da reutilização de trechos de código que podem ser acessados remotamente e, potencialmente, de qualquer lugar do mundo.

Adicionalmente, com o aumento dos usuários domésticos e o início do uso comercial da Internet na década de 90, vem se notando também a crescente necessidade para disponibilizar e interoperar serviços eletrônicos das mais diferentes naturezas.

Desde o início da computação, o desenvolvimento de programas é tarefa exclusivamente humana. E também, desde o início, linguagens, ferramentas, modelos, métodos e outros são criados com o objetivo de apoiar e automatizar, até agora em parte,

[2] Sigla em inglês para Remote Procedure Call.
[3] Sigla em inglês para Service Oriented Architecture.

esta tarefa intrinsecamente cognitiva. Com o advento da Extensible Markup Language (XML) [8], dos serviços Web [9], da arquitetura SOA [10], da Web Semântica [11] e de recursos computacionais cada vez mais disponíveis, dentre outros, novas abordagens para a automação do processo de desenvolvimento de programas têm se tornado objeto de estudo de vários grupos de pesquisa.

Este trabalho tem como tema central a automação do processo de desenvolvimento de programas dentro do contexto da arquitetura SOA ou, mais especificamente, dos serviços Web semânticos [12] [13], a partir do uso de Algoritmos Genéticos (AG) [14] e times assíncronos (A-Teams[4]) [15].

1.1 Composição como Princípio de Integração no Contexto SOA

SOA é uma arquitetura de desenvolvimento de programas baseada na integração de serviços. Este estilo de arquitetura preconiza que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços [10] [16] [17] [18] [19] [20]. Frequentemente, estes serviços são conectados através de um barramento de serviços que disponibiliza interfaces, ou contratos, acessíveis através de serviços Web ou outra forma de comunicação entre as aplicações. Assim sendo, a arquitetura SOA fornece a plataforma que permite a interligação de processos de negócio e recursos operacionais. Entretanto, para Knorr e Rist [21], SOA é fundamentalmente uma ideia e não uma tecnologia, e, portanto, ainda é possível encontrar definições divergentes.

De qualquer forma, em SOA, um programa passa a ser visto como um serviço. Um programa visto como serviço é potencialmente, mas não necessariamente, composto de vários outros serviços, geralmente distribuídos sobre uma infraestrutura de rede, e que pode ser configurado e ligado em tempo de execução. Para Turner et al. [22], a arquitetura SOA representa um paradigma radicalmente diferente na forma como o programa é atualmente construído e mantido. Desde a década de 60, a geração de código binário se baseia em alguma variante do ciclo editar-compilar-ligar. No contexto SOA, o

[4] Sigla em inglês para Asynchronous Teams.

conhecimento necessário ao desenvolvimento de programas passa a ser o da composição, não mais o da edição. Um novo serviço é essencialmente a composição de outros serviços já existentes.

As principais características da arquitetura SOA são [23]:

- Reusabilidade: o desenvolvimento de programas passa a ser visto como a composição de componentes já desenvolvidos. A reusabilidade agiliza o desenvolvimento;
- Interoperabilidade: habilidade para conectar aplicações entre plataformas, linguagens, sistemas operacionais e regiões geográficas. A interoperabilidade é alcançada devido à modularidade e ao fraco acoplamento entre os serviços;
- Escalabilidade: devido à sua característica de fraco acoplamento, as aplicações que utilizam serviços tendem a ser escaláveis;
- Flexibilidade: serviços com fraco acoplamento geralmente são mais flexíveis do que aplicações com acoplamento forte. A natureza assíncrona dos serviços, aliada à padronização da comunicação, permite que os serviços se adaptem às mudanças de requisitos ou às mudanças do ambiente;
- Baixo custo: outras abordagens que integram recursos díspares, tais como sistemas legados, aplicações entre parceiros comerciais e soluções específicas para cada departamento, são caras porque tendem a juntar estes componentes de forma personalizada.

Atualmente, a tecnologia mais promissora para a implantação da arquitetura SOA é representada pelos serviços Web [9] [24]. O maior benefício dos serviços Web está na sua interoperabilidade, resultado do uso de protocolos-padrão amplamente difundidos na Internet, e da existência de uma arquitetura padrão que define os mecanismos necessários para a sua utilização. Assim, os serviços Web tornaram-se a base das novas iniciativas de computação distribuída e de negócios eletrônicos, pois permitem construir redes de aplicações colaborativas distribuídas, intra e extraorganizações, onde os serviços

Web, na forma de módulos autocontidos, são descritos, publicados, localizados e invocados.

Assim sendo, a tecnologia de serviços Web é um *middleware* de suporte para o desenvolvimento de sistemas distribuídos. Oferece funcionalidades básicas que, de outra forma, teriam que ser programadas nas próprias aplicações. Estas funcionalidades básicas reduzem o custo de desenvolvimento de programas.

Tecnicamente, um serviço Web é um programa identificado por um Uniform Resource Locator (URL) que possui uma interface pública e bem definida e que pode ser acessado de forma eletrônica e remota através do uso de protocolos-padrão [25]. Suas interface e ligações são definidas e descritas com o uso da linguagem XML. Desta forma, sua definição (interface ou contrato) pode ser descoberta por outros programas.

Os serviços Web surgiram com a finalidade de substituir as tradicionais estratégias de integração de aplicações corporativas. Inicialmente, eram usados exclusivamente para designar a tentativa de uma empresa interligar suas aplicações internas de negócios, para que dados pudessem ser compartilhados. Recentemente, sua aplicabilidade foi expandida para englobar também a união de dados e processos com parceiros comerciais, governo eletrônico e outras.

É comum a confusão entre os serviços Web e as tecnologias relacionadas às páginas Web e aos *object brokers*, mas Sandeep e Madanmohan [26] estabelecem as diferenças entre eles:

- São programas e não documentos: um serviço Web não é um arquivo estático como uma página HTML em um sítio; é código que executa tarefas de negócio e gera respostas em tempo de execução;
- São otimizados para serem usados por outros programas e não por pessoas: a resposta retornada por um serviço Web não é formatada para que o navegador a apresente a uma pessoa. O programa cliente deve interpretar e operar sobre a informação;

- São acessíveis através do uso de tecnologias amplamente aceitadas na Internet: os serviços Web são construídos sobre as tecnologias estabelecidas da Internet como TCP/IP e XML e de protocolos-padrão cujas especificações são abertas.

Presentemente, os padrões nos quais os serviços Web se sustentam são: SOAP[5], Web Services Description Language (WSDL) e Universal Description, Discovery and Integration (UDDI) [27]:

- SOAP é um protocolo que define um formato de documento XML que descreve como invocar um método de um código remoto [28] [29] [30];
- WSDL é um vocabulário XML que expõe, entre outros, o nome do serviço Web, sua localização física, os nomes de seus métodos e os argumentos daqueles métodos [31] [32];
- UDDI é um repositório de descrições de serviços Web acessível através do protocolo SOAP [33].

Com estes padrões, a interoperabilidade é alcançada independentemente das linguagens de programação e dos sistemas operacionais utilizados. Atualmente, as atividades de padronização dos serviços Web concentram-se em dois grupos: World Wide Web Consortium (W3C) [9] e Organization for the Advancement of Structured Information Standards (OASIS) [34].

Tradicionalmente, o modelo de serviços Web consiste de três entidades, a saber: um fornecedor, um repositório e um consumidor. A Figura 1-1 [35] descreve o relacionamento entre estas entidades e está presente, com pequenas variações em relação aos nomes adotados, em função do contexto, em vários trabalhos [36] [37] [38] [39] [40].

Um serviço Web possui uma interface descrita em um formato que pode ser processável por programas computacionais (especificamente WSDL). Um fornecedor de

[5] Originalmente definido como Simple Object Access Protocol, mas o significado das iniciais caiu em desuso a partir da versão 1.2.

serviços disponibiliza esta interface publicando-a em um repositório. Posteriormente, esta interface pode ser descoberta e utilizada por consumidores para invocar o serviço.

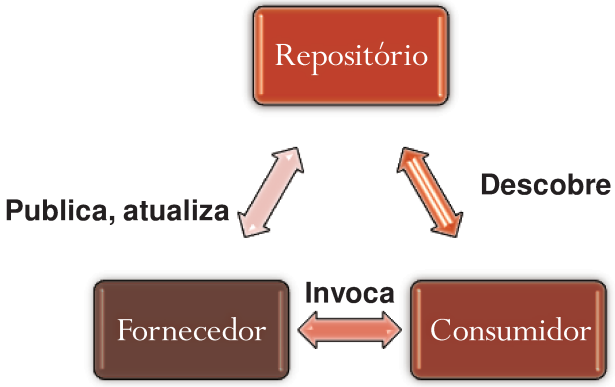

Figura 1-1 - As entidades originais do modelo de serviços Web e o relacionamento entre elas.

A descrição da interface pode ser publicada usando-se uma variedade de mecanismos distintos. A descoberta do serviço depende da sua publicação. Por meio da descrição, um programa, e até mesmo um serviço, pode automaticamente descobrir e localizar outros serviços, de acordo com requisitos e critérios de seleção. Neste processo, o consumidor (Figura 1-1) identifica os potenciais fornecedores cujas ofertas casam com seus requisitos. Qualquer mecanismo que permita que o consumidor do serviço acesse sua descrição e a torne disponível à aplicação em tempo de execução é qualificado como mecanismo de descoberta de serviço [35].

Um consumidor de serviço é o remetente de uma mensagem que solicita ao repositório um serviço Web específico. O solicitante de serviço é comparável ao cliente dentro de um modelo cliente-servidor padrão. O repositório, ou registro, é um local onde o fornecedor publica (ou armazena) suas descrições de serviços; os consumidores as descobrem e obtêm informações sobre como se ligarem ao serviço. A ligação pode ser realizada de forma manual e estática ou automática e dinâmica, aceitando várias gradações entre estes extremos [41] (ver seção 1.2). Para a ligação manual e estática, o repositório de serviços é opcional na arquitetura porque um fornecedor pode emitir a descrição diretamente aos consumidores. Do mesmo modo, consumidores podem obter

uma descrição do serviço de outras fontes além de um repositório UDDI, tal como um registro local de WSDL, um sítio File Transfer Protocol (FTP), um sítio Web etc.

Das tecnologias-padrão inicialmente propostas e acatadas como necessárias à implementação de um serviço Web, a do repositório UDDI é a mais ameaçada. Tanto o é que, em uma nova versão proposta pela W3C para a arquitetura dos serviços Web, a interação entre fornecedor e consumidor foi modificada [27]. As principais mudanças referem-se à retirada do repositório e à introdução de semântica. Várias empresas abandonaram seus projetos de repositório UDDI, entre elas a HP. A concepção de um repositório rígido, com pouca liberdade para alteração e personalização, parece ser um dos principais motivos para o seu abandono. Não menos importante é a ausência de anotações semânticas e o fato de a descrição do serviço propriamente dita ser externa ao repositório. Tais fatos contribuem para eliminar a necessidade do repositório UDDI ou tornam o seu uso improvável.

A alternativa mais promissora à utilização de repositórios ao estilo UDDI parece ser o armazenamento de descrições de interfaces em sítios, ou pastas, que são indexados e que, posteriormente, podem ser acessados por páginas Web ou por uma Interface para Programação de Aplicativos (API[6]). Neste caso, os documentos, semânticos ou não, são simplesmente salvos nesses sítios, ou pastas, para posterior utilização. Sua estrutura, portanto, é mais flexível que a de um repositório UDDI. O Swoogle, por exemplo, é um sítio de busca que faz a indexação de documentos semânticos e que pode servir de repositório de descrição de serviços [42].

1.2 Composição de Serviços

Serviços podem ser usados sob a forma de composições, nas quais dois ou mais serviços são invocados, para um fim comum, por outro serviço. A composição é a transição de um estado inferior de organização para um estado superior de organização. No estado inferior, os serviços são independentes, estão dispersos e são tratados de

[6] Sigla em inglês para Application Program Interface.

forma individualizada. Quando estes serviços passam a interagir entre si e cada um assume um papel específico e distinto, dá-se a composição. O processo de composição é, portanto, a união das partes para formar o todo.

Em outras palavras, a composição é o surgimento de serviços compostos a partir dos serviços previamente disponíveis[7], a formação de serviços complexos a partir de serviços simples, que resulta em *workflows*. A composição envolve a clara definição do escopo de atuação e a definição de sua interação externa dando às partes uma interdependência que constitui a coerência, a organização. A transformação da independência em dependência se dá a partir da definição da troca de mensagens de dados e de controle entre os serviços.

A composição é uma tarefa complexa e desafiante, que cruza muitos temas de interesse, como a modelagem, descrição, descoberta, casamento, composição e execução de serviços. Ainda, mas não de forma exaustiva, pode-se citar: sincronização, coordenação, verificação, orquestração etc. [43]. Os primeiros temas são tratados neste trabalho e chamados, por ora, apenas de sistema de composição de serviços.

Muitas abordagens sobre composição de serviços têm como objetivo último a automação, que promete o rápido desenvolvimento de uma aplicação e a reutilização segura de código, além de facilitar a interação do desenvolvedor com serviços complexos. Geralmente, a composição é descrita por uma linguagem baseada em XML, que é posteriormente executada por uma máquina de composição. Algumas destas linguagens permitem a especificação de uma composição dinâmica [44] [45] [46] [47], em que a composição é automaticamente realizada e possui a capacidade de se adaptar ao contexto no momento de sua execução. Outras linguagens, em usa maioria, permitem apenas a especificação estática de uma composição, ou seja, definida em tempo de projeto. Composições dinâmicas oferecem grandes desafios à computação, entre eles as dificuldades para identificar os serviços candidatos e definir as regras de composição [40] [41] (ver seção 1.5).

[7] No início todos os serviços são atômicos, mas depois do surgimento dos primeiros serviços compostos, novas composições poderão surgir a partir das composições anteriores.

Na perspectiva da composição de serviços, o consumidor especifica um *serviço abstrato* que precisará utilizar um *serviço concreto* criado por um fornecedor [41] [48]. Por ora, é desnecessário ater-se a estes termos, mas convém adiantar que eles serão formalmente definidos no Capítulo 3.

Segundo Klein e Köonig-Ries [36], as técnicas mais promissoras para automatizar a composição de serviços usam semântica. Para aumentar a probabilidade de criar serviços factíveis (com sentido lógico), o processo de composição necessita de semântica.

1.3 Composição de Serviços Web Semânticos

A adição de semântica às tecnologias-padrão dos serviços Web resultou no termo *serviço Web semântico* [11] [12] [13] [49] [50] [51]. A semântica é implementada através de ontologias, que consistem de definições formais e explícitas de conceitos básicos de um domínio e dos relacionamentos entre eles. As ontologias codificam o conhecimento sobre o domínio e o conhecimento que cobre mais de um domínio. Desta forma, elas fazem com que este conhecimento seja reutilizável [52].

As ontologias são utilizadas na Web Semântica para representar a semântica de documentos, explicitando o vocabulário utilizado e possibilitando um padrão para o compartilhamento de informações, de forma a permitir o uso das informações presente nestes documentos por aplicações Web e agentes inteligentes de programas.

A descrição semântica de um serviço permite que um sistema de composição possa determinar quando a composição entre dois ou mais serviços faz sentido lógico e pode ser, consequentemente, realizada. Duas variáveis distintas declaradas em uma linguagem computacional qualquer como, por exemplo, em $int\ A$ e $string\ B$, definidas como parâmetros da entrada de dois serviços independentes, podem, ao final das contas, representar o mesmo conceito (coisa física). Se as variáveis A e B são descritas por uma

ontologia, um algoritmo computacional pode calcular a similaridade semântica entre elas, isto é, pode "descobrir" se a variável A representa, ou não, o mesmo conceito[8] que B.

Muitos modelos de componentes existentes, por exemplo, JavaBeans/Enterprise JavaBeans [53], DCOM [3], CORBA Component Model [54], não especificam como representar ou modelar a semântica dos componentes. Devido à falta da informação semântica, estes modelos exigem desenvolvedores humanos de aplicações para compreender, julgar e combinar estes componentes. A representação da semântica de componentes pode permitir que máquinas de composição autônomas recuperem e combinem semanticamente os componentes. Fugii e Suda [55], por exemplo, utilizam a semântica para modelar um componente em três aspectos: o funcional, o semântico e o lógico.

A adição de informação semântica na descrição dos serviços Web potencializa as vantagens da arquitetura SOA. Em especial, permite:

- Maior reutilização dos serviços: a descrição semântica dos serviços ajuda a encontrar serviços relevantes;
- Maior interoperabilidade entre os serviços: ajuda no mapeamento dos dados trocados entre os serviços fazendo a tradução automática de mensagens entre serviços heterogêneos;
- Automação completa ou parcial do ciclo de vida: permite a configuração (análise de requisitos, descoberta, seleção e composição) e execução do processo (que trata a heterogeneidade em tempo de execução, como a heterogeneidade de dados).

Presentemente, há dois métodos para adicionar informação semântica aos serviços Web:

- Desenvolver e usar linguagens baseadas em ontologias, tal como a Semantic Markup for Web Service (OWL-S) [50]; e,

[8] Ou, de forma mais geral, pode descobrir a relação existente entre A e B.

- Adicionar informação semântica aos padrões estabelecidos para os serviços Web. Por exemplo, Semantic Annotations for WSDL (SAWSDL) [56] é uma iniciativa que visa complementar as especificações em WSDL com informações semânticas.

Não importando o método, é essencial estabelecer as relações entre os atributos dos serviços Web e as ontologias utilizadas [57]. Exemplo de atributos são entradas, saídas, pré-condições e efeitos (IOPE[9]).

1.4 A Necessidade da Composição Automática

A grande quantidade de serviços disponíveis e a sua complexidade crescente, aliadas à necessidade de desenvolvimento rápido e de adaptação aos diversos contextos, estimulam o meio acadêmico e industrial para a criação de técnicas, ferramentas, modelos, linguagens e métodos que permitam a automação do processo de composição de serviços Web. A criação de serviços novos e mais complexos a partir da descoberta, seleção e composição automáticas de serviços pré-existentes é um desafio importante no atual contexto da Tecnologia da Informação.

Em 2003, May afirmou que uma típica companhia de grande porte tem entre 30 a 50 aplicações separadas que não estão integradas [58]. A necessidade de conectar os sistemas dos consumidores e dos fornecedores agravou este problema e deixou as companhias com um grande número de sistemas incompatíveis. Assim sendo, podem-se destacar os seguintes fatores que enfatizam a necessidade e as vantagens da composição automática de serviços:

- O grande aumento da quantidade de serviços disponíveis;
- Aumento na velocidade de desenvolvimento de programas;
- A necessidade de adaptação ao contexto: somente composições automáticas com ligação em tempo de execução permitem que mudanças dos requisitos ou mudanças no ambiente interfiram automaticamente nos resultados da composição

[9] Sigla em inglês para Inputs, Outputs, Preconditions and Eeffects.

sem a obrigatoriedade da refatoração das regras de negócios, ou no caso, de refatoração do serviço abstrato. Por exemplo, o requisito *encontrar o serviço mais barato* pode ser alterado para *encontrar o serviço mais rápido*, sem a necessidade de se alterar o serviço abstrato. As mudanças no ambiente podem incluir a inserção de novos serviços ou remoção de antigos, por exemplo;

- Aumentar a tolerância a falhas: serviços com falhas podem ser automaticamente substituídos por outros;
- Permitir a mobilidade dos serviços: a mudança de endereço de um serviço (URL) não deve afetar o funcionamento do sistema; na verdade, esta mudança deve ser transparente, isto é, um serviço deve ser identificado independentemente de sua localização;
- Aumentar a concorrência entre os fornecedores de serviços: novos serviços podem ser criados; serviços velhos podem ser apagados; e serviços podem alterar suas próprias descrições sem que haja notificação prévia. Parâmetros de QoS que variam em função do tempo podem ser verificados no momento da composição. Em um ambiente dinâmico, a seleção de um serviço é uma função dependente do tempo: os mesmos critérios de seleção executados em momentos diferentes podem resultar em diferentes escolhas;
- Fornecer valor agregado: serviços compostos podem adicionar valor aos serviços atômicos já existentes.

Os problemas característicos da interação entre os serviços são enunciados por Weiss et al. [59]: "*a primeira geração de serviços Web se baseava em duas suposições: (a) que os serviços desenvolvidos isoladamente ou seriam usados isoladamente, ou, se parte de um serviço composto, não iriam se interagir de forma inesperada; e (b) que os usuários tinham completo controle sobre os serviços usados, ou haveria um entendimento comum das operações e dos efeitos colaterais desses serviços. Argumenta-se que estas suposições não são mais válidas para os serviços Web atuais*"[10]. O crescente uso da composição de serviços Web tende a aumentar a gravidade dos problemas relacionados a estas

[10] Livre tradução realizada pelo autor.

suposições. Realizar a composição de forma a não permitir, ou pelo menos tentar minimizar, as interações inesperadas é um dos desafios da composição automática de serviços.

1.5 Composição x Casamento

Utilizado por áreas diversas, o termo "composição" é empregado para designar diferentes estratégias. O mesmo acontece com o emprego dos adjetivos "automático" e "dinâmico" comumente utilizados para qualificá-la. Para estabelecer uma compreensão mais precisa, será utilizada a classificação adotada por Fluegge et al. [41]. Resumidamente, as estratégias de composição são classificadas em dois eixos: criação do serviço abstrato e momento de ligação[11]. A criação do serviço abstrato refere-se à criação de um *workflow* de atividades abstratas, isto é, atividades que não possuem implementação e que, portanto, precisam ser ligadas a serviços concretos para que possam ser executadas. A criação do serviço abstrato pode ser realizada de três formas distintas: manual, semiautomática ou automática. A criação automática é aquela que se contrapõe à criação manual, ou seja, é realizada sem a intervenção humana. Entre estes dois pontos antagônicos, encontra-se a criação semiautomática, na qual homem e programa, revezam-se na tarefa de criar a composição. Por sua vez, a ligação entre os serviços abstrato e concreto pode ocorrer em dois momentos distintos: em tempo de projeto ou em tempo de execução[12].

Estes dois eixos são ortogonais e permitem agrupar os sistemas de composição de serviços em várias categorias, como: criação manual com ligação em tempo de projeto, criação manual com ligação em tempo de execução, criação semiautomática com ligação em tempo de execução e assim por diante. Convém salientar que a fronteira que separa estas classificações não é de forma alguma bem definida, mas sim difusa, de tal forma a permitir que alguns sistemas se situem na fronteira e possam estar ora de um lado, ora de outro.

[11] No original: plan creation e service binding, respectivamente.
[12] No original: early and late binding, respectivamente.

As técnicas baseadas em geração manual do *workflow* pressupõem a existência de um *workflow* inicial (neste caso, um serviço abstrato agindo como consumidor) cujas atividades definem os requisitos (vistos como uma consulta) que precisam ser satisfeitos por serviços concretos, reais. Os requisitos podem ser tanto funcionais quanto não funcionais. A composição, neste caso, significa buscar e selecionar para cada atividade abstrata definida no *workflow* um serviço concreto capaz de realizá-la com sucesso. O método falha caso o serviço concreto não exista, ou não seja encontrado, o que, na prática, resulta no mesmo.

Por sua vez, a geração automática de *workflow* é usada quando o consumidor, ou serviço abstrato, não é representado por um modelo de processo. Consequentemente, o objetivo é gerá-lo a partir dos serviços disponíveis. O resultado final é um novo serviço, entendido como um *workflow*, onde cada atividade representa um serviço concreto.

Presentemente, a composição é tema de interesse, principalmente, dos especialistas de duas grandes áreas: *workflow* e planejamento em Inteligência Artificial (IA). A abordagem adotada pela área de *workflow* baseia-se na geração manual do *workflow* com ligação em tempo de execução enquanto a área de planejamento em IA adota a estratégia de geração automática do *workflow*, podendo a ligação ser realizada tanto em tempo de projeto quanto de execução.

Esta divergência de abordagem adotada pelas áreas de *workflow* e planejamento em IA também é corroborada por Rao [60], Rao e Su [61], Tsetsos et al. [62] e Klusch [63]. Adicionalmente, esta divergência resulta na polissemia do termo “composição”, designando não somente a composição propriamente dita, mas, às vezes, *somente* a busca e seleção de serviços. Por exemplo, Zhang [47] nomeia as técnicas utilizadas por Benatallah et al. [64] e Narayanan e McIlraith [65] como composição adaptativa e interativa. Estas técnicas são baseadas em *templates* que, na verdade, atuam como um *workflow* de atividades abstratas.

Por isto, para diferenciar uma abordagem da outra, Alamri et al. [39] designam a primeira por composição estática e a segunda por composição dinâmica. Já Tsetsos et al.

[62], por casamento direto e casamento indireto. Entretanto, a maioria adota os termos “casamento” e “composição”, respectivamente.

Apesar de a composição, entendida a partir da abordagem adotada pela área de planejamento em IA, oferecer maiores desafios à tecnologia atual que o casamento, ambos possuem vantagens e desvantagens (Tabela 1-1). Um dos objetivos deste trabalho é o desenvolvimento de um modelo que composição que permita aglutinar os valores de cada uma destas áreas.

Algumas vantagens são compartilhadas por ambas as abordagens, por exemplo, a mobilidade dos serviços, a adaptação dinâmica ao contexto e o fornecimento de valor agregado. Outras estão mais associadas a apenas uma delas, como o aumento da reutilização de componentes que estão mais associadas à abordagem de IA. Isto ocorre porque um novo serviço é criado a partir de código já previamente disponível. Isto também ocorre com a abordagem de *workflow*, entretanto, neste caso, o desenvolvedor deve conhecer a priori quais são os recursos disponíveis para que o *workflow* criado possa usufruir de código já existente. Portanto, em um caso a reutilização fica a cargo do desenvolvedor e em outro, de um programa.

O aumento a tolerância a falhas está intimamente ligado ao aumento da concorrência entre os fornecedores, pois vários fornecedores diferentes podem oferecer o mesmo serviço: na falha de um, o concorrente assume. Esta característica pode ser observada em ambas as abordagens. Entretanto, na abordagem de IA, cada nova execução poderá gerar um fluxo diferente, resultante de mudanças diversas no ambiente, e não apenas da falha de um serviço. Portanto, se o objetivo é somente o aumento na tolerância a falhas, a abordagem de *workflow* é mais simples e mais confiável: o fluxo permanecerá o mesmo, alterando-se somente o fornecedor do serviço.

O desacoplamento entre a lógica de negócio, representada por um serviço abstrato, e a sua execução, efetivada por um serviço concreto, permite que um sistema de composição automaticamente se adapte às mudanças do ambiente. Desta forma, serviços concretos podem ser removidos, inseridos ou modificados sem a necessidade de refatorar

o serviço abstrato. Como a abordagem de IA a lógica é automaticamente criada, o desacoplamento só existe, ou só faz sentido, para a abordagem de *workflow*.

Tabela 1-1 – Comparação entre as abordagens das áreas de planejamento em IA e *workflow* para a composição de serviços.

<table>
<tr><th></th><th>Abordagem segundo a área de planejamento em IA</th><th>Abordagem segundo a Área de workflow</th></tr>
<tr><td rowspan="2">Vantagens</td><td>• Aumenta a velocidade do desenvolvimento de programas
• Aumenta a reutilização de componentes de programas
• Criação automática do fluxo</td><td>• Aumenta a tolerância a falhas
• Aumenta a concorrência entre os fornecedores de serviços
• Controle total sobre a criação do fluxo
• Desacoplamento entre a lógica do negócio e a sua execução</td></tr>
<tr><td colspan="2">• Permite a mobilidade dos serviços
• Permite a adaptação dinâmica ao contexto
• Fornecer valor agregado</td></tr>
<tr><td>Desvantagens</td><td>• Oferece pouco controle sobre a criação do fluxo
• Utiliza técnicas mais complexas
• Necessidade de verificar o sentido lógico[13] do fluxo</td><td>• Inflexibilidade do fluxo
• Necessidade de descrição manual completa do fluxo
• Necessidade de conhecimento dos recursos (serviços) disponíveis</td></tr>
</table>

A abordagem adotada pela área de *workflow* apresenta desvantagens, entre elas: inflexibilidade do fluxo, porque ele é manualmente definido em tempo de projeto; necessidade de descrição manual completa da lógica de negócio; e necessidade de conhecimento prévio dos recursos (serviços) disponíveis. Com o objetivo de avaliar o real alcance desta abordagem, Tizzo et al. [66] [67] implementaram um *workflow* usando a linguagem Web Services Business Process Execution Language (WS-BPEL) em que as ligações entre as atividades de um *workflow* e os serviços concretos são realizadas em tempo de execução, segundo os requisitos do usuário. Estes requisitos são especificados na linguagem SPARQL [68], uma linguagem de consulta para o Resource Description Framework (RDF) [69]. A escolha é baseada em requisitos não funcionais, descritos em RDF. As limitações encontradas apontam para a necessidade de criação automática do fluxo.

[13] Em inglês: soundness

Por seu turno, a complexidade e a dificuldade de verificar se a composição realizada tem sentido lógico são desvantagens das técnicas de composição ligadas à área de IA. Isto se deve à complexidade de se obter uma descrição precisa, inequívoca e não ambígua da interação entre os serviços. Esta complexidade também pode ser verificada nas técnicas ligadas à área de *workflow*, mas ficam mais acentuadas nas técnicas ligadas à área de planejamento em IA. Os problemas da composição automática de serviços são apresentados na seção 1.5.

Alguns sistemas de composição baseados na abordagem da área de *workflow* são: EFlow [70][71], Composite Service Definition Language (CSDL) [72] e Polymorphic Process Model (PPM) [73]. Entre as técnicas baseadas em IA, têm-se: cálculo de situações[14] [74], Planning Domain Definition Language (PDDL) [75], Rule-based Planning [76] e Hierarchical Task Network (HTN) [75] [77]. Ainda, Tang e Ai usam algoritmos genéticos para realizar a composição com restrições de Qualidade de Serviço (QoS[15]) [78]; e Tizzo et al. [48] uniram os algoritmos de times assíncronos com operadores genéticos com o objetivo de melhorar o desempenho das técnicas atuais. Estes e outros trabalhos serão apresentados no Capítulo 2.

Por ora, utilizar-se-á indistintamente o termo composição para designar ambas as abordagens. Quando se fizer necessário a diferenciação entre elas, será acrescido ao termo uma menção à área. Os termos casamento e composição serão diferenciados a partir do Capítulo 2.

1.6 Os Problemas da Composição Automática

Segundo Berardi et al. [40], que detalham os problemas apresentados por Weiss et al. [59], as maiores dificuldades da composição estão relacionados com a identificação dos serviços candidatos, com a definição das regras de composição e de seu significado lógico

[14] Em inglês: Situation Calculus.
[15] Sigla em inglês para Quality of Service.

e semântico e também com a verificação de quão próxima uma composição está da requisição.

Quando um serviço, composto ou não, é perfeitamente compatível com uma requisição, a sua escolha é trivial, embora nem sempre o processo para se chegar a tal conclusão o seja. Entretanto, situação mais complexa pode ocorrer quando os serviços especificados pelo fornecedor (serviços abstratos) e os serviços oferecidos ao consumidor (serviços concretos) são apenas parcialmente compatíveis. Neste caso, deve-se recorrer a métricas de comparação que estabeleçam um grau de similaridade entre os serviços. Cabe à aplicação decidir se tal similaridade é adequada ou não. Estabelecer a métrica e qual o grau de similaridade satisfatório ainda são objetos de pesquisa.

Parte da interoperabilidade entre os serviços é alcançada devido à capacidade de se autodescreverem independentemente da plataforma na qual se encontram. A descrição deve acomodar tanto a definição dos atributos funcionais (por exemplo, os atributos IOPE) quanto dos não funcionais (qualidade de serviço, custo e localização, por exemplo). Entretanto, a descrição dos atributos não funcionais não faz parte dos padrões estabelecidos para os serviços Web, fato que tem causando o surgimento de propostas independentes. Ainda, as descrições que pretendem ser completas envolvendo todos os atributos funcionais e não funcionais possíveis de um serviço esbarram no seguinte paradoxo: o completo será complexo demais para ser tratável e o simples será simples demais para ter algum valor prático. A maioria dos trabalhos em composições foca em um ou outro aspecto dos serviços. Os trabalhos que abarcam vários aspectos são denominados de composições multinível, sendo um exemplo encontrado em Medjahed et al. [76].

Adicionalmente à falta de informação semântica, as linguagens disponíveis, comercialmente ou não, não possuem mecanismos que permitam, de forma simples, especificar como se dá a composição. Por exemplo: permitir que o consumidor defina as técnicas de busca e seleção, a técnica de composição e o espaço de busca. Como várias técnicas já foram propostas seria interessante que a linguagem de composição permitisse

escolher uma delas. Controlar o espaço de busca significa, indiretamente, ter controle sobre a escolha dos fornecedores concorrentes e, consequentemente, determinar o grau de confiabilidade dos serviços envolvidos. Atualmente, o espaço de busca e as técnicas utilizadas são características intrínsecas dos sistemas de composição, havendo forte dependência entre a linguagem de composição e a máquina de composição. Isto faz com que as linguagens não sejam portáveis. Uma avaliação das tecnologias atualmente disponíveis para a realização da composição pode ser encontrada em Tizzo et al. [66]. As características fundamentais dos sistemas de composição são abordadas no Capítulo 2.

Finalmente, a comparação de desempenho entre os vários métodos de composição disponíveis é difícil de ser realizada pelos seguintes motivos:

- Falta de uma base de dados de teste comum;
- Falta de avaliação de desempenho em muitos projetos: geralmente o objetivo é mostrar a possibilidade de composição segundo um método qualquer, ou seja, há somente uma avaliação qualitativa do método;
- Grande dependência entre a linguagem utilizada para descrever o serviço e os métodos composição;
- Divergência entre os conceitos de composição: como citado anteriormente existem duas abordagens principais para a composição automática de serviços: a primeira baseada na geração manual do *workflow*; e, a segunda, baseada na sua geração automática;
- Divergência entre os tipos de fluxo possíveis de serem realizados: segundo a Workflow Management Coalition (WfMC) [79] e a Workflow Patterns Initiative [80] existem diversos tipos de fluxo de dados e de controle, entre eles: sequencial, separação paralela, sincronização, escolha-exclusiva e união simples. A maioria dos trabalhos aborda apenas a composição sequencial.

1.7 Objetivos

A partir do contexto e da problemática apresentados, este trabalho, que se concentra na área de composição automática de serviços Web semânticos, possui os seguintes objetivos principais:

- Descrever regras de composição para os tipos de fluxo sequencial, separação paralela, sincronização, escolha-exclusiva, união simples e cíclico a partir da descrição semântica dos parâmetros de entrada e saída dos serviços Web;
- Mostrar que um método de composição que combine as técnicas de *workflow* e de IA apresenta vantagens sobre cada uma isoladamente;
- Mostrar que algoritmos de times assíncronos (A-Teams), quando utilizados em conjunto com operadores genéticos, são eficientes no trato de composição automática de serviços Web semânticos.

Objetivo específico:

- Realizar uma avaliação de desempenho quantitativa para permitir que diferentes algoritmos de composição possam ser comparados entre si.

1.8 Hipótese e Justificativa

Os objetivos terão sido alcançados se as seguintes hipóteses forem verdadeiras:

- É possível descrever regras para a composição de serviços Web semânticos que contemplem os seis tipos de fluxo considerados (sequencial, separação paralela, sincronização, escolha-exclusiva, união simples e cíclico) usando somente os atributos IOPE descritos em arquivos OWL-S;
- Um serviço abstrato pode ser decomposto em atividades abstratas, com a devida descrição do fluxo de dados e de controle entre elas, de tal forma que um serviço abstrato seja, na verdade, um *workflow* de atividades abstratas. Se cada atividade puder se expandir em um novo *workflow,* haverá a combinação das técnicas de

composição referentes às áreas de *workflow* e de planejamento em IA que apresentará vantagens sobre cada uma delas tomadas isoladamente;

- Algoritmos de Times Assíncronos em conjunto com operadores genéticos são adequados ao problema da composição automática de serviços Web semânticos.

Como justificativas às escolhas destas hipóteses, vale citar:

- A linguagem OWL-S[16] é uma das mais populares para a descrição semântica de serviços Web;
- Segundo a Workflow Patterns Initiative [80], dos seis tipos de fluxo escolhidos (sequencial, separação paralela, sincronização, escolha-exclusiva, união simples e cíclico), os cinco primeiros são fluxos de controle básicos, significando, portanto, que os aspectos elementares de fluxo de controle serão contemplados;
- Tomadas isoladamente, a composição de cada atividade comportar-se-á como a composição defendida pela área de *workflow*. Entretanto, se cada atividade puder se expandir em outra composição, as abordagens das duas áreas, *workflow* e planejamento em IA, poderão ser unidas e o resultado terá maior sinergia do que cada uma tomada isoladamente. Isto ocorrerá porque a definição de um serviço abstrato como um *workflow* de operações abstratas irá oferecer, ao desenvolvedor, certo controle sobre o processo de composição e poderá influenciar positivamente o resultado final;
- Tanto A-Teams quanto Algoritmos Genéticos (AG) podem ser considerados métodos fracos: são concebidos para resolver problemas genéricos em mundos genéricos, sendo estes mundos não lineares e não estacionários. A combinação de A-Teams com AG pode ser aplicada a diversos problemas que envolvem busca heurística; em particular, a problemas de planejamento e escalonamento. São especialmente bem sucedidos em fornecer soluções próximas do ótimo para vários problemas NP-completos [81]. A solução para a composição de serviços Web

[16] Embora seja uma ontologia, OWL-S é também frequentemente tratada como linguagem: "Por vezes, nos referimos à ontologia OWL-S como uma linguagem para descrever serviços, refletindo o fato de que ela proporciona um vocabulário padrão que pode ser usado em conjunto com os outros aspectos da linguagem de descrição de OWL para criar descrições de serviço" [50] (tradução livre feita pelo autor).

semânticos deve ser projetada para conseguir uma solução viável (composição com sentido lógico) e ótima (menor quantidade possível de serviços e similaridade semântica exata), ou pelo menos o mais próximo desta possível, satisfazendo o maior número possível de requisitos e restrições do sistema;

- A grande quantidade de serviços disponíveis multiplicado pela quantidade de parâmetros de entrada e saída de cada serviço eleva sensivelmente a quantidade de combinações necessárias para se encontrar uma solução ótima e, possivelmente, caracteriza o problema como NP-completo.

1.9 Método

Para se chegar aos objetivos propostos, o método escolhido foi:

- Estabelecer um sistema de referência para a composição de serviços Web semânticos;
- Definir os graus de similaridade entre dois conceitos quaisquer;
- Definir as regras de casamento entre serviços abstratos e concretos;
- Propor as regras de composição para os padrões de fluxo especificados;
- Propor as regras para a criação de um serviço composto;
- Desenvolver um protótipo baseado em times assíncronos e operadores genéticos para realizar a composição automática de serviços Web semânticos a partir das regras anteriormente definidas;
- Avaliar o desempenho do protótipo em composições de serviços Web semânticos.

1.10 A Estrutura da Tese

A tese está dividida em seis capítulos. O próximo capítulo faz uma abordagem dos conceitos fundamentais envolvidos na composição de serviços. Primeiramente, apresenta-se um sistema de referência para que os principais conceitos possam ser compreendidos a partir de um único enfoque e que servirá de base para os demais capítulos. Posteriormente, são apresentadas as capacidades e limitações de algumas linguagens e

modelos para a especificação e execução de serviços. Em seguida, os principais métodos de casamento e composição são expostos. O capítulo é finalizado com uma discussão sobre as características que um sistema de composição deve possuir.

No capítulo três são desenvolvidas as regras para a composição dos seguintes fluxos de controle: sequencial, separação paralela, sincronização, escolha-exclusiva, união simples e cíclico. Exemplos que ilustram o uso e a potencialidade das regras são apresentados no final do capítulo.

O quarto capítulo descreve o protótipo Esecsy (acrônimo para Evolutionary Service Composition System) implementado usando algoritmo de times assíncronos e operadores genéticos que realiza a composição automática de serviços Web semânticos. A composição é realizada obedecendo às regras definidas no Capítulo 3.

O quinto capítulo apresenta a avaliação de desempenho do Esecsy para alguns cenários típicos.

O último capítulo finaliza este trabalho com as conclusões, faz um resumo e uma análise crítica das contribuições e indica os possíveis trabalhos futuros.

Capítulo 2

Uma Abordagem para a Composição Automática de Serviços Web Semânticos

...dados indutivos caem em cima de nós de todos os lados, como a lava do Vesúvio; ficamos sufocados por fatos descoordenados; nossa mente fica esmagada pelas ciências que geram e se multiplicam num caos especialista por falta de pensamento sintético e de uma filosofia unificadora. Somos, todos, meros fragmentos daquilo que um homem poderia ser.

Durant, Will. A história da filosofia. Editora Nova Cultural, coleção Os Pensadores, Tradução Luiz Carlos do Nascimento Silva, ed. 2000, p 103.

A composição automática de serviços Web semânticos, apesar de avanços, ainda se confunde em algumas antinomias, haja vista as diferentes abordagens adotadas por áreas distintas (ver seção 1.2). Isto também pode ser concluído ao se deparar com a

grande quantidade de linguagens e métodos utilizados para descrever, modelar, compor e executar serviços.

A dificuldade de comparação entre as várias técnicas levou o Stanford Logic Group a desenvolver um sítio com o objetivo de permitir uma compreensão comum das várias tecnologias [82]. Além disto, o grupo pretende também atuar como um agente facilitador no desenvolvimento de novas técnicas que permitam a automação da mediação, da coreografia e da descoberta de serviços Web usando anotações semânticas. Adicionalmente, pretende-se também fazer um levantamento das partes do problema que ainda não foram cobertas.

O presente capítulo não tem a intenção de sintetizar um método de comparação entre as diferentes técnicas. Pretende apenas identificar um sistema de referência para a composição automática de serviços Web semânticos cuja missão é alinhavar os conceitos que são usados neste trabalho. Outros métodos de classificação poderão ser encontrados em Dustdar e Schreiner [38] e Milanovic e Miroslaw [83].

Este capítulo fornecerá uma visão panorâmica de algumas linguagens e modelos utilizados para especificar, e, em alguns casos, executar serviços. Algumas destas linguagens não são semânticas, a exemplo da linguagem de descrição de serviços WSDL. Mas como WSDL está na origem de várias propostas semânticas, esta linguagem também será abordada. Segundo Rajasekaran et al. [84], a forma como os serviços são descritos e descobertos tem impacto direto na composição.

O capítulo se concentra nos processos descritos no sistema de referência: busca, casamento e composição. Os conceitos fundamentais sobre o casamento baseado em lógica, em técnicas de recuperação de informação, em grafos e em técnicas híbridas são apresentados. Posteriormente, é realizada uma breve descrição de algumas técnicas representativas de composição de serviços.

2.1 Sistema de Referência para a Composição Automática de Serviços Web Semânticos

As várias propostas de arquiteturas, modelos, *frameworks*, *middlewares*, sistemas etc. já realizadas na área de composição automática de serviços Web fez brotar uma diversidade de termos técnicos na superfície que, muitas vezes, impede que a atenção caminhe em direção ao núcleo, à essência. Como reflexo, alguns esforços foram despendidos com o intuito de unificar os termos e classificar os métodos de composição. Entretanto, um consenso ainda não foi atingido. Mesmo assim, é possível, e necessário, encontrar pontos em comum entre as diversas abordagens.

Dustdar et al. [38] identificaram que os principais elementos de um *middleware* para a composição de serviços Web são: (a) uma linguagem e um modelo de composição para especificar os serviços envolvidos em uma composição; (b) um ambiente de composição gráfico de serviços Web; e (c) um ambiente para a execução da composição.

Mais centrado no problema da composição automática, Medjahed et al. [76] apresentam uma técnica dividida em quatro fases:

a. Fase de especificação: nesta fase é feita a especificação da composição. Para tal Medjahed utiliza a linguagem Composite Service Specification Language (CSSL);
b. Fase de casamento: esta fase usa regras de composição para gerar planos em conformidade com as especificações do requerente;
c. Fase da seleção: se mais de um plano for gerado, na fase da seleção, o requerente seleciona um plano baseado em parâmetros de qualidade da composição (custo, por exemplo);
d. Fase de geração: a descrição do serviço composto é automaticamente gerada e apresentada ao requerente.

Numa abordagem mais geral que a apresentada por Medjahed, Fluegge et al. [41] propuseram um *framework* em quatro etapas para o ciclo de vida de um serviço composto:

a. Criação de um modelo de processo especificando o fluxo de dados e de controle entre suas atividades;
b. Descoberta dos serviços que casam com as atividades, de acordo com os critérios estabelecidos, geralmente, utilizando-se de um *broker*;
c. Disponibilização do novo serviço para potenciais clientes. Novamente, o *broker* é utilizado, mas agora como meio de publicação e ponto físico de acesso ao serviço;
d. Invocação do serviço composto por uma máquina de execução de processos para coordenar o fluxo de controle e de dados.

Outra proposta de *framework* pode ser encontrada em Rao [60] e em Rao e Su [61]. Todavia, apesar da generalidade e da abrangência de todas estas propostas, nenhuma delas retrata com fidelidade a composição automática de serviços Web semânticos. Particularmente, isso ocorre devido à identificação de dois processos distintos, casamento e composição, que devem se unir em síntese para formar um único sistema. Devido a estas particularidades, um sistema de referência para a composição automática de serviços Web semânticos é apresentado na Figura 2-1 [67]. O restante deste capítulo presta-se ao desenvolvimento conceitual desse sistema e de como outros trabalhos a ele se relacionam.

O sistema de referência é composto por consumidores e por produtores de serviços, que tanto podem ser humanos quanto programas. Os fornecedores alimentam os repositórios de serviços (Figura 2-1 passo 1) e de descrição de serviços (passo 2).

Pelo lado do consumidor, a ação se inicia com a especificação dos requisitos de seleção, que podem ser descritos em linguagem formal ou natural (passo 3). Neste último caso, é realizado um pré-processamento para convertê-la em uma representação formal (passo 4), a exemplo do Semantics-Based Dynamic Web Service Composition que utiliza o Semantic Graph based Service Composition (SeGSeC) para converter uma especificação feita em linguagem natural (especificação externa) em um serviço abstrato (também chamado de especificação formal ou interna) [55].

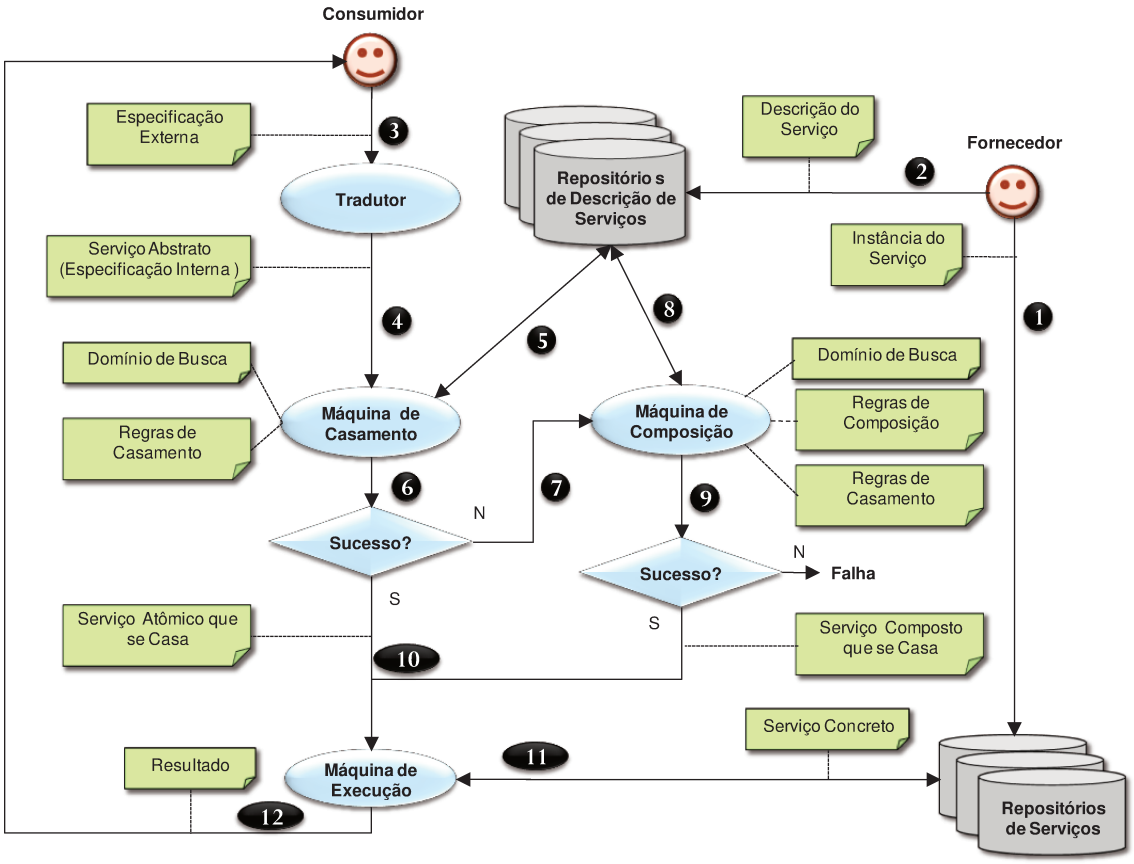

Figura 2-1 – Sistema de referência para a composição automática de serviços Web semânticos.

As especificações, tanto interna quanto externa, de um serviço podem ser muito diversas de um trabalho para outro. Entretanto, são comuns a todos os trabalhos, pelo menos, o uso dos dados de entrada e saída, isto é, parte dos atributos IOPE. Alguns autores utilizam também com parâmetros de Qualidade de Serviço (QoS) e com a descrição de eventos, objetivos, recursos, meta-informações (autor, localização etc.) entre outros. De qualquer forma, nos trabalhos mais recentes, é comum o uso de ontologias para sua descrição. Em *Towards a Precise, Understanding of Service Properties*, os autores identificam muitas das propriedades que um serviço pode ter [85]. A descrição dos requisitos funcionais e não funcionais de um serviço também pode ser encontrada em *Modeling Web Services by Iterative Reformulation of Functional and Non-Functional Requirements* [86].

A máquina de casamento faz uma varredura nos repositórios especificados pelo domínio de busca (passo 5). Na verdade, o domínio de busca pode especificar uma lista de serviços, uma lista de repositórios ou até mesmo uma lista de máquinas de busca [66]. Dirigida pelas especificações do serviço abstrato, a máquina de casamento classifica os serviços concretos encontrados. Se um ou mais for semelhante ao serviço abstrato (passo 6), aquele com maior grau de semelhança, de acordo com as regras de casamento, pode ser enviado para a máquina de execução (passo 10), que acessa o repositório de serviços (passo 11), e o processo é, portanto, finalizado com sucesso, retornado o resultado para o consumidor (passo 12)[17]. Do ponto de vista das técnicas aplicadas pela área de *workflow*, um sistema de composição, finalizado com sucesso ou com falha, se encerra por aqui. Se tomado a rigor, pode-se argumentar que "somente" o casamento foi realizado[18]. E, realmente, doravante este processo será designado por casamento.

A composição propriamente dita entra em cena quando o casamento falha (passo 7). Guiada pelas regras de composição, a máquina de composição empenha-se em construir uma composição a partir da descrição dos serviços existentes nos repositórios (passo 8), de tal forma que ela satisfaça aos requisitos do serviço abstrato. É possível encontrar mais de uma composição. Neste caso, faz-se necessário classificá-las de acordo com o grau de semelhança que apresentam em relação ao serviço abstrato. O serviço composto com maior grau de casamento é selecionado e enviado para execução (passo 10)[19].

2.2 Linguagens e Modelos para a Especificação de Serviços

A interoperabilidade entre serviços é parcialmente alcançada através do uso de protocolos comuns. Entretanto, isto não é suficiente. É necessário conhecer quais são os

[17] Os passos 10 e 11 não precisam, necessariamente, ser executados. O consumidor pode estar simplesmente verificando a existência de um serviço. Neste caso, o resultado retornado para o consumidor é a composição propriamente dita e não o resultado da execução da composição (passo 12).

[18] " 'Somente' o casamento", neste caso, quer enfatizar que a composição não foi realizada. Não tem por intenção incluir os passos 10 e 11 (referentes à execução do serviço), que não fazem parte do processo de casamento.

[19] Vale salientar que casamento e composição são processos distintos, mas vinculados, pois a máquina de composição deve embutir ou usar uma máquina de casamento.

dados de entrada e quais são os de saída, os tipos destes dados, a localização do serviço e tipo de codificação utilizada durante a comunicação. É necessário, portanto, um metaprotocolo para descrever estes parâmetros de forma independente de plataforma e que seja aberto. Com este intuito foram criadas as primeiras linguagens de descrição de serviços Web. Entretanto, com o passar do tempo, verificou-se a necessidade de estender estas linguagens para incorporar outros aspectos, semântica, por exemplo.

Assim, algumas linguagens orientadas à tecnologia de serviços surgiram, tais como: Web Service Description Language (WSDL) [31] [32]; Semantic Annotations for WSDL (SAWSDL) [57] [56] [87]; Web Service Semantics (WSDL-S) [88] [89]; DARPA Agent Markup Language for Service DAML-S [50]; Semantic Markup for Web Service (OWL-S) [50]; Semantic Web Services Ontology (SWSO) [90]; Web Service Modeling Ontology (WSMO) e Web Service Modeling Language (WSML) [91] [92]; Semantic Web Service Framework (SWSF) [93]; Composite Service Specification Language (CSSL) [76]; OCML Modelling Language [94] usada no framework Internet Reasoning Service (IRS-II) [95] e Web Service Execution Specification Language (WSESL) do FUSION [46]. Cabral et al. [96] fazem uma comparação entre as abordagens IRS-II, OWL-S e WSMF.

A Web Services Business Process Execution Language (WS-BPEL ou, simplesmente, BPEL) [97] é uma linguagem de composição focada na lógica de negócio que surgiu a partir da dificuldade encontrada para compor serviços utilizando linguagens convencionais de programação, focadas em API. Para Bolie et al. XML, SOAP, serviços Web e BPEL são os artefatos básicos de qualquer aplicação SOA [98].

Além das linguagens de descrição de serviços propriamente ditos (interface, protocolo de acesso, endereço físico etc.), de fluxo de dados e de controle, há também linguagens para descrever metadados, QoS, coreografia e há, inclusive, linguagens para modelar serviços. Ainda é possível encontrar trabalhos que utilizam linguagens originalmente criadas para outros fins. Por exemplo, Benatallan et al. utilizam os gráficos de estados da linguagem UML para modelar composições de serviços Web [64].

Toda esta diversidade aponta para a necessidade de expansão dos padrões estabelecidos dos serviços Web [27]. De qualquer forma, uma infraestrutura universal não precisa, necessariamente, ser estabelecida. A integração entre serviços pode ser alcançada através do mapeamento entre as diferentes tecnologias já existentes ou que porventura possam vir a coexistir. A especificação MetaObject Facility (MOF) da Object Management Group (OMG) [99], que é parte integrante da Model Driven Architecture (MDA) [1] [100], pode ser utilizada como tecnologia de mapeamento e transformação entre diferentes modelos tecnológicos.

Semantic Markup for Web Services (OWL-S)

A avaliação experimental do protótipo construído neste trabalho utiliza uma coleção de serviços descritos na linguagem OWL-S [50]. OWL-S é uma ontologia de serviços Web baseada em OWL [101] [102]. OWL faz parte da "visão de Web semântica", que preconiza a definição exata da informação na Web e da relação entre informações. Portanto, foi projetada para promover um meio comum de processar a informação na Web por aplicativos computacionais. A linguagem OWL utiliza a notação XML e permite, portanto, que documentos expressos em OWL sejam facilmente trocados entre tipos diferentes de computadores usando tipos diferentes de sistema e de linguagens computacionais. As primeiras versões da linguagem são conhecidas por DAML-S [50].

A OWL foi aprovada como uma recomendação da W3C em fevereiro 2004, sendo compreendida, portanto, como um padrão da Web. A linguagem OWL é dividida em três sublinguagens:

- OWL Lite: suporta basicamente a descrição de uma hierarquia de classificação e restrições simples. Exclui classes enumeradas e cardinalidade arbitrária. É fácil de entender e de usar, permitindo a criação de ferramentas computacionais;
- OWL DL: inclui a OWL Lite. Suporta o máximo de expressividade que se pode conseguir sem perder a completude e decidibilidade computacional. O seu poder de expressão corresponde à lógica de descrição e possui as seguintes restrições: (a) uma classe não pode ser ao mesmo tempo um indivíduo ou um tipo; (b) uma

propriedade não pode ser ao mesmo tempo um indivíduo ou uma classe. Possibilita o raciocínio eficiente e não há compatibilidade total com o Resource Description Framework (RDF);

- OWL Full: inclui a linguagem OWL DL. Possibilita o máximo de expressividade e a liberdade sintática do RDF. Uma classe pode ser tratada simultaneamente como uma coleção de indivíduos ou, simplesmente, como um indivíduo. OWL Full é tão poderosa que é indecidível, portanto, não há suporte completo (ou eficiente) ao raciocínio.

Os subconjuntos OWL Lite e OWL DL são notações variantes às lógicas descritivas conhecidas como SHIF(D) e SHION(D), respectivamente [103].

A ontologia OWL-S define um conjunto de marcadores para a descrição de serviços Web. Adicionalmente, propõe uma estrutura para facilitar a definição de serviços Web. Nesta estrutura, cada serviço tem três conjuntos de informação: *ServiceProfile*, *ServiceModel* e *ServiceGrounding*. Estes três conjuntos são conectados pelas propriedades da classe *Service*, que fornece um ponto de referência à declaração de um serviço Web (Figura 2-2):

- O *ServiceProfile* descreve as propriedades e as capacidades semânticas de um serviço, em especial, descreve um serviço em função de três tipos básicos de informação: a organização que oferece o serviço, a função que o serviço executa, e um conjunto de atributos que especifica as características do serviço;
- O *ServiceModel* especifica o modo como o cliente interage com o serviço. Para detalhar esta interação existe uma classe, *Process*, definida como uma subclasse da classe *ServiceModel*;
- O *ServiceGrounding* fornece as informações para fazer a ligação física com o serviço em tempo de execução, possui uma referência ao arquivo WSDL do serviço. Entre outras coisas, define o protocolo de comunicação, o formato das mensagens, o número das portas de comunicação e as técnicas de serialização empregadas.

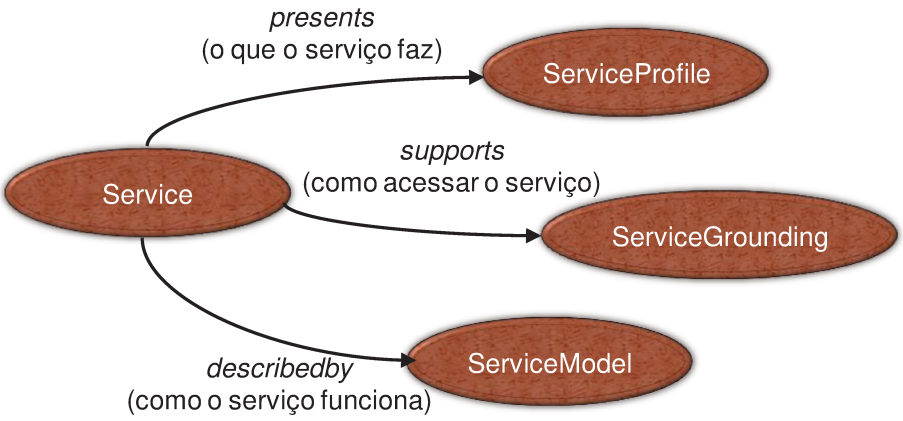

Figura 2-2 – A estrutura da ontologia OWL-S.

A classe *Process*, subclasse da classe *ServiceModel*, possui três subclasses que especificam os três tipos de processos: *AtomicProcess* (Processo Atômico), *SimpleProcess* (Processo Simples) e *CompositeProcess* (Processo Composto), sendo que:

- Os Processos Atômicos correspondem às ações que um serviço pode executar em uma única interação. Os Processos Atômicos não possuem subprocessos e são executados em um único passo;
- Os Processos Compostos são aqueles que podem ser decompostos em outros processos. A decomposição pode especificar estruturas de controle tais como sequências, laços e decisões;
- Os Processos Simples não podem ser invocados (executados) e não estão associados a um *endpoint* (*grounding*), mas, assim como os Processos Atômicos, são concebidos como tendo execuções em um único passo. Os Processos Simples são usados como elementos da abstração.

OWL-S especifica somente duas restrições de cardinalidade: (a) um serviço pode ser descrito por no máximo um *ServiceModel*; (b) e um *ServiceGrounding* deve ser associado a exatamente um *Service*. Finalmente, deve-se notar que, apesar de oferecer uma ontologia superior (geral) própria, uma para o *ServiceProfile,* uma para o *ServiceGrounding* e uma para o *ServiceModel*, a OWL-S permite a construção de abordagens alternativas para cada caso. A intenção não foi prescrever uma única

abordagem, mas fornecer uma abordagem padrão que poderia ser útil para a maioria dos casos.

2.3 Casamento de Serviços Web Semânticos

A tarefa da máquina de casamento é buscar e selecionar um serviço concreto cujos atributos satisfaçam às restrições impostas pelo serviço abstrato, a partir das ontologias disponíveis, e obedecendo às regras de casamento estabelecidas [104] [105].

No âmbito deste trabalho, busca e seleção são compreendidos como técnicas utilizadas pela abordagem da área de *workflow* para realização da composição de serviços. Nestas técnicas, define-se um modelo de processo (serviço abstrato) que inclui um conjunto de atividades e suas respectivas dependências de dados e de controle. Cada atividade define os requisitos (vistos como uma consulta) que serão usados para buscar e selecionar um serviço concreto [61] [63] [106].

Na prática, busca e seleção nem sempre são processos discretos, uma vez que o algoritmo de busca pode igualmente classificar os serviços descobertos de acordo com sua relevância perante os requisitos das atividades do serviço abstrato [62].

Nem sempre é possível encontrar um serviço concreto que satisfaça plenamente os requisitos impostos pelo serviço abstrato. Um serviço concreto que seja a implementação exata de um serviço abstrato pode não existir (ou não ser encontrado, o que, na prática, resulta no mesmo). Por isto, é necessário desenvolver técnicas que possibilitem determinar o quão similar é um serviço concreto de um serviço abstrato. Se esta similaridade for maior que um limiar *a priori* estabelecido, então o casamento pode ser realizado.

A similaridade, ou grau de casamento, é um valor que expressa quão similares são duas entidades, de acordo com a métrica de similaridade escolhida. Quanto maior for este valor, mais próximas estarão as entidades, no caso, o serviço concreto do serviço abstrato.

O cálculo do grau de casamento depende da métrica utilizada, mas geralmente é estabelecido um limite a partir do qual se supõe que haja similaridade.

Para Klusch et al. [107], os métodos de casamento podem ser classificados de acordo com:

- Que tipos e partes da descrição semântica do serviço são considerados para o casamento; e
- Como o casamento é efetivamente executado, isto é, como é feita a comparação entre serviço abstrato e serviço concreto.

Tsetsos et al. [62] classificam os métodos de comparação em:

- Baseados em lógica;
- Baseados em similaridade sintática;
- Emparelhamento de grafos[20].

A seguir, será feita uma breve descrição dos métodos classificados por Tsetsos, estendendo-a também para os métodos híbridos (não classificados por Tsetsos), que são baseados em lógica e em técnicas de similaridade sintática.

2.3.1 Casamento Baseado em Lógica

O casamento baseado em lógica é fundamentado em lógica de descrição e em raciocínio em lógica de primeira ordem, particularmente, no conceito de subsunção. Um conceito subsume outro se o primeiro for mais geral que o segundo, isto é, está mais acima na hierarquia conceitual [62].

O grau de casamento entre dois parâmetros de saída ou de entrada depende da relação entre os conceitos associados a estes parâmetros. Por exemplo, para calcular o grau de casamento entre dois parâmetros de saída, $outA$ (parâmetro de saída de um

[20] Em inglês: graph matching.

serviço concreto)[21] e $outR$ (parâmetro de saída de um serviço abstrato)[22], definidos por uma hierarquia de conceitos, Palocci et al. [105] definem quatro graus de casamento:

- Exato: o conceito $outA$ é equivalente ao conceito $outR$, ou o conceito $outR$ é uma subclasse do conceito $outA$, portanto, $outA$ é um conjunto que inclui $outR$;
- *Plugin*: o conceito $outA$ subsume o conceito $outR$;
- Subsunção: o conceito $outR$ subsume o conceito $outA$.
- Falha: quando não ocorre relação de subsunção entre os conceitos.

Para calcular o grau de casamento entre duas mensagens, uma de saída e outra de entrada, Tran et al. [108] utilizam o método apresentado por Oundhakar et al. [109]. A proximidade semântica entre as mensagens é calculada pela expressão: $Semantic\_Match = max\left\{0, \left(\frac{1}{N}\sum_{i=1}^{N} Si - \alpha 2^{n}\right)\right\}$, sendo N o número total de pares de parâmetros de saída-entrada das mensagens e n o número total de serviços utilizado[23] [108]. O nível de similaridade semântica, Si, para cada par é baseado na similaridade semântica $sim(concept_A, concept_B)$, resumidamente descrito a seguir [109]:

- Similaridade exata: $sim(A, A) = 1$;
- O conceito A subsume diretamente B: o conceito A está exatamente um nível hierárquico acima do conceito B, então, $sim(A, B) = 0{,}75$ e $sim(B, A) = 0{,}3$;
- O conceito A é mais genérico que B: o conceito A está dois níveis hierárquicos, ou mais, acima do conceito B, então, $sim(A, B) = 0{,}75 - 0{,}01 \times 2^{x-1}$ e $sim(B, A) = 0{,}3 - 0{,}05x$, onde x é a distância (em termos de níveis) entre os conceitos A e B;
- Não há relação hierárquica entre os conceitos A e B: $sim(A, B) = 0$.

O valor Si é a média sobre o total de pares de saída-entrada. Além disto, uma penalidade é imposta nos casos em que um grande número de serviços é necessário para satisfazer um pequeno número de saídas. O parâmetro α é um fator ajustável para graduar esta penalidade.

[21] No original, service advertised.
[22] No original, service requested.
[23] No original, service profile.

Um exemplo de casamento baseado em lógica pode ser encontrado em Silva et al. [106], onde os autores propõem um *framework* que utiliza serviços Web semânticos juntamente com Redes de Petri e linguagens de descrição de composição (WS-BPEL) para auxiliar na composição de *workflows* científicos. O objetivo é propor uma estrutura para ajudar a comunidade cientifica a definir e executar experimentos científicos através do uso de serviços Web semânticos, repositórios, ontologias, composição de serviços Web e *workflows* científicos. O *framework* possui as seguintes características:

- O registro e o armazenamento de ontologias (OWL) e de anotações semânticas de serviços Web (OWL-S) em repositórios distribuídos;
- A busca semântica em repositórios distribuídos, com base nos requisitos fornecidos pelo usuário (cientista);
- Análise sintática com base nos requisitos estruturais (parâmetros de entrada e de saída) e também análise semântica dos serviços descobertos, a fim de obter possíveis composições;
- Geração do *workflow* em WS-BPEL baseado nas possíveis composições.

Basicamente, o método consiste na definição de um *workflow* de tarefas abstratas e na descrição semântica do *workflow* e de suas tarefas. Estas descrições devem representar o objetivo global do *workflow*. O sistema executa busca semântica, análise estrutural e casamento, apresentando como resultado três possíveis composições. O usuário solicita a geração de um modelo WS-BPEL de uma delas.

2.3.2 Casamento Baseado em Similaridade Sintática

Os métodos de casamento baseados em similaridade sintática se assentam sobre as técnicas de Recuperação de Informação (RI). Em geral, os registros de descrição de serviços, a exemplo do UDDI, permitem métodos da busca baseados em palavras-chave. Especificamente, os serviços podem ser procurados pela categoria, pelo nome e pela posição entre outros. Extensões a este mecanismo estão sendo propostas com o uso de técnicas de RI.

A ideia principal por trás dos modelos clássicos de RI (booleano, vetorial e probabilístico) considera que cada documento é descrito por um conjunto de palavras-chave, chamadas termos de indexação. Associa-se a cada termo de indexação t_i em um documento d_j um peso $w_{ij} > 0$, que quantifica a correlação entre o termo e o documento. Isto significa que a recuperação de um documento é baseada na análise de frequência com que o termo desejado aparece no documento [110].

Segundo Dong et al. [111], a busca por operações de serviços Web similares é um problema semelhante àquele abordado pelas técnicas de RI em documentos de texto [112] [113]. Semelhantes, mas não idênticos. Por exemplo, os documentos que descrevem um serviço Web são geralmente mais compactos do que outros documentos. Tendo isto em mente Dong et al., desenvolveram um algoritmo de busca que agrupa os nomes dos parâmetros das operações de serviços Web em conceitos semanticamente significativos. Posteriormente, estes conceitos são utilizados para determinar a similaridade entre os parâmetros de entrada, ou de saída, das operações. As operações são consideradas similares se elas possuem entradas similares, produzem saídas similares, e as relações entre as entradas e as saídas são similares.

Dicionário de Sinônimos

Tran et al. [108] realizam a composição de serviços Web a partir dos parâmetros de entrada e saída descritos em arquivos OWL-S. O método desenvolvido utiliza uma abordagem híbrida que combina técnicas semânticas e sintáticas. Antes do casamento semântico, é realizada uma tentativa de casamento dos nomes dos parâmetros de entrada e saída. Se as *strings* destes nomes não coincidem é realizada uma busca por sinônimos no dicionário WordNet [114].

O WordNet é uma grande base de dados léxica do inglês. Os substantivos, os verbos, os adjetivos e os advérbios são agrupados em conjuntos de sinônimos cognitivos (*synsets*), cada um expressando um conceito distinto. Os *synsets* são interligados por meio das relações conceituais semânticas e léxicas. A rede resultante de significados de palavras e conceitos relativos pode ser navegada.

Como último recurso, Tran et al. [108] ainda utilizam a métrica de Jaro (Distância de Jaro) [115] para determinar a similaridade entre duas cadeias de caracteres: duas palavras escritas de maneiras ligeiramente diferentes podem significar a mesma coisa. Numa escala que vai de 0 a 1 (zero significando nenhuma similaridade ortográfica e, um, que as palavras são idênticas), as palavras *DIXON* e *DICKSONX*, por exemplo, possuem uma proximidade de 0,813.

2.3.3 Casamento Baseado em Grafos

Nesta abordagem, a descrição de um serviço é representada por um grafo onde os nós são instâncias de conceitos e os arcos são propriedades destas instâncias. O nó raiz de cada grafo é a representação individual de cada serviço. Os outros nós referem-se aos conceitos tomados das ontologias do domínio. Tais conceitos descrevem a funcionalidade do serviço. O casamento é computado por meio da comparação entre os grafos do serviço abstrato com o do serviço concreto.

A característica deste tipo de casamento reside no fato de que de todas as propriedades dos conceitos, isto é, os arcos do grafo, são examinados a fim de identificar os serviços que se casam, não confiando somente nas propriedades especificadas na ontologia do serviço, por exemplo, os atributos IOPE. Esta técnica de casamento é utilizada por Trastour et al. [116]. Entretanto, esta técnica resulta em um dualismo: ou os serviços se casam ou não se casam. Não é possível estabelecer uma gradação entre as duas possibilidades excludentes entre si.

2.3.4 Casamento Baseado em Métodos Híbridos

Klusch et al. [63] implementaram uma abordagem híbrida com a utilização de descrição formal e técnicas de RI. Os autores alegam que as abordagens tradicionais de casamento semântico de serviços Web semânticos são baseadas na descrição formal e não exploram a semântica implícita existente nos padrões ou na frequência dos termos usados na descrição dos serviços. Por isto, foi desenvolvido pelos autores um sistema que

calcula o grau de casamento semântico entre um serviço concreto e um serviço abstrato[24] através da aplicação sucessiva de cinco filtros diferentes: exato, *plug-in*, subsunção, subsumido-por, vizinho-mais-próximo.

Resumidamente, o método pode ser descrito da seguinte forma: seja T a terminologia da ontologia especificada em OWL-DL ($SHOIN(D)$) e CT_T a hierarquia de subsunção do conceito T. Seja ainda IN_A, OUT_A, IN_C e OUT_C o conjunto dos conceitos dos parâmetros de entrada e saída dos serviços abstratos e concretos, respectivamente; $LSC(C)$ o conjunto dos conceitos menos específicos (descendentes diretos) C' de C, isto é, C' é o sub-conceito imediato de C em CT_T; $LGC(C)$ o conjunto dos conceitos menos genéricos (ascendentes diretos) C' de C, isto é, C' é um superconceito de C em CT_T; $SIM_{IR}(A, B) \in [0,1]$ denota o grau numérico de similaridade sintática entre as cadeias de caracteres A e B de acordo com a métrica de RI escolhida, e $\alpha \in [0,1]$ um dado limiar de similaridade sintática; $\doteq$ e $\dot{\geq}$ denotam os conceitos de equivalência e subsunção, respectivamente. Os filtros operam da seguinte maneira:

- Casamento exato: um serviço concreto C casa-se exatamente com um serviço abstrato $A \Leftrightarrow \forall\, IN_C\, \exists\, IN_A: IN_C \doteq IN_A \wedge \forall\, OUT_A\, \exists\, OUT_C: OUT_A \doteq OUT_C$, isto é, para cada parâmetro de entrada do serviço concreto existe um parâmetro equivalente no serviço abstrato (portanto, C pode ter menos parâmetros de entrada que A, mas os parâmetros que existem em C casam-se exatamente com os de A); e, para cada parâmetro de saída do serviço abstrato existe um parâmetro equivalente no serviço concreto (portanto, A pode ter menos parâmetros de saída que C, mas os parâmetros que existem em A casam-se exatamente com os de C);
- Filtro *plug-in*: um serviço concreto C encaixa-se em um serviço abstrato $A \Leftrightarrow \forall\, IN_C\, \exists\, IN_A: IN_C \dot{\geq} IN_A \wedge \forall\, OUT_A\, \exists\, OUT_C: OUT_C \in LSC(OUT_A)$, isto é, todo parâmetro de entrada do serviço concreto subsume um parâmetro de entrada do serviço abstrato (novamente, C pode ter menos parâmetros de entrada que A); e, para cada parâmetro de saída do serviço abstrato existe um parâmetro

[24] No original: service advertisement e service request, repectivamente.

de saída do serviço concreto de tal forma que os parâmetros de C sejam descendentes diretos dos parâmetros de A (igualmente, A pode ter menos parâmetros de saída que C);

- Filtro subsunção: o serviço abstrato A subsume o serviço concreto $C \Leftrightarrow \forall IN_C \exists IN_A: IN_C \dot{\geq} IN_A \wedge \forall OUT_A \exists OUT_C: OUT_A \dot{\geq} OUT_C$. Idem ao filtro anterior com relação aos parâmetros de entrada. Entretanto, os parâmetros de saída de C não precisam mais ser descendentes diretos de A, precisam apenas ser descendentes;
- Filtro subsumido-por: o serviço concreto C subsume o serviço abstrato $A \Leftrightarrow \forall IN_C \exists IN_A: IN_C \dot{\geq} IN_A \wedge \forall OUT_A \exists OUT_C: (OUT_C \doteq OUT_A \vee OUT_C \in LGC(OUT_A)) \wedge SIM_{IR}(C, A) \geq \alpha$. Idêntico aos filtros *plug-in* e subsunção em relação aos parâmetros de entrada. Não obstante, os parâmetros de saída de C podem ser equivalentes ou mais gerais (ascendentes diretos) que os parâmetros de A, desde que haja similaridade sintática entre eles;
- Filtro vizinho-mais-próximo: o serviço concreto C é o vizinho mais próximo do serviço abstrato $A \Leftrightarrow \forall IN_C \exists IN_A : IN_C \dot{\geq} IN_A \wedge \forall OUT_A \exists OUT_C : OUT_A \dot{\geq} OUT_C \vee SIM_{IR}(C, A) \geq \alpha$. Idem para os três filtros anteriores em relação aos parâmetros de entrada, portanto, a diferença em relação ao filtro anterior refere-se aos parâmetros de saída: basta que haja similaridade sintática entre eles.

Os três primeiros filtros, casamento exato, *plug-in* e subsunção são baseados em lógica enquanto os dois últimos são híbridos, isto é, requerem computação adicional, notadamente técnicas de RI, para calcular o valor da similaridade semântica. O objetivo dos filtros híbridos é melhorar o desempenho onde as duas técnicas isoladas, descrição formal e RI, falham.

Os graus de casamento e as regras de composição desenvolvidas no Capítulo 3 são influenciados por estes filtros.

2.4 Composição de Serviços Web Semânticos

Sob o ponto de vista deste trabalho, a composição é entendida como a geração automática do modelo de processo, e, assim sendo, confunde-se com a abordagem adotada pela área de planejamento em IA aplicada à composição de serviços. Os métodos de planejamento, segundo esta abordagem, são usados quando o consumidor não tem nenhum modelo de processo, mas tem um conjunto de restrições e preferências. Daí o modelo de processo deve ser automaticamente gerado pela máquina de composição (ver Seção 1.2).

O objetivo desta seção é, primeiramente, agrupar algumas abordagens de composição de serviços Web semânticos de acordo com a técnica utilizada pelo algoritmo que efetivamente realiza a composição. Posteriormente, as características comuns a um grupo poderão ser sintetizadas e, entre eles, comparadas.

Entretanto, há objeções na realização deste objetivo. A tarefa de compor é essencialmente uma tarefa de busca. E, neste caso, classificar os algoritmos de composição seria o mesmo que classificar os algoritmos de busca, que, evidentemente, teria pouco valor. Portanto, a classificação levará em consideração outros fatores além do algoritmo de composição propriamente dito, por exemplo, a linguagem ou o modelo adotado para descrever o serviço abstrato. Na maioria das vezes, a linguagem de composição e o algoritmo de composição estão mutuamente relacionados [84].

A seguir, são apresentados alguns trabalhos representativos de composição de serviços Web semânticos a fim de proporcionar um panorama geral sobre a área.

2.4.1 Composição Baseada em Busca em Profundidade

Supondo que um serviço abstrato seja especificado a partir de seus parâmetros de entrada e saída, a busca em profundidade tenta identificar um serviço concreto cujos parâmetros de entrada se casam com os parâmetros de entrada do serviço abstrato. Caso este serviço seja encontrado, uma nova busca é realizada, mas, desta vez, a tentativa de casamento é realizada com os parâmetros de saída do serviço concreto. Desta forma

monta-se um *workflow* de fluxo sequencial onde cada serviço concreto encontrado é imediatamente inserido após o último. A busca é realizada até que os parâmetros de saída do último serviço concreto inserido no *workflow* se casem com os parâmetros de saída do serviço abstrato.

Entretanto, o que ocorre se este serviço concreto não for encontrado? É necessário que o algoritmo tenha outros critérios de parada além deste, por exemplo, quantidade de serviços inseridos ou limite de tempo.

Todavia, problemas mais sutis podem ser encontrados a qualquer momento durante a criação do *workflow*, e não apenas entre os parâmetros de saída do último serviço concreto inserido com os parâmetros de saída do serviço abstrato. O que acontece se, em determinado momento, existirem somente serviços concretos cujos parâmetros de entrada se casam apenas parcialmente com os parâmetros de saída do último serviço concreto inserido no *workflow*? Por casamento parcial, quer-se dizer que alguns parâmetros se casam, mas outros não. Caso a única possibilidade de composição seja o fluxo de controle sequencial, o algoritmo terá que retroceder até o último serviço concreto inserido e tentar outro. Entretanto, as alternativas se multiplicam caso outros tipos de fluxo de controle sejam tratados, por exemplo, o fluxo em paralelo. Isto significa que, dois ou mais serviços concretos compostos em paralelo podem oferecer os parâmetros de entrada, ou de saída, necessários para que casamento seja efetuado com sucesso.

Isto significa que a montagem do *workflow* dependerá dos tipos de fluxo suportados pelo algoritmo de composição. Nos parágrafos acima, o exemplo foi realizado levando-se em consideração apenas fluxo sequencial. Entretanto, segundo a Workflow Management Coalition (WfMC) [79] e Workflow Patterns Initiative [80] existem diversos tipos de fluxo de dados e de controle. Considerando somente os fluxos de controle básicos tem-se: sequencial, separação paralela, sincronização, escolha-exclusiva e união simples.

Tran et al. [108] propõem a composição de serviços Web semânticos baseada em algoritmos de planejamento em IA, realizando busca em profundidade. As características

do sistema implementado por Tran et al. incluem o casamento de nomes, a computação da similaridade semântica e a detecção de ciclos. O sistema permite a alteração de vários parâmetros, por exemplo, em relação à profundidade da composição e ao limiar de similaridade semântica, antes de lançar o processo da composição. A Figura 2-3 apresenta uma visão geral do sistema e, a seguir, tem-se uma breve descrição dos principais módulos. O artigo de Tran et al. [108] traz, ainda, um estudo de caso envolvendo uma ontologia fictícia.

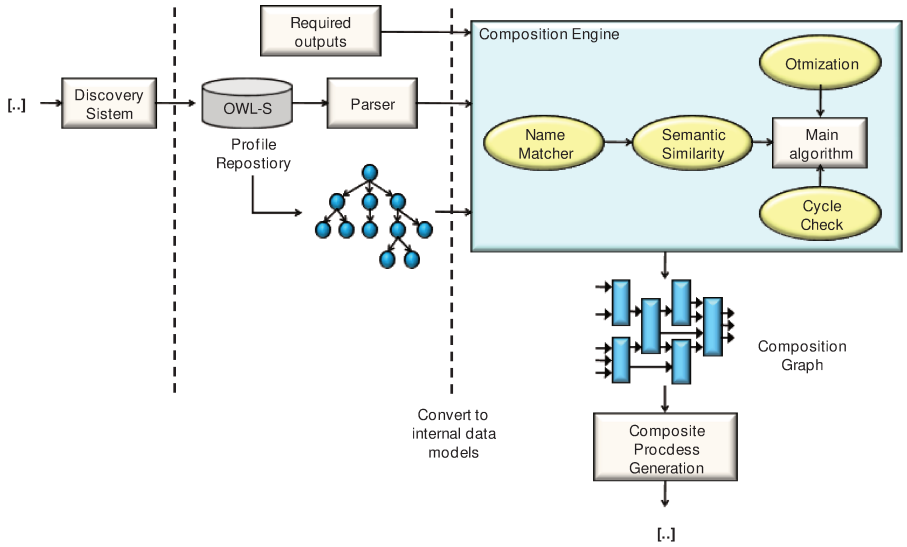

Figura 2-3 – Visão geral do sistema. Fonte: Tran et al. [108].

O casamento de nomes utiliza o dicionário de sinônimos WordNet [114] e a métrica de Jaro (Distância de Jaro) [115] para determinar a similaridade entre duas cadeias de caracteres (ver seção 2.3.2). O módulo da detecção de ciclo detecta quantos ciclos podem existir em cada estágio do processo da composição. Isto elimina a possibilidade de produzir gráficos inválidos de composição que conduzem à não terminação durante a execução. A fim de aperfeiçoar o sistema, um mecanismo de *cache* é introduzido à computação da função $sim(A, B)$ para cada par de A e B: um par de entrada e de saída cuja similaridade semântica deva ser determinada, pode envolver o mesmo par de tipos ontológicos cuja similaridade já foi verificada. O trabalho Tran et al. [108] possui as seguintes características:

- Os serviços são pré-selecionados, nenhuma concepção sobre a descoberta de serviços é realizada;
- Todos os serviços utilizam a mesma ontologia;
- Cada serviço tem somente um *profile*;
- Utiliza o método de planejamento em IA realizando busca em profundidade, a partir das saídas dos serviços;
- Na composição não há estruturas complexas, por exemplo, não há comandos de decisão (IF) e nem laços;
- Somente são consideradas as entradas e saídas, pré- e pós-condições são descartadas;
- Não gera composições em linguagens executáveis, como a BPEL, por exemplo, isto porque o resultado da composição é somente o *profile* do serviço composto;
- Possui detecção de ciclo;
- OWL-S é a ontologia utilizada;
- O número de camadas presentes na solução é definido pelo usuário;
- A similaridade semântica pode ser calculada através do casamento de nomes ou entre sub e superconceitos.

2.4.2 Composição Baseada na Linguagem PowerLoom

O Argos é outro sistema de planejamento em IA criado para gerar automaticamente um *workflow* em resposta a uma requisição do usuário, incluindo a integração de dados e operações de tradução [117]. O *workflow* é gerado por um algoritmo de planejamento cuja operação básica consiste em satisfazer as entradas de um serviço com as saídas de outro. Adaptadores foram criados para realizar as operações de seleção e união de dados, não de fluxo. O adaptador de seleção é utilizado para verificar se um determinado dado está presente em uma estrutura maior e mais complexa.

Um serviço é representado pelos seus dados de entrada e saída, e todos os dados são descritos em uma ontologia comum. Os conceitos e as relações da ontologia do domínio dos dados são apresentados na linguagem de primeira-ordem PowerLoom. O

planejador começa tendo os dados requeridos pelo usuário como objetivo e termina a busca quando encontra um plano completo, isto é, um plano onde todas as entradas de dados são produzidas por outras operações ou fontes. O sistema Argos foi aplicado ao domínio de transporte de bens de consumo.

O sistema Argos não apresenta estruturas complexas de controle de fluxo, tais como comandos de seleção e laço.

2.4.3 Composição Baseada em Inferência Orientada por Objetivos

A inferência orientada por objetivo é utilizada para testar a certeza de uma hipótese (objetivo). Em caso afirmativo a hipótese poderá ser acrescentada à base do conhecimento como fato comprovado.

Fujii e Suda [55] implementaram um sistema completo para a definição, composição e execução de serviços Web que utiliza a inferência orientada por objetivo. A arquitetura do sistema é divida em três partes: Component Service Model with Semantics (CoSMoS), Component Runtime Engine (CoRE) e Semantic Graph based Service Composition (SeGSeC).

CoSMoS é um modelo abstrato que representa um componente como um gráfico semântico. CoRE fornece uma interface unificada para descobrir e acessar os componentes implementados em várias tecnologias diferentes. SeGSeC gera um *workflow* a partir de uma requisição do usuário feita em linguagem natural. SeGSeC igualmente realiza o casamento semântico para confirmar que a semântica do *workflow* gerado combina com a semântica do pedido do usuário. SeGSeC é projetado como uma coleção de agentes, a saber, *RequestAnalyzer*, *ServiceComposer*, *Reasoner*, e *ServicePerformer*. O *RequestAnalyzer* analisa gramaticalmente um pedido de usuário na forma de texto (por exemplo, "*print a direction from home to restaurant*") e o transforma em uma representação gráfica semântica do CoSMoS. Depois disto, o *RequestAnalyzer* envia o pedido ao *ServiceComposer*. Uma vez recebido o pedido do *RequestAnalyzer*, o *ServiceComposer* primeiramente procura por uma operação (operação *print* de um

componente de impressora, por exemplo) que execute o predicado do pedido. Então, o *ServiceComposer* gera um *workflow* conectando entradas de uma operação com as saídas de outras operações ou propriedades de outros componentes. Após ter gerado o *workflow*, o *ServiceComposer* envia-o juntamente com o pedido do usuário ao *Reasoner* para executar o casamento semântico. O *Reasoner* é um agente que executa o casamento semântico para verificar se o pedido do usuário pode ser satisfeito aplicando as regras de casamento semântico no *workflow*. Estas regras são projetadas para derivar significados semânticos a partir das interações dos componentes em um *workflow*. Após ter terminado o casamento semântico, o *workflow* é enviado ao *ServicePerformer* para execução.

Alamri et al. [39] indicam alguns pontos fracos deste projeto. Apesar de a técnica de composição ser de alguma forma controlada pelo usuário final, o conjunto de consultas[25] que pode ser usado para testar a funcionalidade do sistema é limitado. Adicionalmente o sistema foi projetado para cenários muito restritos e precisa de muitas adições para poder satisfazer outros tipos de consultas.

2.4.4 Composição Baseada em Cálculo de Situações

O cálculo de situações [118] é uma linguagem de primeira ordem (com algumas características de segunda ordem) especialmente projetada para representar dinamicamente mudanças nos mundos. Todas as mudanças nos mundos são resultados de ações. Uma possível história do mundo, que é simplesmente uma sequencia de ações, é representada por um termo de primeira ordem chamado situação. A constante S_o é usada para denotar a situação inicial, isto é a situação na qual nenhuma ação ocorreu. Há um símbolo distinto para uma função binária, do; $do(\alpha, s)$ denota a situação sucessora a s resultante da realização da ação α. As ações podem ser parametrizadas, por exemplo, $put(x, y)$ pode representar a ação de colocar o objeto x no objeto y, na qual $do(put(A, B), s)$ denota a situação resultante de colocar A em B quando o mundo está na situação s. No cálculo de situações, as ações são denotadas por termos de primeira

[25] Em inglês: queries.

ordem, e situações (histórias do mundo) também são termos de primeira ordem. Por exemplo, $do\left(putdown(A), do\left(walk(L), do(pickup(A), S0)\right)\right)$ é uma situação que denota que a história do mundo consiste da sequência das ações $[pickup(A), walk(L), putdown(A)]$. A sequência das ações na história, na ordem na qual elas ocorrem, é obtida do termo situacional pela leitura das ações da direita para a esquerda.

Golog é uma linguagem de programação em alto-nível para especificar e executar ações complexas em domínios dinâmicos. Golog foi construída sobre o cálculo de situações [119].

McIlraith et al. [74] [120] [65] adaptaram e estenderam a linguagem Golog para a descoberta, execução e composição automática de serviços Web. Os autores abordam o problema de composição de serviços Web através da provisão de procedimentos genéricos de alto-nível e da otimização de restrições. A linguagem Golog é adotada como um formalismo natural para representar e raciocinar sobre este problema. A requisição do usuário (procedimento genérico) e os requisitos são descritos na linguagem do cálculo de situações. Os autores concebem cada serviço Web como uma ação primitiva (*PrimitiveAction*) ou complexa (*ComplexAction*). As ações primitivas são concebidas como ações que mudam o estado do conhecimento. As ações complexas são composições de ações primitivas. A base de conhecimento do agente fornece uma codificação lógica das condições prévias e dos efeitos das ações do serviço Web na linguagem do cálculo de situações. Os agentes usam as construções da linguagem de programação procedimental compostas dos conceitos definidos para os serviços e as restrições usam máquinas dedutivas. Um serviço composto é um conjunto de serviços atômicos conectados por construções da linguagem de programação procedimental (*if-then-else*, *while* e assim por diante).

A linguagem Golog é capaz de executar fluxos complexos onde estão presentes construções como *if-then-else* e *while*, mas não é capaz de gerar automaticamente estas construções. A redução do espaço de busca é alcançada, segundo os autores, com os

parâmetros de entrada e as restrições do usuário. Peer [121] levanta, ainda, que a criação de um programa Golog é difícil, impedindo a adoção da sua tecnologia e que a escalabilidade do cálculo de situações pode ser questionada.

2.4.5 Composição Baseada na Linguagem PDDL

O grande interesse da comunidade de planejamento em IA pela área de composição de serviços Web deve-se à similaridade entre as representações de DAML-S [50] e de PDDL (Planning Domain Definition Language) [61]. A PDDL é largamente reconhecida como uma entrada padrão para planejadores avançados [121]. Foi inicialmente desenvolvida para tornar possíveis as competições do International Planning Competitions (1998/2000) [122]. É possível mapear a linguagem PDDL no cálculo de situações [123].

PDDL é uma linguagem centrada na ação, inspirada pela formulação STRIPS de problemas de planejamento. O seu núcleo é uma padronização de uma sintaxe para expressar semânticas familiares de ações, usando pré- e pós-condições para descrever a aplicabilidade e os efeitos das ações.

As tarefas de planejamento especificadas em PDDL são separadas em dois arquivos: o arquivo de domínio para predicados e ações, e um arquivo de problema para especificação dos objetos, dos estados iniciais e do objetivo. Os componentes de um planejador em PDDL são:

- Objetos: coisas do mundo que são de interesse. A definição do domínio contém os predicados e operadores do domínio (chamados ações no PDDL). Ele também contém tipos, constantes, fatos estatísticos entre outros;
- Predicados: propriedade dos objetivos que são de interesse; podem ser falsos ou verdadeiros;
- Estado inicial: estado inicial do mundo;

- Objetivo: uma descrição do objetivo é usada para especificar os objetivos desejados em um problema de planejamento e também as precondições de uma ação;
- Ações/Operadores: modos de mudar o estado do mundo.

A linguagem PDDL separa a descrição de ações parametrizadas, que caracterizam o comportamento do domínio, da descrição das condições iniciais e dos objetivos que caracterizam a instância do problema. Desta forma, o problema de planejamento é criado pela junção da descrição do domínio com a descrição do problema. Uma mesma descrição de domínio pode fazer par com várias descrições de problemas diferentes para produzir diferentes problemas de planejamento dentro de um mesmo domínio.

Peer [121] argumenta que a diversidade de domínios dos serviços Web requer uma combinação de técnicas complementares de raciocinadores de planejamento baseados em IA. Assim sendo, foi criado o WSPlan, um *framework* flexível e não monolítico, que permite conectar os planejadores mais apropriados para certos domínios e tarefas, essencial para lidar com mudanças ou novas exigências dos usuários. Para a integração de tal estrutura, foi usada a linguagem PDDL [124]. O WSPlan transforma, a partir da base de conhecimento do agente, dados e informação da descrição (anotações) do serviço Web em documentos PDDL. Um planejador capaz de processar PDDL que combina as exigências impostas pelas definições de um domínio em particular pode processar aqueles documentos e eventualmente identificar um plano. O plano resultante pode ser transformado em um fluxo de operações a serem invocadas. Em resumo, esta técnica, segundo o autor, permite que se faça a seleção do melhor planejador para uma dada tarefa de composição. Uma ressalva importante: é possível fazer a marcação semântica das operações dos serviços Web, entretanto, o WSPlan não processa esta informação. Uma limitação que, segundo os autores, seria tema de futuros trabalhos.

McDermott [75] expande a PDDL para realizar a composição de serviços Web. Segundo o autor, falta à PDDL a habilidade de especificar que a execução de uma ação cria determinada informação. Por exemplo, pode ser necessário que a ação de enviar uma

mensagem a um agente qualquer gere uma identificação que possa ser usada em futura resposta. A existência desta identificação não é propriamente um efeito[26] no sentido da PDDL. Para resolver este problema, McDermott introduziu um novo tipo de conhecimento, chamado valor da ação[27]. Isto permite distinguir a transformação da informação e a mudança de estado produzida pela execução de um serviço. A informação, que é apresentada pelos parâmetros de entrada e saída, é provavelmente reutilizável, permitindo que os valores de dados podem ser duplicados para a execução de múltiplos serviços. Contrariamente, os estados do mundo são mudados pela execução do serviço. Esta mudança é interpretada como o desaparecimento de estados velhos e a produção de estados novos.

Sucintamente, o algoritmo de McDermott funciona da seguinte forma. Dado um objetivo e uma situação inicial, o algoritmo realiza uma busca situação-espaço[28] por uma sequência de passos que possam atingir o objetivo. Isto é chamado de situação atual do estado. Um estado da busca é uma série de passos factíveis começando por uma situação inicial e culminando em uma situação que se espera esteja mais próxima de satisfazer o objetivo. A busca é guiada pela seguinte heurística: para cada estado da busca, construa um grafo casamento-regressão[29] a partir do objetivo em direção à situação atual do estado da busca. O grafo é essencialmente uma árvore E/OU em que em nós “E” correspondem às pré-condições de uma ação, e os nós “OU” correspondem a um conjunto de ações (apenas uma delas é suficiente para se atingir uma condição). Algumas limitações do método podem ser elencadas: as operações dos serviços Web precisam ser descritas (ou traduzidas) na linguagem do planejador, que não é uma linguagem padronizada para os serviços Web, ou seja, a resposta não é propriamente um *workflow*, mas uma sequência de passos que se executados sequencialmente, atingem o objetivo. Isto quer dizer que não há a determinação de fluxo de controle. Ademais, e de forma mais grave, não há o emprego de semântica, uma vez que se está interessado nas técnicas que a utilizem.

[26] Em PDDL: :effect.
[27] No original: value of action.
[28] No original: situation-space.
[29] No original: regression-match graph.

2.4.6 Composição Baseada em HTN

A Rede Hierárquica de Tarefas (HTN[30]) é uma técnica de planejamento da área de IA baseada no conhecimento de controle e na suposição de mundo fechado (informalmente, isto significa que todos os "blocos de construção" são conhecidos *a priori*). O planejamento HTN é como o planejamento clássico da área de IA em que cada estado do mundo está representado por um conjunto de átomos, e cada ação corresponde a uma transição determinística de estado. Entretanto, os planejadores HTN diferem dos planejadores clássicos da IA em dois sentidos: (a) no que eles planejam e (b) como eles planejam [125]. A finalidade de um planejador HTN é produzir uma sequencia de ações que executem alguma atividade ou tarefa. A descrição do domínio de planejamento inclui um conjunto de operadores similares ao planejamento clássico (no caso de serviços Web, são as operações), e também um conjunto de métodos, cada um dos quais é uma prescrição de como decompor uma tarefa maior em tarefas menores. A decomposição segue até que as tarefas primitivas (serviços Web executáveis, concretos) sejam encontradas.

Sirin et al. [77] sugerem um método de planejamento HTN para realizar a composição automática de serviços Web descritos com OWL-S. Eles descrevem um algoritmo para traduzir as descrições OWL-S dos serviços Web para o domínio do SHOP2 (SHOP2 é um sistema de planejamento HTN independente do domínio [125]) e então um plano é encontrado pela decomposição de tarefas complexas. O objetivo é encontrar uma coleção de instâncias de processos atômicos que formem um caminho de execução para algum processo composto definido num nível de abstração mais alto. Neste sentido, a entrada do planejador é um serviço composto, isto é, uma descrição de como compor uma sequencia de ações simples. Segundo os autores, a escalabilidade do sistema é pequena: ela não está adaptada para trabalhar na escala da Web, que pode conter centenas, até mesmo milhares, de fatos e axiomas.

[30] Sigla em inglês para Hierarchical Task Network.

Peer [121] levanta que um dos problemas do uso de HTN é a falta de execução em paralelo, uma característica frequentemente necessária ao uso eficiente dos serviços Web. Realmente, o SHOP2 não considera esta construção, mas Sirin at. al. [77] dizem ser possível implementá-la com Concurrent Golog [126], uma extensão da linguagem Golog que permite execução em paralelo.

Considerando o procedimento de decomposição de tarefas, os métodos baseados em HTN estão principalmente relacionados com a possibilidade da decomposição da tarefa, isto é, se um plano pode ser encontrado. Entretanto, um plano pode falhar por várias razões, tais como: um serviço pode não mais existir; ou devido aos problemas nas interações características dos serviços Web [59].

2.4.7 Composição Baseada em Grafo

Prazeres [127] modelou os serviços Web semânticos como um grafo direcionado. Cada serviço corresponde a um nó no grafo e os serviços são conectados entre si com base na similaridade semântica (via regra de subsunção) entre seus parâmetros de entrada e saída, ou seja, uma aresta no grafo é criada entre dois serviços quaisquer, por exemplo, S_1e S_2, se um dos parâmetros de saída do serviço S_1 for similiar ao parâmetro de entrada do serviço S_2. A cada uma das arestas é associado um custo, referente à utilização do serviço. A composição automática é realizada utilizando algoritmos de cálculo do caminho de custo mínimo. Como limitações este trabalho apresenta:

- Encontra apenas um único caminho mínimo (uma única composição), mesmo que outras existam;
- O tratamento de ciclos não foi realizado;
- Os caminhos são criados a priori e, portanto, não é possível realizar a seleção dinâmica.

O método proposto por Shina et al. [128] especifica explicitamente a semântica funcional de um serviço Web baseado em uma ontologia de domínio, utilizando para isto Web Service Modeling Ontology (WSMO). A especificação semântica descreve o que um

serviço pode oferecer aos clientes quando for invocado; e inclui a capacidade e categorização do serviço. Nesta abordagem, a funcionalidade de um serviço é representada por um par *ação* e *objeto da ação*. A informação sobre os serviços é organizada e armazenada em um modelo de grafo de duas camadas: em uma camada é descrita a dependência de dados entre os serviços; e, na outra, é descrita as relações entre as ações de domínio. A ligação entre as duas camadas é feita através de um mapeamento.

Dada uma requisição do usuário, é realizada a busca por caminhos no grafo e a posterior construção de um serviço composto a partir dos caminhos descobertos. O método proposto para a composição de serviços Web consiste de três etapas:

- Construção de um conjunto de serviços Web candidatos: a partir dos requisitos do usuário é construído um grafo ligando a saída de um serviço à entrada de outro;
- Encontrar caminho no grafo de serviços;
- Construção de serviços compostos a partir dos caminhos encontrados.

A abordagem utilizada é capaz de encontrar mais de uma composição, caso exista, mas realiza somente composição para o padrão de fluxo de controle sequencial.

Song et al. [57] desenvolveram um framework para a composição de serviços Web semânticos. Dado um processo de negócio e um conjunto de serviço Web semânticos descritos em SAWADL, para cada atividade do processo de negocio é realizado um casamento com um serviço. O casamento leva em consideração três aspectos: categoria do serviço, parâmetros de entrada e saída e QoS. O algoritmo que realiza o casamento dos parâmetros de entrada e saída é baseado em grafos. Dado um grafo onde os nós representam os serviços Web e os arcos entre dois nós representam as ligações entre os dados de entrada e saída dos serviços, é possível computar o melhor caminho entre o nó inicial (estado inicial) e o nó final (estado final). O caminho representa uma composição. Portanto, uma atividade do processo pode casar com uma composição. Entretanto, o algoritmo de composição não foi implementado. Segundo os autores, devido à

diversidade, flexibilidade e complexidade, a composição seria abordada em futuros projetos.

Wu et al. [129] estendem um planejamento em grafo, o GraphPlan, que é um algoritmo de planejamento em IA para gerar automaticamente o fluxo de controle de processos Web. O modelo de composição foca somente na parte abstrata de um serviço Web, isto é, operações e mensagens, não considerando as informações de *binding*. Para o domínio da composição de serviços Web, os autores fazem uma extensão da linguagem STRIPS para que possam utilizá-la como linguagem representacional do modelo apresentado. O sistema implementado gera um processo na linguagem BPEL com habilidade de gerar os fluxos de controle do tipo sequencial, separação e laço.

A geração automática do fluxo de controle em laço é baseada na observação dos padrões de iteração frequentemente usados. Por exemplo, o pedido de compras pode incluir múltiplas linhas de itens (uma matriz de linha de itens) enquanto uma operação de adição requer apenas uma única linha como entrada de dados. Portanto, a operação precisa iterar todas as linhas de itens para realizar um pedido. A seção 3.4.4, que descreve as regras para a Composição em Laço, parte da desta mesma observação.

Hashemian e Mavaddat [130] usaram uma linguagem de modelagem especifica, chamada interface automata, para descrever composições de serviços Web semânticos. A linguagem é um modelo baseado em estados, similar aos diagramas de estados finitos, usada para representar o comportamento de componentes de software. A linguagem permite a especificação formal da composição entre dois componentes. Cada serviço é representado como um conjunto de parâmetros de entrada, um conjunto de parâmetro de saída e um conjunto de informações de dependência entre as diferentes entradas e saídas do serviço. Da mesma forma, uma requisição é especificada através da definição destes três conjuntos. A partir destes conjuntos é possível montar um grafo, tal que os nós representem as entradas e as saídas dos serviços disponíveis, e os arcos representem as associações entre os serviços. O grafo é, portanto, um conjunto de relações entre uma entrada e uma saída. A partir do grafo é possível encontrar uma árvore binária que

representa um serviço composto e esta árvore binária, por sua vez, é mapeada em um *workflow*. Entretanto, a criação da árvore binária é possível graças à restrição imposta de que uma operação de um serviço Web pode ter apenas uma única entrada e uma única saída.

2.4.8 Composição Baseada em Computação Evolutiva

Tang e Ai [78] usam algoritmos genéticos para resolver o problema da composição de serviços Web com restrições de QoS. O usuário descreve um serviço abstrato (definido como um *workflow* de operações) e o algoritmo seleciona para cada operação do serviço abstrato, uma implementação (um serviço Web concreto) de tal maneira que a qualidade da composição seja máxima. Os critérios de qualidade selecionados são: tempo de resposta, preço, reputação, confiabilidade e disponibilidade. Para cada um destes critérios um o peso é estabelecido, e uma função de *fitness* é criada a fim de refletir a qualidade total da composição.

Para cada operação, diversos serviços concretos com seus respectivos parâmetros de QoS são aleatoriamente criados. Estes serviços concretos são estruturas de dados capazes de armazenar os parâmetros de QoS. O algoritmo implementa as operações de cruzamento e mutação. A validação do algoritmo é realizada através de várias simulações, por exemplo, variando a quantidade de operações dos serviços abstratos ou variando a quantidade de serviços concretos existentes para cada operação. Os resultados são apresentados em relação ao *fitness* médio e ao tempo de resposta do algoritmo.

2.4.9 A Escolha de Um entre Vários

Chen et al. [131] trabalham com a hipótese de que, no mundo real, vários planos podem solucionar um processo de negócio especifico definido em alto-nível. Por exemplo, uma viagem pode ser realizada de carro, avião ou trem. A escolha será baseada em requisitos não funcionais definidos pelo usuário, tais como preço. Para tanto,

desenvolveram um modelo para combinar o modelo de processo de decisão de Markov (MDP[31]) e o planejamento HTN dirigido à composição automática de serviços.

Os autores fazem uma modificação no processo de planejamento HTN proposto por Sirin et al. [77] para que mais de um plano possa ser encontrado. Posteriormente, estes planos são enviados a um processo de decisão de Markov que escolhe o melhor deles. A escolha é baseada em parâmetros de QoS definidos pelo usuário. Para mostrar a potencialidade da abordagem, os autores fazem um estudo de caso utilizando o exemplo de uma viagem. No cenário proposto, o usuário deseja primeiro fazer alguns dias de excursão pela cidade e somente depois ir ao congresso. Para conseguir este objetivo, o usuário pode ter mais de um plano. Depois de encontrar todos os planos possíveis, o usuário pode escolher aquele que será executado ou então, confiar no método MDP para que identifique o plano ótimo baseado nos aspectos não funcionais. Algumas considerações sobre o método:

- A escolha de todos os planos possíveis poderia comprometer a escalabilidade do método. Entretanto, os autores contornam este problema com a técnica de recompensa imediata[32]. A recompensa imediata é um valor que mede a qualidade da decomposição de uma tarefa. Quando uma tarefa é decomposta em tarefas primitivas ela recebe uma recompensa. Um algoritmo utiliza o valor da recompensa imediata para fazer o controle estratégico da expansão da decomposição e assegurar que nem todos os ramos serão decompostos;
- Excetuando-se a limitação de se ter somente um plano, este método possui as mesmas limitações das técnicas descritas em Sirin et al. [77], uma vez que utiliza os mesmo métodos de transformação de OWL-S para SHOP2 e vice-versa;
- Os autores asseguram que o impacto de possíveis falhas na execução de um plano é reduzido, uma vez que o método oferece múltiplos planos ao usuário. Entretanto, tal consideração requer tratamento específico a falhas, uma vez que, tendo ocorrido um erro durante a execução de um plano, não basta simplesmente

[31] Sigla em inglês para Markov Decision Process.
[32] Em inglês: immediate reward.

trocar de plano. É necessário ativar mecanismos de compensação ou de *rollback*. Muitas vezes estes mecanismos não são fornecidos pelo serviço e, portanto, devem ser projetados de maneira *ad-doc*.

2.5 Método Híbrido entre Casamento e Composição

Medjahed et al. [76] apresentam um modelo de composição em quatro fases: especificação, casamento, seleção e geração. Na fase de especificação, o usuário utiliza a linguagem CSSL, uma linguagem onde a composição de serviços é especificada como interações entre operações abstratas. Durante o casamento, o algoritmo tenta encontrar para cada operação abstrata uma ou mais operações pertencentes a serviços Web concretos que com ela se casam. Caso mais de uma operação seja encontrada, haverá a criação de mais de um plano de composição. Um destes planos é selecionado de acordo com parâmetros de qualidade de composição. O plano selecionado é enviado para a fase de geração, onde detalhes referentes ao controle de fluxo, por exemplo, são adicionados.

Há várias regras para determinar se dois ou mais serviços podem ser compostos. As regras são divididas em duas categorias que comparam as propriedades sintáticas e semânticas dos serviços Web. As regras sintáticas verificam a possibilidade de composição entre os modos e os protocolos de ligação[33]. As regras semânticas verificam as mensagens, as operações, as propriedades qualitativas das operações e o sentido lógico[34] da própria composição.

Na interação entre os serviços é necessário que os modos e protocolos de comunicação se casem, por exemplo, uma operação de modo *solicit-response* somente poderá se casar com uma operação de modo *request-response*; e ambos os serviços devem aceitar os mesmos protocolos de comunicação, por exemplo, o SOAP.

Na composição de mensagens, são considerados a quantidade de parâmetros de entrada e saída, os tipos de dados e as unidades dos tipos de dados. No que diz respeito

[33] No original: binding protocols.
[34] No original: soundness.

às operações, tem-se, por exemplo, que um pedido de cotação de carro é diferente do pedido de reserva de quarto de hotel, e, portanto, não podem ser compostas. Em relação à possibilidade de composição qualitativa, são consideradas, por exemplo, a criptografia e a não repudiação entre as mensagens. A validação do sentido lógico de uma composição deve concluir que combinar os serviços de reserva de hotel com locação de automóveis faz sentido, mas combinar reserva de hotel com venda de materiais de construção não é usual.

Dois serviços Web somente podem ser compostos se a mensagem da saída de um serviço é compatível com a mensagem da entrada de outro serviço, ou seja, se os tipos de dados dos parâmetros de saída são compatíveis com os tipos de dados dos parâmetros de entrada. Há a definição de dois tipos de composição de mensagem: direta e indireta. Dois parâmetros são diretamente compatíveis se eles possuem os mesmos tipos de dados. Dois parâmetros são indiretamente compatíveis se um pode ser derivado do outro, por exemplo, $positiveInteger$ e $short$.

Embora o método obrigue o usuário a especificar uma composição em CSSL, a especificação é realizada em alto nível: os detalhes de integração entre os serviços são automaticamente realizados. Por isto, o método estabelece uma forma de compromisso entre as técnicas de casamento e de composição. O modelo de composição proposto no Capitulo 3 tentará ampliar esta característica permitindo que cada operação abstrata possa se expandir em um novo *workflow*.

2.6 Considerações Finais

Neste capítulo, foi apresentado um sistema de referência de composição de serviços com a missão de definir alguns conceitos que serão utilizados no restante deste trabalho, em especial, casamento e composição. Adicionalmente, algumas das abordagens desenvolvidas para realizar o casamento e a composição foram brevemente descritas.

Em relação às abordagens apresentadas, o trabalho aqui proposto focará na busca de soluções para as seguintes limitações:

- Técnicas de casamento e composição utilizadas em separado (ver seções 1.5, 2.3 e 2.4);
- Falta de semântica (ver seções 1.3 e 2.2);
- Composições geradas apenas para parte dos tipos cinco tipos básicos de fluxo de controle [79] [80]: sequencial, separação paralela, sincronização, escolha-exclusiva, união simples;
- Ausência de resultados quantitativos;
- Apresentam apenas uma única solução (ver seção 2.4.9).

Tomadas isoladamente, é possível encontrar estas limitações solucionadas em várias abordagens, conforme apresentado no corrente capítulo. Entretanto, em nenhuma delas todas estas limitações são tratadas. Para solucioná-las, propõe-se uma abordagem com as seguintes características essenciais:

- Desenvolver um método que combine as vantagens das técnicas de geração manual do *workflow* (casamento) e geração automática do *workflow* (composição). O serviço abstrato definido pelo usuário pode conter uma ou mais de uma operação. Para cada operação do serviço abstrato, o método tentará encontrar um serviço concreto. Caso não o encontre, tentará realizar uma composição. O resultado desta composição será apresentado como um único serviço. Desta forma, o casamento será feito entre uma operação e um serviço composto;
- Utilizar semântica: casamento e composição levarão em conta a descrição semântica dos parâmetros de entrada e saída dos serviços descritos em arquivos OWL-S;
- Desenvolvimento de técnicas de composição para fluxo de controle sequencial, separação paralela, sincronização, escolha-exclusiva, união simples e laço;
- A máquina de composição será implementada usando times assíncronos e computação evolutiva. Em consequência, poderá ser encontrada mais de uma solução;

- Desenvolvimento de regras de casamento baseadas em regras lógicas e técnicas de RI;
- Utilização de base de serviços Web semânticos OWLS-TC3 [132] para a validação e análise desempenho do sistema implementado, gerando resultados numéricos que permitam a análise quantitativa;
- O resultado final da composição deverá ser um serviço executável.

No próximo capítulo, serão desenvolvidas das regras que permitem determinar o grau de similaridade e de casamento e as regras de composição de serviços.

Capítulo 3

Graus de Casamento e Regras de Composição para Serviços Web Semânticos

Existe miséria maior do que ter que se esforçar na direção do básico?

Pessanha, Juliano Garcia. Instabilidade Perpétua. Ateliê Editorial, 2009, p 44.

Quando um serviço concreto pode substituir um serviço abstrato? Ou, quando dois serviços podem ser sequencialmente compostos? Estas são algumas das questões que norteiam o presente capítulo. Para respondê-las, serão definidas as regras que expressam o grau de similaridade e de casamento entre um serviço abstrato e um serviço concreto. O grau de casamento é calculado em relação à similaridade sintática e semântica dos parâmetros de entrada e saída entre estes dois serviços. Na verdade, entre uma *operação abstrata* e um *serviço concreto*, conforme será apresentado. Em última análise, as regras

de casamento permitem que um serviço concreto seja interpretado como uma implementação de uma operação abstrata.

Além das regras de casamento, as regras que definem a composição de seis padrões de fluxo de controle são também definidas. Segundo WfMC [79] e Workflow Patterns Initiative [80], cinco destes padrões de fluxo são básicos: sequencial, separação paralela, união síncrona, escolha-exclusiva e união simples; e um é estrutural: laço.

A definição destas regras é importante, porque permite que algoritmos computacionais as utilizem para automatizar o processo de composição de serviços (tema que será abordado no Capítulo 4).

A apresentação da terminologia utilizada pelas regras inicia o presente capítulo, que se encerra com exemplos e com a análise do grau de complexidade do problema em análise.

3.1 Serviços e Operações: Terminologia

Um serviço é formado por operações. As operações podem representar quaisquer comandos necessários à definição de processos, tais como, comandos de laço, decisão, invocação, terminação etc. Entretanto, este trabalho é focado nas operações que invocam outros serviços e, salvo menção contrária, doravante o termo operação será utilizado somente neste sentido. A invocação pode ser explícita, feita através da especificação da URL do serviço (oportunamente, esta operação será definida como concreta); ou implícita, feita através da especificação das mensagens abstratas de entrada e saída do serviço (operação abstrata). Um serviço também pode apresentar pré e pós-condições. A Figura 3-1 é uma representação simplificada de um serviço (a) através de um diagrama de classes da notação Unified Modeling Language (UML) e (b) de seu diagrama de processo usando notação BPMN [133].

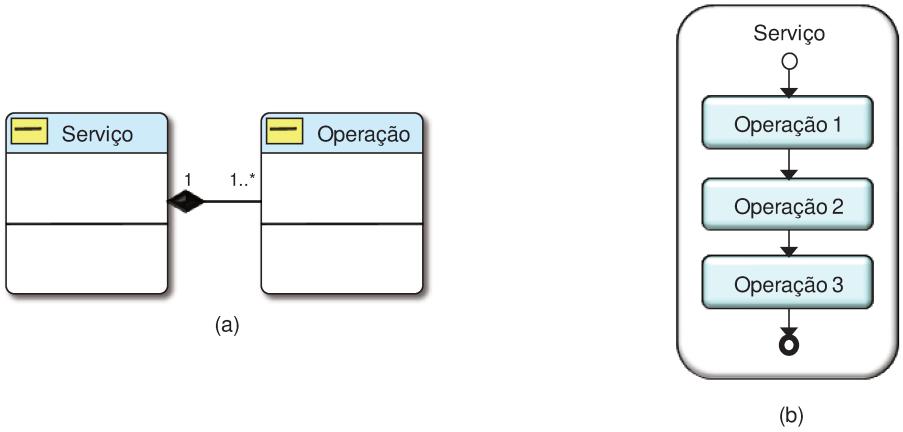

Figura 3-1 – Estrutura interna simplificada de um serviço, em duas versões: (a) o modelo de classes usando a notação UML e (b) um exemplo de diagrama de processo usando notação BPMN.

Definição 3-1 (Terminologia) *Seja T a terminologia da ontologia utilizada pelos graus de casamento e pelas regras de composição de serviços especificada em OWL-DL SHOIN(D) e CT^T a hierarquia de subsunção de conceitos de T.*

Definição 3-2 (Serviço) *Um serviço S possui um conjunto de parâmetros de entrada, $IN^S = \{in_1^S, in_2^S, \dots, in_\alpha^S\}$; um conjunto de parâmetros de saída, $OUT^S = \{out_1^S, out_2^S, \dots, out_\beta^S\}$, onde α e β representam a quantidade total dos parâmetros de entrada e saída, respectivamente, sendo $\alpha, \beta > 0$; e um conjunto de operações $OP^S = \{op_1^S, op_2^S, \dots, op_\gamma^S\}$, onde γ é a quantidade total de operações, sendo $\gamma \geq 1$. Por sua vez, cada operação op_i^S $(0 < i \leq \gamma)$ possui um conjunto de parâmetros de entrada, $IN^{op_i^S} = \{in_1^{op_i^S}, in_2^{op_i^S}, \dots, in_\delta^{op_i^S}\}$, e um conjunto de parâmetros de saída, $OUT^{op_i^S} = \{out_1^{op_i^S}, out_2^{op_i^S}, \dots, out_\varepsilon^{op_i^S}\}$, onde δ e ε representam a quantidade total dos parâmetros de entrada e saída, respectivamente; adicionalmente $\delta, \varepsilon > 0$. Um serviço também possui um conjunto de pré-condições, $PRE^S = \{pre_1^S, pre_2^S, \cdots, pre_\theta^S\}$, e pós-condições, $POS^S = \{pos_1^S, pos_2^S, \cdots, pos_\lambda^S\}$, sendo θ a quantidade de pré- condições e λ, a de pós-condições $(\theta, \lambda \geq 0)$. Os parâmetros in_j^S $(0 < j \leq \alpha)$, out_k^S $(0 < k \leq \beta)$, $in_l^{op_i^S}$ $(0 < l \leq \delta)$ e $out_m^{op_i^S}$ $(0 < m \leq \varepsilon)$ são conceitos terminológicos em CT^T.*

Definição 3-3 (Condição) *Uma condição C^A é uma expressão lógica dependente da variável A, cuja função de avaliação, $f(C^A)$, resulta em verdadeiro ou falso. A é um conceito terminológico em CT^T.*

Definição 3-4 (Pré-condição) *Uma pré-condição pre_i^S $(0 < i \leq \theta)$ é uma condição que deve ser satisfeita antes da execução do serviço para que a execução deste serviço seja bem sucedida.*

Definição 3-5 (Pós-condição) *Uma pós-condição pos_i^S $(0 < i \leq \lambda)$ é uma condição avaliada como verdadeira após a execução bem sucedida de um serviço.*

As operações de um serviço fazem parte de um *workflow* que descreve o fluxo de dados e de controle entre estas operações. O termo *workflow* é usado como definido pela WfMC [79].

Os serviços, assim como suas respectivas operações, são classificados em dois eixos distintos: no primeiro, podem ser atômicos ou compostos; no segundo, abstratos ou concretos.

Definição 3-6 (Serviço Atômico) *Seja AtS um serviço com uma única operação, isto é, $\gamma = 1$; então, AtS é um serviço atômico.*

Os serviços atômicos possuem implementações que geralmente oferecem acesso aos recursos computacionais do sistema em que tomam parte (banco de dados e sistemas legados, por exemplo) e são utilizados pelos serviços compostos na execução de tarefas maiores e mais complexas.

Definição 3-7 (Serviço Composto) *CpS é um serviço composto se as duas condições seguintes forem satisfeitas: (a) suas operações se confundem com as atividades de um workflow; e (b) uma ou mais de suas operações invocam outros serviços.*

A Definição 3-7 é de cunho estrutural, pois, sob o ponto de vista externo, mesmo um serviço composto é funcionalmente considerado atômico, uma vez que sua estrutura interna não é exposta.

Definição 3-8 (Repositório de Serviços Concretos) *Seja CcS um repositório de serviços concretos; então $CcS = \{CcS_1, CcS_2, \dots, CcS_\mu\}$, onde CcS_i $(1 \leq i \leq \mu)$ é um serviço concreto, e μ é a quantidade de serviços concretos disponíveis.*

Definição 3-9 (Serviço Concreto) *Seja CcS_i um serviço concreto; então, CcS_i $(1 \leq i \leq \mu)$ pertence a um repositório de serviços concretos ($CcS_i \in CcS$) e suas operações também são concretas.*

Definição 3-10 (Operação Concreta) *Uma operação op_i^{CcS} é concreta se uma das duas condições a seguir for satisfeita: (a) possuir implementação; (b) invocar um serviço concreto.*

Pelas Definições 3-9 e 3-10b, um serviço concreto possui operações concretas que invocam serviços concretos. Para que esta definição não seja infinitamente recursiva, basta que o último elo desta cadeia (ou seja, a última operação) tenha uma implementação (Definição 3-10a).

Definição 3-11 (Serviço Abstrato) *Um serviço AbS é abstrato se as três condições seguintes forem satisfeitas: (a) $PRE^{AbS} = \{\ \}$, isto é, $\theta = 0$; (b) $POS^{AbS} = \{\ \}$ $(\lambda = 0)$; e (c) suas operações forem abstratas.*

Definição 3-12 (Operação Abstrata) *Uma operação op_i^{AbS} $(0 < i \leq \gamma)$ é abstrata se possuir apenas a especificação da sua interface*[35]*.*

Os parâmetros de entrada e de saída de uma operação abstrata podem ser compreendidos como requisitos que devem ser satisfeitos por algum serviço concreto ou, em outras palavras, são tomados como uma, ou parte de uma, consulta (*query*) ao repositório de serviços concretos.

Como consequência destas definições, tem-se que os serviços, concretos e abstratos, também podem ser classificados como atômicos ou compostos.

[35] Em um documento WSDL, a definição de serviço inclui a definição de uma interface e uma de implementação. A descrição da interface, independentemente dos detalhes de implementação, é chamada de abstrata. A localização e as informações de implementação específicas de um serviço Web em particular constituem as partes concretas de um documento WSDL [32].

3.2 Graus de Similaridade

Definição 3-13 (Similaridade) *Sejam A e B dois conceitos em CT^T; então, $SIM(A,B)$ denota a similaridade entre os conceitos A e B, tal que:*

$$SIM(A,B) \Leftrightarrow SIM^E(A,B) \vee SIM^{AI}(A,B) \vee SIM^{DI}(A,B) \vee SIM^S(A,B) \qquad (3.1)$$

Definição 3-14 (Similaridade Exata) *$SIM^E(A,B) \Rightarrow$ O conceito A é equivalente ao conceito B em CT^T.*

Definição 3-15 (Similaridade Ascendência Direta) *$SIM^{AD}(A,B) \Rightarrow$ O conceito A é superconceito imediato do conceito B em CT^T.*

Definição 3-16 (Similaridade Ascendência Indireta) *$SIM^{AI}(A,B) \Rightarrow$ O conceito A é superconceito do conceito B em CT^T.*

Definição 3-17 (Similaridade Descendência Direta) *$SIM^{DD}(A,B) \Leftrightarrow SIM^{AD}(B,A)$. Isto é, o conceito A é subconceito direto do conceito B em CT^T.*

Definição 3-18 (Similaridade Descendência Indireta) *$SIM^{DI}(A,B) \Leftrightarrow SIM^{AI}(B,A)$. Ou seja, o conceito A é subconceito do conceito B em CT^T.*

Definição 3-19 (Similaridade Sintática) *$SIM^S(A,B) \Rightarrow$ O conceito A é similar ao conceito B se o grau numérico de similaridade sintática entre as strings A e B de acordo com a métrica de RI escolhida for igual ou maior que o limiar η estabelecido, sendo $\eta \in [0,1]$.*

O conceito de similaridade definido pela Equação 3.1 é bastante abrangente. Entretanto, dependendo da necessidade, pode ser mais restritivo. Basta restringir os tipos de similaridade utilizadas na Equação 3.1.

3.3 Graus de Casamento

Os graus de casamento indicam a similaridade entre um serviço concreto e uma operação abstrata. Em última instância, eles sinalizam a qualidade com que um serviço

concreto pode ser interpretado como uma implementação de uma operação abstrata. Os graus foram adaptados a partir dos filtros propostos por Klusch et al. [63] (doravante designados por "filtros de Klusch", em atenção à facilidade de comparação e diferenciação).

Os três primeiros graus de casamento são baseados apenas em regras lógicas. Os dois últimos são regras híbridas que também levam em consideração a similaridade sintática. Os graus são apresentados em ordem decrescente de similaridade: do primeiro ao quinto. Mas, antes, é necessário definir os termos soma, precedência e operação polivalente.

Definição 3-20 (Soma) *Sejam A e B dois conjuntos de conceitos terminológicos em CT^T; então, $A \dot{+} B$ é o conjunto formado pelos conceitos de A e pelos conceitos de B, de tal forma que, em caso de similaridade entre os conceitos, mantenha-se o mais genérico deles, de acordo com os graus de similaridade anteriormente estabelecidos.*

Por exemplo, considere os conjuntos $A = \{a, b1\}$ e $B = \{b, c\}$, sendo $a, b1, b$ e c conceitos terminológicos em CT^T, de tal forma que $SIM^{AI}(b, b1)$; então, $A \dot{+} B = \{a, b1\} \dot{+} \{b, c\} = \{a, b, c\}$.

Definição 3-21 (Precedência) *Uma operação op_i^S precede uma operação op_j^S se op_i^S é sempre executada antes de op_j^S, ou seja, $op_i^S \prec op_j^S \; para \; 0 < i, j \leq \gamma$.*

Definição 3-22 (Operação Polivalente) *Uma operação polivalente, pv_i^{AbS}, é uma operação fictícia relativa à operação abstrata $op_i^{AbS} (0 < i \leq \gamma)$ do serviço abstrato AbS. Os parâmetros de entrada da operação pv_i^{AbS} são formados por todos os parâmetros de entrada e saída das operações precedentes à operação op_i^{AbS}, incluindo os desta última; e, os parâmetros de saída de pv_i^{AbS} são iguais aos da operação op_i^{AbS}. Toda operação op_i^{AbS} possui sua equivalente pv_i^{AbS}. Formalmente, tem-se que:*

$$IN^{pv_i^{AbS}} = IN^{op_1^{AbS}} \dot{+} IN^{op_2^{AbS}} \dot{+} \ldots \dot{+} IN^{op_i^{AbS}} \dot{+}$$

$$OUT^{op_1^{AbS}} \dot{+} OUT^{op_2^{AbS}} \dot{+} \ldots \dot{+} OUT^{op_{i-1}^{AbS}} \quad (3.2)$$

$$OUT^{pv_i^{AbS}} = OUT^{op_i^{AbS}} \quad (3.3)$$

Doravante, considerar:

- $0 < i \leq \gamma$ (sendo γ a quantidade total de operações de um serviço, isto é, $\gamma = |\mathrm{OP^S}|$);
- Os serviços concretos CcS_k e CcS_l, tal que $0 < k, l \leq \mu$ (sendo μ a quantidade de serviços concretos disponíveis, $\mu = |\mathrm{CcS}|$);
- $1 < m \leq |IN^{CcS_k}|$ ($|IN^{CcS_k}|$ denota a quantidade total de parâmetros de entrada de CcS_k);
- $1 < r \leq |OUT^{CcS_k}|$;
- $1 < n \leq \left|IN^{pv_i^{AbS}}\right|$;
- $1 < q \leq \left|OUT^{pv_i^{AbS}}\right|$;
- $1 < p \leq |IN^{CcS_l}|$;
- $1 < h \leq |PRE^{CcS_k}|$;
- $1 < j \leq |PRE^{CcS_l}|$.

3.3.1 Casamento de Primeiro Grau

O casamento de primeiro grau entre um serviço concreto CcS_k e uma operação abstrata op_i^{AbS} ocorre quando: (a) existir para todo parâmetro de entrada do serviço concreto, um parâmetro de entrada da operação polivalente pv_i^{AbS} com similaridade

exata; e, (b) existir para cada parâmetro de saída da operação polivalente, um parâmetro de saída do serviço concreto com similaridade também exata. Formalmente, tem-se que:

$$\left(\forall\, in_m^{CcS_k} \exists\, in_n^{pv_i^{AbS}} : SIM^E\left(in_m^{CcS_k}, in_n^{pv_i^{AbS}}\right)\right) \wedge$$

$$\left(\forall\, out_q^{pv_i^{AbS}} \exists\, out_r^{CcS_k} : SIM^E\left(out_q^{pv_i^{AbS}}, out_r^{CcS_k}\right)\right) \quad (3.4)$$

Esta regra permite que a similaridade dos parâmetros de entrada do serviço concreto possa ser verificada com qualquer parâmetro, tanto de entrada quanto de saída, previamente disponível no *workflow*, porque a similaridade é verificada com os parâmetros da operação polivalente pv_i^{AbS} e não com os da operação abstrata op_i^{AbS}. Ainda, esta regra permite que um serviço concreto tenha menos parâmetros de entrada que a operação polivalente e, ao mesmo tempo, possua mais parâmetros de saída, ou seja, permite que o serviço concreto "ofereça mais por menos". Consequentemente, a possibilidade de casamento é aumentada. Esta característica é mantida pelas próximas regras, afrouxando-se somente as restrições de similaridade.

A Figura 3-2 ilustra o casamento de primeiro grau que ocorre entre a operação polivante pv_i^{AbS} e o serviço concreto CcS_k. A operação polivalente possui três parâmetros de entrada, A1.1, B1 e C2, e um único parâmetros de saída R1. Por sua vez, o serviço concreto, CcS_k, possui dois parâmetros de entrada A1.1 e B1, e dois parâmetros de saída, R1 e S1.2. Supor que o parâmetro $A1.1$, na verdade um conceito terminológico em CT^{T}, é um subconceito de $A1$; da mesma forma que $A1$ é subconceito de A. Raciocínio análogo é válido para os demais parâmetros. O serviço CcS_k possui casamento de primeiro grau com a operação polivante pv_i^{AbS} porque todos os seus parâmetros de entrada possuem similaridade exata de com um dos parâmetros de entrada da operação polivante, da mesma forma que todos parâmetros de saída da operação polivante possuem similaridade exata com um dos parâmetros de saída do serviço concreto. Abordagens equivalentes serão utilizadas para ilustrar os próximos graus de casamento.

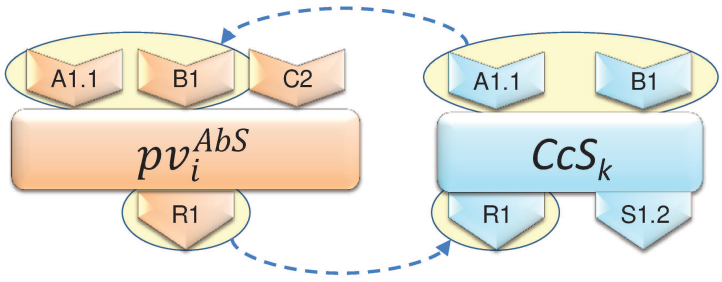

Figura 3-2 – Casamento de primeiro grau.

3.3.2 Casamento de Segundo Grau

O casamento de segundo grau ocorre quando os parâmetros de entrada do serviço concreto possuem similaridade do tipo exata ou ascendência direta com um parâmetro de entrada da operação polivalente, enquanto o parâmetro de saída do serviço concreto pode ter similaridade do tipo exata ou descendência direta com um parâmetro da operação polivalente:

$$\left(\forall\, in_m^{CcS_k} \exists\, in_n^{pv_i^{AbS}} : SIM^E\left(in_m^{CcS_k}, in_n^{pv_i^{AbS}}\right) \vee SIM^{AD}\left(in_m^{CcS_k}, in_n^{pv_i^{AbS}}\right)\right) \wedge$$
$$\left(\forall\, out_q^{pv_i^{AbS}} \exists\, out_r^{CcS_k} : SIM^E\left(out_r^{CcS_k}, out_q^{pv_i^{AbS}}\right) \vee SIM^{DD}\left(out_r^{CcS_k}, out_q^{pv_i^{AbS}}\right)\right) \tag{3.5}$$

Resumidamente, casamento de grau dois indica que os parâmetros de entrada do serviço concreto podem ser mais genéricos, e os parâmetros de saída mais específicos do que os parâmetros correspondentes da operação polivalente (Figura 3-3).

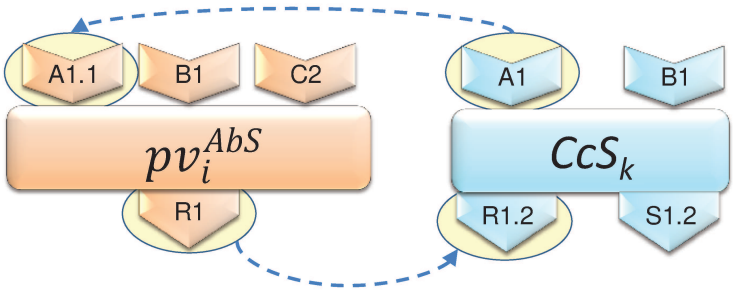

Figura 3-3 – Casamento de segundo grau.

3.3.3 Casamento de Terceiro Grau

No casamento de terceiro grau, a similaridade passa a ser indireta. Os parâmetros de entrada do serviço concreto podem ter tanto similaridade do tipo exata quanto do tipo ascendência indireta. E os parâmetros de saída do serviço concreto podem ter similaridade do tipo exata ou do tipo descendência indireta em relação aos respectivos parâmetros da operação polivalente:

$$\left(\forall\, in_m^{CcS_k}\ \exists\, in_n^{pv_i^{AbS}} : SIM^E\left(in_m^{CcS_k}, in_n^{pv_i^{AbS}}\right) \vee SIM^{AI}\left(in_m^{CcS_k}, in_n^{pv_i^{AbS}}\right)\right) \wedge$$
$$\left(\forall\, out_q^{pv_i^{AbS}}\ \exists\, out_r^{CcS_k} : SIM^E\left(out_r^{CcS_k}, out_q^{pv_i^{AbS}}\right) \vee SIM^{DI}\left(out_r^{CcS_k}, out_q^{pv_i^{AbS}}\right)\right) \tag{3.6}$$

A Figura 3-4 ilustra o casamento de terceiro grau.

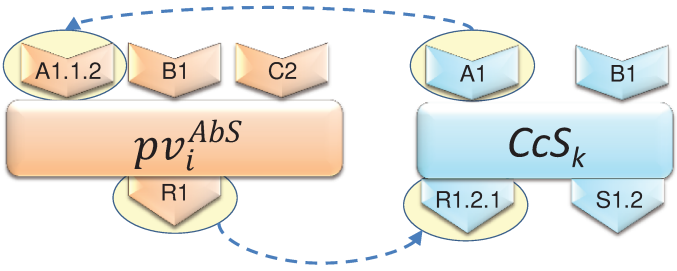

Figura 3-4 – Casamento de terceiro grau.

3.3.4 Casamento de Quarto Grau

No casamento de quarto grau, as regras para os parâmetros de entrada permanecem as mesmas do grau anterior, entretanto, os parâmetros de saída do serviço concreto agora são ascendentes diretos dos respectivos parâmetros da operação polivalente, além da necessidade deve haver similaridade sintática entre eles:

$$\left(\forall\, in_m^{CcS_k}\ \exists\, in_n^{pv_i^{AbS}} : SIM^E\left(in_m^{CcS_k}, in_n^{pv_i^{AbS}}\right) \vee SIM^{AI}\left(in_m^{CcS_k}, in_n^{pv_i^{AbS}}\right)\right) \wedge$$

$$\left(\forall\, out_q^{pv_i^{AbS}}\ \exists\, out_r^{CcS_k} : \left(SIM^E\left(out_r^{CcS_k}, out_q^{pv_i^{AbS}}\right) \vee SIM^{AD}\left(out_r^{CcS_k}, out_q^{pv_i^{AbS}}\right)\right) \wedge\right.$$

$$\left. SIM^S\left(out_r^{CcS_k}, out_q^{pv_i^{AbS}}\right)\right) \tag{3.7}$$

Os parâmetros de saída do serviço concreto devem ser ascendentes diretos dos parâmetros da operação polivalente para eliminar o retorno de dados genéricos demais (Figura 3-5). Entretanto, dependendo do domínio da aplicação esta restrição pode ser abrandada. Esta é uma regra híbrida uma vez que extrapola os domínios do raciocínio baseado em lógica pela adição de técnicas de recuperação de informação. A próxima regra também é híbrida.

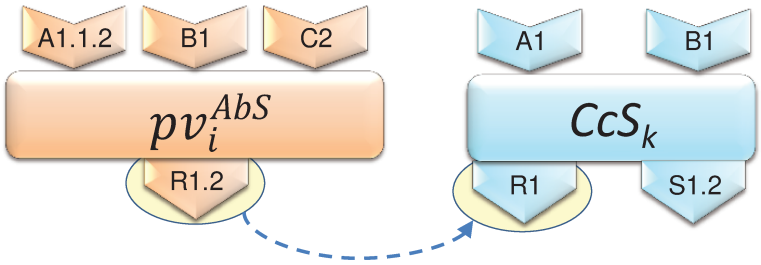

Figura 3-5 – Casamento de quarto grau.

3.3.5 Casamento de Quinto Grau

Neste último tipo de casamento, as regras com relação aos parâmetros de entrada continuam as mesmas das duas regras anteriores, entretanto, os parâmetros de saída do serviço concreto não precisam mais ser ascendentes diretos dos respectivos parâmetros da operação polivalente, mas mantém-se a necessidade de haver similaridade sintática entre eles. Este é o grau mais abrangente entre todos:

$$\left(\forall\, in_m^{CcS_k} \exists\, in_n^{pv_i^{AbS}} : SIM^E\left(in_m^{CcS_k}, in_n^{pv_i^{AbS}}\right) \vee SIM^{AI}\left(in_m^{CcS_k}, in_n^{pv_i^{AbS}}\right)\right) \wedge$$

$$\left(\forall\, out_q^{pv_i^{AbS}} \exists\, out_r^{CcS_k} : SIM^E\left(out_r^{CcS_k}, out_q^{pv_i^{AbS}}\right) \vee SIM^{AI}\left(out_r^{CcS_k}, out_q^{pv_i^{AbS}}\right) \vee\right.$$

$$\left. SIM^S\left(out_r^{CcS_k}, out_q^{pv_i^{AbS}}\right)\right) \tag{3.8}$$

A Figura 3-6 ilustra o casamento de quinto grau.

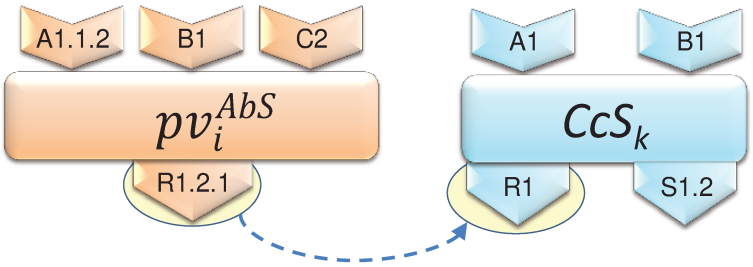

Figura 3-6 – Casamento de quinto grau.

3.4 Regras de Composição

As regras de composição fazem uso da mesma terminologia T utilizada pelos graus de casamento. Os objetivos das regras de composição são: (a) verificar se dois serviços podem ser compostos; (b) sendo possível a composição, estabelecer o modo de fazê-la para os seguintes padrões de fluxo de controle: sequencial, separação paralela, união síncrona, escolha-exclusiva, união simples e laço.

Antes das regras propriamente ditas, é necessário definir os conceitos de diferença, exclusividade e mais específico.

Definição 3-23 (Diferença) *Seja $\dot{-}$ a denotação para o conceito de diferença, de tal forma que, $A \dot{-} B$ é o conjunto formado pelos conceitos de A que não possuem similaridade com nenhum conceito de B, sendo A e B dois conjuntos de conceitos terminológicos em CT^T.*

Definição 3-24 (Exclusividade) *Sejam A e B dois conjuntos de conceitos terminológicos em CT^T; então, $Exclusividade(A,B)$ é o conjunto dos conceitos de A e de B que não possuem similaridade entre si, ou seja:*

$$Exclusividade(A,B) = (A \dot{-} B) \dot{+} (B \dot{-} A) \tag{3.9}$$

Definição 3-25 (Conceito Mais Específico) *Sejam A e B dois conjuntos de conceitos terminológicos em CT^T; então, CME(A, B) é o conjunto formado pelos conceitos mais específicos (descendentes diretos ou indiretos) de A e de B que possuem similaridade entre si.*

3.4.1 Composição Sequencial

O padrão de fluxo de controle sequencial entre dois serviços quaisquer pode ocorrer quando pelo menos um dos parâmetros de saída de um dos serviços possui similaridade com algum dos parâmetros de entrada do outro serviço.

Definição 3-26 (Composição Sequencial) *Seja $\prec$ a denotação para a composição sequencial, então a composição $CcS_k \prec CcS_l$ pode ser realizada se $PRE^{CcS_l} = \{\ \}$ e se houver similaridade entre os parâmetros de entrada de CcS_l e os de saída de CcS_k:*

$$\exists\left(in_p^{CcS_l}, \mathrm{out}_r^{CcS_k}\right): SIM\left(in_p^{CcS_l}, \mathrm{out}_r^{CcS_k}\right) \tag{3.10}$$

Se for desejável, é possível restringir a composição sequencial apenas designando uma similaridade mais específica (ver seção 3.2).

Uma vez verificado que é possível realizar a composição sequencial entre os serviços CcS_k e CcS_l, é necessário descrever o resultado desta composição, ou seja, como o serviço composto, designado por CpS, é criado (o símbolo ← denota atribuição). Segue a descrição de como criá-lo:

- É criada uma operação, op_1^{CpS}, com os mesmos parâmetros de entrada e de saída que o serviço CcS_k: $IN^{op_1^{CpS}} \leftarrow IN^{CcS_k}$ e $OUT^{op_1^{CpS}} \leftarrow OUT^{CcS_k}$. Esta operação invoca o serviço CcS_k;

- É criada uma operação, op_2^{CpS}, com os mesmos parâmetros de entrada e saída que o serviço CcS_l: $IN^{op_2^{CpS}} \leftarrow IN^{CcS_l}$ e $OUT^{op_2^{CpS}} \leftarrow OUT^{CcS_l}$. Esta operação invoca o serviço CcS_l;
- O serviço composto CpS é criado e formado pelo conjunto de operações $OP^{CpS} \leftarrow \{op_1^{CpS}, op_2^{CpS}\}$, sendo que $(op_1^{CpS} \prec op_2^{CpS})$;
- As pré-condições do serviço CpS são as mesmas do serviço CcS_k: $PRE^{CpS} \leftarrow PRE^{CcS_k}$;
- O conjunto das pós-condições do serviço CpS é formado pelas pós-condições dos serviços CcS_k e CcS_l: $POS^{CpS} \leftarrow POS^{CcS_k} + POS^{CcS_l}$;
- Para os parâmetros de entrada do serviço CpS : $IN^{CpS} \leftarrow Exclusividade\left(IN^{op_1^{CpS}}, IN^{op_2^{CpS}} \dot{-} OUT^{op_1^{CpS}}\right) \dot{+} CME\left(IN^{op_1^{CpS}}, IN^{op_2^{CpS}} \dot{-} OUT^{op_1^{CpS}}\right)$;
- Para os parâmetros de saída do serviço CpS : $OUT^{CpS} \leftarrow OUT^{op_2^{CpS}} \dot{+} \left(OUT^{op_1^{CpS}} \dot{-} IN^{op_2^{CpS}}\right)$.

As duas últimas regras garantem a factibilidade do novo serviço criado, uma vez que: (a) são adicionados como parâmetros de entrada do serviço CpS os parâmetros exclusivos (não possuem similaridade) entre $IN^{op_1^{CpS}}$ e $\left(IN^{op_2^{CpS}} \dot{-} OUT^{op_1^{CpS}}\right)$; (b) e, entre os similares, apenas os mais específicos: $CME\left(IN^{op_1^{CpS}}, IN^{op_2^{CpS}} \dot{-} OUT^{op_1^{CpS}}\right)$. Deve-se lembrar que $IN^{op_2^{CpS}} \dot{-} OUT^{op_1^{CpS}}$ designa todos os parâmetros de entrada da operação op_2^{CpS} que não possuem similaridade com os parâmetros de saída da operação op_1^{CpS}.

Por sua vez, parâmetros de saída do serviço CpS são formados por todos os parâmetros de saída da operação op_2^{CpS} e por todos os parâmetros de saída da operação op_1^{CpS} que não possuem similaridade com os parâmetros de entrada da operação op_2^{CpS}. Em caso de similaridade, são adicionados os mais genéricos (por definição, a soma mantém os conceitos mais genéricos, ver Definição 3.20).

3.4.2 Composição Paralela

A Workflow Patterns Initiative [80] define dois tipos diferentes de fluxos relacionados à execução paralela: separação paralela e sincronização. Neste trabalho, toda separação paralela é sempre seguida pela sincronização. Na prática, esta limitação significa que todas as vezes que um fluxo é dividido, ele será sincronizado novamente. Esta restrição garante a factibilidade da composição, conforme definido a seguir.

Definição 3-27 (Composição paralela) *Seja* ‖ *a denotação para a composição paralela, então dois serviços concretos quaisquer,* CcS_k *e* CcS_l*, podem ser compostos em paralelo,* $CcS_k \parallel CcS_l$*, se* $PRE^{CcS_k} = PRE^{CcS_l} = \{\ \}$.

Para a criação do serviço composto CpS, os passos são:

- É criada uma operação, op_1^{CpS}, com os mesmos parâmetros de entrada e de saída que o serviço CcS_k: $IN^{op_1^{CpS}} \leftarrow IN^{CcS_k}$ e $OUT^{op_1^{CpS}} \leftarrow OUT^{CcS_k}$. Esta operação invoca o serviço CcS_k;
- É criada uma operação, op_2^{CpS}, com os mesmos parâmetros de entrada e saída que o serviço CcS_l: $IN^{op_2^{CpS}} \leftarrow IN^{CcS_l}$ e $OUT^{op_2^{CpS}} \leftarrow OUT^{CcS_l}$. Esta operação invoca o serviço CcS_l;
- O serviço composto CpS é criado e formado pelo conjunto de operações $OP^{CpS} \leftarrow \{op_1^{CpS}, op_2^{CpS}\}$, tal que $(op_1^{CpS} \parallel op_2^{CpS})$;
- O conjunto das pós-condições do serviço CpS é formado pelas pós-condições dos serviços CcS_k e CcS_l: $POS^{CpS} \leftarrow POS^{CcS_k} + POS^{CcS_l}$;
- Os parâmetros de entrada do serviço CpS : $IN^{CpS} \leftarrow Exclusividade\left(IN^{op_1^{CpS}}, IN^{op_2^{CpS}}\right) \dot{+} CME\left(IN^{op_1^{CpS}}, IN^{op_2^{CpS}}\right)$, isto é, o conjunto dos parâmetros de entrada do serviço CpS é formado pelos parâmetros exclusivos e mais específicos entre as duas operações;
- Os parâmetros de saída do serviço CpS: $OUT^{CpS} \leftarrow OUT^{op_1^{CpS}} \dot{+} OUT^{op_2^{CpS}}$.

Algumas variações destas duas últimas regras podem ser adotadas. Por exemplo, pode-se copiar para o serviço CpS todos os parâmetros de entrada da operação op_1^{CpS} mais os parâmetros de entrada da operação op_2^{CpS} que não se casam com os parâmetros de entrada da operação op_1^{CpS}; ou ainda, pode-se optar por copiar sempre os parâmetros mais genéricos entre as duas operações. Para a adoção de uma destas variações deve-se levar em julgamento o tipo de aplicação, a ontologia em uso etc.

3.4.3 Composição do Tipo Escolha-Exclusiva

A composição do tipo escolha-exclusiva ocorre quando uma operação faz restrições explícitas, na forma de pré-condições, quanto aos valores que seus parâmetros de entrada podem assumir. A forma como se dará a composição dependerá da relação entre as pré-condições dos serviços envolvidos. Em alguns casos, por exemplo, é possível realizar uma composição sequencial e eliminar a restrição imposta pela pré-condição. Em outros não. Primeiramente, analisar-se-á o inter-relacionamento entre duas condições.

Definição 3-28 (Condições Complementares) *Sejam C^A e C^B duas condições; então, $CompCond(C^A, C^B) \Rightarrow SIM^E(A,B) \wedge \big(dom(A) = dom(B)\big) \wedge (\forall x \in dom(A): A = x, B = x \Rightarrow f(C^A) \vee f(C^B) = true)$; sendo $dom(A)$ e $dom(B)$ o domínio das varáveis A e B, respectivamente.*

Em linhas gerais, duas condições são complementares se satisfizer as seguintes restrições: a) houver similaridade exata entre os conceitos terminológicos de suas variáveis; b) as variáveis das duas condições possuírem os mesmos domínios; e, c) se, para qualquer valor que suas variáveis possam assumir, a avaliação de pelo menos uma das condições for verdadeira.

Definição 3-29 (Condições Parcialmente Complementares) *Sejam C^A e C^B duas condições então: $PartCompCond(C^A, C^B) \Rightarrow SIM^E(A,B) \wedge \big(dom(A) = dom(B)\big) \wedge (\exists x \in dom(A): A = x, B = x \Rightarrow f(C^A) \vee f(C^B) = false)$.*

Duas condições são parcialmente complementares se houver um valor que suas variáveis possam assumir tal que as avaliações de ambas as condições, para este valor de variável, resultem em falso.

Definição 3-30 (Composição do tipo escolha-exclusiva) *Seja $\oplus$ a denotação para a composição de acordo com o padrão fluxo escolha-exclusiva, então dois serviços concretos quaisquer, CcS_k e CcS_l, poderão ser composto de acordo com este padrão sempre que uma das seguintes condições for satisfeita (devem ser verificadas na ordem em que são apresentadas):*

a) As pré-condições de CcS_k e CcS_l, PRE^{CcS_k} e PRE^{CcS_l}, respectivamente, são complementares: $\forall\left(pre_h^{CcS_k}, pre_j^{CcS_l}\right) \Rightarrow CompCond\left(pre_h^{CcS_k}, pre_j^{CcS_l}\right)$, sendo $pre_h^{CcS_k} \in PRE^{CcS_k}$ e $pre_j^{CcS_l} \in PRE^{CcS_l}$;

b) As pré-condições de CcS_k e CcS_l são parcialmente complementares: $\forall\left(pre_h^{CcS_k}, pre_j^{CcS_l}\right) \Rightarrow PartCompCond\left(pre_h^{CcS_k}, pre_j^{CcS_l}\right)$, sendo $pre_h^{CcS_k} \in PRE^{CcS_k}$ e $pre_j^{CcS_l} \in PRE^{CcS_l}$.

Se a condição descrita em (a) for satisfeita, a criação do serviço composto, CpS, segue os passos ditados abaixo:

i. É criada uma operação, op_1^{CpS}, com os mesmos parâmetros de entrada e de saída que o serviço CcS_k: $IN^{op_1^{CpS}} \leftarrow IN^{CcS_k}$ e $OUT^{op_1^{CpS}} \leftarrow OUT^{CcS_k}$. Esta operação invoca o serviço CcS_k;

ii. É criada uma operação, op_2^{CpS}, com os mesmos parâmetros de entrada e saída que o serviço CcS_l: $IN^{op_2^{CpS}} \leftarrow IN^{CcS_l}$ e $OUT^{op_2^{CpS}} \leftarrow OUT^{CcS_l}$. Esta operação invoca o serviço CcS_l;

iii. O serviço composto CpS é criado e formado pelo conjunto de operações $OP^{CpS} \leftarrow \{op_1^{CpS}, op_2^{CpS}\}$, tal que $(op_1^{CpS} \oplus op_2^{CpS})$;

iv. Para os parâmetros de entrada do serviço CpS : $IN^{CpS} \leftarrow Exclusividade\left(IN^{op_1^{CpS}}, IN^{op_2^{CpS}}\right) \dot{+} CME\left(IN^{op_1^{CpS}}, IN^{op_2^{CpS}}\right)$;

v. Para os parâmetros de saída do serviço CpS: $OUT^{CpS} \leftarrow OUT^{op_1^{CpS}} \dot{+} OUT^{op_2^{CpS}}$;

vi. O conjunto de pós-condições do serviço CpS é formado pelas pós-condições dos serviços CcS_k e CcS_l: $POS^{CpS} \leftarrow POS^{CcS_k} + POS^{CcS_l}$;

vii. Um comando de decisão é criado para avaliar as pré-condições de CcS_k, PRE^{CcS_k}. A saída verdadeira é ligada à operação op_1^{CpS} e, a falsa, à operação op_2^{CpS}. As saídas das operações op_1^{CpS} e op_2^{CpS}são ligadas pelo fluxo de união simples. O novo serviço criado não possui pré-condição.

Definição 3-31 (Operação op^{end}) *A operação op^{end} provoca o término abrupto do fluxo de execução de um workflow. Não possui dados de entrada e nem de saída.*

Se a condição (b) e não a (a) for satisfeita, então os passos acima devem ser transcritos como:

i. Idem;

ii. Idem;

iii. CpS é criado e formado pelo conjunto de operações $OP^{CpS} \leftarrow \{op_1^{CpS}, op_2^{CpS}, op^{end}\}$, tal que $(op_1^{CpS} \oplus op_2^{CpS} \oplus op^{end})$;

iv. Idem;

v. Idem;

vi. Idem;

vii. Dois comandos de decisão são criados, o primeiro para avaliar as pré-condições de CcS_k, PRE^{CcS_k}. A saída verdadeira deste é ligada à operação op_1^{CpS} e, a falsa, ao segundo comando de decisão, que avalia as pré-condições de CcS_l, PRE^{CcS_l}. A saída verdadeira deste segundo comando de decisão é ligada à operação op_2^{CpS}; e a saída falsa, à operação op^{end}. As saídas das operações op_1^{CpS} e op_2^{CpS}são ligadas pelo fluxo união simples;

viii. A pré-condição $PRE^{CpS} = \{pre^{CcS_k}, pre^{CcS_l}\}$ é formada pela união das pré-condições dos serviços CcS_k e CcS_l.

3.4.4 Composição em Laço

O laço é caracterizado por um ponto no *workflow* onde uma ou mais atividades podem ser executadas repetidamente. A necessidade da composição em laço é detectada pela existência de parâmetros dimensionais em serviços, tais como vetores, listas etc. Considere-se o seguinte exemplo: a entrada de um serviço é o número de identificação de um estudante e, sua saída, uma lista de todos os cursos em que o estudante está atualmente matriculado. Este serviço não pode ser diretamente composto através de fluxo sequencial com outro serviço que recebe apenas um curso, e não uma lista de cursos. Assim, a solução é criar uma composição em laço para retirar, ou ler, cada curso da lista. Wu et al. [129] fizeram esta mesma observação.

Definição 3-32 (Composição em laço) *Seja ⊙ a denotação para a composição em laço, então dois serviços quaisquer, por exemplo, CcS_k e CcS_l, devem ser compostos em laço, $CcS_k \odot CcS_l$, se um parâmetro de entrada do serviço CcS_l possuir dimensão $n-1$ e similaridade com um dos parâmetros de saída do serviço CcS_k cuja dimensão é n. A composição em laço é formada por: (a) um buffer de armazenamento de dados; (b) um comando de leitura do buffer; (c) um comando de decisão; (d) e um comando de desvio de execução. Adicionalmente, $PRE^{CcS_k} = PRE^{CcS_l} = \{\ \}$.*

Seguem as regras para formação do serviço composto CpS segundo o fluxo de controle em laço:

- É criada uma operação, op_1^{CpS}, com os mesmos parâmetros de entrada e de saída que o serviço CcS_k: $IN^{op_1^{CpS}} \leftarrow IN^{CcS_k}$ e $OUT^{op_1^{CpS}} \leftarrow OUT^{CcS_k}$. Esta operação invoca o serviço CcS_k;
- É criada uma operação, op_2^{CpS}, com os mesmos parâmetros de entrada e saída que o serviço CcS_l: $IN^{op_2^{CpS}} \leftarrow IN^{CcS_l}$ e $OUT^{op_2^{CpS}} \leftarrow OUT^{CcS_l}$. Esta operação invoca o serviço CcS_l;

- O serviço composto CpS é criado e formado pelo conjunto de operações $OP^{CpS} \leftarrow \{op_1^{CpS}, op_2^{CpS}\}$, tal que $(op_1^{CpS} \circledcirc op_2^{CpS})$. Os dados de saída do serviço op_1^{CpS} são armazenados no *buffer*. O primeiro parâmetro de dimensão $n-1$ é lido do *buffer*. O comando de decisão verifica se o parâmetro é não nulo. Em caso positivo, o fluxo de execução segue para op_2^{CpS}; após a execução desta operação, o fluxo retorna para a leitura do próximo parâmetro do armazenado no *buffer*. Em caso negativo, o fluxo é desviado para o térmico do *workflow*;
- Para os parâmetros de entrada do serviço CpS : $IN^{CpS} \leftarrow$ $Exclusividade\left(IN^{op_1^{CpS}}, IN^{op_2^{CpS}} \dot{-} OUT^{op_1^{CpS}}\right) \dot{+}$ $CME\left(IN^{op_1^{CpS}}, IN^{op_2^{CpS}} \dot{-} OUT^{op_1^{CpS}}\right)$;
- Para os parâmetros de saída do serviço CpS : $OUT^{CpS} \leftarrow OUT^{op_2^{CpS}} \dot{+} \left(OUT^{op_1^{CpS}} \dot{-} IN^{op_2^{CpS}}\right)$.

3.5 Potencialidades e Consequências dos Graus de Casamento e das Regras de Composição

O objetivo desta seção é ilustrar o potencial e as implicações das regras de casamento e de composição de serviços. Entretanto, somente algumas características, as mais interessantes, serão abordadas. Mas antes, será necessário apresentar a ontologia e os serviços utilizados.

A ontologia utilizada é ilustrada na Figura 3-7. Para a melhor compreensão e realização dos exemplos, os nomes conceituais são definidos de forma arbitraria, mas sistematicamente. Como o foco de atenção é o relacionamento hierárquico entre os conceitos, somente os relacionamentos de subsunção são essenciais e são definidos como segue: os conceitos A, B, C … são subconceitos de $Thing$. Os conceitos $A1, A2$ … são subconceitos de A; da mesma forma como $A1.1, A1.2$ … são subconceitos de $A1$ e assim

por diante. Nem todos os conceitos e subconceitos definidos nesta ontologia serão diretamente utilizados nos estudos de caso a seguir.

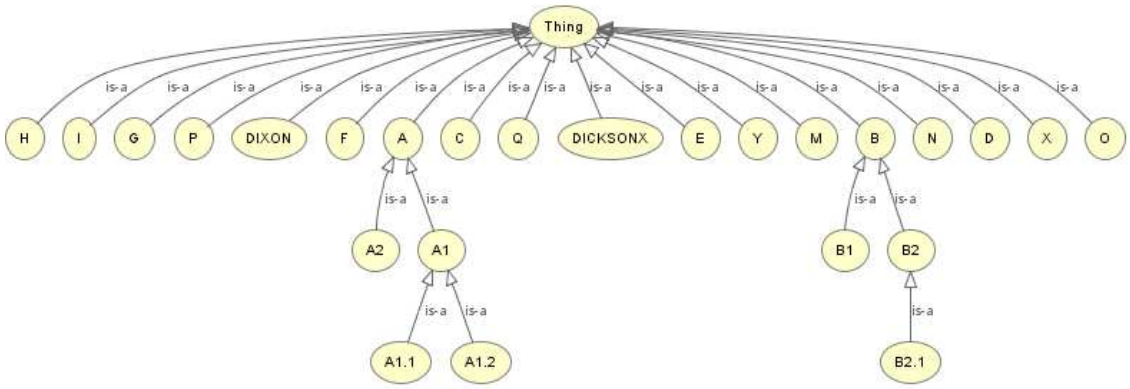

Figura 3-7 – Relacionamento hierárquico da ontologia de teste.

Seja o serviço abstrato AbS_1 formado pela operação abstrata $OP^{AbS_1} = \{op_1^{AbS_1}\}$. Como este serviço possui uma única operação, os parâmetros de entrada e saída de ambos coincidem, portanto, $IN^{AbS_1} = IN^{op_1^{AbS_1}} = \{A, B1\}$ e $OUT^{AbS_1} = OUT^{op_1^{AbS_1}} = \{C, D\}$. O serviço abstrato AbS_2é especificado como $OP^{AbS_2} = \{op_1^{AbS_2}, op_2^{AbS_2}\}$, sendo que $op_1^{AbS_2} \prec op_2^{AbS_2}$, $IN^{AbS_2} = \{A, B1\}$, $OUT^{AbS_2} = \{C, D\}$ e, para as operações, $IN^{op_1^{AbS_2}} = \{A, B1\}$, $OUT^{op_1^{AbS_2}} = \{X\}$, $IN^{op_2^{AbS_2}} = \{X\}$ e $OUT^{op_2^{AbS_2}} = \{C, D\}$. Estes e outros serviços abstratos estão descritos na Tabela 3-1.

Tabela 3-1 – Serviços abstratos utilizados para exemplificação.

Serviços Abstratos	P. Entrada do Serviço	P. Saída do Serviço	Tipo de Composição	Operações	P. de Entrada da Operação	P. de Saída da Operação
$\boldsymbol{AbS_1}$	*A, B1*	*C, D*	$op_1^{AbS_1}$	$op_1^{AbS_1}$	*A, B1*	*C, D*
$\boldsymbol{AbS_2}$	*A, B1*	*C, D*	$op_1^{AbS_2} \prec op_2^{AbS_2}$	$op_1^{AbS_2}$	*A, B1*	*X*
				$op_2^{AbS_2}$	*X*	*C, D*
$\boldsymbol{AbS_3}$	*M, N*	*P, Q*	$op_1^{AbS_3} \prec op_2^{AbS_3}$	$op_1^{AbS_3}$	*M, N*	*O*
				$op_2^{AbS_3}$	*O*	*P, Q*
$\boldsymbol{AbS_4}$	*O, I, Y*	*H, B2*	$op_1^{AbS_4} \parallel op_2^{AbS_4}$	$op_1^{AbS_4}$	*O, I*	*H*
				$op_2^{AbS_4}$	*O, Y*	*B2*

Sejam também os serviços concretos disponíveis no repositório $CcS = \{CcS_1, CcS_2, CcS_3, CcS_4, CcS_5, CcS_6, CcS_7, CcS_8, CcS_9, CcS_{10}, CcS_{11}\}$, cujos parâmetros de entrada, saída, pré e pós-condições estão descritos na Tabela 3-2.

Tabela 3-2 – Serviços concretos disponíveis.

Serviços Concretos	Parâmetros de Entrada	Parâmetros de Saída	Condição
$\boldsymbol{CcS_1}$	*A, B*	*C, D*	
$\boldsymbol{CcS_2}$	*A, B1*	*X*	
$\boldsymbol{CcS_3}$	*X*	*C, D*	
$\boldsymbol{CcS_4}$	*C, E*	*F, G*	
$\boldsymbol{CcS_5}$	*M, N*	*O*	
$\boldsymbol{CcS_6}$	*O, N*	*P, Q*	
$\boldsymbol{CcS_7}$	*O, Y*	*P, Q*	
$\boldsymbol{CcS_8}$	*O, I*	*H*	$pre_1^{CcS_8} = (I < 10)$
$\boldsymbol{CcS_9}$	*O, Y*	*Dixon*	
$\boldsymbol{CcS_{10}}$	*Dicksonx*	*B2.1*	
$\boldsymbol{CcS_{11}}$	*O, I*	*H*	$pre_1^{CcS_{11}} = (I \geq 10)$

A seguir, alguns estudos de caso serão realizados para ilustrar: (a) a influência da operação polivalente; (b) o funcionamento dos graus de casamento e das regras de composição; (c) a ligação existente entre composição sequencial e paralela; e (d) o uso das regras na criação de composições de composições.

3.5.1 Operação Polivalente

Os parâmetros previamente disponíveis no *workflow* podem influenciar positivamente o casamento entre as operações abstratas e os serviços concretos. Para exemplificar, supor que se queira encontrar o melhor casamento para o serviço AbS_3.

O serviço AbS_3 possui duas operações: $op_1^{AbS_3}$ e $op_2^{AbS_3}$, sendo que $op_1^{AbS_3} \prec op_2^{AbS_3}$. Destas, a operação $op_1^{AbS_3}$ casa-se exatamente com o serviço CcS_5 (casamento de primeiro grau), pois ambas possuem os mesmos parâmetros: M, N (entradas) e O (saída). Mas a operação $op_2^{AbS_3}$ não se casa com nenhum serviço concreto disponível. Entretanto, existem dois serviços, CcS_6 e CcS_7, que são bastante próximos desta operação; eles possuem os mesmos parâmetros de saída (P e Q) e, dos dois parâmetros de entrada, um se casa (parâmetro O) e o outro não (o parâmetro N de CcS_5 e o parâmetro Y de CcS_6). Se os parâmetros previamente disponíveis no *workflow* fossem ignorados, nenhum deles se casaria com a operação $op_2^{AbS_3}$.

Mas resultados diferentes ocorrem quando se leva em consideração os parâmetros previamente disponíveis no *workflow*. Para realizar o casamento da operação $op_2^{AbS_3}$, é criada a operação polivalente $pv_2^{AbS_3}$, tal que $IN^{pv_2^{AbS_3}} = \left\{IN^{op_1^{AbS_3}} \dotplus IN^{op_2^{AbS_3}} \dotplus OUT^{op_1^{AbS_3}}\right\} = \{M, N, O\}$ e $OUT^{pv_2^{AbS_3}} = \left\{OUT^{op_1^{AbS_3}}\right\} = \{P, Q\}$. Agora, o serviço CcS_6 possui casamento de primeiro grau com a operação $op_2^{AbS_3}$.

Observe-se que, para o casamento, não é necessário que todos os parâmetros de entrada da operação abstrata, ou melhor, da operação polivalente, possuam similaridade com os respectivos parâmetros do serviço concreto, mas todos os parâmetros de entrada do serviço concreto devem possuir similaridade com os parâmetros de entrada da

operação polivalente. Com relação aos parâmetros de saída, tem-se que todos da operação polivalente devem possuir similaridade com os do serviço concreto, mas nem todos do serviço concreto precisam possuir similaridade com os da operação polivalente.

3.5.2 Casamento e Composição

Para ilustrar a sinergia existente entre os graus de casamento e as regras de composição, considere-se, por exemplo, que se queira encontrar o melhor casamento para o serviço abstrato AbS_1. De todos os serviços presentes no repositório, o serviço concreto CcS_1 realiza o melhor casamento, de segundo grau, com o serviço AbS_1, porque o parâmetro de entrada B do serviço CcS_1 é ascendente direto do parâmetro de entrada de $B1$ do serviço abstrato AbS_1, enquanto os demais parâmetros possuem similaridade exata. Este é o melhor casamento possível, desconsiderando a possibilidade de composição.

Observe-se que o serviço abstrato AbS_2 possui os mesmos parâmetros que o serviço AbS_1, entretanto, AbS_2 é especificado como um *workflow* envolvendo as operações $op_1^{AbS_2}$ e $op_2^{AbS_2}$. Estas duas operações possuem casamento de primeiro grau com os serviços concretos CcS_2 e CcS_3, respectivamente.

Mas, para tornar o cenário mais interessante, suponha-se que o serviço CcS_1 não exista. Neste caso, nenhum serviço concreto satisfaria os requisitos de AbS_1 e, portanto, a composição $CcS_2 \prec CcS_3$ precisaria ser realizada.

A composição sequencial entre os serviços CcS_2 e CcS_3 ocorre da seguinte forma:

- É criada a operação $op_1^{CpS_1}$, sendo que $IN^{op_1^{CpS_1}} \leftarrow \{\mathrm{A}, \mathrm{B1}\}$ e $OUT^{op_1^{CpS_1}} \leftarrow \{\mathrm{X}\}$;
- É criada a operação $op_2^{CpS_1}$, sendo que $IN^{op_2^{CpS_1}} \leftarrow \{\mathrm{X}\}$ e $OUT^{op_2^{CpS_1}} \leftarrow \{\mathrm{C}, \mathrm{D}\}$;
- É criado um serviço composto CpS_1 (Tabela 3-3) formado pelas operações $OP^{CpS_1} \leftarrow \{op_1^{CpS_1}, op_2^{CpS_1}\}$, sendo que $op_1^{CpS_1} \prec op_2^{CpS_1}$;
- $IN^{CpS_1} \leftarrow \{\mathrm{A}, \mathrm{B1}\}$;
- $OUT^{CpS_1} \leftarrow \{\mathrm{C}, \mathrm{D}\}$.

Esta composição oferece um casamento de primeiro grau para o serviço AbS_1, em contraposição ao casamento de segundo grau, que é realizado sem a interferência da composição. Este resultado é semelhante à lógica de composição do serviço AbS_2, que teve casamento de primeiro grau. Há, portanto, um relacionamento entre o nível de detalhamento do *workflow* e a qualidade do casamento. É bem possível que este relacionamento seja diretamente proporcional na maioria dos casos, ou seja, que quanto maior for o nível de detalhamento do *workflow* melhor será o casamento. Entretanto, não se deve assumi-lo como universal, por exemplo, se o serviço concreto CcS_3 não existe, a qualidade do casamento oferecido pela composição seria bem pior. Disto, conclui-se que a qualidade do casamento é um compromisso estabelecido entre o nível de detalhamento do *workflow* e os serviços concretos existentes. As várias possibilidades com que um desenvolvedor pode especificar (decompor) um serviço abstrato oferece a ele a liberdade para estabelecer este compromisso.

Em decorrência do sistema de referência para a composição automática de serviços estabelecida na seção 2.1, o processo de composição é inibido sempre que o casamento atingir o grau desejado. Esta inibição deve ocorrer porque os custos computacionais envolvidos na composição de serviços são maiores que aqueles envolvidos no casamento, e o resultado da composição é ignorado *a priori*.

3.5.3 Composição Sequencial x Paralela

Como descrito anteriormente, a composição sequencial é realizada quando pelo menos um parâmetro de entrada de um serviço qualquer possui similaridade com algum parâmetro de saída de outro serviço. Os parâmetros que não possuem similaridade transformam-se em parâmetros de entrada ou saída do serviço composto criado. Por exemplo, seja a composição sequencial entre os serviços concretos CcS_1 e CcS_4. Um novo serviço, CpS_2 é criado da seguinte forma: $OP^{CpS_2} = \{op_1^{CpS_2}, op_2^{CpS_2}\}$ sendo que $op_1^{CpS_2} \prec op_2^{CpS_2}$. Os parâmetros das operações são: $IN^{op_1^{CpS_2}} = \{A, B\}$, $OUT^{op_1^{CpS_2}} = \{C, D\}$, $IN^{op_2^{CpS_2}} = \{C, E\}$, $OUT^{op_2^{CpS_2}} = \{F, G\}$. Para os parâmetros do serviço composto, tem-se que: $IN^{CpS_2} = \{A, B, E\}$, $OUT^{CpS_2} = \{D, F, G\}$ (ver o resultado final da criação do

serviço composto CpS_2 na Tabela 3-3). Em um primeiro momento, pode-se estranhar a presença dos parâmetros E e D como entrada e saída, respectivamente, do serviço CpS_2. Mas como o parâmetro E não se casa com nenhum parâmetro de saída e nem de entrada do serviço CcS_1, ele se torna um parâmetro de entrada do serviço composto CpS_2. Raciocínio análogo deve ser usado para o parâmetro de saída D. Isto faz com que o serviço CpS_2 seja factível ou, em outras palavras, executável.

Porém, esta não é a única forma possível de se realizar a composição sequencial. A regra pode ser mais flexível ou mais rígida, conforme a necessidade, que aquela estabelecida na seção 3.4.1. Por exemplo, a composição sequencial somente poderia ocorrer quando todos os parâmetros de entrada do serviço sucessor tivessem similaridade com algum dos parâmetros de saída do serviço precedente. Em outro extremo, poder-se-ia permitir a composição sequencial mesmo quando nenhum dos parâmetros tivesse similaridade. Neste extremo, a composição sequencial comportar-se-ia, estruturalmente (mas não funcionalmente), como uma composição paralela. Estas diferentes possibilidades podem ser oportunamente aproveitadas em algoritmos de composição automática de serviços.

Tabela 3-3 – Serviços compostos.

Serviços Abstratos	P. Entrada do Serviço	P. Saída do Serviço	Tipo de Composição	Operações	P. Entrada da Operação	P. Saída da Operação
$\boldsymbol{CpS_1}$	*A, B1*	*C, D*	$op_1^{CpS_1} \prec op_2^{CpS_1}$	$op_1^{CpS_1}$	*A, B1*	*X*
				$op_2^{CpS_1}$	*X*	*C, D*
$\boldsymbol{CpS_2}$	*A, B, E*	*D, F, G*	$op_1^{CpS_2} \prec op_2^{CpS_2}$	$op_1^{CpS_2}$	*A, B*	*C, D*
				$op_2^{CpS_2}$	*C, E*	*F, G*
$\boldsymbol{CpS_3}$	*O, I*	*H*	$op_1^{CpS_3} \oplus op_2^{CpS_3}$	$op_1^{CpS_3}$	*O, I*	*H*
$\boldsymbol{CpS_4}$	*O, Y*	*B2*	$op_1^{CpS_4} \prec op_2^{CpS_4}$	$op_1^{CpS_4}$	*O, Y*	*DIXON*
				$op_2^{CpS_4}$	*DICKSONX*	*B2.1*

3.5.4 Composição de Composições

Uma vez que é criado um novo serviço composto, ele torna-se um potencial serviço concreto que pode ser utilizado para novas composições, permitindo, assim, o uso recursivo das composições. A seguir, pretende-se mostrar a recursividade das regras anteriormente definidas a partir da possibilidade de criação de composição de composições. Também aproveitar-se-á esta subseção para exemplificar por completo o processo de transformação de um serviço abstrato em um serviço concreto.

Seja o serviço abstrato AbS_4 descrito como a composição paralela entre duas operações, $op_1^{AbS_4} \parallel op_2^{AbS_4}$, conforme descrito na Tabela 3-1. Para realizar o casamento da operação $op_1^{AbS_4}$, é necessário, antes, criar uma composição do tipo escolha-exclusiva entre os serviços CcS_8 e CcS_{11}, pois ambos possuem pré-condições, $pre_1^{CcS_8} = (I < 10)$ e $pre_1^{CcS_{11}} = (I \geq 10)$, respectivamente. Seja, então, a criação do serviço CpS_3 composto por duas operações, $op_1^{CpS_3}$ e $op_2^{CpS_3}$ (Tabela 3-3). A primeira operação é ligada ao serviço CcS_8 e, a segunda, ao serviço CcS_{11}. Como as duas pré-condições são complementares (ver seção 3.4.3), um comando de decisão, $I < 10$, é criado para avaliar as pré-condições. A saída verdadeira do comando de decisão é ligada à operação $op_1^{CpS}(CcS_8)$ e, a falsa, à operação $op_2^{CpS_3}(CcS_{11})$. As saídas das operações $op_1^{CpS_3}$ e $op_2^{CpS_3}$ são ligadas pelo fluxo de união simples. O novo serviço criado não possui mais pré-condição.

A segunda operação do serviço abstrato AbS_4, $op_2^{AbS_4}$, não se casa com nenhum serviço disponível no repositório, mas casa-se com o resultado da composição sequencial entre os serviços CcS_9 e CcS_{10}. Esta composição sequencial acontece via casamento de quinto grau (todos os outros falham): utilizando a métrica de Jaro para determinar a similaridade sintática entre as palavras $DIXON$ e $DICKSONX$, o resultado obtido é 0,813 (ver seção 2.1). Portanto, é criado um serviço, CpS_4, tal que $OP^{CpS_4} = \{op_1^{CpS_4}, op_2^{CpS_4}\}$ sendo $op_1^{CpS_4} \prec op_2^{CpS_4}$. A operação $op_1^{CpS_4}$ é ligada ao serviço CcS_9 e, a operação $op_2^{CpS_4}$, ao serviço CcS_{10}. Os parâmetros de entrada do serviço CpS_4 são: $IN^{CpS_4} = \{O, Y\}$; enquanto o parâmetro de saída é: $IN^{CpS_4} = \{B2.1\}$. Desta forma, o serviço CpS_4 possui

casamento de segundo grau com a operação $op_2^{AbS_4}$, porque o parâmetro de saída $out_1^{CpS_4} = B2.1$ é descendente direto do parâmetro de saída $out_1^{Op_2^{AbS_4}} = B2$. O serviço CpS_4 está resumidamente descrito na Tabela 3-1 e ilustrado na Figura 3-8.

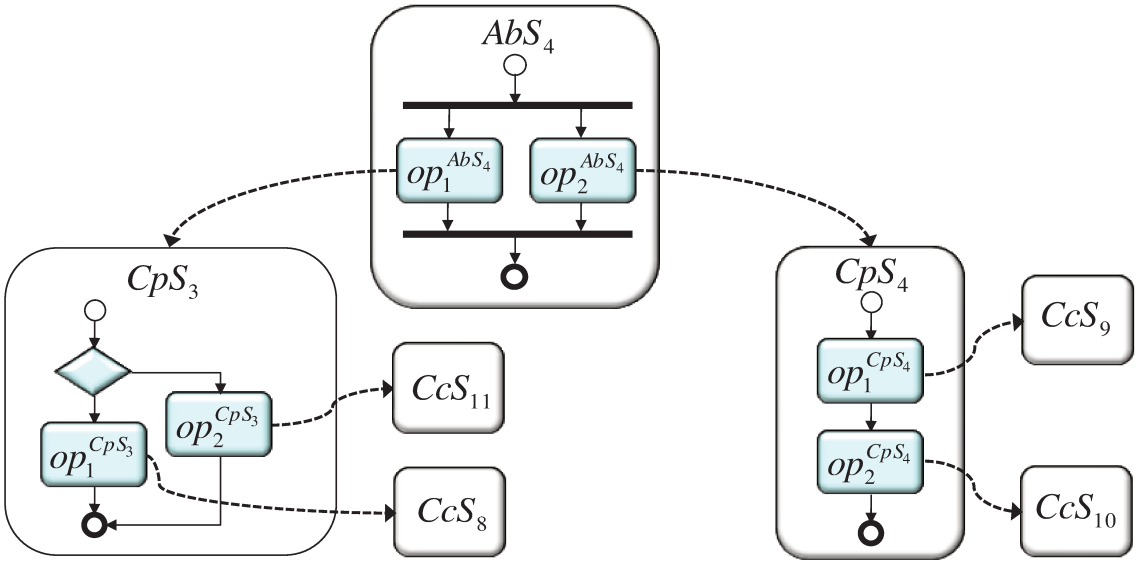

Figura 3-8 – Exemplo de composição de composições.

Assim mostrou-se que é possível a criação de diversos tipos diferentes de fluxo de controle e de dados a partir do uso de composições de composições de forma recursiva.

3.6 Análise de Complexidade do Problema de Composição de Serviços

Seja $CcS = \{CcS_1, CcS_2, \dots, CcS_\mu\}$ o repositório de serviços concretos onde μ representa a quantidade de serviços disponíveis no repositório. No pior caso, a quantidade de composições é dada pela fórmula abaixo:

$$f(\mu) = \frac{\mu!}{(\mu-(\mu-0))!} + \frac{\mu!}{(\mu-(\mu-1))!} + \frac{\mu!}{(\mu-(\mu-2))!} + \cdots + \frac{\mu!}{\left(\mu-(\mu-(\mu-1))\right)!} \qquad (3.11)$$

A Equação 3.11 significa que a quantidade de composições possíveis é o somatório do arranjo simples de μ serviços tomados μ a μ, $(\mu-1)$ a $(\mu-1)$, $(\mu-2)$ a $(\mu-2)$, $\cdots$,

$\big(\mu-(\mu-1)\big)$ a $\big(\mu-(\mu-1)\big)$. O arranjo simples é qualquer ordenação de r elementos dentre os μ elementos, em que cada maneira de tomar os elementos se diferencia pela ordem e natureza dos elementos. A fórmula para cálculo de arranjo simples é dada por [134]:

$$A_r^{\mu} = \frac{\mu!}{(\mu-\mathrm{r})!} \tag{3.12}$$

Para cada tipo de composição, supõe-se que haja $f(\mu)$ possibilidades, o que leva finalmente ao resultado de:

$$f_n(\mu) = n \times f(\mu) \tag{3.13}$$

Sendo n a quantidade de tipos de composições possíveis. Como exemplo, para $\mu = 10$ e levando-se em consideração os quatro tipos de composições descritos na seção 3.4, tem-se:

$$f_n(\mu) = 4 \times \frac{10!}{0!} + \frac{10!}{1!} + \frac{10!}{2!} + \frac{10!}{3!} + \frac{10!}{4!} + \frac{10!}{5!} + \frac{10!}{6!} + \frac{10!}{7!} + \frac{10!}{8!} + \frac{10!}{9!} = 39.456.400 \tag{3.14}$$

possibilidades de composições diferentes. Para $\mu = 11$, $f_n(\mu) = 434.020.400$ composições. Em outras palavras, um algoritmo que verificasse todas as composições possíveis teria que comparar $434.020.400$ alternativas, entre apenas os onze serviços para encontrar a melhor delas. Este número ainda representa um valor menor do que o esperado em verdade, uma vez que esta análise de complexidade do problema da composição de serviços está simplificada porque se está desprezando os vários tipos de casamento de parâmetros que podem ocorrer para cada tipo de fluxo de controle. A Figura 3-9 apresenta um gráfico da quantidade de composições possíveis em função da quantidade de serviços (μ).

Fazendo-se uma analogia com o conhecido problema do caixeiro viajante [135], pode-se dizer que cada serviço presente em uma composição assemelha-se a uma cidade de um determinado caminho do caixeiro, e, desta forma, é possível especular que o problema da composição de serviços Web semânticos é da mesma ordem de

complexidade do problema do caixeiro viajante, um clássico representante do conjunto de problemas NP-Completo.

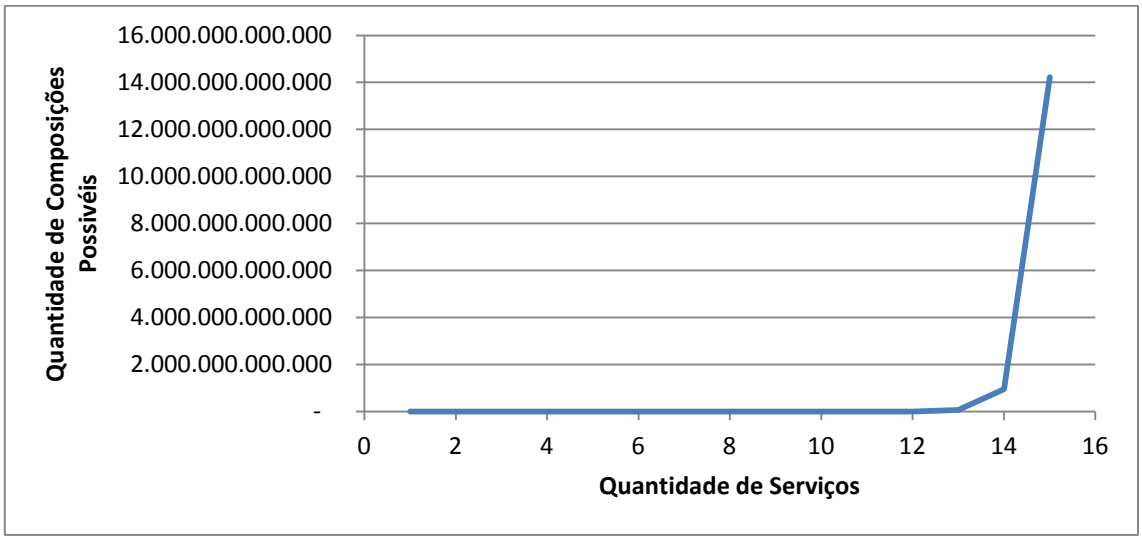

Figura 3-9 – Quantidade de soluções possíveis em relação à quantidade de serviços.

3.7 Considerações Finais

Neste capítulo foram desenvolvidos os graus de casamento e as regras que descrevem a composição de serviços Web semânticos. As regras de composição contemplam os cinco tipos básicos de fluxo de controle: sequencial, separação paralela, sincronização, escolha exclusiva e união simples; e um padrão estrutural: fluxo de controle em laço. Estas regras serão utilizadas, no próximo capítulo, para implementar um sistema que realiza a composição automática de serviços Web semânticos.

É importante ressaltar que os graus de casamento desenvolvidos neste trabalho (ver seção 3.3) vão além dos filtros desenvolvidos por Klusch et al. [63] (ver seção 2.3.4). Os filtros de Klusch são utilizados somente para o casamento (*matching*) de um para um, ou seja, de um serviço abstrato (no caso, uma operação abstrata) para um serviço concreto. As entradas e saída previamente disponíveis no *workflow* não são consideradas. Aliás, em Klusch, não há a noção de *workflow*. A introdução de um *workflow* de operações abstratas dentro de um único serviço abstrato possibilita o casamento de vários para um, isto é, permite que parâmetros de operações precedentes sejam utilizados para o

casamento da operação em questão (ver seção 3.5.1). Esta possibilidade concretizou-se a partir da introdução do conceito de operação polivalente (ver seção 3.3).

Ainda, a definição de um serviço abstrato como um *workflow* de operações abstratas oferece, ao desenvolvedor, várias formas diferentes para definir um serviço abstrato e, em última análise, de controlar a qualidade do casamento, uma vez que o nível de detalhamento do *workflow* e os serviços concretos existentes influenciam no grau do casamento realizado (ver seção 3.5.3). Também é possível controlar a qualidade dos casamentos manipulando o uso dos diversos graus de similaridade (ver seção 3.2).

A possibilidade de utilizar serviços compostos para realizar novas composições permite a criação de vários tipos diferentes de fluxo de controle e de dados, de forma recursiva (ver seção 3.5.4).

Capítulo 4

Esecsy _ Um Sistema Evolutivo para Composição Automática de Serviços Web Semânticos

"Ha, ha, ha! Depois disso, o senhor encontrará prazer mesmo numa dor de dentes!", exclameis rindo.

_ Como não? Há prazer mesmo numa dor de dentes _ respondo. _ Tive dor de dentes um mês inteiro; sei o que é isto.

Dostoiévski, Fiódor. Memórias do subsolo. Editora 34, São Paulo, 5ª. Ed, 2005, p 26.

Este capítulo descreve o Esecsy, um sistema para composição automática de serviços Web semânticos que utiliza time assíncrono (A-Team) com operadores genéticos. O Esecsy explora a semântica dos conceitos definida através das marcações *hasInput* e *hasOutput* dos arquivos OWL-S dos serviços [50]. Esecsy é o acrônimo para Evolutionary Service Composition System.

Tanto os algoritmos genéticos quanto os times assíncronos são especialmente bem sucedidos em fornecer soluções para vários problemas do tipo NP-Completo [81]. A solução para a composição de serviços Web semânticos deve ser projetada para conseguir uma solução viável (composição com sentido lógico) e ótima (menor quantidade possível de serviços e casamento de primeiro grau entre eles), ou, pelo menos o mais próximo dela possível, de maneira a satisfazer o maior número de requisitos e restrições do sistema. A grande quantidade de serviços disponíveis multiplicada pela quantidade de parâmetros de entrada e saída de cada serviço eleva sensivelmente a quantidade de cálculos necessários para se encontrar uma solução ótima e caracteriza o problema como sendo de elevada complexidade computacional (ver seção 3.6).

Resumidamente, o Esecsy tenta encontrar uma solução para o seguinte problema: dado um serviço abstrato qualquer, segundo a Definição 3-11, e um repositório de serviços concretos, segundo a Definição 3-8, deseja-se criar um novo serviço a partir da composição dos serviços presentes no repositório (segundo as regras apresentadas no Capítulo 3), que possua a menor quantidade possível de serviços e grau de similaridade com o serviço abstrato maior do que qualquer um dos serviços presentes no repositório. Se o serviço criado possuir similaridade exata com o serviço abstrato, o *fitness* deste serviço composto será igual a um, significando, portanto, que a solução é a de melhor *fitness*.

O Esecsy utiliza os graus de similaridade e casamento e as regras de composição definidas no capítulo anterior. Foram implementadas as composições do tipo sequencial e paralela com o propósito de validar os principais conceitos das regras de casamento e para avaliar a capacidade do sistema em realizar composições. Ambos os serviços, abstrato e concreto, são descritos na linguagem OWL-S.

4.1 Times Assíncronos

Segundo Talukdar et al. [15], um time assíncrono, ou A-Team, é um conjunto de agentes autônomos e um conjunto de memórias, interconectados para formar uma rede

cíclica de dados, isto é, uma rede em que cada agente está em um laço fechado (Figura 4-1). Os agentes cooperam entre si, seguindo regras e estruturas pré-definidas, na busca de um objetivo comum. As principais características de um A-Team são:

- Todos os agentes atuam de forma autônoma, não existe controle de dependência entre eles, isto é, agentes externos não podem interferir nas decisões internas de um dado agente;
- Os fluxos de dados são cíclicos, com os dados de saída de um agente servindo potencialmente como dados de entrada para outros agentes; e
- A comunicação entre os agentes é assíncrona e se faz exclusivamente através de memórias compartilhadas.

Em geral, as memórias armazenam soluções (parciais ou completas) de algum problema. Estas soluções formam populações, assim como nos algoritmos genéticos. Os agentes leem uma determinada memória, transformam os dados desta memória e escrevem o resultado em outra memória. Eventualmente as memórias de leitura e de escrita podem ser a mesma. A cooperação implica que a memória de leitura de um agente deve ser a memória de escrita de outro. Os agentes de um A-Team são implementados usando-se algoritmos conhecidos, como, por exemplo, os operadores de Computação Evolutiva [15]. As populações são variantes no tempo: novos membros são continuamente adicionados por agentes construtores, enquanto membros menos adaptados ao meio são apagados por agentes exterminadores.

Um tipo especial de agente construtor é o agente iniciador, que é encarregado de preencher as memórias com soluções iniciais que serão melhoradas durante a execução do A-Team. Uma vez iniciada a memória, os demais agentes entram em execução, lendo-a e atualizando-a num ciclo contínuo, até que as soluções ali armazenadas não possam mais ser melhoradas ou que tenha sido alcançado um limite de tempo de processamento previamente definido. Nesse instante, o A-Team interrompe sua execução e apresenta a melhor solução como saída.

O outro tipo especial de agente é o exterminador. Ele tem a função de eliminar periodicamente algumas soluções da memória, abrindo espaço para que novas soluções sejam armazenadas. As soluções a serem removidas são selecionadas segundo alguma política de destruição que depende do problema e da estrutura do A-Team em questão.

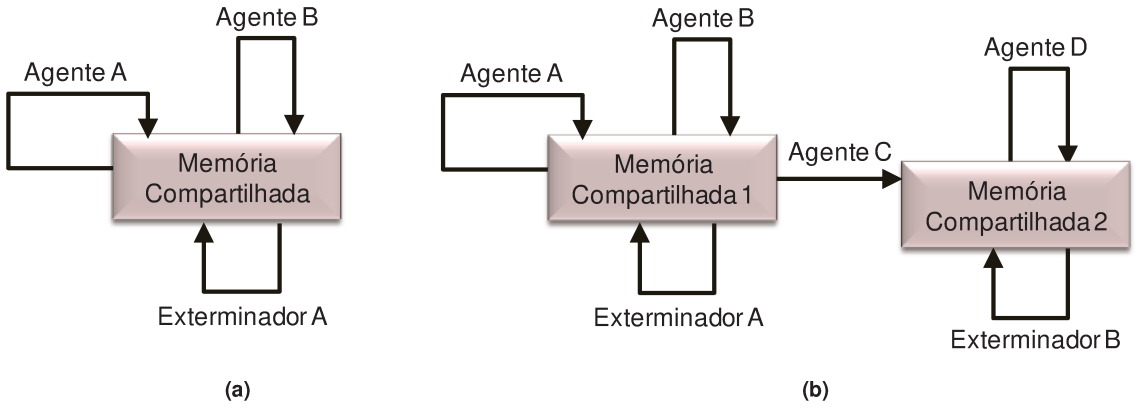

Figura 4-1 – Dois exemplos de A-Teams: (a) com uma única memória compartilhada; (b) com duas memórias compartilhadas.

Experiências realizadas com este tipo de organização mostram que o resultado produzido por um A-Team tende a ser melhor do que a "soma" dos resultados obtidos isoladamente por cada um dos agentes que o compõe. Além disso, mostram que o time tende a ser eficiente em escala, ou seja, o seu desempenho pode melhorar com a introdução de novos elementos (agentes e memórias). Outro ponto observado é que a comunicação assíncrona e a autonomia dos membros da organização permitem que os agentes sejam executados paralelamente, o que, juntamente com as outras características dos A-Teams, aumentam em muito a chance de se encontrar uma solução próxima à ótima global para o problema que está sendo tratado [15].

4.2 Algoritmos Genéticos

Os algoritmos clássicos de computação evolutiva normalmente são formados por um laço em que cada operador é executado em uma ordem pré-estabelecida. Após a criação de uma população inicial, os indivíduos são avaliados de acordo com o seu grau de adaptação, expresso por uma variável denominada *fitness*. O *fitness* é um valor que

quantifica, dada uma função de avaliação, a qualidade relativa de um indivíduo. Em seguida, tem início um laço onde os indivíduos são selecionados, se reproduzem, sofrem mutação e são novamente avaliados para formar uma nova população. Este ciclo se repete até que os indivíduos não possam mais ser melhorados ou que tenha sido alcançado um limite de tempo de processamento previamente definido. Neste momento, o melhor indivíduo é apresentado como solução. O pseudocódigo deste algoritmo é mostrado na Figura 4-2 [14].

```
procedure evolution program
begin
  t ←0
  inicialize P(t)
  evaluate P(t)
  while (not terminate-condition) do
  begin
    t ←t+1
    select P(t) from P(t-1)
    alter P(t)
    evaluate P(t)
  end
end
```

Figura 4-2 – Pseudocódigo de um programa evolutivo.

4.3 Time Assíncrono com Operadores Genéticos

Nos algoritmos genéticos, há dois tipos básicos de operadores responsáveis pela alteração do código genético da população: cruzamento[36] e mutação. Cada um destes pode ter muitas variações. As variações permitem que o operador se adapte ao problema específico. Segundo Michalewicz [14], os algoritmos genéticos clássicos operam sobre *strings* de *bits*. A aplicação destes algoritmos em estruturas de dados complexas acaba por modificar os próprios algoritmos, entretanto, ainda assim podem ser classificados como algoritmos genéticos.

[36] Em inglês: crossover.

Nos A-Teams, os operadores genéticos podem atuar em conjunto com outros operadores, mas não precisam seguir a ordem pré-estabelecida dos algoritmos genéticos, pois cada operador é autônomo em estabelecer sua política de execução. Desta forma, a noção de geração, naturalmente distinguível nos algoritmos genéticos, só pode ser artificialmente definida nos A-Teams.

Outra diferença significativa entre as duas técnicas diz respeito ao objetivo da seleção de indivíduos: enquanto os algoritmos genéticos selecionam os sobreviventes, os A-Teams selecionam os que deverão morrer.

4.4 Estrutura de Dados do Esecsy

A memória compartilhada do Esecsy representa a população, formada por uma lista de árvores binárias (Figura 4-3a). Um indivíduo desta população (uma árvore binária) representa um serviço concreto, que pode ser de dois tipos diferentes: atômico ou composto. Um serviço atômico é representado por uma árvore com exatamente um nó; enquanto a árvore de um serviço composto possui raiz, folhas e, possivelmente, nós intermediários (Figura 4-3b). Cada nó armazena: o tipo de serviço (atômico ou composto); se composto, o tipo de composição (sequencial ou paralela); o nome e tipo dos parâmetros de entrada; e o nome e tipo dos parâmetros de saída (Figura 4-3c). O nome é uma *string* e os parâmetros de entrada e saída são do tipo OWLType [136] (Figura 4-3d).

O serviço composto é uma composição de serviços atômicos. As folhas representam serviços atômicos; a raiz e os nós intermediários representam o tipo de composição que liga o lado esquerdo com o lado direito da árvore.

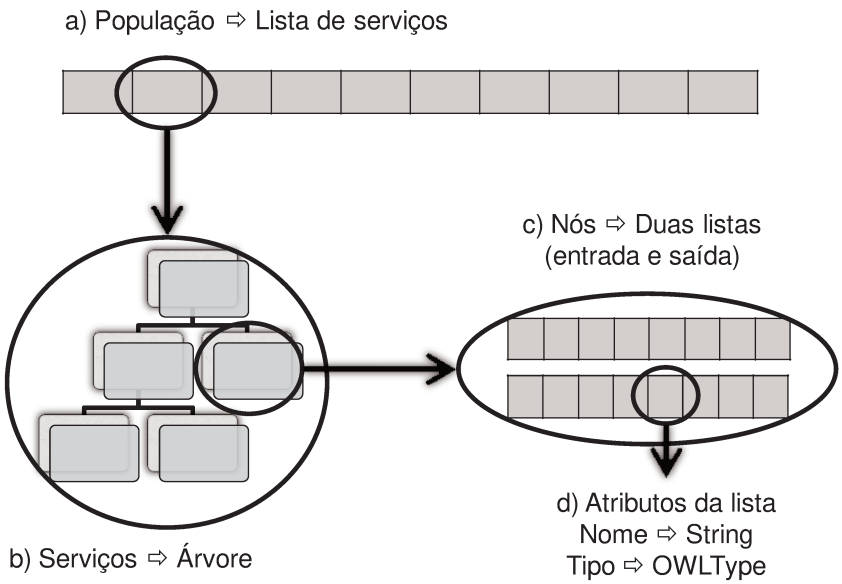

Figura 4-3 – Estrutura de dados do Esecsy.

Por exemplo, a Figura 4-4a ilustra a árvore da composição dos serviços atômicos AtS_1, AtS_2 e AtS_3. A leitura deve ser realizada em intraordem: o nó intermediário CpS_1_Sq indica que os serviços AtS_1 e AtS_2 são sequencialmente compostos. Por sua vez, CpS_1_Pr indica que CpS_1_Sq é paralelamente composto com AtS_3, em suma $CpS_1\_Pr = (AtS_1 \prec AtS_2) \parallel AtS_3$. O *workflow* resultante desta composição é mostrado na Figura 4-4b.

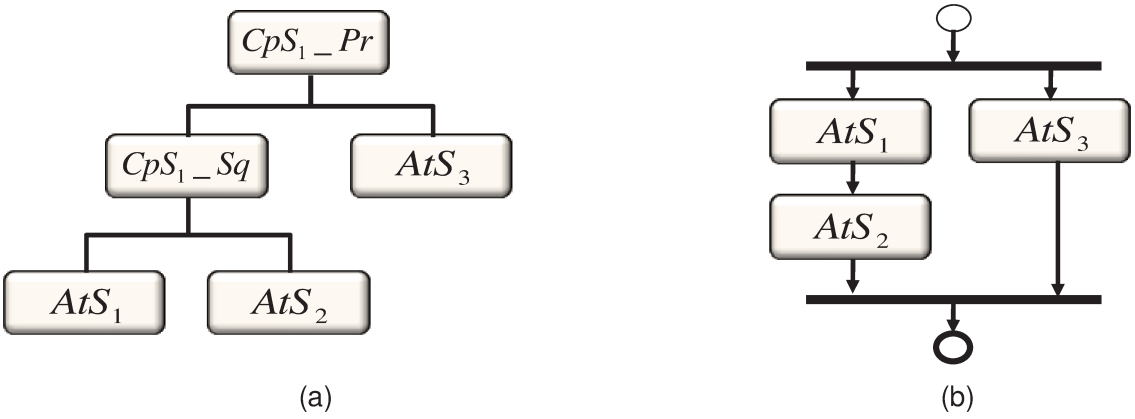

Figura 4-4 – Exemplo (a) da estrutura de dados de um serviço composto, e (b) do *workflow* correspondente.

A estrutura de dados do Esecsy foi projetada levando-se em consideração que cada serviço possui apenas uma única operação. Desta forma, serviço e operação se confundem, permitindo que algumas simplificações possam ser feitas em relação às definições de serviço e, consequentemente, em relação às equações de similaridade, de casamento e de composição apresentadas no capítulo anterior. Esta decisão não interfere no objetivo do Esecsy de comprovar a validade destas mesmas equações e nem de realizar composições. Para um serviço concreto com *n* operações, pode-se pensar em *n* serviços com apenas uma única operação. O efeito é o mesmo. Talvez um único ponto mereça atenção. Se o serviço abstrato possui apenas uma única operação, como poderá ser comprovada a eficiência da introdução do conceito de operação polivalente para o casamento? A resposta é simples, embora requeira um pouco mais de cuidado e controle durante a fase de testes. A interferência da operação polivalente poderá ser comprovada inserindo-se, artificialmente, novos parâmetros de entrada em um serviço abstrato. Estes parâmetros inseridos comportam-se exatamente como os parâmetros previamente disponíveis no *workflow* e que fazem parte da definição da operação polivalente. Portanto, as definições apresentadas no capítulo anterior devem ser compreendidas como uma generalização desta particularidade. Ademais, a base de dados utilizada para a validação do sistema, OWLS-TC3 [132], possui apenas serviços com uma única operação.

4.5 Diagrama de Classes do Esecsy

O Esecsy utiliza a OWL-S API [137], que fornece uma API Java para o acesso programático à leitura, execução e escrita de descrições de serviço em OWL-S. Para acessar os arquivos OWL-S, esta API necessita que os mesmos estejam armazenados em um servidor Web. Para tal fim foi utilizado o Tomcat [138].

A implementação foi estruturada em cinco pacotes: *ateams*, *evolution*, *ga*, *tests* e *utilities*. No pacote *tests,* foram implementadas várias classes com o objetivo exclusivo de validar e analisar o desempenho do sistema. Os experimentos, assim como os resultados obtidos, são apresentado no Capítulo 5. No pacote *utilities* são implementadas

funcionalidades tais como manipulação de arquivos, ordenação de vetores e listas, acesso a servidores Web etc. Os pacotes mais interessantes são os três primeiros.

4.5.1 O Pacote Ateam

O diagrama de classes das principais classes do pacote *ateams* está ilustrado na Figura 4-5, são elas: *Agente*, *AgMutação*, *AgIniciador*, *AgObservador*, *AgCruzamento*, *AgParalelo*, *AgSequencial*, *AgExterminador*. Seguem abaixo os detalhes referentes às decisões de implementação destas classes. Os detalhes relevantes ao funcionamento são explicitados nas seções posteriores deste capítulo.

A classe *Agente* é abstrata e estende a classe *Thread* do Java. Ela oferece acesso aos arquivos de *log* do sistema e sinaliza o término da execução. Para criar um objeto do tipo *Agente* é necessário passar uma referência a um objeto do tipo *SAbstrato* e duas referências a objetos do tipo *População*: uma para o reservatório gênico e outra para a população (memória compartilhada). O reservatório gênico possui os serviços atômicos disponíveis e é invariante durante toda a execução do sistema.

As classes *AgMutação*, *AgIniciador*, *AgObservador*, *AgCruzamento*, *AgParalelo*, *AgSequencial* e *AgExterminador* são subclasses da classe *Agente*. Os detalhes de cada uma destas classes são tratados na seção 4.7.

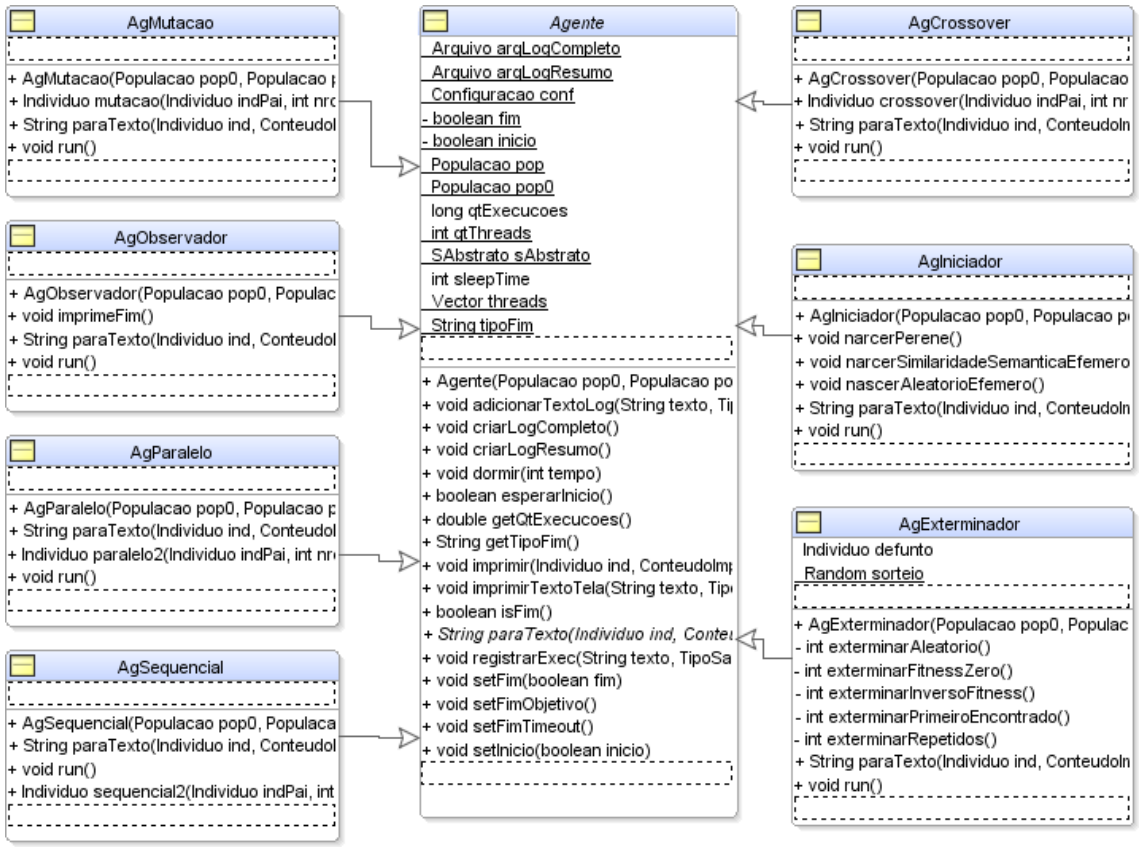

Figura 4-5 – Diagrama de classes do pacote ateams.

4.5.2 O Pacote Evolution

As principais classes do pacote *evolution* são: *SAbstrato*, *Indivíduo*, *Gene* (nó da árvore na Figura 4-3) , *Proteína* (atributos do nó da arvore na Figura 4-3), *FitnessInd*, *População*, *FitnessPop*, *Configuração* e *Ontologia* (Figura 4-6).

A classe *SAbstrato* estende a classe *Indivíduo* e é responsável pelo acesso e armazenamento dos dados referentes ao serviços abstratos. Estes dados são obtidos a partir da leitura de um arquivo OWL-S armazenado no servidor *Tomcat*. A leitura é realizada pela classe *Ontologia* que utiliza a OWL-S API [137].

A classe *FitnessInd* armazena os dados referentes ao *fitness* do indivíduo, e a classe *FitnessPop* armazena dados estatísticos da população, tais como *fitness* médio desvio padrão do *fitness*, *fitness* do melhor indivíduo etc.

A classe *Configuração* possui as variáveis de configuração do sistema, como por exemplo, serviço inicial, serviço final, tamanho da população e tipo de nascimento. O significado de cada uma destas variáveis está descrito na seção 5.3.

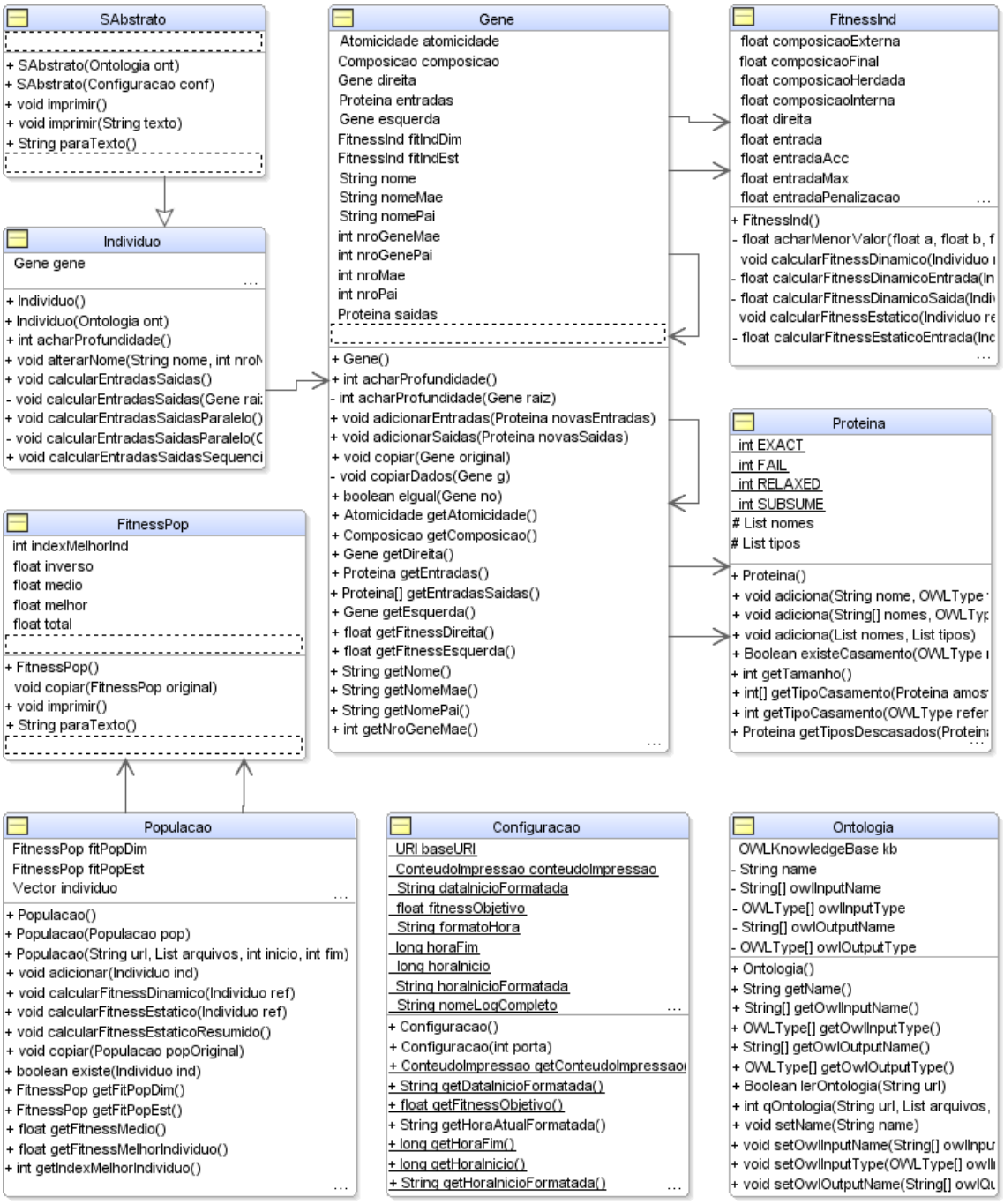

Figura 4-6 – Diagrama de classes do pacote Evolution.

A classe *População* armazena as informações de *fitness* da população, e é, essencialmente, uma lista (implementada como uma classe *Vector* do Java) da classe *Indivíduo*. Entre os métodos que merecem destaque estão aqueles que selecionam um individuo e calculam o *fitness* da população. A classe *Indivíduo* é responsável pelo cálculo de *fitness* de um indivíduo (serviço), pela determinação dos parâmetros de entrada e saída e pela seleção dos genes para a realização do cruzamento e da mutação. A classe *Indivíduo* faz uma referencia à classe *Gene*. A classe *Gene* implementa a estrutura de árvore que representa um serviço e que está ilustrada na Figura 4-3. Ela armazena informações sobre o *fitness* do nó e de como ele foi gerado. Possui duas referencias à classe *Proteína*: uma para especificar os parâmetros de entrada e outra para os de saída. A classe *Proteína* possui duas listas, uma para armazenar os nomes dos tipos de dados e outra para armazenar os tipos de dados, propriamente dito, ou seja, um *OWLType*.

4.5.3 O Pacote GA

O pacote *GA* foi implementado com o objetivo de comparar o desempenho do sistema utilizando um algoritmo de times assíncronos e um algoritmo genético clássico. As principais classes deste pacote são: *Operador*, *OpNovaPop*, *OpParalelo*, *OpCrossover*, *OpSequencial*, *OpMutação*. A classe *Operador* é a super classe de todos os operadores e possui papel similar à classe *Agente* do pacote *Ateam*. Aliás, as classes *OpMutação*, *OpParalelo*, *OpCrossover* e *OpSequencial* são funcionalmente semelhantes às classes com o mesmo sufixo, respectivamente, do pacote *Ateams*. A classe *OpNovaPop* realiza a seleção de indivíduos para uma nova geração.

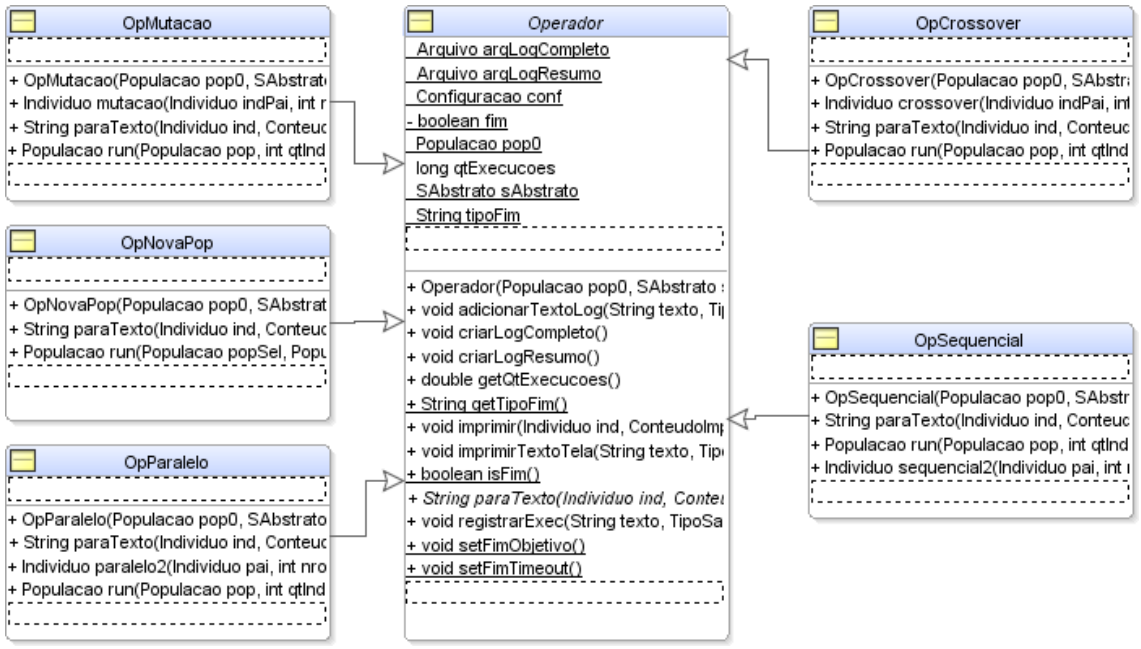

Figura 4-7 – Diagrama de classes do pacote GA.

O pseudocódigo do algoritmo genético clássico implementado está descrito na seção 4.8.

4.6 A Função de *Fitness*

Em termos gerais, o *fitness* é um valor que quantifica, dada uma função de avaliação, o grau de adaptação de um indivíduo. A comparação dos valores individuais de *fitness* resultará em uma competição pela sobrevivência e reprodução no ambiente, havendo uma vantagem seletiva daqueles indivíduos com valores elevados de *fitness* [14]. No Esecsy, o *fitness* é um valor numérico entre zero e um que, dependendo do tipo do serviço envolvido, expressa diferentes significados:

- Para um serviço atômico: é um valor numérico que expressa o grau de casamento entre o serviço atômico e o serviço abstrato. O valor zero revela nenhuma similaridade, e o valor um simboliza um casamento de primeiro grau;
- Para um serviço composto: é um valor numérico que expressa o grau de casamento entre o serviço composto e o serviço abstrato e também a qualidade da

composição. O valor zero pode revelar ausência de similaridade ou baixa qualidade da composição. Por sua vez, o valor um revela excelsa similaridade e composição.

A função de *fitness* de um serviço S é definida como:

$$Fitness^S = InternalFitness^S \times ExternalFitness^S \quad (4.1)$$

sendo que:

$$InternalFitness^S = lowerValueOf\ (CompFitness^S, InternalFitness^{left}, InternalFitness^{right}) \quad (4.2)$$

$$ExternalFitness^S = \frac{\frac{InputFitness^S}{InputFitnessMax^S} + \frac{OutputFitness^S}{OutputFitnessMax^S}}{2} \quad (4.3)$$

$$InputFitness^S = \sum_{m=0}^{|IN^S|} HigherSimilarityDegree(in_m^S, IN^{AbS}) \quad (4.4)$$

$$InputFitnessMax^S = 3 \times |IN^S| \quad (4.5)$$

$$OutputFitness^S = \sum_{q=0}^{|OUT^{AbS}|} HigherSimilarityDegree\left(out_q^{AbS}, OUT^S\right) \quad (4.6)$$

$$OutputFitnessMax^S = 3 \times |OUT^{AbS}| \quad (4.7)$$

onde:

- AbS é um serviço abstrato;
- $|IN^S|$ é a quantidade total dos parâmetros de entrada do serviço S;
- $|OUT^{AbS}|$ é a quantidade total dos parâmetros de saída do serviço AbS.
- $InternalFitness^S$ é o *fitness* interno do serviço S, depende do tipo de serviço (ver seções 4.6.1e 4.6.2);
- $ExternalFitness^S$ é o *fitness* externo do serviço S, depende dos seus parâmetros de entrada e saída;
- $CompFitness^S$: é o *fitness* da composição que gerou o serviço S; depende do tipo de composição;

- $InternalFitness^{left}$: é o *fitness* interno da composição do ramo esquerdo da árvore que gerou o serviço S;
- $InternalFitness^{right}$: é o *fitness* interno da composição do ramo direito da árvore que gerou o serviço S;
- $InputFitness^{S}$ é o *fitness* de entrada do serviço S, calculado em relação aos seus parâmetros de entrada;
- $InputFitnessMax^{S}$ é o *fitness* de entrada máximo que o serviço S poderia alcançar;
- $OutputFitness^{S}$ é o *fitness* de saída do serviço S, calculado em relação aos seus parâmetros de saída;
- $OutputFitnessMax^{S}$ é o *fitness* de saída máximo que o serviço S poderia alcançar.

Definição 4-1 (Fitness composto de um serviço atômico) *Se S é um serviço atômico, então o seu fitness composto é um, $CompFitness^{S} = 1$.*

Definição 4-2 (Fitness do ramo esquerdo de um serviço atômico) *Se S é um serviço atômico, então o fitness do seu ramo esquerdo é um, $InternalFitness^{left} = 1$.*

Definição 4-3 (Fitness do ramo direito de um serviço atômico) *Se S é um serviço atômico, então o fitness do seu ramo direito é um, $InternalFitness^{right} = 1$.*

As seções 4.6.1 e 4.6.2 descrevem o significado destas equações e definições.

4.6.1 Cálculo do *Fitness* para um Serviço Atômico

A partir das Definições 4.1, 4.2 e 4.3 demonstra-se que o *fitness* interno de um serviço atômico, AtS, é igual a um:

$$InternalFitness^{AtS} = lowerValueOf\ (1,1,1) = 1 \qquad (4.8)$$

E, portanto, seu *fitness* resume-se ao *fitness* externo:

$$Fitness^{AtS} = ExternalFitness^{AtS} = \frac{\frac{InputFitness^{AtS}}{InputFitnessMax^{AtS}} + \frac{OutputFitness^{AtS}}{OutputFitnessMax^{AtS}}}{2} \quad (4.9)$$

Os parâmetros de entrada e de saída contribuem igualmente para o valor do *fitness*. O *fitness* de entrada (Equação 4-4) é o somatório do maior grau de similaridade encontrado entre cada parâmetro de entrada do serviço atômico (in_m^{AtS}) com os parâmetros de entrada do serviço abstrato (IN^{AbS}). Para uma similaridade do tipo exata é atribuído o valor três, do tipo ascendência, dois, e do tipo descendência, um[37]. A similaridade sintática não foi contemplada. O valor máximo que o *fitness* de entrada (Equação 4.5) pode ter é $3 \times |IN^{AtS}|$, isto é, todos os parâmetros de entrada do serviço atômico possuem similaridade exata com algum parâmetro de entrada do serviço abstrato.

Raciocínio análogo pode ser feito em relação à parte do *fitness* referente aos parâmetros de saída: o *fitness* de saída (Equação 4.6) é o somatório do maior grau de similaridade encontrado entre cada parâmetro de saída do serviço abstrato $\left(out_q^{AbS}\right)$ com os parâmetros de saída do serviço atômico (OUT^{AtS}). O valor máximo que o *fitness* de saída (Equação 4.7) pode ter é $3 \times |OUT^{AbS}|$, isto é, todos os parâmetros de saída do serviço abstrato possuem similaridade exata com algum parâmetro de saída do serviço atômico.

Naturalmente, o *fitness* tenderá a ser baixo quando ocorrer uma ou mais das seguintes condições:

- O serviço atômico possui mais parâmetros de entrada do que o serviço abstrato;
- O serviço atômico possui menos parâmetros de saída do que o serviço abstrato;
- Há pouca similaridade entre os respectivos parâmetros de entrada e saída dos serviços atômico e abstrato.

[37] A implementação de um matchmaker realizada pelo Semantic Web Research Group utilizou os valores EXACT = 0, SUBSUME = 1, RELAXED = 2 e FAIL = 3 [137].

Para exemplificar o cálculo do *fitness*, supor os serviços AbS e AtS conforme ilustrado da Figura 4-8. Os parâmetros do serviço abstrato são: $IN^{AbS} = \{A1.1, B1\}$ e $OUT^{AbS} = \{R1.1\}$. E os parâmetros do serviço atômico são: $IN^{AtS} = \{A1, C1, D2\}$ e $OUT^{AtS} = \{R1, S1.2\}$. A definição da hierarquia de subsunção entre os parâmetros destes serviços é dada pela Figura 3-7 apresentada na seção 3.5. Para este exemplo, os valores parciais do *fitness* são:

- $InputFitness^{AtS} \leftarrow 2$: a única similaridade ocorre entre o parâmetro $A1$ do serviço AtS que é ascendente direto do parâmetro $A1.1$ do serviço AbS. Dois pontos são atribuídos para esta similaridade;
- $InputFitnessMax^{AtS} \leftarrow 9$: o serviço atômico possui três parâmetros de entrada, cada um vale três pontos (pontuação máxima considerando que todos eles tivessem similaridade exata com algum parâmetro do serviço abstrato);
- $OutputFitness^{AtS} \leftarrow 1$: o parâmetro $R1.1$ do serviço AbS é descendente direto do parâmetro $R1$ do serviço AtS;
- $OutputFitnessMax^{AtS} \leftarrow 3$: o serviço abstrato possui um único parâmetro de saída.

Portanto, o *fitness* do serviço AtS, em relação ao serviço AbS, é: $Fitness^{AtS} = \frac{\frac{2}{9}+\frac{1}{3}}{2} = 0{,}278$.

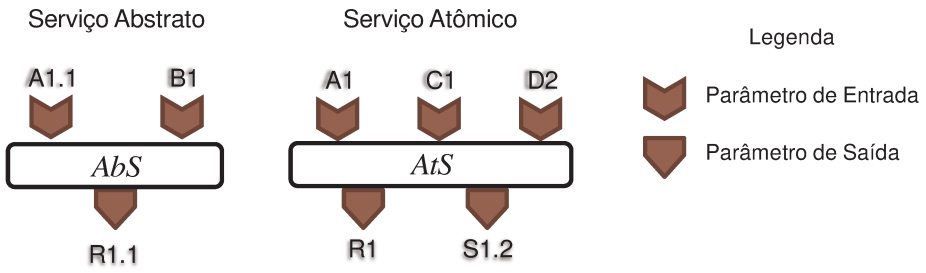

Figura 4-8 – Os serviços AbS e AtS utilizados para exemplificar o cálculo do *fitness* para serviços atômicos.

Resumidamente, para um serviço atômico, o *fitness* é calculado em relação à quantidade total de pontos que seriam possíveis de se obter se todos os seus parâmetros possuíssem similaridade exata com os parâmetros do serviço abstrato.

4.6.2 Cálculo do *Fitness* para um Serviço Composto

Qual deve ser o *fitness* de uma composição sequencial? Ou paralela? Mas como uma composição pode ser a composição de outras composições, além destas questões, o cálculo do *fitness* de um serviço composto esbarra em outra: qual deve ser a repercussão do *fitness* de uma composição que participa de uma nova composição? Intuitivamente, um serviço S_1 não pode ter *fitness* igual a 1 se ele é composto por outra composição, por exemplo, $S_2 \prec S_3$, cujo *fitness* é menor que um. Portanto, o *fitness* do serviço S_1 não pode ser maior do que *fitness* de suas composições. Isto significa que uma composição com baixo *fitness* deve propagar este *fitness* para todos os serviços que a utilizam. Mas uma ressalva, o *fitness* final de um serviço é o produto do *fitness interno* pelo *fitness externo*. O *fitness* externo reflete a similaridade dos parâmetros de entrada e saída de um serviço com os respectivos parâmetros do serviço abstrato. Já o *fitness* interno reflete a qualidade da composição (no caso de um serviço composto), e é, portanto, este *fitness* que deve ser propagado.

O *fitness* interno de um serviço composto é o menor valor de *fitness* entre todas as composições que fazem parte deste serviço (Equação 4.2). Para um serviço composto, CpS, formado pela composição dos serviços CcS_k e CcS_l, a Equação 4.2 é transcrita como:

$$InternalFitness^{CpS} = lowerValueOf\ (CompFitness^{CpS}, InternalFitness^{CcS_k}, InternalFitness^{CcS_l}) \quad (4.10)$$

Onde:

- $InternalFitness^{CcS_k}$: é o *fitness* interno do serviço CcS_k;
- $InternalFitness^{CcS_l}$: é o *fitness* interno do serviço CcS_l;
- $0 < k, l \leq \mu$ (sendo μ a quantidade de serviços concretos do repositório).

A Equação 4.10 é uma função recursiva, uma vez que o calculo do *fitness* interno de um serviço, CpS, depende do cálculo do *fitness* interno dos serviços que o compõe, CcS_k e CcS_l.

O *fitness* da composição, $CompFitness^{CpS}$, é dependente do tipo de composição. Para uma composição sequencial entre CcS_k e CcS_l ($CcS_k \prec CcS_l$), tem-se:

$$CompFitness^{CcS_k \prec CcS_l} = \frac{\sum_{m=0}^{|IN^{CcS_l}|} HigherSimilarityDegree\left(in_m^{CcS_l}, OUT^{CcS_k}\right)}{3 \times |IN^{CcS_l}|} \qquad (4.11)$$

Para exemplificar, supor os serviços CcS_z, CcS_k e CcS_l, sendo:

- $IN^{CcS_z} = \{A1.1, B1\}$ e $OUT^{CcS_z} = \{C1\}$;
- $IN^{CcS_k} = \{C1\}$ e $OUT^{CcS_k} = \{E1.1\}$;
- $IN^{CcS_l} = \{E1\}$ e $OUT^{CcS_l} = \{R1.1\}$.

Ademais, supor $CpS_1 = (CcS_k \prec CcS_l)$ e $CpS_2 = CcS_z \prec CpS_1$, ou seja, $CpS_2 = CcS_z \prec (CcS_k \prec CcS_l)$, $0 < z, k, l \leq \mu$, conforme ilustrado na Figura 4-9. Sendo CcS_z, CcS_k e CcS_l serviços atômicos, então:

- $InternalFitness^{CcS_z} = 1$;
- $InternalFitness^{CcS_k} = 1$;
- $InternalFitness^{CcS_l} = 1$;
- $CompFitness^{CpS_1} \leftarrow \frac{2}{3}$;
- $InternalFitness^{CpS_1} \leftarrow lowerValueOf\ (\frac{2}{3}, InternalFitness^{CcS_k}, InternalFitness^{CcS_l})$;
- $CompFitness^{CpS_2} \leftarrow \frac{3}{3}$;
- $InternalFitness^{CpS_2} \leftarrow lowerValueOf\ (\frac{3}{3}, InternalFitness^{CcS_z}, \frac{2}{3})$;
- $ExtenalFitness^{CpS_2} \leftarrow \frac{\frac{6}{6}+\frac{3}{3}}{2}$.

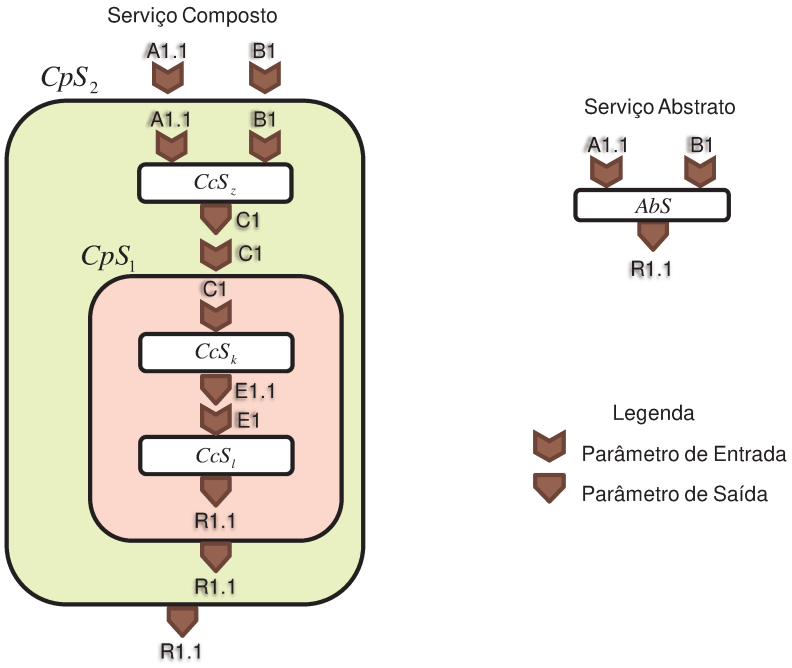

Figura 4-9 – Os serviços AbS e $CpS_2 = CcS_z \prec (CcS_k \prec CcS_l)$ utilizados para exemplificar o cálculo do *fitness* para serviços compostos sequencialmente.

Portanto, o *fitness* final do serviço CpS_2 é $Fitness^{CpS_2} = \frac{2}{3} \times 1 = 0{,}667$. Há um casamento de grau um entre os parâmetros externos, parâmetros de entrada e saída, dos serviços CpS_2 e AbS. Entretanto, o serviço CpS_1, que participa da composição do serviço CpS_2, possui uma composição interna de segundo grau (por causa dos parâmetros $E1.1\ e\ E1$). É exatamente este casamento de segundo grau que penaliza o *fitness* da composição como um todo (o *fitness* do serviço CpS_2). Naturalmente, o serviço CpS_1 também possui *fitness* externo, mas, neste caso, ele é irrelevante na formação do *fitness* da composição.

Definição 4-4 (Fitness composto de uma composição paralela) *Se CpS é um serviço resultante de uma composição paralela entre CcS_k e CcS_l, então o seu fitness composto é um: $CompFitness^{CcS_k \| CcS_l} = 1$.*

Para exemplificar, supor o serviço abstrato AbS e o serviço composto $CpS = CcS_k \parallel CcS_l$, conforme ilustrado na Figura 4-10:

- $IN^{AbS} = \{A1.1, B1\}$ e $OUT^{AbS} = \{R1.1\}$;

- $IN^{CcS_k} = \{A1.1, B1\}$ e $OUT^{CcS_k} = \{E1\}$;
- $IN^{CcS_l} = \{D1, C1\}$ e $OUT^{CcS_l} = \{R1.1\}$;
- $IN^{CpS} = \{A1.1, B1, D1, C1\}$ e $OUT^{CcS_l} = \{E1, R1.1\}$.

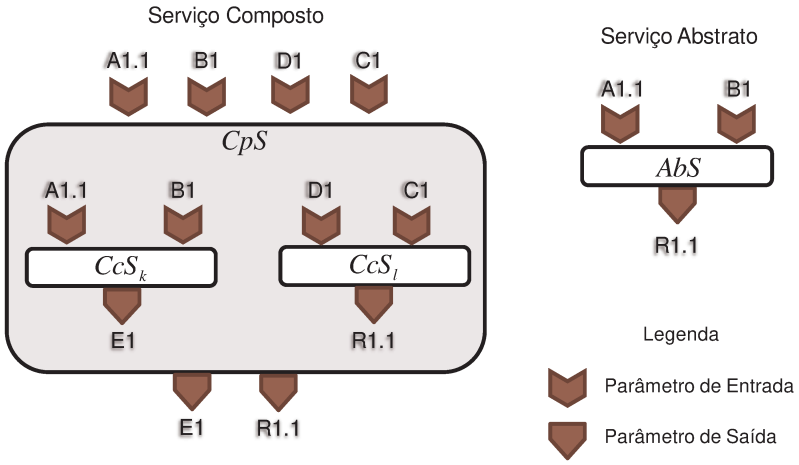

Figura 4-10 – Os serviços AbS e $CpS = CcS_k \prec CcS_l$ utilizados para exemplificar do cálculo do *fitness* para serviços compostos paralelamente.

Calculando-se o *fitness* do serviço CpS, chega-se ao seguinte valor:

- $InternalFitness^{CcS_k} = 1$;
- $InternalFitness^{CcS_l} = 1$;
- Portanto, $Fitness^{CpS} = 1 \times \frac{\frac{6}{12}+\frac{3}{3}}{2} = 0{,}75$.

Apesar de satisfazer o parâmetro de saída e de possuir dois parâmetros com similaridade exata aos parâmetros de entrada do serviço abstrato, o serviço CpS possui dois parâmetros de entrada a mais que o serviço abstrato, portanto, seu *fitness* é menor que um.

4.6.3 Análise da Função de Fitness

O cálculo do *fitness* de um individuo é realizado atribuindo-se um valor para cada par de parâmetros similares e não atribuindo um único valor correspondente ao grau de casamento realizado entre as interfaces do serviço concreto e do serviço abstrato. Por quê? Porque a primeira opção oferece uma gradação melhor do que a segunda, ou seja, a

classificação dos serviços fica mais bem distribuída. Por exemplo, o serviço atômico AtS da Figura 4-8 não realiza nenhum tipo de casamento com o serviço abstrato AbS, porque possui mais parâmetros de entrada do que o serviço abstrato. Portanto, se fosse atribuído um valor ao *fitness* correspondente ao tipo de casamento, este valor deveria ser zero. Entretanto, o valor zero omite o fato de que alguns parâmetros deste serviço são similares ao serviço abstrato e que, em uma listagem classificatória, deverá ser apresentado antes de outros serviços cujos parâmetros não possuem nenhuma similaridade.

Mas surge outra questão, decorrente da primeira: para que, então, servem os graus de casamento? Eles não são adequados para determinar o valor do *fitness* enquanto se realiza o processo de composição, mas são adequados para se classificar o resultado final da composição. Algoritmos evolutivos podem oferecer várias soluções para um mesmo problema. Classificar estas soluções em diferentes grupos é muitas vezes desejável ou necessário.

4.7 Os Agentes Assíncronos

O Esecsy foi implementado como um algoritmo de time assíncrono (A-Team) [15] com cinco agentes construtivos (iniciador, paralelo, sequencial, mutação e cruzamento), um agente exterminador e um agente especial chamado observador. Os agentes paralelo, sequencial, mutação e cruzamento executam estratégias de diversificação que forçam a busca a examinar regiões ainda não visitadas. O Esecsy possui ainda um repositório gênico para armazenar os serviços atômicos disponíveis (Figura 4-11).

Duas variáveis foram definidas para manter o tamanho da população sob controle: $popMin$ e $popMax$. A quantidade de indivíduos nunca será menor que $popMin$, exceto no início da execução do sistema; e tampouco será maior do que $popMax$. Se a quantidade de indivíduos alcançar o valor $popMax$, somente o agente exterminador será executado; os outros dormirão. Enquanto a quantidade de indivíduos for inferior a $popMin$ (no início da execução do sistema) o exterminador ficará dormindo. Cada agente é implementado como uma *thread*. A seguir, explanações detalhadas sobre cada agente.

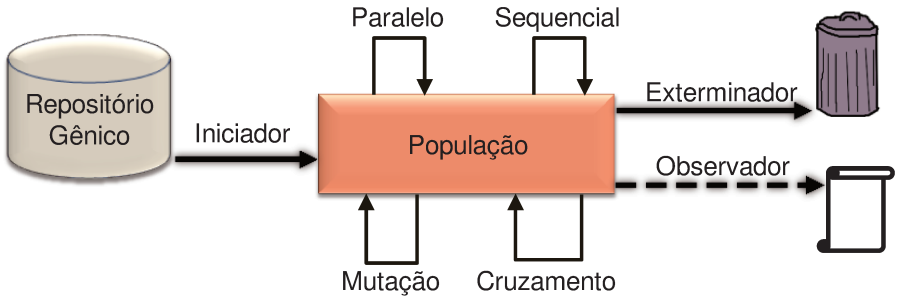

Figura 4-11 – A estrutura e os agentes assíncronos do sistema Esecsy.

4.7.1 O Agente Iniciador

Este é o agente responsável por introduzir indivíduos, ou melhor, serviços atômicos na população. Os serviços atômicos disponíveis estão armazenados em um repositório gênico que é alimentado por terceiros. O agente iniciador pode ser configurado de três maneiras diferentes, chamadas tipos de nascimento:

- Aleatório: periodicamente serviços atômicos são inseridos na população. O intervalo de tempo é configurável. Os serviços são selecionados de forma cega;
- Semântico: introduz periodicamente serviços atômicos na população, mas somente se o serviço possuir similaridade com o serviço abstrato ou com algum serviço já presente na população. Parte-se do princípio de que somente serviços com algum tipo de similaridade podem formar composições;
- Perpétuo: no início da execução insere todos os serviços presentes no repositório gênico na população inicial. Esta operação é realizada uma única vez.

Os tipos de nascimento aleatório e semântico simulam domínios de busca dinâmicos (ver seção 2.1) onde o reservatório gênico pode ser constantemente alterado pela criação, eliminação ou atualização dos serviços disponíveis. Portanto, nesta configuração, o iniciador fica constantemente inserindo serviços do repositório gênico na população durante a execução do sistema. Como a seleção é aleatória, nem todos os serviços são inseridos, e é possível que o iniciador insira serviços atômicos já presentes na população (serviços duplicados). A responsabilidade de manter o tamanho da população sob controle e de eliminar os indivíduos duplicados é do agente exterminador.

O nascimento do tipo perpétuo é usado quando todos os serviços atômicos são conhecidos *a priori*. O iniciador insere na população todos os serviços atômicos do repositório gênico e fica inoperante durante o restante da execução. Neste contexto, o agente exterminador não pode eliminar nenhum serviço atômico. Este tipo de nascimento é usado em ambientes estáticos ou quando o usuário quer ter o controle total sobre os serviços atômicos que podem participar de uma composição.

4.7.2 O Agente Sequencial

O agente sequencial é responsável pela criação de serviços compostos segundo as regras definidas na seção 3.4.1. A cada execução, o agente sequencial seleciona três serviços, por exemplo, CcS_k, CcS_l, e CcS_m. Estes três serviços são selecionados de acordo com um dos métodos de seleção descritos na seção 4.9, e eles precisam ser diferentes entre si $(k \neq l \neq m)$. Depois da seleção são criados dois novos serviços, resultantes da composição entre $CcS_k \prec CcS_l$ e $CcS_m \prec CcS_k$. Se alguma destas composições tiver o *fitness* interno (Equação 4.10) igual a zero, ela é automaticamente descartada (não é inserida na população). Com *fitness* interno igual a zero, estes serviços iriam zerar o *fitness* de toda composição da qual tomassem parte e, consequentemente, mais cedo ou mais tarde, seriam eliminados pelo agente exterminador.

O *fitness* do serviço CcS_k é sempre calculado em relação ao serviço abstrato. Mas, para direcionar a busca, o *fitness* do serviço CcS_l é calculado em relação ao *serviço jusante* e, o *fitness* do serviço CcS_m, em relação ao *serviço montante*.

O serviço jusante é um serviço fictício que possui como parâmetros de entrada, os parâmetros de saída do serviço concreto CcS_k e, como parâmetros de saída, os parâmetros de saída do serviço abstrato. Ou seja, o serviço jusante estabelece uma ponte entre as saídas do serviço concreto CcS_k e as saídas do serviço abstrato. Em outros termos, ele é o serviço que, se fosse sequencialmente composto com o serviço concreto CcS_k, resultaria em um *fitness* de saída (Equação 4.6) máximo. Neste sentido, é como se a busca fosse direcionada a partir dos parâmetros de entrada para os parâmetros de saída.

Definição 4-5 (Serviço Jusante) *Seja CcS_k um serviço concreto e AbS um serviço abstrato, então JsS^{CcS_k} é o serviço jusante de CcS_k, tal que:*

$$IN^{JsS^{CcS_k}} \leftarrow OUT^{CcS_k} \; e \; OUT^{JsS^{CcS_k}} \leftarrow OUT^{AbS} \tag{4.12}$$

De forma análoga, o serviço montante facilita a busca em direção contrária, ou seja, a partir dos parâmetros de saída em direção aos parâmetros de entrada. É o serviço que, se fosse sequencialmente composto com o serviço concreto CcS_k, resultaria em um *fitness* de entrada (Equação 4.4) máximo. Ele estabelece uma ponte entre as entradas do serviço concreto CcS_k e as entradas do serviço abstrato.

Definição 4-6 (Serviço Montante) *Seja CcS_k um serviço concreto e AbS um serviço abstrato, então MtS^{CcS_k} é o serviço montante de CcS_k, tal que:*

$$IN^{MtS^{CcS_k}} \leftarrow IN^{AbS} \; e \; OUT^{MtS^{CcS_k}} \leftarrow IN^{CcS_k} \tag{4.13}$$

Por exemplo, considere o serviço concreto CcS_k, tal que: $IN^{CcS_k} = \{B1\}$ e $OUT^{CcS_k} = \{C1\}$; e o serviço abstrato AbS, tal que: $IN^{AbS} = \{A1\}$, $OUT^{AbS} = \{D1\}$. Então, o serviço jusante, JsS^{CcS_k}, é criado de tal forma que: $IN^{JsS^{CcS_k}} \leftarrow C1$ *e* $OUT^{JsS^{CcS_k}} \leftarrow D1$; já para o serviço montante, MtS^{CcS_k}, tem-se que: $IN^{MtS^{CcS_k}} \leftarrow \{A1\}$ e $OUT^{MtS^{CcS_k}} \leftarrow B1$ (Figura 4-12).

Além de direcionarem a busca, os serviços jusante e montante eliminam o problema do serviço intermediário com *fitness* zero. Suponha, por exemplo, o serviço abstrato AbS e os serviços concretos CcS_1, CcS_2 e CcS_3, conforme Figura 4-13. Obviamente, a solução a este serviço abstrato é a composição sequencial $CcS_1 \prec CcS_2 \prec CcS_3$, pois $in_1^{AbS} = in_1^{CcS_1} = A1$, $out_1^{CcS_1} = in_1^{CcS_2} = B1$, $out_1^{CcS_2} = in_1^{CcS_3} = C1$ e $out_1^{AbS} = out_1^{CcS_3} = D1$. Entretanto, o *fitness* do serviço CcS_2, em relação ao serviço abstrato, é zero e, assim sendo, sua probabilidade de ser escolhido, caso a escolha fosse proporcional a este *fitness*, é nula. E, portanto, a composição jamais seria levada a cabo. Para contornar esta anomalia, um serviço pode ser selecionado levando-se em consideração o seu *fitness* calculado em relação aos serviços jusante e montante. Tal

procedimento é detalhado a seguir através do pseudocódigo do agente sequencial ilustrado na Figura 4-14.

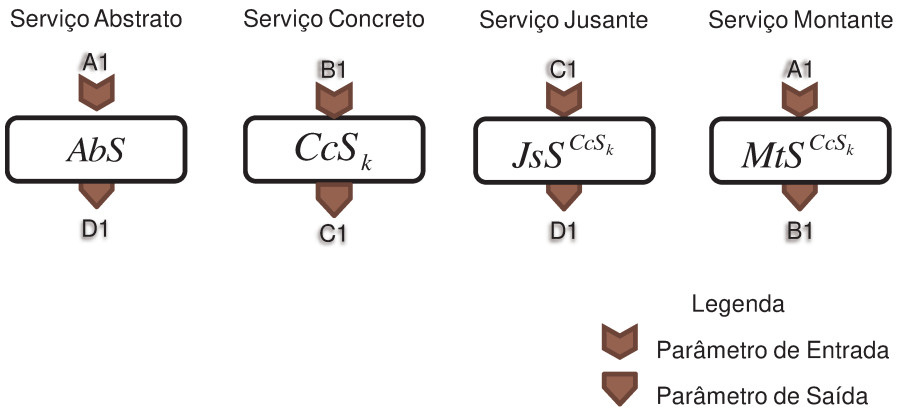

Figura 4-12 – Ilustração dos conceitos de serviço jusante e montante.

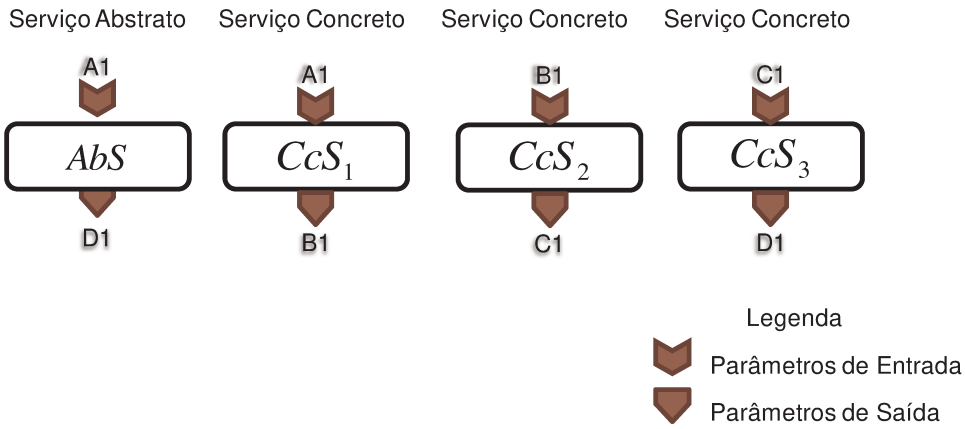

Figura 4-13 – O serviço concreto CcS_2 possui *fitness* zero em relação ao serviço abstrato AbS, mas *fitness* diferente de zero em relação aos serviços jusante e montante.

O agente sequencial só começa a execução após perceber que o agente iniciador já criou a população inicial. Esta percepção é feita através do monitoramento de uma variável compartilhada por todos os agentes (linha 1). Se não ocorreu o fim do programa (linha 2), e o tamanho da população for menor que um tamanho máximo especificado pela variável *PopMax* (linha 3), o agente, efetivamente, tenta encontrar uma solução para o problema. O fim pode ocorrer quando um agente encontra uma solução ou quando acontece *timeout*. Há vários agentes sendo executados em paralelo, quando um deles encontra uma solução, uma variável compartilhada indica a todos os demais agentes que

a execução do sistema deve ser finalizada. Se o fim ocorrer por *timeout*, é de responsabilidade do agente observador instanciar tal variável. Após estes testes, um serviço, no caso CcS_k, é selecionado de acordo com o método de seleção escolhido (linha 4). Caso o método de seleção escolhido seja proporcional ao *fitness*, então, neste caso, será utilizado o *fitness* calculado em relação ao serviço abstrato. Posteriormente, é criado o serviço jusante, JsS_k (linha 5), e o *fitness* de cada indivíduo da população é calculado em relação a ele (linha 6). Um serviço, CcS_l, é selecionado levando-se em consideração o *fitness* calculado em relação ao serviço jusante (linha 7). O serviço composto, $CpS_i = CcS_k \prec CcS_l$, é criado (linha 8) e, se o seu *fitness* interno for diferente de zero e a sua profundidade (quantidade de níveis da árvore) for menor que um valor pré-estabelecido, ele é inserido na população (linha 9), caso seu *fitness* seja maior ou igual ao *fitness* desejado, a variável de fim de programa é ativada (linha 10). As linhas 11, 12, 13, 14, 15 e 16 são análogas às linhas 5, 6, 7, 8, 9 e 10, porém, utilizam como referência o serviço montante, MtS_k. Uma variável permite especificar um tempo, em milissegundos, em que o agente deve dormir a cada iteração (linha 17). Isto permite controlar a quantidade de vezes que um agente é executado em relação ao demais.

```
1  Aguarde inicio
2  Enquanto não fim, faça:
3   Se tamanho da população < popMax, faça:
4    Selecione serviço CcS_k
5    Crie o serviço jusante JsS_k
6    Calcule o fitness de cada indivíduo da população em relação ao jusante
7    Selecione o serviço CcS_l
8    Crie o serviço composto CpS_i = CcS_k ≺ CcS_l
9    Se InternalFitness^CpS_i ≠ 0 e profundidade < permitida, insira CpS_i na população
10   Se Fitness^CpS_i ≥ fitness desejado, fim de execução
11   Crie o serviço montante MtS_k
12   Calcule o fitness de cada indivíduo da população em relação ao montante
13   Selecione o serviço CcS_m
14   Crie o serviço composto CpS_n = CcS_m ≺ CcS_k
15   Se InternalFitness^CpS_n ≠ 0 e profundidade < permitida, insira CpS_n na população
16   Se Fitness^CpS_n ≥ fitness desejado, fim de execução
17   Se tempo de dormência ≠ 0, durma
```

Figura 4-14 – Pseudocódigo do agente sequencial.

O pseudocódigo do agente sequencial (Figura 4-14) é bastante semelhante ao pseudocódigo do agente paralelo.

4.7.3 O Agente Paralelo

O agente paralelo é responsável pela criação de serviços compostos segundo as regras definidas na seção 3.4.2. Os serviços que formarão o novo serviço composto, CcS_ke CcS_l, são selecionados por um dos métodos de seleção descritos na seção 4.9; estes serviços precisam ser diferentes um do outro ($k \neq l$). Caso o método de seleção seja proporcional ao *fitness*, o primeiro serviço, CcS_k , é selecionado levando-se em consideração o *fitness* calculado em relação ao serviço abstrato. Mas, para direcionar a busca da composição paralela, o segundo serviço, CcS_l, é selecionado levando-se em consideração o *fitness* calculado em relação ao serviço fronteira. A função exercida pelo o serviço fronteira no agente paralelo é semelhante à função exercida pelos serviços montante e jusante no agente sequencial.

Definição 4-7 (Serviço Fronteira) *Seja* CcS_k *um serviço concreto e* AbS *um serviço abstrato, então,* FrS^{CcS_k} *é o serviço fronteira de* CcS_k*, tal que:*

$$IN^{FrS^{CcS_k}} \leftarrow IN^{AbS} \dot{-} IN^{CcS_k} \text{ e } OUT^{FrS^{CcS_k}} \leftarrow OUT^{AbS} \dot{-} OUT^{CcS_k} \qquad (4.14)$$

A Equação 4.14 indica que o serviço fronteira é formado pelos parâmetros de entrada e de saída do serviço abstrato menos os respectivos parâmetros do serviço concreto.

O pseudocódigo deste agente é semelhante ao pseudocódigo do agente sequencial (Figura 4-14), excetuando-se o fato de que o serviço fronteira realiza as funções dos serviços montante e jusante.

4.7.4 O Agente Mutação

O agente mutação substitui um nó da árvore de um serviço composto por um serviço atômico. Os serviços composto e atômico são selecionados por um dos métodos de seleção descritos na seção 4.9; o nó da árvore do serviço composto é selecionado aleatoriamente. A Figura 4-15 exemplifica o processo de mutação. Primeiramente, é selecionado um serviço, CpS_4_Sq $(CpS_4\_Sq = AtS_1 \prec AtS_2 \prec AtS_3 \prec AtS_4 \prec AtS_5)$, e um nó da árvore deste serviço, CpS_1_Sq. Depois é a vez de se selecionar um serviço atômico armazenado no repositório gênico, AtS_9. O nó CpS_1_Sq é trocado pelo serviço AtS_9, resultando em um novo serviço, chamado de serviço mutante, no caso, $CpS_4\_Sq\_muta\text{ção}: 1$.

Toda vez que uma mutação é realizada, o sufixo $\_muta\text{ção}: n$ é colocado no nome do nó mutante (nó que sofreu a mutação) e de todos os seus ascendentes. n é um número sequencial incrementado de um a cada execução do agente. Isto assegura um nome adicionado para cada serviço e, igualmente, funciona como um histórico genético do serviço.

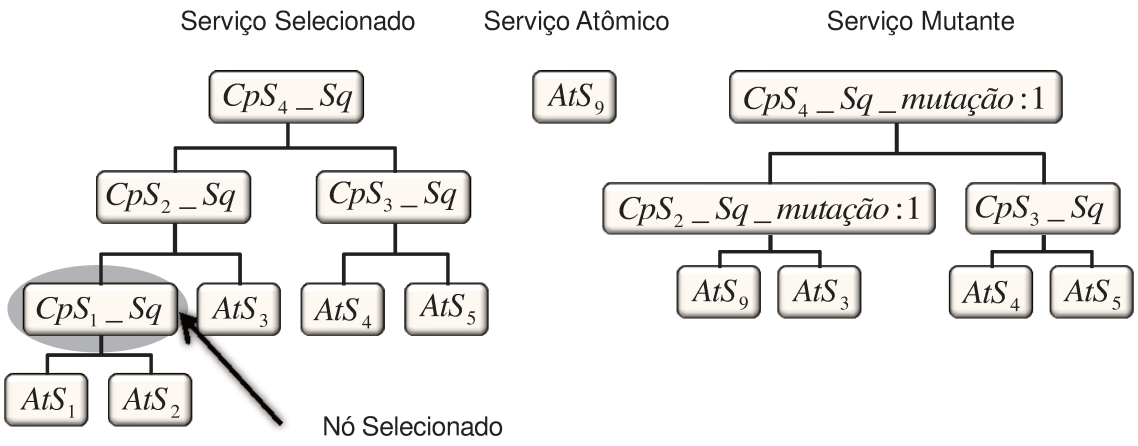

Figura 4-15 – Exemplo do processo de mutação. O serviço $CpS_4\_Sq\_mutação:1$ é resultado da mutação do nó CpS_1_Sq pelo serviço atômico AtS_9.

A mutação é sempre realizada com serviços atômicos, isto é, o nó que sofrerá a mutação sempre será trocado por um serviço atômico. Isto diferencia a mutação do cruzamento e permite, como efeito colateral, manter sobre controle a profundidade da árvore.

4.7.5 O Agente Cruzamento[38]

Como um operador convencional de cruzamento de um algoritmo genético, o agente cruzamento do Esecsy mistura os genes de dois indivíduos, neste caso, ele troca os nós das árvores de dois serviços. Novamente, os serviços são selecionados por um dos métodos da seleção descritos na seção 4.9. Estes serviços devem ser diferentes entre si. Os nós de cada serviço são aleatoriamente selecionados.

Como exemplo, supor dois serviços, $CpS_2\_Sq = AtS_1 \prec AtS_2 \prec AtS_3$ e $CpS_3\_Sq = AtS_4 \prec (AtS_6 \parallel AtS_5)$, sendo escolhido o nó AtS_3 do serviço CpS_2_Sq e o nó $CpS_1\_Pr = AtS_6 \parallel AtS_5$ do serviço CpS_3_Sq como pontos de cruzamento (Figura 4-16). O nó AtS_3 é colocado no lugar do nó CpS_1_Pr e vice-versa, resultando, a cada iteração do agente paralelo, em dois novos serviços:

- $CpS_2\_Sq\_crossover: 1 = (AtS_1 \prec AtS_2) \prec (AtS_6 \parallel AtS_5)$;

[38] Em inglês: crossover

- $CpS_2\_Sq\_crossover: 2 = AtS_4 \prec AtS_3$.

A Figura 4-16 ilustra somente o primeiro resultado. O sufixo $\_crossover: n$, análogo ao sufixo utilizado na mutação, é adicionado ao nome de todos os ascendentes do nó que sofreu cruzamento.

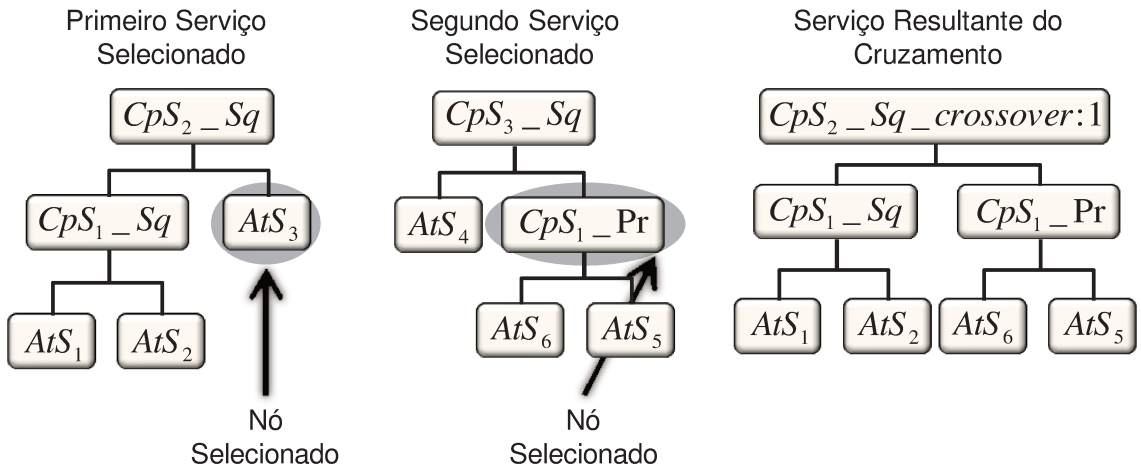

Figura 4-16 – Exemplo do processo de cruzamento: o serviço $CpS_2\_Sq\_crossover: 1$ é resultado do cruzamento entre os serviços CpS_2_Sq e CpS_3_Sq.

4.7.6 O Agente Exterminador

Para manter o tamanho da população sob o controle, o agente exterminador periodicamente elimina indivíduos de acordo com os limites mínimo $(popMin)$ e máximo $(popMax)$ estabelecidos pelo usuário do sistema. Quando a população alcança o limite máximo, somente o agente exterminador pode operar.

No início da execução (pseudocódigo na Figura 4-17), quando a quantidade de indivíduos está abaixo do limite mínimo, o exterminador não pode atuar: ele fica aguardando um sinal de início gerado pelo agente iniciador (linha 1). Enquanto não ocorrer o fim da execução do sistema (linha 2), e o tamanho da população for maior que o tamanho mínimo estabelecido (linha 3), o agente tenta eliminar um indivíduo. O fim da execução pode ser sinalizado quando um dos agentes construtores (sequencial, paralelo, mutação e cruzamento) encontrar uma solução; ou mesmo, quando ocorrer *timeout*. A eliminação dos indivíduos ocorre na seguinte ordem de prioridade:

- Duplicados: os agentes construtores não se importam com a duplicação de indivíduos. É responsabilidade do exterminador eliminá-los (linha 4);
- Roleta inversa: os indivíduos com baixo *fitness* têm maior probabilidade de eliminação (linha 5);
- *Fitness* igual a zero: elimina os indivíduos com *fitness* igual a zero (linha 6). O *fitness* igual a zero é decorrente do *fitness* externo igual a zero, uma vez que indivíduos com *fitness* internos iguais a zero são automaticamente descartados quando gerados;
- Cega: em um caso muito especial, quando não há nenhum individual duplicado e quando todos os indivíduos têm *fitness* igual a zero, o método de eliminação cega precisa ser executado, e somente neste caso, para garantir o processo evolucionário (linha 7).

Primeiramente, o exterminador elimina os serviços duplicados. Se não houver nenhum, ele tenta eliminar utilizando o método da roleta inversa, se este não conseguir eliminar nenhum, é feita uma tentativa de eliminar os indivíduos com fitness igual a zero e, em último caso, a eliminação cega é feita. Excetuando a eliminação de indivíduos duplicados, os outros métodos oferecem risco. O *fitness* utilizado na eliminação por roleta inversa e por *fitness* igual a zero é o *fitness* calculado em relação ao serviço abstrato. Entretanto, um serviço com *fitness* igual a zero pode ser essencial na busca por uma solução. A eliminação do serviço CcS_2 na Figura 4-13, por exemplo, impede a descoberta da solução. À primeira vista, a solução seria basear a eliminação no *fitness* calculado em relação aos serviços jusante, montante ou fronteira, dependendo do caso. Mas isto não é possível porque não se sabe sobre quem realizar este cálculo, uma vez que o cálculo deste *fitness* depende da escolha de outros serviços (no caso da Figura 4-13, dos serviços CcS_1, ou CcS_3). Entretanto, alguma forma de manter o tamanho da população sobre controle é necessária. Por isto algum o risco deve ser absorvido. O melhor indivíduo nunca é eliminado.

```
1  Aguarde inicio
2  Enquanto não fim, faça:
3     Se o tamanho da população ≥ popMin, então:
4       Tente eliminar os indivíduos repetidos
5       Se nenhum foi eliminado, tente eliminar utilizando a roleta inversa
6       Se nenhum foi eliminado, tente eliminar que possua fitness = 0
7       Se nenhum foi eliminado, faça eliminação cega
```

Figura 4-17 – Pseudocódigo do agente exterminador.

4.7.7 O Agente Observador

O observador é um agente especial que monitora a população e o tempo de execução do sistema. A cada *t* ms, o observador seleciona o melhor indivíduo e armazena-o em arquivo de *log* (o tempo *t* é um parâmetro de configuração do usuário). Outras informações também são armazenas, tais como: os valores parciais do *fitness* de cada indivíduo, a quantidade de vezes que o *fitness* foi calculado, o tempo gasto para se encontrar uma solução e a árvore que representa a composição de um serviço.

A Figura 4-18 é um fragmento do arquivo de *log* e ilustra a forma como um serviço é armazenado. O serviço de nome sequencial:74 indica que este serviço foi obtido após 74 execuções do agente sequencial. O *fitness* deste serviço é 1.0 (significa que o objetivo foi alcançado). Após o nome, são listados os nomes e tipos dos parâmetros de entrada e saída e os resultados parciais do *fitness*. Em seguida, os serviços que constituem a composição sequencial são indicados (AMOUNT-OF-MONEY__4WHEELEDCAR_SERVICE_NEW e 4WHEELEDCAR_PRICE_SERVICE), com seus respectivos parâmetros de entrada, saída e resultados parciais e finais do *fitness*.

```
Sequencial:74   - Fitness:1.0
  Parâmetros de Entrada
    Nome: http://127.0.0.1/services/amount-of-money3wheeledcar.owls#_AMOUNT-OF-MONEY
    Tipo: http://127.0.0.1/ontology/portal.owl#Amount-Of-Money
  Parâmetros de Saída
    Nome: http://127.0.0.1/services/1.1/4wheeledcar.owls#_PRICE
    Tipo: http://127.0.0.1/ontology/concept.owl#Price
  Fitness de Entrada - Max:3.0 - Acumulado:3.0 - Resultado:1.0
  Fitness de Saída   - Max:3.0 - Acumulado:3.0 - Resultado:1.0
  Fitness da Composição: 1.0
  AMOUNT-OF-MONEY__4WHEELEDCAR_SERVICE_NEW   - Fitness:0.5
    Parâmetros de Entrada
      Nome: http://127.0.0.1/services/amount-of-money3wheeledcar.owls#_AMOUNT-OF-MONEY
      Tipo: http://127.0.0.1/ontology/portal.owl#Amount-Of-Money
    Parâmetros de Saída
      Nome: http://127.0.0.1/services/amount-of-money3wheeledcar.owls#_4WHEELEDCAR
      Tipo: http://127.0.0.1/ontology/my_ontology.owl#FourWheeledCar
    Fitness de Entrada - Max:3.0 - Acumulado:3.0 - Resultado:1.0
    Fitness de Saída   - Max:3.0 - Acumulado:0.0 - Resultado:0.0
    Fitness da Composição: 1.0
 4WHEELEDCAR_PRICE_SERVICE  - Fitness:0.5
  Parâmetros de Entrada
      nome: http://127.0.0.1/services/4wheeledcar.owls#_4WHEELEDCAR
      tipo: http://127.0.0.1/ontology/my_ontology.owl#FourWheeledCar
    Parâmetros de Saída
      Nome: http://127.0.0.1/services/4wheeledcar.owls#_PRICE
      Tipo: http://127.0.0.1/ontology/concept.owl#Price
    Fitness de Entrada - Max:3.0 - Acumulado:0.0 - Resultado:0.0
    Fitness de Saída   - Max:3.0 - Acumulado:3.0 - Resultado:1.0
    Fitness da Composição: 1.0
```

Figura 4-18 – Exemplo de um serviço armazenado no arquivo de *log*.

4.8 Algoritmo AG Clássico

Um algoritmo AG Clássico foi implementado para que o desempenho do algoritmo A-Team pudesse ser comparado a outro algoritmo.

O pseudocódigo do AG é ilustrado na Figura 4-19. Enquanto a solução do problema não for encontrada e não ocorrer *timeout*, o algoritmo fica executando em laço (linha 1). Na linha 2, ocorre a chamada do método que realiza composição sequencial, onde:

- *pop* é a população atual;

- *qtPopSeq* é a quantidade de indivíduos que serão gerados por esta composição; e,
- *popSeq* é a população resultante da composição sequencial.

Métodos equivalentes são definidos para a composição paralela (linha 3) e para a geração de indivíduos por cruzamento (linha 4) e mutação (linha 5). O método *opSeleção.run* seleciona o melhor indivíduo da população atual e, caso seja necessário, seleciona mais indivíduos para completar a quantidade necessária (linha 6). A população seguinte é formada pelos indivíduos gerados por cada método (linha 7).

```
1  while (!fim) {
2    popSeq = opSequencial.run(pop, qtPopSeq);
3    popPar = opParalelo.run(pop, qtPopPar);
4    popCross = opCruzamento.run(pop, qtPopCross);
5    popMut = opMutacao.run(pop, qtPopMut);
6    popSel = opSelecao.run(pop, qtPopSel);
7    pop = opNovaPop.run(popSeq, popPar, popCross, popMut, popSel);
8    geracao++;
9  }
```

Figura 4-19 – O pseudocódigo do AG clássico implementado.

Para efeito de comparação entre os algoritmos, AG Clássico versus A-Teams, foram realizados testes experimentais para determinar qual deveria ser a quantidade de indivíduos gerados por cada método para produzir o melhor desempenho. Os resultados são apresentados no Capítulo 5.

4.9 Métodos de Seleção

Três tipos de métodos de seleção de indivíduos foram implementados:

- Cego: um indivíduo é aleatoriamente selecionado, isto é, despreza-se seu *fitness*;
- Roleta[39]: o método clássico de seleção de indivíduos [14], isto é, a probabilidade de escolha de um indivíduo é diretamente proporcional ao seu *fitness*;

[39] Em inglês: roulette wheel.

- Torneio: seleciona o melhor indivíduo entre n indivíduos selecionados de forma cega, sendo n um parâmetro de configuração do sistema.

Os agentes que necessitam selecionar indivíduos podem ser configurados para usar qualquer um destes métodos seleção.

4.10 Considerações Finais

Neste capítulo foi apresentado o algoritmo empregado pelo Esecsy[40], um sistema para composição automática de serviços Web semânticos. Suas principais características são:

- Utiliza algoritmo de time assíncrono e possui sete agentes: iniciador, sequencial, paralelo, mutação, cruzamento, exterminador e observador;
- Utiliza dois operadores genéticos: cruzamento e mutação;
- Realiza composição segundo os fluxos de controle sequencial e paralelo (separação paralela e sincronização);
- Explora a semântica dos parâmetros *hasInput* e *hasOutput* dos arquivos OWL-S.

No próximo capítulo são apresentados os experimentos conduzidos para avaliar o desempenho do sistema.

[40] Acrônimo para Evolutionary Service Composition System.

Capítulo 5

Resultados e Avaliação de Desempenho

Sem dúvida, nada há de mais natural, hoje em dia, do que ver as pessoas trabalharem de manhã à noite e optarem, em seguida, por perder nas cartas, no café e em tagarelices o tempo que lhes resta para viver... Em Oran, como no resto do mundo, por falta de tempo e de reflexão, somos obrigados a amar sem saber.

Camus, Albert. A Peste. Record. Rio de Janeiro. p 1.

Neste capítulo, são apresentados e discutidos os experimentos realizados para avaliar o desempenho do Esecsy em resolver o problema da composição automática de serviços Web semânticos.

O Esecsy foi desenvolvido na linguagem Java versão 1.5.0 e necessita da instalação de um servidor Web e da OWL-S API [137]. A OWL-S API fornece uma API Java para o acesso programático à leitura, execução e escrita de descrições de serviços em OWL-S.

Para acessar os arquivos OWL-S, esta API necessita que os mesmos estejam armazenados em um servidor Web. Para tal fim foi utilizado o servidor Tomcat [138].

Para avaliar o desempenho do sistema, o seguinte método foi adotado:

- Estabelecer um ambiente de execução: como o desempenho do sistema pode variar em função do *hardware* e do sistema operacional, dentre outros fatores, todos os experimentos foram realizados no ambiente IBM do CENAPAD-SP [139] (ver seção 5.1);
- Estabelecer uma base de dados de descrições de serviços Web semânticos: foi adotada a coleção de teste chamada OWLS-TC3 [132] (ver seção 5.2);
- Estabelecer uma configuração de referência: o desempenho do sistema pode depender de muitos fatores. Para alguns destes fatores foram estabelecidas variáveis cujos valores pudessem ser manipulados. Entretanto, existem variáveis que foram definidas apenas para aumentar a flexibilidade da implementação em futuras versões. Em alguns casos, pré-testes tiveram de ser realizados para estabelecer um valor apropriado para algumas variáveis ou, pelo menos, estabelecer uma faixa de variação. Estes valores pré-estabelecidos serão doravante designados por *valores de referência*. O conjunto das variávies com seus respectivos valores de referência forma a *configuração de referência*. Na avaliação de desempenho, as variáveis não mencionadas durante um experimento qualquer assumem seus valores de referência (ver seção 5.3);
- Estabelecer uma métrica de comparação: os resultados são expressos em relação ao Tempo de Resposta Médio (TRM). Como o TRM pode variar em função do *hardware*, do sistema operacional etc. os resultados também são oferecidos em relação à quantidade de vezes em que a função de *fitness* é calculada (doravante simplificado por QtF). Quando conveniente, o desvio padrão (DP) em relação ao TRM também é mencionado. Isto permitirá eventuais comparações com outros sistemas;

- Estabelecer a quantidade de amostras necessárias: a quantidade de amostras irá determinar o grau de confiabilidade dos resultados (ver seção 5.4). Uma amostra significa uma execução do Esecsy até que se obtenha uma solução do problema, que pode ser atingida por *fitness* ou por *timeout*. Nos casos de omissão, a amostra é conseguida por *fitness*;
- Determinar a influência da função de *fitness*: realizar um experimento para identificar a influência da função de *fitness* no desempenho do sistema (ver seção 5.5);
- Estabelecer as variáveis independentes e as variáveis de resposta: fatores como tamanho da população, quantidade de serviços disponíveis, complexidade do serviço composto, quantidade de agentes em execução, método de seleção, taxa de cruzamento, taxa de mutação etc. serão tratados como variáveis independentes. A variável medida será o TRM e, às vezes, o QtF e o DP (ver seções 5.6 à 5.15). O objetivo é avaliar como cada um destes fatores influencia no desempenho do sistema, e ao mesmo tempo, determinar qual a melhor configuração para o Esecsy;
- Finalmente, comparar o tempo de resposta entre dois algoritmos distintos: A-Teams e AG clássico. Ambos implementados pelo Esecsy.

Os resultados poderão fornecer dados para validar as regras estabelecidas no Capítulo 3, assim como prover indícios da eficiência do algoritmo apresentado no Capítulo 4.

A apresentação deste capítulo segue basicamente a mesma ordem estabelecida pelo método.

5.1 Ambiente de Execução

Todos os experimentos foram realizados no ambiente do Centro Nacional de Processamento de Alto Desempenho (CENAPAD) de Campinas, São Paulo [139]. O CENAPAD-SP disponibiliza um ambiente computacional baseado em máquinas RISC (IBM)

e Intel/Itanium2 (SGI), com sistema operacional baseado em Unix. O ambiente IBM é composto por 40 nós computacionais SMP modelo IBM P750, que utilizam processadores Power7. Cada nó possui 4 processadores IBM Power7 com 8 núcleos de 3.55 GHz. Totalizando, portanto, 32 núcleos de processamento; 128 GB de memória RAM e 908,8 GFlops de desempenho teórico, totalizando 1.280 cores, 5 TB de memória RAM e capacidade teórica de processamento de aproximadamente 37 TFLOPs. O sistema operacional instalado é o AIX 6.1 TL6 SP1.

A execução de programas (chamados de *Jobs*) no ambiente CENAPAD é realizada através da submissão dos *jobs* a um ambiente de filas, controlado por um programa chamado *LoadLeveler*. Porém, antes de submeter um *job* para execução, é necessária a criação de um arquivo de comandos, que descreve como o *job* deverá ser executado. Este arquivo de comandos assemelha-se a um *shell script*, que contém comandos dados por palavras chaves do *LoadLeveler* (diretivas). Um exemplo de arquivo de comando utilizado para executar o Esecsy é mostrado na Figura 5-1.

```
# @ notification = always
# @ notify_user = nlpvtzz@gmail.com
#
# @ job_name = esecsy
# @ job_type = parallel
# @ node = 1
#
# @ output = log/esecsy.out
# @ error = log/esecsy.err
# @ class = paralela
# @ queue
~/apache-tomcat-6.0.32/bin/startup.sh
java -cp .:./classes:../../owl-s-1.1.0-beta/lib/owl-s.jar testes/Esecsy
~/apache-tomcat-6.0.32/bin/shutdown.sh
```

Figura 5-1 – Arquivo de comandos do Esecsy enviado ao LoadLeveler.

Resumidamente, as diretivas utilizadas no arquivo de comando do Esecsy são:

- *notification*: condição em que o usuário receberá e-mail (início e fim de execução, erro etc.);
- *notify_user*: e-mail do usuário que receberá notificação;
- *job_name*: nome do *job* submetido;
- *job_type*: pode especificar execução serial ou paralela;
- *node*: número nós necessários para o processamento do *job*;
- *output*: nome do arquivo onde será gravada saída gerada pelo programa;
- *error*: nome do arquivo onde serão gravados os erros gerados pelo programa;
- *class*: *jobs* paralelos são processados por meio da submissão para as classes paralelas: par128 ou exp512. Cada uma destas classes tem características diferenciadas, como por exemplo: tempo de uso da CPU, tempo máximo (*wall_clock*) de execução e número de processadores alocados (pequena, media ou grande);
- *queue*: coloca uma cópia do *job* na fila, ou seja, indica o final de um *job*.

As três últimas linhas do arquivo de comandos indicam, respectivamente: a) a inicialização do servidor Tomcat [138]; b) execução do Esecsy; e c) o encerramento do Tomcat.

Aparentemente há uma contradição entre o *job* ser paralelo e necessitar de apenas um nó para sua execução (diretiva `# @ node = 1`). Mas o Esecsy foi implementado com *threads*, deixando o paralelismo a cargo do sistema operacional. Experimentos realizados com mais de um nó demonstraram que não houve diminuição no tempo de resposta do Esecsy. Portanto, os experimentos forma realizados em um nó apenas.

5.2 Base de Serviços Web Semânticos

A coleção de teste chamada OWLS-TC3 [132] foi a base de serviços utilizada para avaliar e medir o desempenho do sistema. Esta coleção consiste em 1.007 serviços especificados em OWL-S 1.1 de sete domínios diferentes: comunicação, economia, educação, alimentação, medicina, viagem e armamento. A coleção inclui também as

ontologias usadas na marcação dos serviços. Está publicamente disponível no endereço projects.semwebcentral.org/projects/owls-tc/. Atualmente, está na versão 4.

Esta coleção foi inicialmente criada para avaliar algoritmos de busca de serviços Web semânticos. Assim, algumas modificações precisaram ser feitas a fim adaptá-la ao problema da composição de serviços. Como na coleção original não há nenhum serviço cujo parâmetro de saída pode ser usado como o parâmetro de entrada de outro serviço, novos serviços tiveram de ser criados. Caso contrário, a composição sequencial, por exemplo, seria impossível. Apesar destas modificações, nenhum tipo novo de domínio ou do parâmetro (conceito terminológico) foi criado. Por isto foram acrescentados cinco novos serviços: _00_esecsy_university_researcher_service.owls, _01_esecsy_researcher_book_service.owls, _02_esecsy_city_book_service.owls, _03_esecsy_book_genre.owls e _04_esecsy_book_grade.owls. Na configuração deste projeto, a porta 8099 foi adicionada ao endereço do host, 127.0.0.1:8099, em todos os arquivos da OWLS-TC3. Doravante, para efeito de simplificação, estes serviços serão designados por S_0, S_1, S_2, AtS_3 e S_4, respectivamente. Os parâmetros de entrada e de saída destes serviços são definidos de forma arbitraria, mas sistematicamente, com o objetivo de validar e avaliar o desempenho do Esecsy e não possuem, necessariamente, significado físico. As ontologias referenciadas são parte da OWLS-TC3 (Figura 5-2).

```
Serviço: _00_esecsy_university_researcher_service
Parâmetros de Entrada
 Nome: http://127.0.0.1:8099/services/1.1/university_researcher_service.owls#_UNIVERSITY
 Tipo: http://127.0.0.1:8099/ontology/portal.owl#University
Parâmetros de Saída
 Nome: http://127.0.0.1:8099/services/1.1/university_researcher_service.owls#_RESEARCHER
 Tipo: http://127.0.0.1:8099/ontology/portal.owl#Researcher

Serviço: _01_esecsy_researcher_book_service
Parâmetros de Entrada
 Nome: http://127.0.0.1:8099/services/1.1/researcher_book_service.owls#_RESEARCHER
 Tipo: http://127.0.0.1:8099/ontology/portal.owl#Researcher
Parâmetros de Saída
 Nome: http://127.0.0.1:8099/services/1.1/researcher_book_service.owls#_BOOK
 Tipo: http://127.0.0.1:8099/ontology/books.owl#Book

Serviço: _02_esecsy_city_book_service.owls
Parâmetros de Entrada
 Nome: http://127.0.0.1:8099/services/1.1/city_book_service.owls#_CITY
 Tipo: http://127.0.0.1:8099/ontology/travel.owl#City
Parâmetros de Saída
 Nome: http://127.0.0.1:8099/services/1.1/city_book_service.owls#_BOOK
 Tipo: http://127.0.0.1:8099/ontology/books.owl#Book

Serviço: _03_esecsy_book_genre_service.owls
Parâmetros de Entrada
 Nome: http://127.0.0.1:8099/services/1.1/book_genre_service.owls#_BOOK
 Tipo: http://127.0.0.1:8099/ontology/books.owl#Book
Parâmetros de Saída
 Nome: http://127.0.0.1:8099/services/1.1/book_genre_service.owls#_GENRE
 Tipo: http://127.0.0.1:8099/ontology/books.owl#Genre

Serviço: _04_esecsy_book_grade_service.owls
Parâmetros de Entrada
 Nome: http://127.0.0.1:8099/services/1.1/book_grade_service.owls#_BOOK
 Tipo: http://127.0.0.1:8099/ontology/books.owl#Book
Parâmetros de Saída
 Nome: http://127.0.0.1:8099/services/1.1/book_grade_service.owls#_GRADE
 Tipo: http://127.0.0.1:8099/ontology/books.owl#Grade
```

Figura 5-2 – Os nomes e tipos dos parâmetros de entrada e saída dos serviços S_0, S_1, S_2, S_3 e S_4 acrescentados à base de dados OWLS-TC3.

Além dos cinco serviços que foram acrescentados, 81 foram eliminados por falta de parâmetros de entrada ou de saída, restando, portanto, 931 serviços dos 1.007

originais. A soma dos parâmetros de entrada e de saída de todos estes serviços contabiliza 1.247 e 1.362, respectivamente.

Os serviços originais desta base, assim como aqueles criados exclusivamente para a validação e avaliação deste trabalho são identificados como os serviços atômicos utilizados durante o processo de composição. Entretanto, nem todos os serviços atômicos presentes no repositório são usados em todos os experimentos. Os serviços atômicos usados em cada experimento são definidos pelas variáveis de configuração do sistema (ver seção 5.3).

5.3 Configuração de Referência do Esecsy

O Esecsy possui muitas variáveis de configuração. Algumas delas podem influenciar diretamente o desempenho do sistema. Para estas variáveis foram estabelecidos valores de referência, obtidos de forma intuitiva ou através de pré-testes. Estes valores são apenas valores iniciais que deverão ser testados. Aliás, o objetivo de muitos experimentos é, exatamente, encontrar um valor para algumas destas variáveis que otimize o desempenho do sistema. Os valores iniciais estabelecidos são chamados de valores de referência. As variáveis com seus respectivos valores de referência são:

- Serviço inicial ← 0. São considerados os serviços da base de dados OWLS-TC3 listados em ordem alfabética. As variáveis *serviço inicial* e *final* indicam a faixa de serviços que serão utilizados;
- Serviço final ← 19. Os vinte primeiros serviços compartilham as mesmas ontologias;
- Tamanho mínimo da população (popMin) ← 20. Na prática indica que o agente exterminador não pode eliminar nenhum indivíduo se a quantidade de indivíduos presentes na população estiver abaixo deste valor. Talukdar et al. [15] utiliza times assíncronos para resolver o problema do caixeiro viajante e após alguns experimentos escolhem uma população de tamanho igual à quantidade de cidades;

- Tamanho máximo da população (popMax) ← 30. Quando a população atinge o seu valor máximo, os agentes construtores não podem ser executados, portanto, somente o agente exterminador pode atuar;
- Profundidade ← 8. As árvores, que representam os indivíduos, estão limitadas a oito níveis de profundidade;
- *Fitness* desejado ← 1.0. Valor utilizado como critério de parada;
- Método de seleção ← roleta. Também podem ser utilizados os métodos cego e torneio. A seleção por roleta é a padrão (ver seção 4.9);
- Tipo de Nascimento ← perene. Podem ser configurados três tipos: aleatório, similaridade sintática e perene. Os tipos aleatório e similaridade sintática são efêmeros (ver seção 0);
- Taxa de amostragem ← 100 ms. Tempo utilizado pelo agente observador para acompanhar a execução e atualizar o arquivo de *log*;
- Tempo de dormência da *thread* ← 0 ms. Período de tempo em que cada *thread* deverá dormir depois de cada iteração;
- Quantidade de agentes ← 1. Define a quantidade de *threads* em execução de cada agente.

Os valores destas variáveis sofrem alterações de acordo com o objetivo de cada experimento. Haverá indicação sempre que ocorrerem alterações, caso contrário, serão utilizados os valores de referência.

5.3.1 Serviço Abstrato de Referência

O serviço abstrato utilizado na configuração de referência é chamado query_2seq_2par_university_genre_grade.owls, possui um parâmetro de entrada e dois parâmetros de saída, conforme ilustrado na Figura 5-3. Este serviço foi criado exclusivamente para avaliação de desempenho do Esecsy.

```
Query: query_2seq_2par_university_genre_grade.owls
  Parâmetros de Entrada
    Nome: http://127.0.0.1:8099/services/1.1/book_price_service.owls#_UNIVERSITY
    Tipo: http://127.0.0.1:8099/ontology/portal.owl#University
  Parâmetros de Saída
    Nome: http://127.0.0.1:8099/services/1.1/book_price_service.owls#_GRADE
    Tipo: http://127.0.0.1:8099/ontology/books.owl#Grade
    Nome: http://127.0.0.1:8099/services/1.1/book_price_service.owls#_GENRE
    Tipo: http://127.0.0.1:8099/ontology/books.owl#Genre
```

Figura 5-3 – Serviço abstrato de referência.

5.3.2 Composições Encontradas para o Serviço Abstrato de Referência

Considerando o serviço abstrato de referência (Figura 5-3) e base OWLS-TC3[41] (Figura 5-2), foram realizados experimentos para verificar a exatidão do Esecsy na obtenção de composições que atendem aos requisitos do serviço abstrato apresentado. Duas soluções eram esperadas: $(S_0 \prec S_1) \prec (S_4 \parallel S_3)$ e $S_0 \prec \big(S_1 \prec (S_4 \parallel S_3)\big)$. A Figura 5-4 representa a estrutura em árvore destas duas soluções.

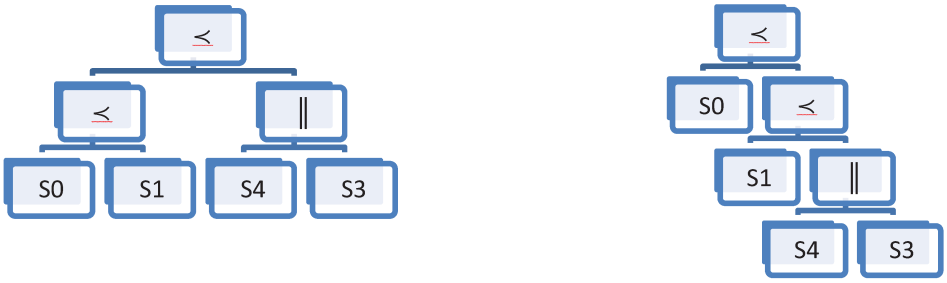

Figura 5-4 – A estrutura em árvore das duas soluções esperadas.

O serviço S_0possui apenas um parâmetro de entrada e um de saída: $University$ e $Researcher$, respectivamente. O parâmetro de saída, $Researcher$, é exatamente do mesmo tipo do parâmetro de entrada do serviço S_1. Por sua vez, o parâmetro de saída do serviço S_1, $Book$, é o parâmetro de entrada dos serviços e S_3 e S_4; entretanto, S_3 possui $Genre$, como parâmetro de saída, enquanto, S_4, $Grade$. A composição paralela entre S_3 e

[41] Com as alterações apresentadas na seção 5.2.

S_4 possui $Book$ como parâmetro entrada e $Genre$ e $Grade$ como parâmetros de saída. Estas duas composições atendem exatamente aos requisitos estabelecidos pelo serviço abstrato, representando a solução ótima do problema (Figura 5-5).

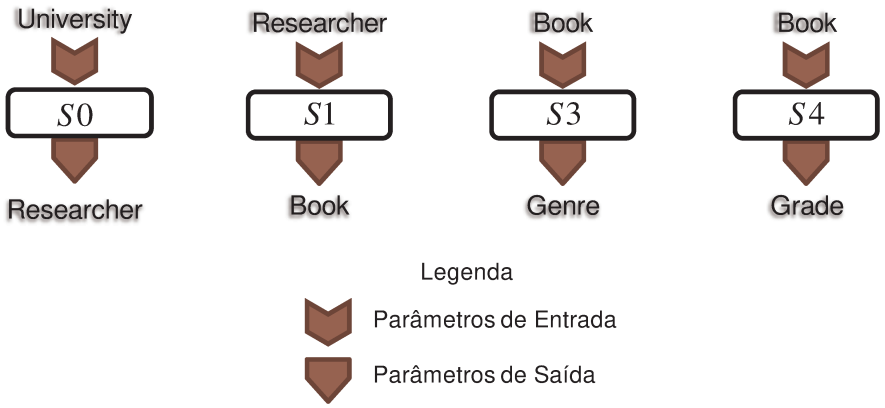

Figura 5-5 – Os serviços S_0, S_1, S_3 e S_4 que fazem parte da solução ótima do serviço abstrato de referência.

As duas soluções apresentadas na Figura 5-4 são ótimas porque apresentam *fitness* igual a um e suas árvores possuem sete nós, sendo que três representam tipos de composições (nós intermediários) e, quatro, serviços atômicos (nós folhas). Esta é a menor árvore possível considerando os serviços disponíveis na coleção OWLS-TC3 modificada. Entretanto, além destas duas soluções ótimas, o Esecsy encontrou outras soluções. Algumas delas estão representadas na Figura 5-6. Todas possuem *fitness* igual a um, isto é, todos os casamentos internos e externos são exatos. Mas não são soluções ótimas porque a quantidade de nós excede a quantidade apresentada pela solução ótima.

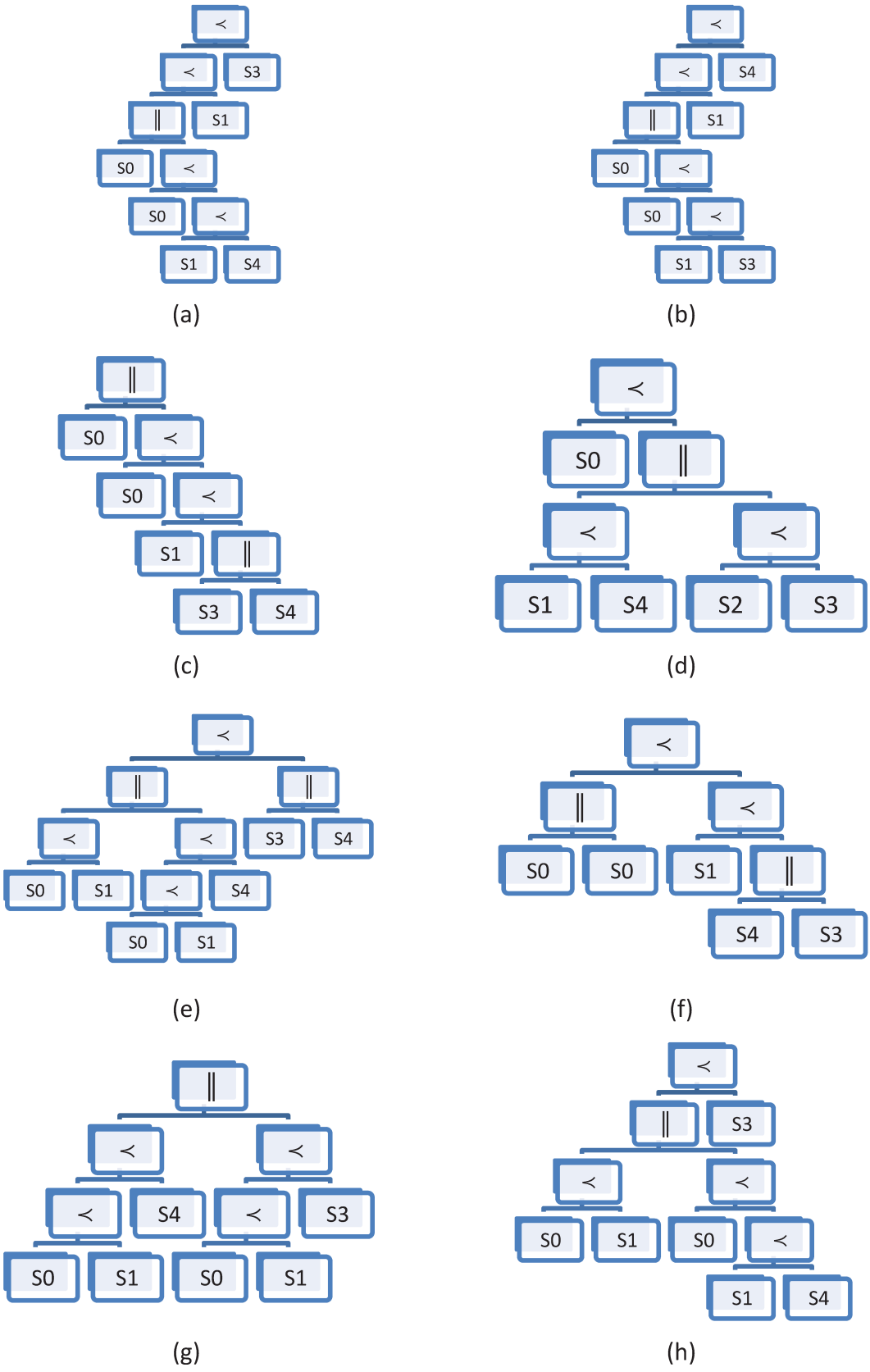

Figura 5-6 – Algumas soluções encontradas pelo Esecsy.

A análise destas árvores revela algumas soluções originais. A composição apresentada pela árvore b (Figura 5-6) possui três parâmetros de saída, $Researcher$, $Genre$ e $Grade$. Isto significa que esta composição oferece mais do que o necessário para satisfazer os requisitos do serviço abstrato. Esta característica é consequência direta das regras de composição definidas no Capítulo 3. O ramo $\left(S_0 \parallel (\mathrm{S}_0 \prec \mathrm{S}_1 \prec \mathrm{S}_3)\right)$ possui dois parâmetros de saída: $Researcher$ e $Genre$. Destes, o parâmetro $Researcher$ de S_0 compõe-se sequencialmente com o parâmetro de entrada de S_1. E o parâmetro de saída de S_1 compõe-se com o parâmetro de entrada de S_3. O parâmetro de saída de S_3, $Genre$, não se casa com nenhum parâmetro de entrada de S_1 e, portanto, torna-se parâmetro de saída da composição final.

Na árvore d, os serviços S_1 e S_4 são sequencialmente compostos, assim como os serviços S_1 e S_3. Estas duas composições são novamente compostas em paralelo e, então, o serviço S_0 é sequencialmente composto com este resultado parcial. Isto é possível porque o parâmetro de entrada do serviço S_1, $Researcher$, casa-se exatamente com a saída do serviço S_0. A análise das outras árvores também poderá revelar soluções interessantes, mas não é necessário delongar-se neste ponto. Estas soluções foram obtidas depois 21.000 amostras realizadas.

5.4 Análise da Função Densidade de Probabilidade do Tempo de Resposta Médio (TRM)

Experimentos preliminares realizados com 2.000 amostras mostraram que a função de densidade de probabilidade para o TRM, utilizando o método Kolmogorov-Smirnov, converge para a função de distribuição Weibull (Equação 5.1), cujo domínio é $\gamma \leq x \leq +\infty$, sendo: α parâmetro de curva contínuo $(\alpha > 0)$; β parâmetro escalar contínuo $(\beta > 0)$; e, γ parâmetro de localização contínuo.

$$f(x) = \frac{\alpha}{\beta}\left(\frac{x-\gamma}{\beta}\right)^{\alpha-1} e^{-\left(\frac{x-\gamma}{\beta}\right)^{\alpha}} \quad (5.1)$$

A Figura 5-7 ilustra a função de densidade de probabilidade para o TRM do Esecsy realizadas com a configuração de referência para 2.000 amostras. Os valores da distribuição Weibull são: $\alpha = 0{,}81511$, $\beta = 1311{,}4$ e $\gamma = 14{,}0$.

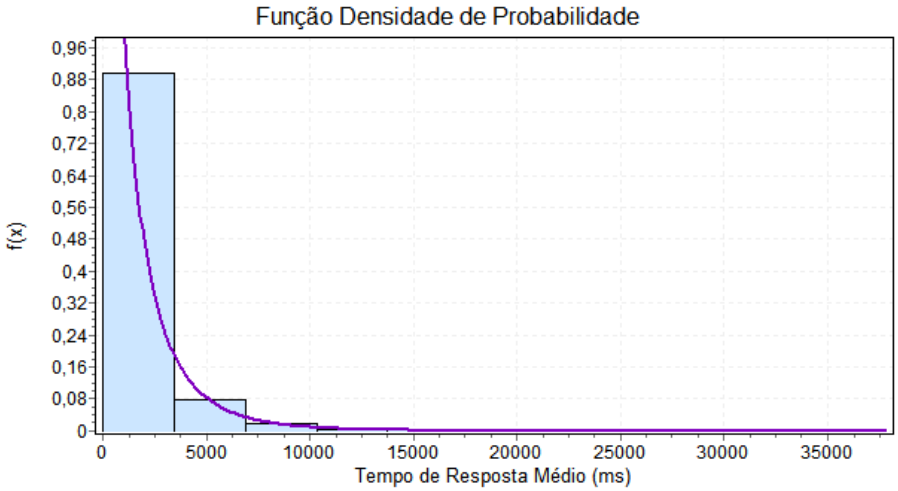

Figura 5-7 – Função densidade de probabilidade do TRM para a configuração de referência.

Para avaliar o desvio padrão (DP) do TRM foram realizados experimentos variando-se a quantidade de amostras entre 100 e 2.000. A partir de 600 amostras, o DP se estabiliza. Os resultados são ilustrados na Figura 5-8 (no gráfico, o eixo das ordenadas representa tanto o tempo, em ms para TRM e DP, quanto a unidade, para QtF; esta dupla função permanecerá nos gráficos seguintes). A quantidade de vezes que a função de *fitness* é avaliada (QtF) mantém-se praticamente constante desde o início. As pequenas oscilações presentes no DP também são sentidas no QfF, demonstrando, portanto, alta correlação entre ambas.

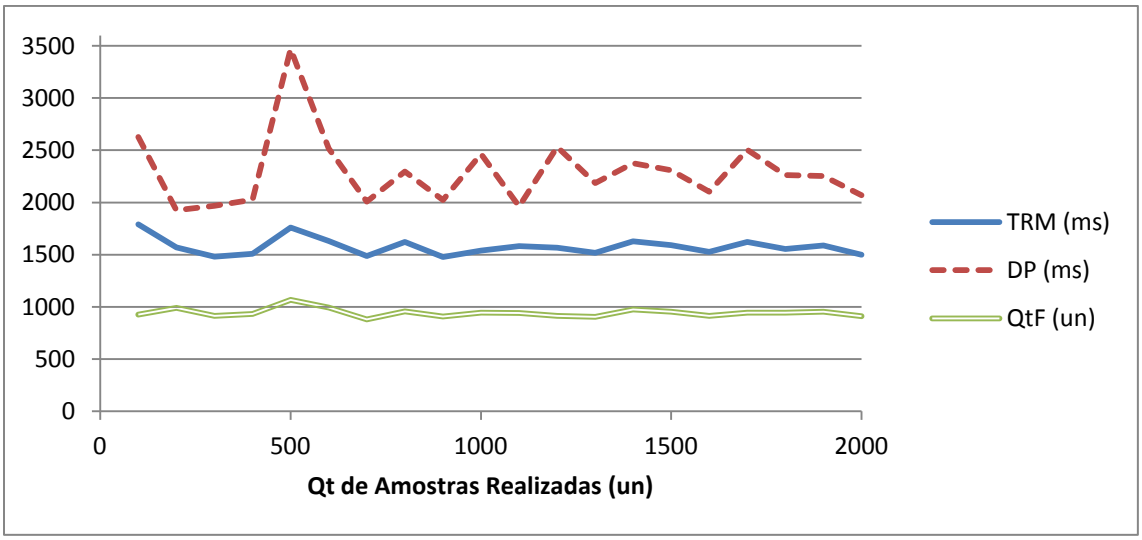

Figura 5-8 – TRM, DP e QtF em função da quantidade de amostras realizadas. O eixo das ordenadas representa tanto o tempo (em ms para TRM e DP) quanto a unidade (para QtF).

Quantidade de Amostras Realizadas

Sabendo-se que para uma amostragem de tamanho n, o intervalo de confiança de uma média populacional é [140, p. 216]:

$$\bar{x} \pm z \frac{s}{\sqrt{n}} \tag{5.2}$$

sendo: $\bar{x}$ a média populacional; z a probabilidade da distribuição entre 0 e *Z*; e s, o DP.

Para o experimento com 2.000 amostras supracitado, o TRM foi de 1.501 ms e o DP, 2.069 ms. Para a distribuição Weibull, $Z_{0,95} = 2{,}0781$. Portanto, tem-se que $n = 58$. Isto significa que, para um intervalo de confiança de 95% pelo menos 58 amostras são necessárias. Entretanto, doravante, os experimentos serão realizados com pelo menos 200 amostras, o que dá, aproximadamente, uma confiança de 98%.

5.5 Avaliação da Função de *Fitness*

Para avaliar a eficiência da função de *fitness*, será feita a comparação do desempenho do sistema através da seleção por roleta e cega. Na seleção por roleta, a probabilidade de um indivíduo ser sorteado é proporcional ao seu *fitness*: quanto maior o

fitness, maior será a probabilidade de ser selecionado; e vice-versa. Este método de seleção é utilizado em todos os casos em que é necessário selecionar um indivíduo, isto é, nos casos de mutação, cruzamento, composição sequencial e paralela. Também é utilizado pelo exterminador, mas neste caso, de forma inversa, isto é, quanto maior o *fitness*, menor será probabilidade de o indivíduo ser selecionado.

Na seleção cega, o *fitness* do indivíduo é ignorado, significando que todos possuem a mesma probabilidade de serem selecionados.

A comparação entre seleção cega e por roleta foi realizada para dois serviços abstratos diferentes. Para o primeiro, a solução é uma composição sequencial entre os serviços $S0$ e $S1$, tal que, $S0 \prec S1$; para o segundo, a solução é a composição $(S0 \prec S1) \prec (S4 \parallel S3)$. A Tabela 5-1 apresenta os resultados da comparação.

Tabela 5-1 – TRM, DP e QtF para a seleção por roleta e cega.

Composição	Tipo de Seleção	TRM (ms)	DP (ms)	QtF (un)
$\boldsymbol{S0 \prec S1}$	Cega	1.304	1.371	942
	Roleta	28	27	31
$\boldsymbol{(S0 \prec S1) \prec (S4 \parallel S3)}$	Cega	1.421.340	1.532.955	847.345
	Roleta	1.667	2.781	841

Para a composição $S0 \prec S1$, o TRM da seleção por roleta é 46,57 vezes menor que o TRM da seleção cega; isto representa um ganho de 97,85%. Na seleção por roleta, ocorre aproximadamente o calculo de 1 *fitness* por ms, enquanto na seleção cega, este índice cai para 0,72. Também se pode observar que o valor do DP para a seleção por roleta é bem menor do que o mesmo valor para a seleção cega.

Valores mais díspares podem ser observados para a composição $(S0 \prec S1) \prec (S4 \parallel S3)$. A seleção por roleta é 852,63 vezes mais rápida ou, em termos porcentuais, representa um ganho de 99,88%. Efetivamente, estes valores atestam não só que a heurística representada pela função de *fitness* é adequada ao problema, mas que também quanto maior é complexidade da composição, maior é a eficiência desta heurística.

Validação de Hipótese

Para validar a hipótese de que a função de *fitness* realmente aumenta a eficiência da busca em relação à busca cega, será utilizado o método de intervalo de confiança de um único lado [140, p. 213]. Primeiramente será validada a hipótese para a composição $S0 \prec S1$.

A diferença do TRM entre a seleção por roleta e cega é de -1.276 ms. O DP desta diferença é 96,96. O número efetivo dos graus de liberdade é 199,15 ms. Utilizando a distribuição Weibull, o valor de Z para 95% de intervalo de confiança é $z_{0,95} = 2{,}0781$, que resulta em um intervalo de confiança entre -1.276,00 e -1.074,50 ms. Como o intervalo não inclui o zero pode-se efetivamente concluir que a heurística utilizada na função de *fitness* diminui o TRM do problema.

Como os valores para a composição $(S0 \prec S1) \prec (S4 \parallel S3)$ são mais díspares, pode-se concluir, sem a necessidade do teste de hipótese, que a função de *fitness* também diminui o TRM do problema. Somente para registrar, o intervalo de confiança, neste caso, é de -1.419.673 a -1.194.415 ms.

Resumidamente, este experimento mostra a eficiência da heurística adotada pela função de *fitness* face à seleção cega; e que quanto maior a complexidade da composição, maior é a eficiência desta heurística.

5.6 Taxa de Cruzamento

Geralmente, a taxa de cruzamento é um importante fator de especificação dos algoritmos genéticos. A taxa de cruzamento determina a porcentagem de indivíduos, em relação à população total, que serão produzidos por cruzamento a cada nova geração. Como não há o conceito de geração nos algoritmos A-Teams, mas o Esecsy utiliza o operador cruzamento, duas perguntas surgem: é possível especificar uma taxa de cruzamento? E, caso seja possível, tem esta taxa influência no desempenho do sistema?

Primeiramente, é possível sim especificar uma taxa de cruzamento, especificando um tempo de dormência (*sleep time*) da *thread* que executa o cruzamento. A cada execução, a *thread* produz um indivíduo (ou dois, considerando-se a inversão entre os papéis de pai e mãe) e dorme. Quanto maior o tempo de dormência desta *thread*, menor será a quantidade de indivíduos produzidos por ela.

Para responder à segunda pergunta, foi realizado um experimento variando-se o tempo de dormência da *thread* em 0 a 1.000 ms, com intervalos de 10 ms. Para cada experimento foram realizadas 200 amostras. Os resultados são apresentados na Figura 5-9. Observa-se uma relação linear entre o TRM e o tempo de dormência, dada pela equação $y = 0{,}0289x + 1137{,}9$. O QtF acompanha a evolução do TRM.

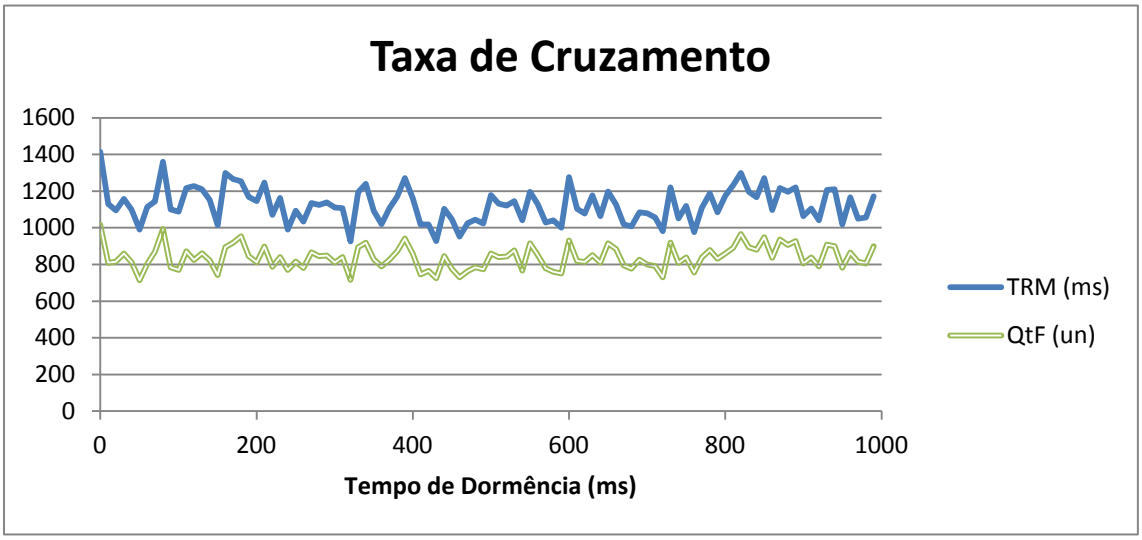

Figura 5-9 – TRM e QtF em função do tempo de dormência da *thread* de cruzamento.

Sendo o coeficiente angular muito próximo de zero, 0,0289, pode-se concluir que a taxa de cruzamento influencia de forma bastante suave o TRM. Ou seja, a *thread* de cruzamento praticamente não interfere na busca da solução. Este resultado aparentemente invalida a afirmação de que é possível especificar uma taxa de cruzamento, mas há uma hipótese para este resultado: a evolução dos indivíduos está sendo mais influenciado pelos outros agentes. Para testar esta hipótese, será necessário

manipular o tempo de dormência da mutação, da composição paralela e da composição sequencial.

5.7 Taxa de Mutação

Neste experimento foram mensurados o TRM e o QtF do sistema em relação ao tempo de dormência da *thread* que realiza a mutação. A taxa de dormência variou de 0 a 1.000 ms, intervalos de 10 ms, e para cada variação foram realizadas 200 amostras.

O desempenho do sistema é registrado no gráfico da Figura 5-10. O TRM em relação à taxa de dormência da *thread* de mutação varia linearmente com o tempo de dormência ($y = 0{,}0132x + 1207{,}2$).

Similar ao comportamento da taxa de cruzamento, a taxa de mutação também possui pouca influência no desempenho do sistema.

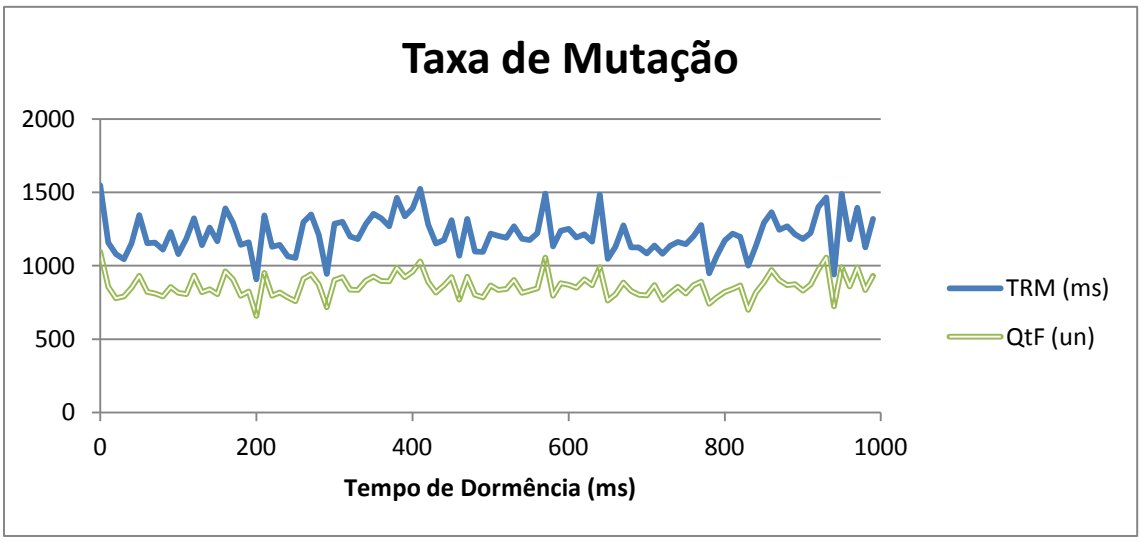

Figura 5-10 – TRM e QtF em função do tempo de dormência da *thread* de mutação.

5.8 Taxa de Composição Paralela

Neste experimento foram mensurados o TRM e a QtF do sistema em relação ao tempo de dormência da *thread* que realiza a composição em paralelo. A taxa de

dormência variou de 0 a 200 ms, intervalos de 10 ms, e para cada variação foram realizadas 200 amostras.

É possível verificar no gráfico da Figura 5-11 que o TRM é diretamente proporcional à taxa de dormência da *thread* de paralelo. Com relação à QtF, ele praticamente acompanha a curva do TRM.

A dependência do desempenho do sistema em relação ao tempo de dormência desta *thread* explica-se pelo fato de que a solução ao serviço abstrato apresentado possui uma composição paralela. Portanto, este resultado avaliza a atuação precisa e correta desta *thread*. Indica, também, que quanto maior a quantidade de composições em paralelo maior a dependência em relação a esta *thread*, ou de forma mais generalista, o sistema deve apresentar uma dependência direta entre o tipo de composição e o tempo de resposta da *thread* correspondente. Se esta hipótese for verdadeira, comportamento semelhante deverá ser encontrado nos experimentos referentes à taxa de dormência da *thread* sequencial.

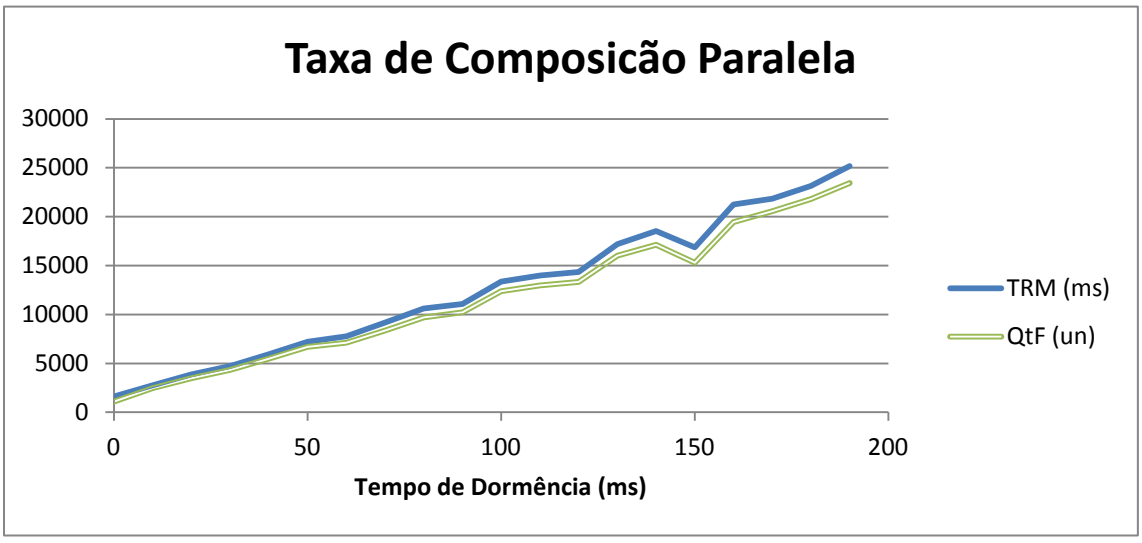

Figura 5-11 – TRM e QtF em função do tempo de dormência da *thread* de composição em paralelo.

5.9 Taxa de Composição Sequencial

Com efeito, este experimento demonstra a forte relação existente entre o tipo de composição presente na resposta do sistema e a taxa de execução da *thread* responsável por produzir tal composição. Usando regressão polinomial, obtém-se a a aproximação dada pela equação $y = 76{,}045x^3 - 3798{,}4x^2 + 57983x - 39954$ $(R^2 = 0{,}9921)$, onde y representa o TRM e x o tempo de dormência da *thread* sequencial (Figura 5-12). O tempo de dormência variou de 0 a 50 ms, intervalo de 10 ms, e para cada intervalo foram realizadas 200 amostras.

Voltando-se à Figura 5-6, a quantidade de composições sequenciais presentes na maioria das soluções é maior que a quantidade de composições paralelas. Por isto, a dependência em relação a esta *thread* é maior do que em relação à *thread* paralela. Estes resultados também pode explicar a pouca influência das *threads* de cruzamento e mutação.

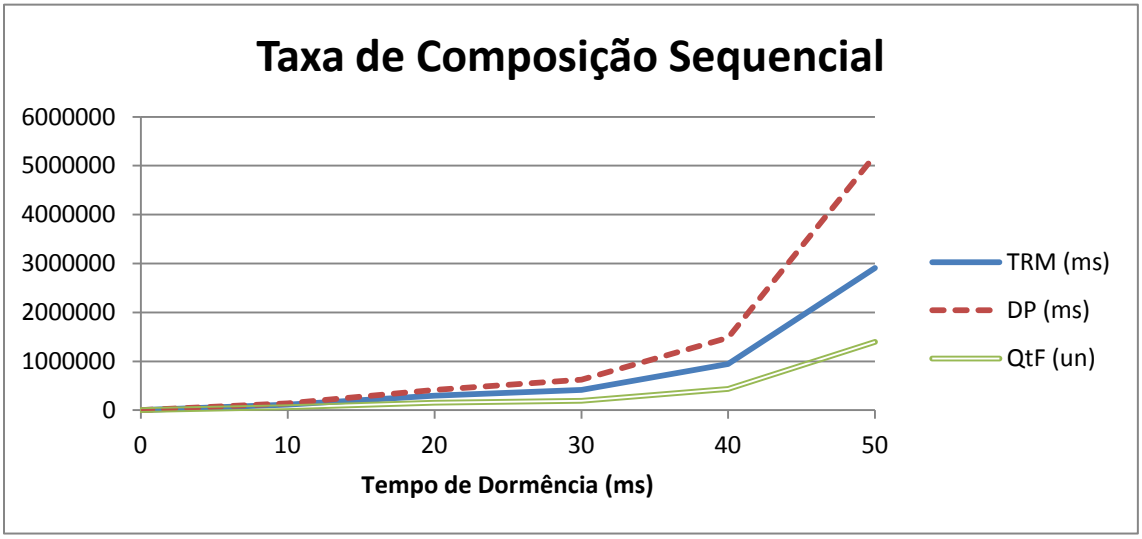

Figura 5-12 – TRM e QtF em função do tempo de dormência da *thread* de composição sequencial.

5.10 Valor dos Operadores Genéticos

Diante da baixa influência das *threads* de cruzamento e mutação no TRM, foi realizado um experimento sem estas *threads*. Os resultados estão dispostos na Figura

5-13. Realmente, constata-se que o Esecsy fica mais rápido quando estas *threads* são retiradas. Todos os índices, TRM, DP e QtF, ficam melhores sem o cruzamento e a mutação. Entretanto, para manter a uniformidade entre os ensaios, estas *threads* serão mantidas nos próximos experimentos.

Tanto o ponto (nó da árvore) de mutação quanto o ponto de cruzamento são escolhidos de forma cega, enquanto o serviço que será posicionado neste nó é escolhido por roleta. A cada mutação ou cruzamento, o *fitness* de cada serviço presente na população é recalculado levando-se em consideração os parâmetros de entrada e saída que precisariam estar presentes no serviço escolhido para que este nó da árvore tivesse *fitness* igual a 1. Em suma, se o ponto de mutação ou de cruzamento é cegamente selecionado, o serviço que tomará lugar neste ponto, não o é. Isto leva à hipótese de que a baixa interferência das *threads* de mutação e cruzamento no desempenho do sistema é resultante da escolha cega dos pontos de cruzamento e mutação.

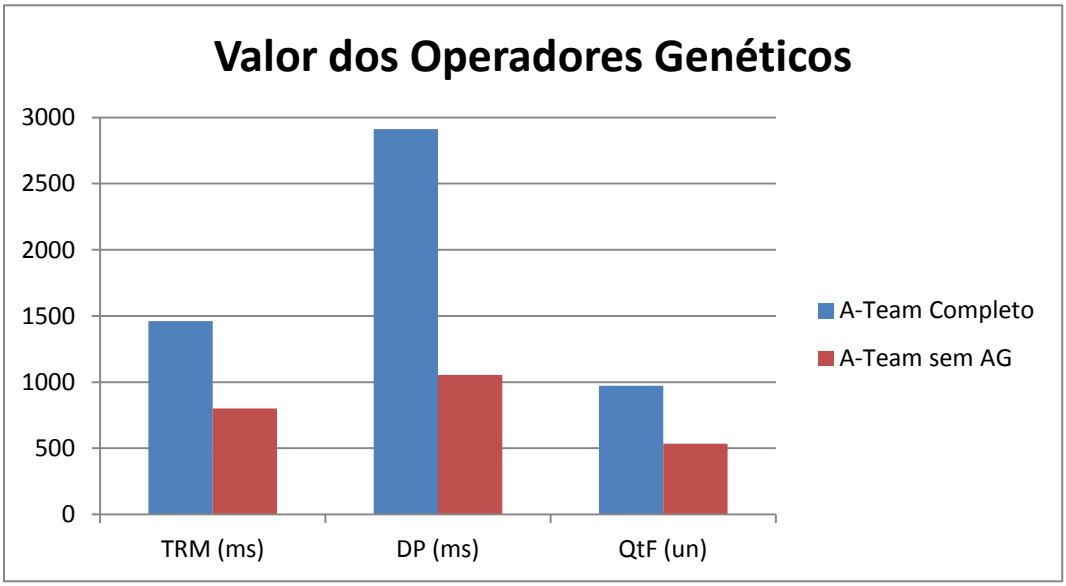

Figura 5-13 – Comparação entre o A-Team com e sem os operadores genéticos.

5.11 Seleção por Torneio

Três métodos de seleção foram implementados: roleta, torneio e cego. E neste experimento é feita análise de desempenho da seleção por torneio em função da

quantidade de competidores. Os resultados são explicitados na Tabela 5-2. Os experimentos foram feitos variando-se a quantidade de competidores entre dois e dezenove. Com um único competidor, a seleção torna-se cega e o resultado deste método de seleção está transcrito na Tabela 5-1.

Tabela 5-2 – TRM, DP e QtF em função da quantidade de competidores por torneio.

Qt Competidores por Torneio (un)	TRM (ms)	DP (ms)	QtF (un)
2	61.178	61.867	43.226
3	21.686	42.921	13.858
4	8.227	10.630	5.292
5	5.146	9.770	3.119
6	3.247	5.125	2.196
7	1.880	3.971	1.338
8	1.649	2.395	1.213
9	1.886	4.460	1.344
10	1.797	2.872	1.401
11	1.637	2.203	1.319
12	1.526	1.684	1.276
13	1.953	2.228	1.513
14	2.937	4.431	1.871
15	2.989	4.712	1.933
16	4.446	6.222	2.866
17	7.967	18.290	4.816
18	14.167	23.777	9.172
19	27.923	31.773	19.799

O TRM decresce à medida que a quantidade de competidores aumenta entre dois e doze, atingindo o valor mínimo neste último: 1.526 ms. Neste intervalo, o TRM cai potencialmente à taxa de $y = 180073x^{-2,089} (R^2 = 0,9344)$. A partir de doze

competidores, o TRM cresce exponencialmente à taxa de $y = 9.9815e^{0,4005x} (R^2 = 0,9473)$.

Para o método de seleção por roleta, o TRM é de 1.667 ms (Tabela 5-1), portanto, acima apenas do tempo obtido para onze e doze competidores por torneio, 1.637 e 1.526 ms, respectivamente (Tabela 5-2). Como todos estes valores estão muito próximos, será necessário fazer a validação da hipótese para poder concluir que o TRM obtido pelo método de seleção por torneio com doze competidores é realmente menor do que o TRM obtido pelo método de seleção por roleta.

Validação de Hipótese

A diferença do TRM obtido pelo método da seleção por torneio com doze competidores e pelo método da seleção por roleta é de -141 ms. O DP desta diferença é de 119,02 ms. O número efetivo de graus de liberdade é 199,10. Utilizando a distribuição Weibull, o valor de Z para 95% de intervalo de confiança é $z_{0,95} = 2,0781$, que resulta em um intervalo de confiança entre -141,00 e 106,49 ms. Como o intervalo inclui o zero, rejeita-se a hipótese de que a seleção por torneio é mais rápida do que a seleção por roleta.

5.12 Tipo de Nascimento

Aqui o TRM é analisado em relação ao tipo de nascimento: efêmero, perene ou similaridade semântica. A Tabela 5-3 mostra que o tipo de nascimento perene é mais rápido do que os tipos de nascimento efêmero e similaridade semântica. O fato do tipo de nascimento efêmero perder seus indivíduos durante o processo de evolução e do tipo de nascimento similaridade semântica ficar procurando por serviços durante o processo evolutivo, provoca o aumento dos seus respectivos TRM. Entretanto, estes dois tipos de nascimento continuam sendo opções conceituais interessantes para ambientes mais dinâmicos (ver seção 4.7.1).

Tabela 5-3 – TRM, DP e QtF em função do tipo de nascimento.

Tipo de Nascimento	TRM (ms)	DP (ms)	QtF (un)
Efêmero	5.765	8.250	1.898
Perene	1.667	2.781	841
Similaridade Semântica	3.968	6.121	1.465

A rapidez com que o tipo de nascimento perene encontra a solução pode ser ainda mais conclusiva quando analisado o QtF em relação ao TRM. Para o tipo de nascimento efêmero, um *fitness* é calculado a cada 0,33 ms, para a similaridade semântica a cada 0,37 ms e, para o tipo perene, a cada 0,67 ms.

Validação de Hipótese

Será feita a validação para as seguintes hipóteses:

- O tipo de nascimento perene é mais rápido do que o tipo de nascimento similaridade semântica; e,
- O tipo de nascimento similaridade semântica é mais rápido do que o tipo efêmero. Se tais hipóteses forem verdadeiras, conclui-se, consequentemente que o tipo perene é o mais rápido entre todos.

A diferença do TRM entre o tipo de nascimento perene e similaridade semântica é de -2.301 ms. O DP desta diferença é 432,82ms. O número efetivo dos graus de liberdade é 199,00. Utilizando a distribuição Weibull, o valor de Z para 95% de intervalo de confiança é $z_{0,95} = 2{,}0781$, que resulta em um intervalo de confiança entre –2.031,00 e -1.401,55 ms. Como o intervalo exclui o zero, aceita-se a hipótese de que o tipo de nascimento perene é mais eficiente do que o tipo similaridade semântica.

A diferença do TRM entre o tipo de nascimento similaridade semântica e efêmero é de -1.797 ms. O DP desta diferença é 726,39 ms. O número efetivo dos graus de liberdade é 368,83. Utilizando a distribuição Weibull, o valor de Z para 95% de intervalo de confiança é $z_{0,95} = 2{,}0781$, que resulta em um intervalo de confiança entre –1.797,00 e -

287,48 ms. Como o intervalo exclui o zero, aceita-se a hipótese de que o tipo de nascimento similaridade semântica é mais eficiente do que o tipo efêmero. E, portanto, o tipo de nascimento perene o mais rápido entre eles.

5.13 Tamanho da População

Os próximos dois experimentos avaliam a influência do tamanho da população no TRM do sistema. No primeiro experimento, as variáveis *popMin* (tamanho mínimo da população) e *popMax* (tamanho máximo da população) foram gradualmente incrementadas, mas mantendo fixa a diferença absoluta entre elas. No segundo experimento, a variável *popMin* é mantida constante, enquanto a variável *popMax* é gradativamente aumentada, portanto, a diferença entra ambas é, também, gradativamente aumentada.

5.13.1 Diferença Fixa

A Figura 5-14 mostra o TRM em relação ao tamanho da população. Neste experimento, as variáveis *popMin* e *popMax* foram gradualmente incrementadas de 20 até 990 e de 30 até 1000, respectivamente, em intervalos de 10 (a figura mostra somente o valor da variável *popMax*). O tamanho da população não pode ser menor do que 20 porque o iniciador é configurado para o tipo de nascimento perene e a quantidade de serviços atômicos disponíveis é 20. Isto significa que o iniciador insere 20 serviços na população e morre. Se algum destes serviços não for inserido, ou se for perdido durante a evolução, e caso ele seja necessário para a composição, a solução não será encontrada (supondo *fitness* desejado = 1). Neste contexto, o tamanho mínimo da população deve ser 20.

O TRM varia de acordo com a equação $y = 6{,}4878x + 2173{,}3$ $(R^2 = 0{,}798)$, sendo y o TRM e, x, o tamanho da população. Isto significa que o sistema mantém uma relação linear entre o TRM e a quantidade de indivíduos (serviços) presentes na população. Assim como no caso dos algoritmos genéticos clássicos, este experimento mostra que populações grandes requerem maiores esforços computacionais.

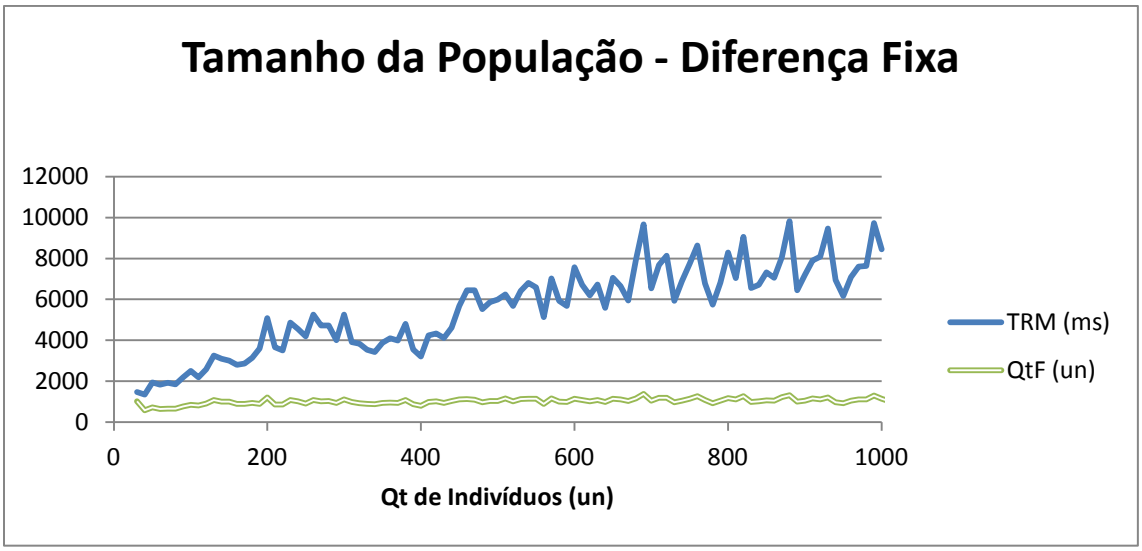

Figura 5-14 – TRM e QtF em função do tamanho da população, mantendo-se fixa a diferença entre as variáveis *PopMin* e *PopMax*.

5.13.2 Diferença Variável

Neste experimento, é avaliado como TRM se comporta em função do aumento do valor da variável *popMax*. A variável *popMim* manteve-se constante igual a 20 e a variável *popMaX* foi gradualmente incrementada de 30 até 1000 em intervalos de 10 (a Figura 5-15 mostra somente o valor da variável *popMax*).

Há uma leve tendência de aumento do TRM à medida que se aumenta o tamanho da população, visto que o coeficiente angular obtido pela regressão linear é positivo: $y = 0{,}9936x + 3439{,}7$. Entretanto, o valor de $R^2 = 0{,}1585$ é muito pequeno, significando baixa relação entre as duas variáveis. A aproximação por uma linha de tendência do tipo potência, $y = 2191{,}5x^{0{,}096}$, produz um $R^2 = 0{,}228$ um pouco melhor, e indica uma estabilidade do TRM. Mas, se assim o for, porque o TRM independe do valor de *popMax*? A análise da QtF também revela uma linha de tendência praticamente constante. A hipótese é que a quantidade de indivíduos mantida pelo sistema é praticamente constante e próxima do valor de *popMin*.

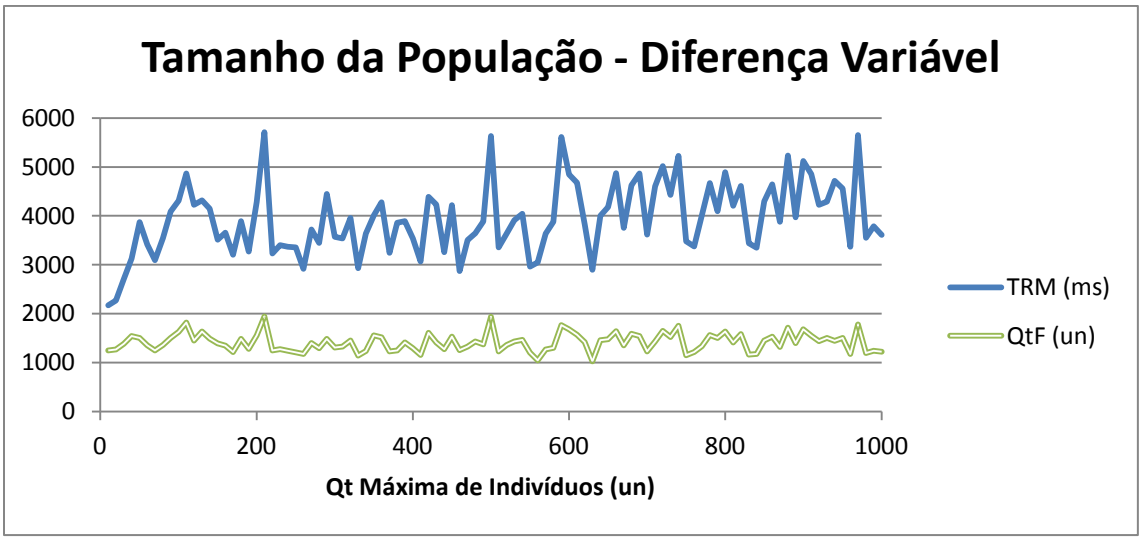

Figura 5-15 – TRM e QtF em função do tamanho da população, aumentado-se gradativamente a diferença entre as variáveis *PopMin* e *PopMax*

O experimento realizado na seção 5.13.1, tanto a variável *popMin* quanto a *popMax* foram incrementadas, isto é, a diferença entre *popMin* e *popMax* era constante. Mas neste experimento, a diferença foi gradativamente incrementa. Isto estabeleceu uma "largura de população" muito maior, e em última análise, uma quantidade de indivíduos muito mais maleável. Mas o exame dos arquivos de *log* revelou que a quantidade de indivíduos mantida pelo sistema foi, em média, 35. E é por isto que o TRM do sistema não se degrada à medida que o valor de *popMax* é incrementado, fato que leva à validação da hipótese de que o tamanho da população é mantido próximo de *popMin*.

5.14 Quantidade de Serviços Atômicos

Como analisado na seção 3.6, a quantidade de serviços disponíveis no repositório (μ) é fator determinante na caracterização da complexidade do algoritmo e, consequentemente, no tempo de resposta do sistema. Neste experimento, o objetivo é determinar, empiricamente, o tempo de resposta do Esecsy em função de μ.

Os experimentos foram realizados alterando-se a quantidade de serviços de 20 para 90, incrementos de 10. Isto significa que a cada execução, 10 novos serviços atômicos, retirados do repositório, foram inseridos na população. Foi utilizado o tipo de

nascimento perene (configuração de referência) e, como neste tipo de nascimento os serviços atômicos não podem ser exterminados, o tamanho da população também foi incrementado de 10 a cada iteração. Os resultados são apresentados na Figura 5-16.

No gráfico, o QtF permanece praticamente constante à medida que o TRM aumenta. Com uma quantidade maior de serviços atômicos, a QtF deveria aumentar. Como isto não ocorre, o TRM acaba se degradando.

Maior quantidade de serviços atômicos implica em maior esforço computacional, em compensação, há a possibilidade de se criar um número maior de composições. É previsível que com poucos serviços apropriados, o TRM seja pequeno. No limite, a melhor resposta deverá ser obtida quando a solução for uma composição dois serviços: os dois únicos serviços atômicos presentes na população.

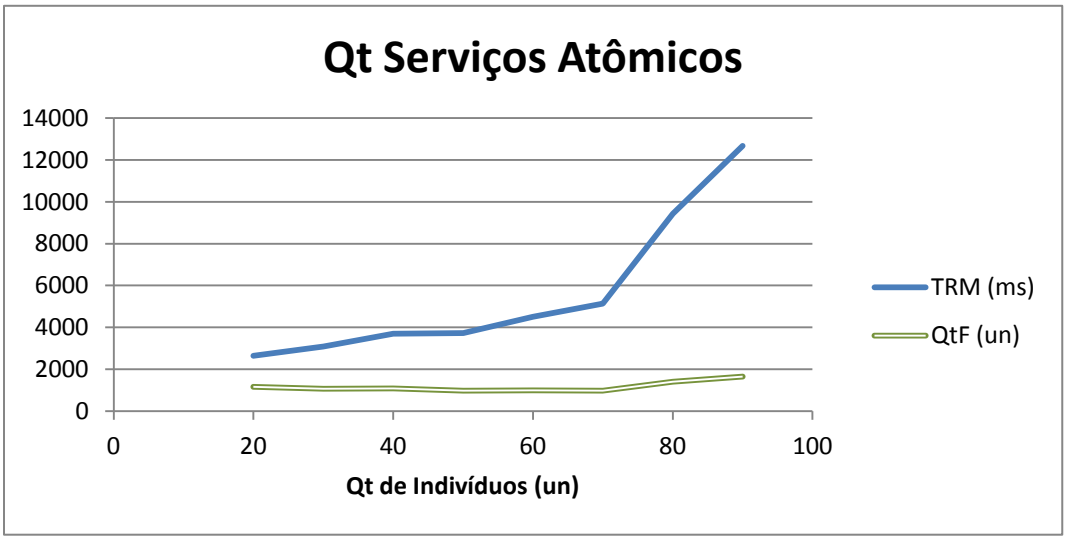

Figura 5-16 – TR e QtF em função da quantidade de serviços atômicos utilizados.

O TRM em função da quantidade de serviços atômicos é representado por um polinômio de terceiro grau: $y = 0{,}0655x^3 - 7{,}7148x^2 + 320{,}8x - 1215{,}8$ $(R^2 = 0{,}9839)$. Isto significa que, apesar de não garantir a melhor solução, o TRM do Esecsy é da ordem de $O(\mu^3)$.

5.15 Quantidade de Agentes

O tempo de dormência pode atrasar a execução de uma *thread*, mas não pode fazer com que ela seja executada mais rápido ou um número maior de vezes. Uma grande vantagem dos algoritmos A-Teams é a possibilidade de instanciar vários agentes clones (agentes com o mesmo código) de uma única vez. Como os processadores modernos possuem vários núcleos, a instanciação de vários agentes pode melhorar o desempenho do sistema (lembrando que os experimentos foram realizados em processadores IBM Power7 com 8 núcleos). Para atingir este ponto, os próximos experimentos serão feitos com a instanciação de vários agentes (ou, no caso, *threads*). Entretanto, deve existir um ponto ótimo na relação entre a quantidade de agentes em execução e o TRM do sistema, uma vez que a existência de uma região de exclusão mútua impede que mais de um agente acesse-a simultaneamente. Ademais, o gerenciamento e o tempo de chaveamento entre as *threads* devem causar alguma depreciação no sistema.

A Figura 5-17 ilustra a relação entre a quantidade de agentes de cruzamento instanciados e o TRM. Foram realizados 10 experimentos, variando-se a quantidade de agentes de cruzamento um em um. Os demais agentes, mutação, paralelo e sequencial, permanecem com apenas uma instância. O TRM aumenta à medida que a quantidade de agentes aumenta. Este resultado já era, de certa forma esperado, uma vez o tempo de dormência da *thread* de cruzamento (ver seção 5.6) teve pouca influencia no TRM.

Resultado semelhante é verificado quanto se aumenta a quantidade de agentes de mutação (Figura 5-18).

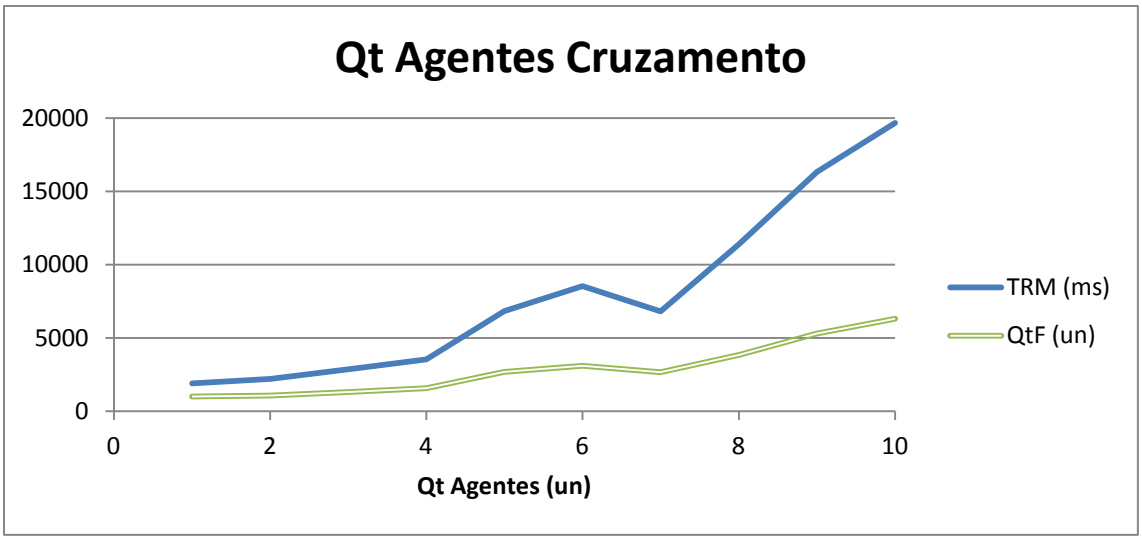

Figura 5-17 – TRM e QtF em função da quantidade de *threads* cruzamento.

Com relação à quantidade de agentes de composição paralela, o melhor desempenho do sistema é verificado quando se instancia três agentes (Figura 5-19). Os demais agentes continuam com apenas uma instância. Como a solução do problema apresentado possui uma composição em paralelo, um aumento na quantidade de agentes deveria contribuir para uma redução do TRM. Isto realmente se verificou.

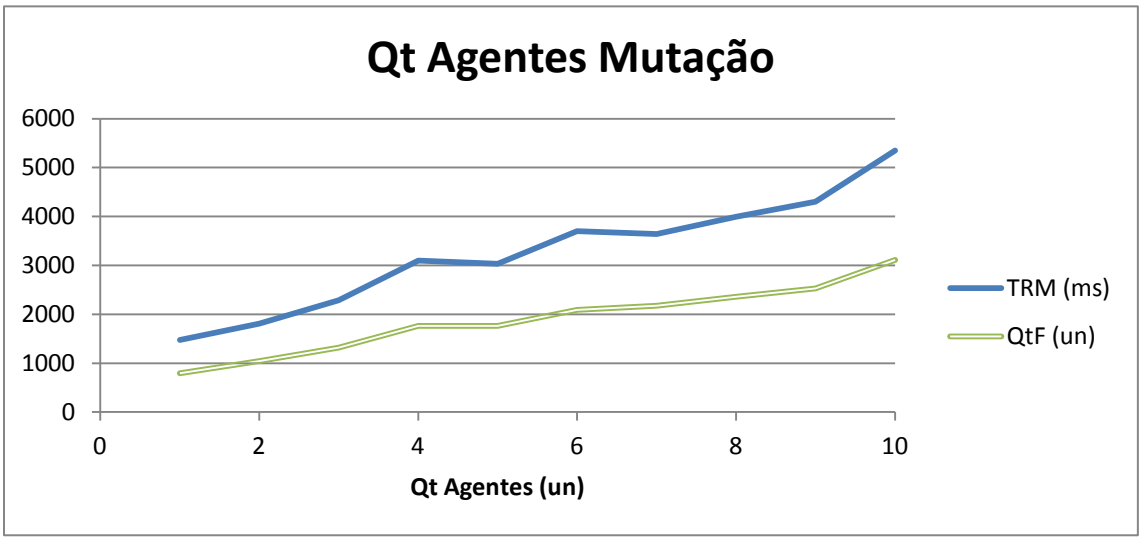

Figura 5-18 – TRM e QtF em função da quantidade de *threads* mutação.

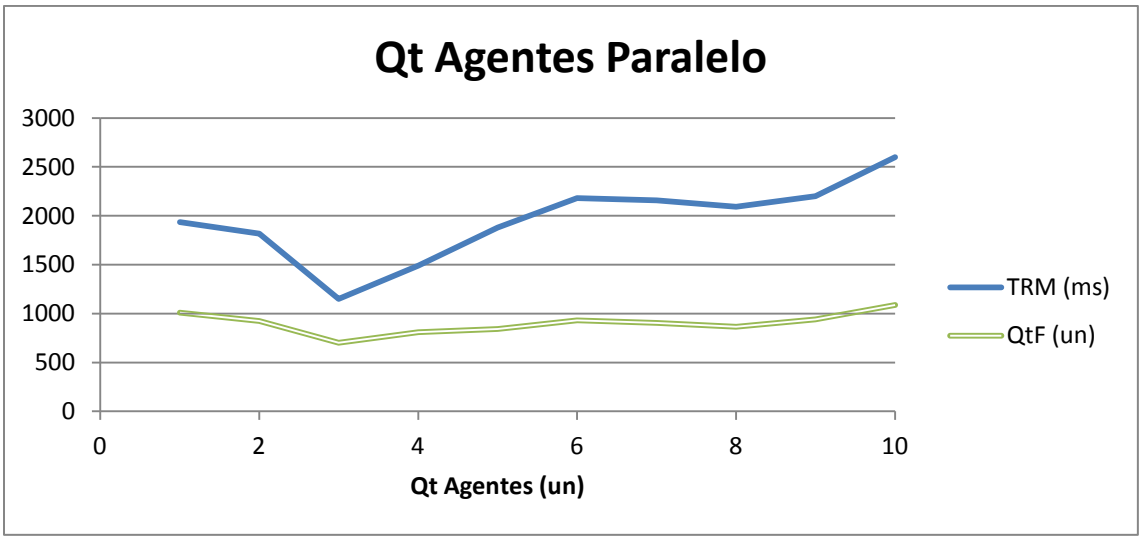

Figura 5-19 – TRM e QtF em função da quantidade de *threads* paralelo.

Com relação à quantidade de agentes de composição sequencial, o melhor desempenho ficou entre dois e sete agentes (Figura 5-20), com TRM abaixo de 1.300 ms. Conforme era esperado, devido à dependência entre este agente e a solução do problema, o TRM realmente deveria melhorar.

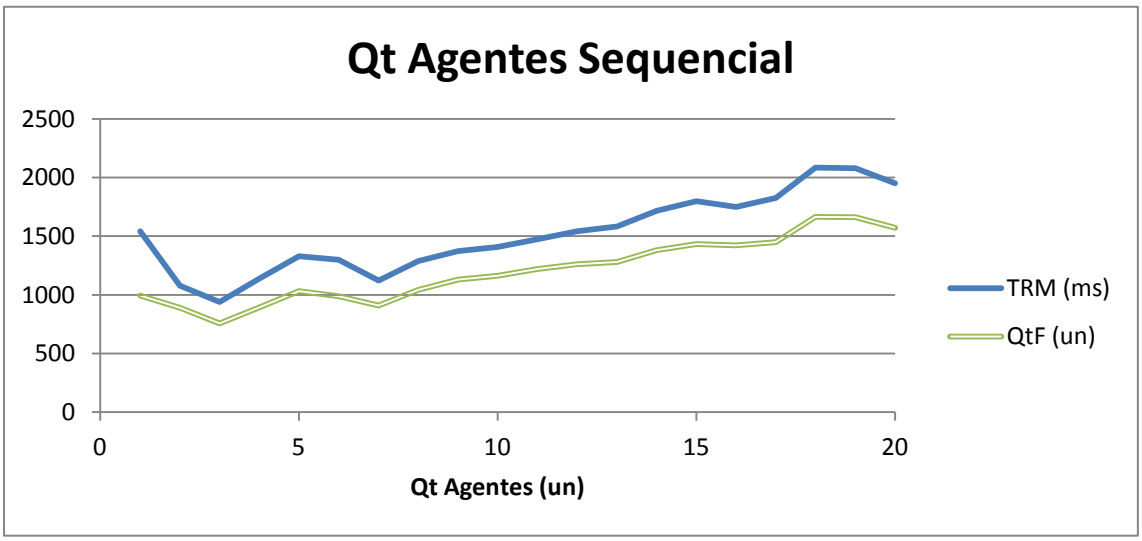

Figura 5-20 – TRM e QtF em função da quantidade de *threads* sequencial.

Tomados os resultados destes quatro últimos experimentos isoladamente, o sistema apresentou melhor desempenho quando configurado com um agente de

cruzamento, um agente de mutação, três agentes de composição em paralelo e três agentes de composição sequencial. Estes dados serão utilizados no próximo experimento.

5.16 A-Teams x AG Clássico

Nesta seção, o objetivo é comparar o TRM do algoritmo A-Teams com o do algoritmo genético clássico (implementado como uma única *thread*). Como o TRM de ambos os algoritmos pode depender de parâmetros de configuração, foram realizados experimentos preliminares para se obter quais valores estes parâmetros devem assumir para que se obtenha o melhor TRM possível de cada algoritmo. Assim sendo, os algoritmos serão comparados em seus melhores desempenhos.

A diferença essencial entre os dois algoritmos se reduz a:

- Pelo lado do A-Teams: quantidade de agentes instanciados e taxa de dormência de suas *threads*;
- Pelo lado do AG clássico: nas taxas de cruzamento, mutação, paralelo e sequencial de seus operadores.

Para os A-Teams serão utilizadas os valores determinados nas seções 5.6, 5.7, 5.8, 5.9 e 5.15, isto é, taxa de cruzamento: 50 ms; taxa de mutação: 30 ms; taxa de paralelo: 0 ms; taxa de sequencial: 0 ms; quantidade de agentes de cruzamento: um; quantidade de agentes de mutação: 1; quantidade de agentes de composição paralela: três; quantidade de agentes de composição sequencial: três.

Para o AG clássico é necessário fazer experimentos adicionais para determinar o valor das taxas de cruzamento, mutação, paralelo e sequencial. Estas taxas são mapeadas na quantidade de indivíduos gerados pelos respectivos métodos. Os experimentos foram feitos variando-se a quantidade de indivíduos gerados de 1 a 10, para cada taxa isoladamente. Os resultados estão apresentados na Figura 5-21, Figura 5-22, Figura 5-23 e Figura 5-24. Para o método paralelo, o melhor TRM foi obtido quando a quantidade de indivíduos gerados foi igual a um (Figura 5-21) e para o método sequencial, três (Figura

5-22). Isto se explica porque a solução do problema apresenta duas composições sequenciais e uma paralela. Portanto, uma taxa de composição sequencial maior aumenta as chances de se encontrar a solução. Disto também se pode concluir que a configuração ideal de um AG é fortemente dependente das características do problema.

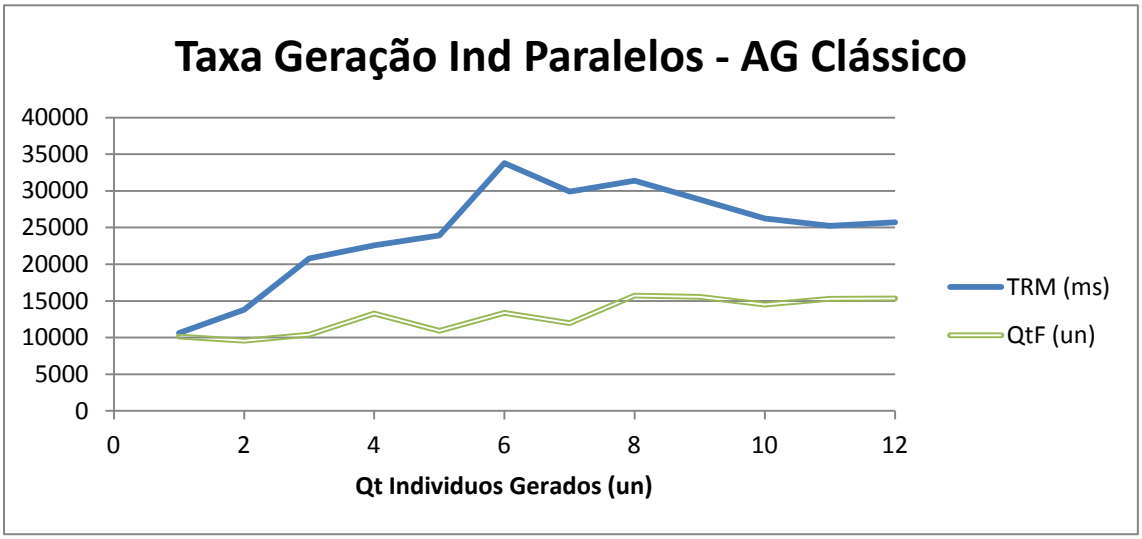

Figura 5-21 – TRM do AG clássico em função da quantidade de indivíduos gerados pela composição em paralelo utilizando o algoritmo genético clássico.

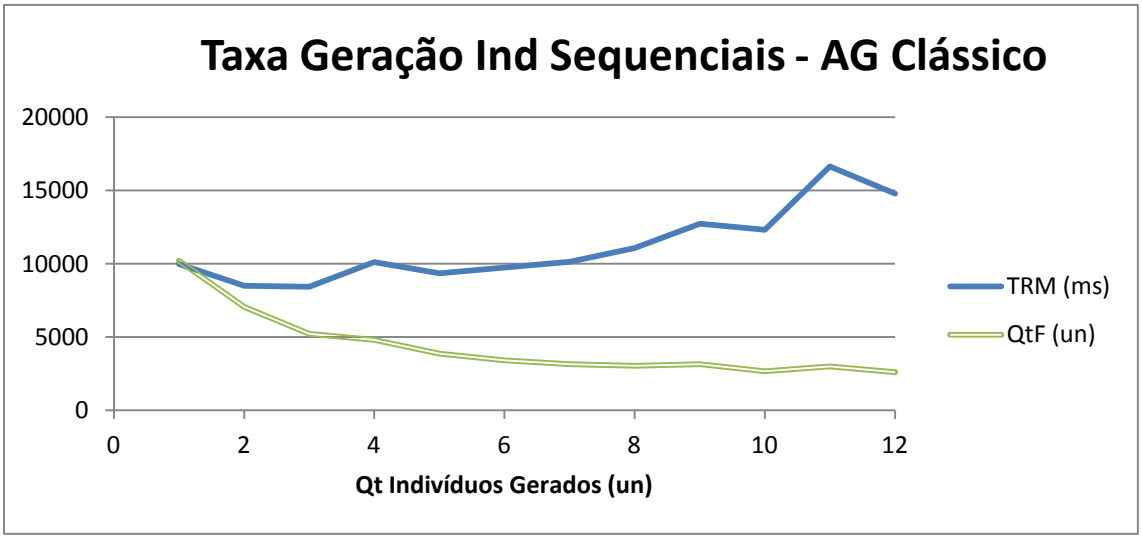

Figura 5-22 – TRM do AG clássico em função da quantidade de indivíduos gerados pela composição sequencial utilizando o algoritmo genético clássico.

Com relação ao cruzamento e à mutação, o menor TRM foi obtido com apenas um único indivíduo, Figura 5-23 e Figura 5-24, respectivamente.

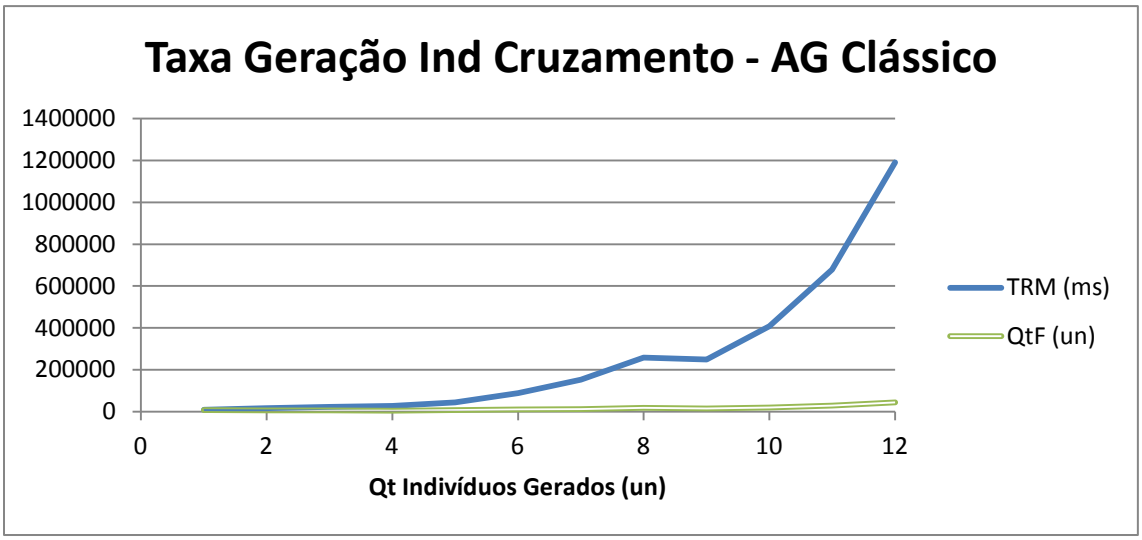

Figura 5-23 – TRM do AG clássico em função da quantidade de indivíduos gerados pelo cruzamento utilizando o algoritmo genético clássico.

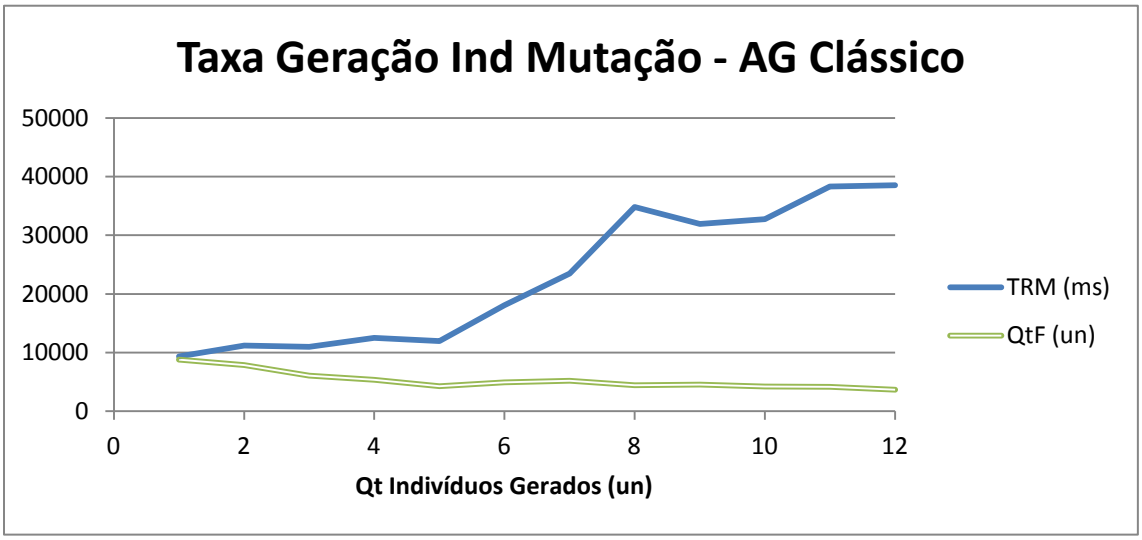

Figura 5-24 – TRM do AG clássico em função da quantidade de indivíduos gerados pela mutação utilizando o algoritmo genético clássico.

A Tabela 5-4 resume a configuração de melhor desempenho, em termos de TRM, para ambos os algoritmos. Para as demais variáveis serão mantidas os valores de referência.

Tabela 5-4 – Configuração de melhor desempenho para os algoritmos A-Teams e AG Clássico.

A-Teams	AG Clássico
Quantidade de *threads* de cruzamento = 1; Quantidade de *threads* de mutação = 1; Quantidade de *threads* de paralelo = 3; Quantidade de *threads* de sequencial = 3; Tempo de dormência da *thread* cruzamento = 50 ms; Tempo de dormência da *thread* mutação = 30 ms; Tempo de dormência da *thread* paralelo = 0 ms; Tempo de dormência da *thread* sequencial = 0 ms.	Quantidade de indivíduos de cruzamento = 1; Quantidade de indivíduos de mutação = 1; Quantidade de indivíduos de paralelo = 1; Quantidade de indivíduos de sequencial = 3;

Os resultados podem ser analisados a partir do gráfico da Figura 5-25 e da Tabela 5-5. Os A-Teams possuem TRM igual a 808 ms e realiza 564 cálculos de *fitness*; o AG clássico possui TRM igual a 3.958 ms e realiza 1.484 cálculos de *fitness*. Levando-se em consideração o TRM, o A-Team é 4,90 vezes mais rápido do que o AG Clássico, ou 79,58%. Analisando a quantidade de vezes que o *fitness* é calculado em relação ao TRM dos algoritmos, tem-se que:

- A-Teams: $\frac{564}{808} = 0{,}70$ cálculos de *fitness* por ms;
- AG clássico: $\frac{1484}{3958} = 0{,}37$ cálculos de *fitness* por ms.

Portanto, com relação ao cálculo do *fitness*, os A-Teams são praticamente duas vezes mais rápidos.

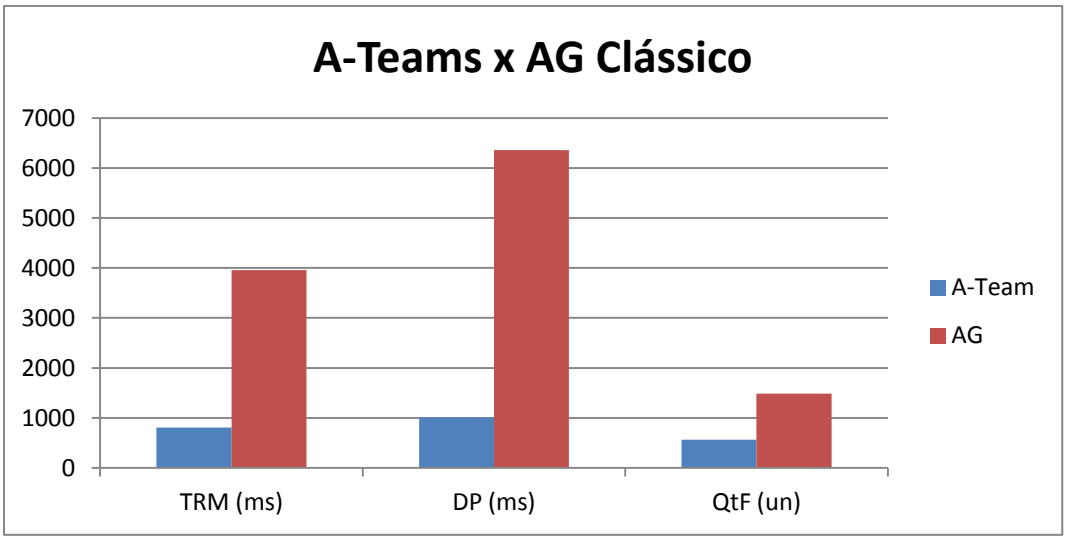

Figura 5-25 – TRM, DP e QtF dos A-Teams em contraposição ao AG clássico.

Validação de Hipótese

Para validar a hipótese de que os A-Teams são mais rápidos do que o AG clássico será utilizado o método de intervalo de confiança de um único lado [140, p. 213]. O TRM e o DP de ambos os métodos estão listados na Tabela 5-5.

Tabela 5-5 – TRM dos A-Teams versus TRM do AG clássico.

Algoritmo	Quantidade de Amostras	TRM (ms)	DP (ms)	QtF (un)
A-Teams	200	808	1.006	564
AG Clássico	200	3.958	6.359	1484

A diferença entre as médias é -3.135. O DP desta diferença é 455,24. O número efetivo dos graus de liberdade é 209,05. Utilizando a distribuição Weibull, o valor de Z para 95% de intervalo de confiança é $z_{0,95} = 2{,}0781$, que resulta em um intervalo de confiança entre (-3.150, -2.203,96). Como o intervalo não inclui o zero, pode-se efetivamente concluir que o Esecsy utilizando A-Teams é mais rápido do que o Esecsy utilizando AG clássico.

5.17 Considerações Finais

Este capítulo apresentou os resultados referentes à avaliação de desempenho do Esecsy. Vários experimentos foram realizados. Alguns com objetivo de validar os graus de casamento e as regras composição apresentadas no Capítulo 3. Outros com o objetivo de determinar a configuração para que o sistema apresentasse o menor tempo resposta. E, por último, alguns experimentos compararam o TRM do Esecsy em relação aos AG clássicos.

O critério de parada estabelecido para todos os experimentos foi *fitness* desejado igual a um porque se sabia *a priori* que havia uma solução. Entretanto, em cenários reais isto é pouco provável. Desta forma, outro critério de parada deve ser utilizado: *timeout* ou valor de *fitness* menor que um. Mesmo assim, em ambos os casos, pode ser que o sistema encontre uma solução com *fitness* = 1. Caso contrário, o sistema apresentará a melhor solução encontrada acompanhada de informações relevantes sobre sua composição. Cada serviço que toma parte da composição virá acompanhado do seu *fitness* interno e externo. A análise destes valores permitirá detectar qual destes serviços possui o menor *fitness* e, consequentemente, qual serviço deveria ser implementado para que a composição final tivesse um *fitness* maior. Suponha dois exemplos. No primeiro, toma-se o serviço abstrato representado na Figura 5-3 e supõe-se que após t ms, sendo t o valor do *timeout* estabelecido como critério de parada, o Esecsy apresente como solução o serviço denominado S_{100}, tal que, $S_{100} = \big((S_0 \prec S_1) \prec S_4\big)$. Os parâmetros de entrada, saída e valores parciais do *fitness* deste serviço estão representados na Figura 5-26. Este serviço possui *fitness* final de 0,75. De onde vem este valor? Basta analisar os valores parciais do seu *fitness*. O *fitness* de entrada é máximo, 1. Isto significa que todos os seus parâmetros de entrada se casam os parâmetros de entrada do serviço abstrato. O seu *fitness* de saída é 0,5, significando que somente metade dos seus parâmetros de saída se casa com os parâmetros de saída do serviço abstrato (da pontuação máxima possível, 6, o acumulado, ou conseguido, é 3). Sendo o seu *fitness* interno um, todas as suas composições internas são exatas. Disto, pode-se concluir que o motivo do *fitness* ser

menor que um deve-se exclusivamente aos seus parâmetros de saída. Como este serviço é uma composição sequencial entre os serviços S_0, S_1 e S_4, também se pode concluir que o serviço $S4$ não oferece todos os parâmetros de saída desejados.

```
Serviço: S100   - Fitness:0.75
Parâmetros de Entrada
  Nome: http://127.0.0.1:8099/services/1.1/university_researcher_service.owls#_UNIVERSITY
  Tipo: http://127.0.0.1:8099/ontology/portal.owl#University
Parâmetros de Saída
  Nome: http://127.0.0.1:8099/services/1.1/university_researcher_service.owls#_GRADE
  Tipo: http://127.0.0.1:8099/ontology/books.owl#Grade
Fitness de Entrada - Max:3.0 - Acumulado:3.0 - Resultado:1.0
Fitness de Saída   - Max:6.0 - Acumulado:3.0 - Resultado:0.5
Fitness Composição - Interno:1.0
```

Figura 5-26 – Um fragmento do arquivo de log que permite a análise dos valores parciais do *fitness* de um serviço composto, S_{100}.

No segundo exemplo, supõe-se que a solução apresentada é a composição $S_{200} = \big((S_0 \prec S_1) \prec (S_4 \parallel S_5)\big)$. Os parâmetros de entrada e saída do serviço S_5 estão disponíveis na Figura 5-27 (serviço criado exclusivamente para este exemplo). Na ontologia http://127.0.0.1:8099/ontology/books.owl o tipo de dado $Comic$ é subclasse do tipo de dado $Grade$.

```
Serviço: S5
Parâmetros de Entrada
  Nome: http://127.0.0.1:8099/ontology/researcher_book_service#_BOOK
  Tipo: http://127.0.0.1:8099/ontology/books.owl#Book
Parâmetros de Saída
  Nome: http://127.0.0.1:8099/ontology/researcher_book_service#_COMIC
  Tipo: http://127.0.0.1:8099/ontology/books.owl#Comic
```

Figura 5-27 – Parâmetros de entrada e saída do serviço _05_esecsy_book_A (serviço S_5).

Diferentemente do caso anterior, aqui não está faltando nenhum parâmetro, mas um dos parâmetros de saída oferece casamento de segundo grau, resultando em um *fitness* igual 0,92.

```
Serviço: S200   - Fitness:0.92
Parâmetros de Entrada
  Nome: http://127.0.0.1:8099/services/1.1/university_researcher_service.owls#_UNIVERSITY
  Tipo: http://127.0.0.1:8099/ontology/portal.owl#University
Parâmetros de Saída
  Nome: http://127.0.0.1:8099/services/1.1/university_researcher_service.owls#_GRADE
  Tipo: http://127.0.0.1:8099/ontology/books.owl#Grade
  Nome: http://127.0.0.1:8099/ontology/researcher_book_service#_COMIC
  Tipo: http://127.0.0.1:8099/ontology/books.owl#Comic
Fitness de Entrada - Max:3.0 - Acumulado:3.0 - Resultado:1.0
Fitness de Saída   - Max:6.0 - Acumulado:5.0 - Resultado:0.83
Fitness Composição - Interno:1.0
```

Figura 5-28 – Um fragmento do arquivo de log que permite a análise dos valores parciais do *fitness* de um serviço composto, S_{200}.

De posse destas informações, desenvolvedores de programas conseguem: (a) aumentar a reutilização de código; e (b) determinar exatamente o que está faltando.

Capítulo 6

Conclusões, Contribuições e Trabalhos Futuros

Sim, com Deus! Liberdade, vida nova, ressurreição dentre os mortos... que inefável momento!

Dostoiévski, Fiódor. Recordações das casas dos mortos. Edição Saraiva, São Paulo, 1949, p 260.

Neste trabalho foi desenvolvido um sistema, chamado de Esecsy, que realiza a composição automática de serviços Web semânticos. O Esecsy utiliza algoritmo de times assíncronos com operadores genéticos que realizam a composição de acordo com os tipos de fluxo de controle sequencial, paralelo e sincronização. A composição é descrita através de regras lógicas semânticas que levam em consideração os parâmetros de entrada e saída dos serviços Web semânticos obtidos a partir de documentos OWL-S.

Este último capítulo contempla uma reflexão sobre as contribuições, as conclusões e possíveis linhas de continuidade da pesquisa aqui desenvolvida.

6.1 Contribuições

As principais contribuições deste trabalho são: (a) definição dos graus de casamento e das regras lógicas de composição de serviços Web semânticos (ver Capítulo 3); (b) definição de um algoritmo de composição de serviços Web semânticos baseado nos graus de casamento e nas regras lógicas de composição (ver Capítulo 4); e, (c) estabelecimento de um *benchmark* para o tempo de resposta de um sistema de composição de serviços Web semânticos utilizando um subconjunto das regras (ver Capítulo 5). A seguir, cada uma das contribuições é comentada.

6.1.1 Graus de Casamento e Regras Lógicas

Os graus de casamento indicam a similaridade entre um serviço concreto e uma operação abstrata. Em última análise, eles sinalizam a qualidade com que uma operação concreta pode ser interpretada como uma implementação de uma operação abstrata. Foram definidos cinco graus de casamento. Os três primeiros graus são baseados apenas em regras lógicas, e os dois últimos são regras híbridas que também levam em consideração a similaridade sintática (ver seções 3.2 e 3.3).

A partir dos graus de casamento foi possível encontrar as regras de composição que contemplam os cinco tipos básicos de fluxo de controle: sequencial, separação paralela, sincronização, escolha exclusiva e união simples; e um padrão estrutural: fluxo de controle em laço. As regras de casamento são utilizadas para verificar se dois serviços podem ser compostos e, sendo possível a composição, estabelecer o modo de realizá-la (ver seção 3.4).

A utilização do conceito de operação polivalente (ver definição 3-22 da seção 3.3.1) e *workflow* de operações abstratas são duas características importantes destas regras. A definição um *workflow* de operações abstratas oferece, ao desenvolvedor, várias formas

diferentes para definir um serviço abstrato. E, indiretamente, de controlar a qualidade do casamento, uma vez que o nível de detalhamento do *workflow* e os serviços concretos existentes influenciam no grau do casamento realizado. Também é possível controlar a qualidade dos casamentos manipulando o uso dos diversos graus de similaridades. A possibilidade de utilizar serviços compostos para realizar novas composições permite a criação de vários tipos diferentes de fluxo de controle e de dados (ver seção 3.3).

O conceito de operação polivalente aumentou a possibilidade de casamento ao permitir que parâmetros previamente disponíveis no *workflow* fossem utilizados durante o processo de casamento, conforme ilustrado na seção 3.5.1.

A implementação de um algoritmo eficiente de composição automática de serviços Web semânticos só é possível a partir da definição das regras de casamento e de composição.

6.1.2 Definição e Implementação do Esecsy

O Esecsy é um sistema computacional que foi desenvolvido primordialmente para realizar a composição automática de serviços Web semânticos utilizando algoritmos de A-Teams (ver Capítulo 4). Adicionalmente, o Esecsy implementa outros algoritmos com o objetivo de comparar o desempenho entre eles. As principais características do Esecsy são:

- Realiza busca com algoritmo de time assíncrono e operadores genéticos;
- Realiza busca com algoritmos genéticos clássicos;
- Realiza busca cega;
- Implementa a composição sequencial e paralela;
- Explora a semântica dos parâmetros *hasInput* e *hasOutput* dos arquivos OWL-S.

Dentro deste sistema convém destacar a função de *fitness* e os conceitos de serviços jusante, montante e fronteira. A função de *fitness* é capaz de refletir tanto a qualidade da composição interna (casamento interno) quanto o grau de similaridade entre o serviço abstrato e o concreto (casamento externo) (ver seção 4.6).

Os serviços jusante e montante são utilizados na composição do tipo sequencial (ver seção 4.7.2). Jusante é um serviço fictício que possui como parâmetros de entrada os parâmetros de saída de um serviço S qualquer e, como parâmetros de saída, os parâmetros de saída do serviço abstrato, ou seja, conceitualmente, estabelece uma ponte entre o serviço S e o serviço abstrato. O serviço jusante é, portanto, o serviço que sê composto com o serviço S fará com que este último tenha os mesmos parâmetros de saída do serviço abstrato. Neste sentido é como se a busca fosse direcionada a partir dos parâmetros de entrada para os parâmetros de saída.

O serviço montante realiza a mesma função conceitual do serviço jusante, porém em direção contrária, ou seja, a partir dos parâmetros de saída em direção aos parâmetros de entrada.

O serviço fronteira é utilizado na composição paralela para direcionar a busca (ver seção 4.7.3). Ele é formado pelos parâmetros de entrada e de saída que estão presentes no serviço abstrato mas não no serviço concreto.

6.1.3 Benchmark

Não existe, até onde se saiba, um *benchmark* para o TRM de um sistema de composição de serviços Web semânticos. As pesquisas são normalmente direcionadas para uma abordagem mais qualitativa do que quantitativa. São poucos os trabalhos que apresentam análise de desempenho em função do tempo de resposta do sistema. As pesquisas realizadas na área de composição de serviços Web têm como objetivo projetar algoritmos que “simplesmente” resolvam o problema da composição, avaliando, posteriormente, a qualidade da solução apresentada. Neste âmbito, o trabalho aqui desenvolvido avança o estado da arte ao apresentar resultados que permitem a comparação quantitativa (ver Capítulo 5).

6.2 Conclusões

Nos experimentos realizados, os algoritmos de times assíncronos com operadores genéticos mostraram ser eficientes para lidar com o problema da composição automática de serviços Web semânticos:

- Para a composição sequencial envolvendo apenas dois serviços, a função de *fitness* definida pelo Esecsy provocou uma redução de 46,57 vezes no tempo de resposta do sistema em relação à seleção cega. Para uma composição um pouco mais complexa, esta redução chegou a 852,63 vezes (ver seção 5.5);
- Comparado a um AG Clássico, os A-Teams são 4,90 vezes mais rápidos para uma composição envolvendo quatro serviços (ver seção 5.16).

O desempenho do Esecsy, tanto em relação ao TRM, quanto em relação às composições realizadas (ver seção 5.3.2), comprovam a eficácia de suas regras e de seu algoritmo para a resolução de um problema que se acredita ser NP-Completo (ver seção 3.6). A heurística utilizada, representada pela função de *fitness* (Equação 4.1), apesar de não garantir a melhor solução, melhora o tempo de resposta do sistema.

Adicionalmente, pode-se ressaltar a capacidade de determinar o serviço faltante para que uma composição seja realizada com sucesso, como é descrito na seção 5.17.

6.3 Trabalhos Futuros

São várias as sugestões de continuidade deste trabalho. A primeira delas refere-se à base de serviços utilizada para os experimentos. Algumas modificações tiveram que ser feitas na base OWLS-TC3 para adaptá-la ao problema da composição de serviços. Como as adaptações realizadas foram as mínimas necessárias para validar os objetivos deste trabalho, a OWLS-TC3 ainda apresenta limitações quanto ao uso abrangente em composição de serviços. Por isto, uma base de dados pública, robusta e totalmente adaptada ao problema da composição é fundamental não só para o desenvolvimento de novos produtos, mas também para que os algoritmos possam ser comparados entre si.

Atualmente está disponível a versão 4 desta base. A análise desta versão poderá revelar se ela é ou não adequada ao problema da composição. Caso não o seja, seria conveniente a criação de uma nova base.

As regras de composição propostas são fundamentadas em cinco graus de casamento. Os três primeiros graus são baseados apenas em regras lógicas, e os dois últimos são regras híbridas que também levam em consideração a similaridade sintática. O Esecsy implementa somente as regras lógicas. E, das quatro regras de composição formuladas, a saber, sequencial, paralela-sincronização, escolha exclusiva e laço, somente as duas primeiras foram implementadas, em consequência, a continuação natural deste trabalho passa pela implementação dos dois graus de casamentos híbridos e das outras duas regras.

Os valores atribuídos às similaridades dos tipos exata, ascendência e descendência, três, dois e um, respectivamente, foram inspirados nos valores adotados pelo Semantic Web Research Group para a representação numérica os seus graus de similaridade [137]. Embora, inicialmente, estes valores tenham sido escolhidos sem outros critérios, eles se mostraram razoáveis diante dos resultados obtidos nos experimentos. Entretanto, outros valores devem ser testados. Talvez valores mais apropriados pudessem ser determinados com o usa de técnicas de lógica fuzzy [141].

A hipótese levantada na seção 5.12 de que os tipos de nascimento efêmero e similaridade semântica possuem TRM melhor em ambientes dinâmicos deverá ser testada.

Heurísticas mais adaptadas ao problema devem ser analisadas ou desenvolvidas para melhorar o desempenho da mutação e do cruzamento. A baixa interferência das *threads* de mutação e de cruzamento no desempenho do sistema pode encontrar explicação extra no fato dos genes, de mutação e de cruzamento, serem aleatoriamente escolhidos.

Fica, ainda, a sugestão de se utilizar Sistemas Imunológicos Artificiais (SIA) e computação de DNA para a solução da composição automática de serviços Web e posterior comparação com o desempenho do Esecsy. Os SIA surgiram a partir de tentativas de modelar e aplicar princípios imunológicos no desenvolvimento de novas ferramentas computacionais para reconhecimento de padrões, detecção de falhas e anomalias, segurança computacional, otimização, controle, robótica, escalonamento, análise de dados e aprendizagem de máquina, dentre outras [142] [143]. A computação de DNA utiliza técnicas da biologia molecular para simular problemas matemáticos e resolvê-los. A ideia faz uso da propriedade da molécula de DNA de codificar e guardar informações. Estas informações podem ser manipuladas de forma a resolver problemas matemáticos [144]. A técnica já foi utilizada para resolver o problema do caixeiro viajante.

O Esecsy foi implementado em Java com o uso de *threads*, portanto, o paralelismo depende do hardware e do sistema operacional utilizados. Em casos extremos, como por exemplo, processadores com somente um núcleo, nenhuma *thread* é realmente executada em paralelo. Transformar e posteriormente avaliar o desempenho do Esecsy como um sistema realmente paralelo seria interessante.

Finalmente, novos cenários de testes poderiam ser elaborados. Por exemplo, de todas as variáveis testadas, qual é aquela que mais influencia nos resultados? Ou, melhor ainda, seria descobrir o grau de influência de cada uma delas.

Abre-te Sésamo, quero sair!

Referências Bibliográficas

1. FRANKEL, D. S. **Model Driven Architecture:** Applying MDA to Enterprise Computing. [S.l.]: Wiley, 2003. 352 p. ISBN 0471319201.

2. THE OPEN GROUP. DCE Home Page. **DCE -- OpenDCE -- Portal**, 2005. Disponivel em: <http://www.opengroup.org/dce/>. Acesso em: 05 Apr 2011.

3. MICROSOFT. Distributed Component Object Model (DCOM) Remote Protocol Specification. **Microsoft Developer Network**, 2011. Disponivel em: <http://msdn.microsoft.com/library/cc201989.aspx>. Acesso em: 05 Apr 2011.

4. OBJECT MANAGEMENT GROUP, INC. (OMG). Welcome To The OMG's CORBA Website. **CORBA**, 2011. Disponivel em: <http://www.corba.org/>. Acesso em: 05 Apr 2010.

5. ORACLE. Remote Method Invocation Home. **Oracle Technology Network**, 2011. Disponivel em: <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html>. Acesso em: 05 Apr 2011.

6. MICROSOFT..NET Framework Developer Center. **Microsoft Developer Network**, 2011. Disponivel em: <http://msdn.microsoft.com/pt-br/netframework/>. Acesso em: 05 Apr 2011.

7. WHITE, J. E. **High-level framework for network-based resource sharing**. Proceedings of the June 7-10, 1976, National Computer Conference and Exposition. New York, NY, USA: ACM. 1976. p. 561–570. URL: http://doi.acm.org/10.1145/1499799.1499878.

8. WALSH, N. A Technical Introduction to XML. **XML.com**, October 03, 1998. Disponivel em:

<http://www.xml.com/pub/a/98/10/guide0.html>. Acesso em: 05 Apr 2011.

9. W3C. Web Services Activity. **World Wide Web Consortium (W3C)**, 2002. Disponivel em: <http://www.w3.org/2002/ws/>. Acesso em: 16 October 2008.

10. KRAFZIG, D.; BANKE, K.; SLAMA, D. **Enterprise SOA:** Service-Oriented Architecture Best Practices. [S.l.]: Prentice Hall PTR, 2004. 408 p. ISBN ISBN-10: 0131465759.

11. STUDER, R.; GRIMM, S.; ABECKER, A. **Semantic Web Services:** Concepts, Technologies, and Applications. 1 edition. ed. [S.l.]: Springer, June 11, 2007.

12. ANTONIOU, G.; VAN HARMELEN, F. **A Semantic Web Primer**. 2. ed. London: MIT Press, 2008. 287 p. ISBN ISBN 978-0-262-01242-3.

13. FENSEL, D.; BUSSLER, C.; MAEDCHE, A. Semantic Web Enabled Web Services. **Lecture Notes in Computer Science**, Berlin / Heidelberg, 2342/2002, 2002. 1-2. http://www.springerlink.com/content/vwxtdyremg1p1w42/.

14. MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs**. 3rd edition. ed. [S.l.]: Springer, 1996.

15. TALUKDAR, S. et al. Asynchronous Teams: Cooperation Schemes for Autonomous Agents. **Journal of Heuristics**, v. 4, p. 295-321, 1998. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=42833637E8C67472DE174C1F94F02EF0?doi=10.1.1.119.8685&rep=rep1&type=pdf.

16. WHYTE, R. Manage and distribute service reference and metadata in an available and distributed SOA. **developerWorks**, February 2005. Disponivel em: <http://www.ibm.com/developerworks/library/ws-largescale-soa/>. Acesso em: 18 March 2011.

17. ROSEN, M. Where Does One End and the Other Begin. **Business Process and Trendes**, January 2006. Disponivel em: <http://www.bptrends.com/publicationfiles/01-06%20COL%20SOA%20-Where%20Does%20One%20End%20-%20Rosen.pdf>. Acesso em: 18 March 2011.

18 IBM. Service Oriented Architecture — SOA. **IBM**. Disponivel em: <http://www-
. 01.ibm.com/software/solutions/soa/>. Acesso em: 28 March 2011.

19 LUBLINSKY, B. Defining SOA as an architectural style. **developerWorks**, 09 January 2007.
. Disponivel em: <http://www.ibm.com/developerworks/library/ar-soastyle/>. Acesso em: 18 March 2011.

20 SPROTT, D.; WILKES, L. Understanding Service-Oriented Architecture. **The Architecture Journal**,
. January 2004. Disponivel em: <http://msdn.microsoft.com/en-us/library/aa480021.aspx>. Acesso em: 18 March 2011.

21 KNORR, E.; RIST, O. 10 steps to SOA. **InfoWord**, 7 November 2005. Disponivel em:
. <http://www.infoworld.com/article/05/11/07/45FEsoaintro_1.html>. Acesso em: 18 March 2011.

22 TURNER, M.; BUDGEN, D.; BRERETON, P. Turning Software into a Service. **Computer**, 36, n. 10,
. October 2003. 38-44.

23 ORT, E. Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools.
. **Sun Developer Network (SDN)**, April 2005. Disponivel em: <http://java.sun.com/developer/technicalArticles/WebServices/soa2/SOATerms.html#soawhy>. Acesso em: 18 March 2011.

24 ALONSO, G. et al. **Web Services Concepts, Architectures and Applications**. 1. ed. [S.l.]:
. Springer, 2003. 354 p. ISBN ISBN 3-540-44008-9.

25 HAAS, H.; BROWN, A. Web Services Glossary. **W3C**, 2004. Disponivel em:
. <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/>. Acesso em: 03 mar. 2012.

26 SANDEEP, P.; MADANMOHAN, T. R. **Web Services: Frameworks, Architecture and Evolution -**
. **An Interpretative Analysis**. Indian Institute of Management Bangalore. [S.l.].

27 BOOTH, D. et al. Web Services Architecture. **World Wide Web Consortium (W3C)**, 11 February
. 2004. Disponivel em: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>. Acesso em:

07 July 2008.

28. MITRA, N.; LAFON, Y. SOAP Version 1.2 Part 0: Primer (Second Edition). **World Wide Web Consortium**, 27 April 2007. Disponivel em: <http://www.w3.org/TR/2007/REC-soap12-part0-20070427/>. Acesso em: 16 October 2008.

29. GUDGIN, M. et al. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). **World Wide Web Consortium**, 27 April 2007. Disponivel em: <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>. Acesso em: 16 October 2008.

30. GUDGIN, M. et al. SOAP Version 1.2 Part 2: Adjuncts (Second Edition). **World Wide Web Consortium**, 27 April 2007. Disponivel em: <http://www.w3.org/TR/2007/REC-soap12-part2-20070427/>. Acesso em: 16 October 2008.

31. BOOTH, D.; KEVIN LIU, C. World Wide Web Consortium. **Web Services Description Language (WSDL) Version 2.0 Part 0:** Primer, 26 June 2007. Disponivel em: <http://www.w3.org/TR/wsdl20-primer/>. Acesso em: 07 July 2008.

32. CHRISTENSEN, E. et al. Web Services Description Language (WSDL) 1.1. **World Wide Web Consortium (W3C)**, 15 March 2001. Disponivel em: <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>. Acesso em: 18 March 2011.

33. UDDI XML.ORG. Online community for the Universal Description, Discovery and Integration OASIS Standard. **OASIS**, 1993-2009. Disponivel em: <http://uddi.xml.org/>.

34. OASIS. OASIS Commitees by Category: Web Services. **Organization for the Advancement of Structured Information Standards**. Disponivel em: <http://www.oasis-open.org/committees/tc_cat.php?cat=ws>. Acesso em: 18 March 2011.

35. CHAMPION, M. et al. Web Services Architecture. **World Wide Web Consortium (W3C)**, 14 November 2002. Disponivel em: <http://www.w3.org/TR/2002/WD-ws-arch-20021114/>. Acesso em: 18 March 2011.

36. KLEIN, M.; KÖONIG-RIES, B. Combining Query and Preference – An Approach to Fully Automatize Dynamic Service Binding. **Proceedings of the IEEE International Conference on**

. **Web Services (ICWS'04)**, 2004. 788.

37 KREGER, H. **Web Services Conceptual Architecture (WSCA 1.0)**. IBM. [S.l.], p. 41. 2001.
. http://whitepapers.silicon.com/0,39024759,60030666p,00.htm.

38 DUSTDAR, S.; SCHREINER, W. A survey on web services composition. **Int. J. Web and Grid**
. **Services**, 1, n. 1, 2005. 1-30. http://www.infosys.tuwien.ac.at/Staff/sd/papers/A%20survey%20on%20web%20services%20composition_Dustdar_Schreiner_inPress.pdf.

39 ALAMRI, A.; EID, M.; EL SADDIK, A. Classification of the state-of-the-art dynamic web services
. composition techniques. **International Journal of Web and Grid Services**, 2, n. 2, 2006. 148-166.

40 BERARDI, D. et al. Automatic Composition of E-services That Export Their Behavior. **Proc. 1st**
. **Int. Conf. on Service Oriented Computing (ICSOC)**, 2003. 43–58. http://www.dis.uniroma1.it/~mecella/publications/eService/BCDLM_ICSOC2003.pdf.

41 FLUEGGE, M. et al. Challenges and techniques on the road to dynamically compose web
. services. **ACM International Conference Proceeding Series**, 263, 2006. 40 - 47.

42 FININ, T. et al. Swoogle Semantic Web Search, 2004. Disponivel em:
. <http://swoogle.umbc.edu/>. Acesso em: 10 August 2008.

43 YANG, J.; P. PAPAZOGLOU, M.; HEUVEL, W.-J. Tackling the Challenges of Service Composition in
. E-Marketplaces. **Twelfth International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems**, 2002. 125-133. http://www.uvt.nl/infolab/pub/db/ride02.pdf.

44 GHANDEHARIZADEH, S. et al. Proteus: system for dynamically composing and intelligently
. executing Web Services. **Proceedings of the First International Conference on Web Services (ICWS)**, Las Vegas, NV, June 2003. http://dblab.usc.edu/Users/shkim/papers/proteus.pdf.

45 AMBITE, J. L. et al. Argos: Dynamic Composition of Web Services for Goods Movement Analysis and Planning. **In Proceedings of the 5th Annual National Conference on Digital Government**

. **Research**, 2004. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.5323&rep=rep1&type=pdf.

46 VANDERMEER, D. et al. FUSION: a system allowing dynamic web service composition and
. automatic execution. **IEEE International Conference on E-Commerce Technology (CEC'03)**, June 2003. 399-404. http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/8596/27235/01210276.pdf?tp=&isnumber=&arnumber=1210276.

47 ZHANG, R. **Ontology-driven web services composition techniques**. University of Georgia.
. Georgia. May, 2004. http://www.cs.uga.edu/~budak/ruoyan_thesis.pdf.

48 TIZZO, N. P.; ADÁN-COELLO, J. M.; CARDOZO, E. **Automatic Composition of Semantic Web**
. **Services Using A-Teams with Genetic Agents**. Proceedings of the 2011 IEEE Congress on Evolutionary Computation. New Orleans, USA: IEEE Press. 2011. p. 370–377.

49 MCILRAITH, S. A.; SON, T. C.; ZENG, H. Semantic Web Services. **Intelligent Systems, IEEE**, 16
. Issue:2, Mar-Apr 2001. 46 - 53.

50 MARTIN, D. et al. OWL-S: Semantic Markup for Web Services. **World Wide Web Consortium**
. **(W3C)**, 22 November 2004. Disponivel em: <http://www.w3.org/Submission/OWL-S/>. Acesso em: 1 September 2008.

51 DAML. Semantic Web Services Home Page. **The DARPA Agent Markup Language Homepage**,
. 2004. Disponivel em: <http://www.daml.org/services/>. Acesso em: 05 Apr 2011.

52 GRUBER, T. R. A translation approach to portable ontology specifications. **Knowledge**
. **Acquisition**, London, UK, 5, n. 2, 1993. 199-220.

53 BURKE, B.; MONSON-HAEFEL, R. **Enterprise JavaBeans 3.0**. [S.l.]: O'Reilly Media, Inc., 2006.
.

54 OBJECT MANAGEMENT GROUP, INC. (OMG). CORBA Component Model (CCM). **CORBA**, 2008.
. Disponivel em: <http://www.omg.org/spec/CORBA/3.1/>. Acesso em: 05 Apr 2011.

55 FUJII, K.; SUDA, T. Semantics-based dynamic service composition. **IEEE Journal on Selected** . **Areas in Communications**, 12, December 2005. 2361 - 2372. http://netresearch.ics.uci.edu/kfujii/dsc/projects.htm.

56 W3C. Semantic Annotations for WSDL Working Group. **World Wide Web Consortium**, 2007. . Disponivel em: <http://www.w3.org/2002/ws/sawsdl/>. Acesso em: 18 March 2011.

57 SONG, T.-X. et al. Web Services' Semantic Annotation and Auto-Matching Based on SAWSDL. . **Information Science and Engieering, 2008. ISISE '08. International Symposium on**, 2, December 2008. 577-580.

58 MAY, M. **Business process management:** integration in a web-enabled environment. London: . Financial Times Prentice Hall, 2003. ISBN ISBN: 0273661086 9780273661085.

59 WEISS, M.; ESFANDIARI, B.; LUO, Y. Towards a classification of web service feature interactions. . **Computer Networks**, 51, February 2007. 359-381. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.7236.

60 RAO, J. **Semantic Web Service Composition via Logic-based Program Synthesis**. Trondheim: . Norwegian University of Science and Technology, 2004. ISBN ISBN 82-471-6463-9.

61 RAO, J.; SU, X. A Survey of Automated Web Service Composition Methods. In: _____ **Lecture** . **Notes in Computer Science**. Berlin / Heidelberg: Springer, 2005. p. 43-54.

62 TSETSOS, V.; ANAGNOSTOPOULOS, C.; HADJIEFTHYMIADES, S. Semantic Web Service . Discovery: Methods, Algorithms and Tools. In: CARDOSO, J. **Semantic Web Services:** Theory, Tools and Applications. [S.l.]: IDEA Group Publishing, 2007. ISBN ISBN 978-1-59904-045-5 (hardcover), ISBN 978-1-59904-047-9.

63 KLUSCH, M.; FRIES, B.; SYCARA, K. Automated semantic web service discovery with OWLS-MX. . **Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems**, Hakodate, Japan, May 2006. 915-922.

64 BENATALLAH, B. et al. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. **Proceedings of the 18th International Conference on Data Engineering (ICDE)**,

. Washington, DC, USA, 2002. 297-308. http://portal.acm.org/citation.cfm?id=878987.

65 NARAYANAN, S.; A. MCILRAITH, S. Simulation, verification, automated composition of web . services. **Proceedings of the 11th International World Wide Web**, Honolulu, Hawaii, USA, May 2002. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.142.1846.

66 TIZZO, N. P.; ADÁN-COELLO, J. M.; CARDOZO, E. Improving Dynamic Web Service Selection in . WS-BPEL Using SPARQL. **Proceedings 2011 IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC 2011)**, Anchorage, AK, USA, 9-12 October 2011. 864 - 871. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6083760.

67 TIZZO, N. P.; ADÁN-COELLO, J. M.; CARDOZO, E. Dynamic Service Matching in WS-BPEL with . SPARQL. **International Transactions on Systems Science and Applications**, v. 7, n. 1, 2012. ISSN ISSN 1751-1461 (print).

68 PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL Query Language for RDF**, January 2008. . Disponivel em: <http://www.w3.org/TR/rdf-sparql-query/>. Acesso em: 14 April 2009.

69 KLYNE, G.; J. CARROLL, J. **Resource Description Framework (RDF):** Concepts and Abstract . Syntax, 10 February 2004. Disponivel em: <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>. Acesso em: 23 April 2009. Latest version: http://www.w3.org/TR/rdf-concepts/.

70 CASATI, F. et al. **eFlow: a Platform for Developing and Managing Composite e-Services**. HP . Laboratories. Palo Alto, CA, p. 9. 2000. (HPL-2000-36). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.3676.

71 CASATI, F. et al. Adaptive and Dynamic Service Composition in eFlow. **Lecture Notes in . Computer Science**, Berlin / Heidelberg, 1789/2000, 01 January 2000. 13-31. http://www.springerlink.com/content/56c5x7gq056c1a9d/.

72 CASATI, F.; SAYAL, M.; SHAN, M.-C. Developing E-Services for Composing E-Services. In: . HEIDELBERG, S. B. /. **Advanced Information Systems Engineering**. Lecture Notes in Computer Science. ed. [S.l.]: [s.n.], v. 2068, 2001. p. 171-186. URL http://dx.doi.org/10.1007/3-540-45341-5_12.

73. SCHUSTER, H. et al. Modeling and Composing Service-Based nd Reference Process-Based Multi-enterprise Processes. **Proceedings of the 12th International Conference on Advanced Information Systems Engineering**, London, UK, 2000. 247–263. URL http://portal.acm.org/citation.cfm?id=646088.680058.

74. MCILRAITH, S.; CAO SON, T. Adapting Golog for Composition of Semantic Web Services. **Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR2002)**, Toulouse, France, April 2002. 482-493. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.9235.

75. MCDERMOTT, D. Estimated-regression planning for interactions with Web services. **AAAI Press**, 2002. 204-211. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.3548.

76. MEDJAHED, B.; BOUGUETTAYA, A.; K. ELMAGARMID, A. Composing Web services on the Semantic Web. **The VLDB Journal — The International Journal on Very Large Data Bases**, 12, n. 4, November 2003. 333 - 351.

77. SIRIN, E. et al. HTN planning for Web Service composition using SHOP2. **Web Semantics: Science, Services and Agents on the World Wide Web**, 1, n. 4, October 2004. 377-396. http://dx.doi.org/10.1016/j.websem.2004.06.005.

78. TANG, M.; AI, L. A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition. **Proceeding of the 2010 World Congress on Computational Intelligence**, Centre de Convencions Internacional de Barcelona, Barcelona, 18-23 July 2010. http://eprints.qut.edu.au/33293/1/c33293.pdf.

79. WORKFLOW MANAGEMENT COALITION (WFMC). **Terminology and glossary**. Workflow Management Coalition. [S.l.]. 1999. (Technical Report Document Number WFMC-TC-1011, Issue 3.0). http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf.

80. WORKFLOW PATTERNS INITIATIVE. Patterns. **Workflow Patterns Initiative**, 2010. Disponivel em: <http://www.workflowpatterns.com/patterns/>. Acesso em: 18 Feb 2011.

81. A. DE JONG, K.; M. SPEARS, W. **Using Genetic Algorithms to Solve NP-Complete Problems**.

. Proceedings of the third international conference on Genetic algorithms. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 1989. p. 124-132. URL: http://dl.acm.org/citation.cfm?id=93126.93172.

82 STANFORD LOGIC GROUP. Challenge on Automating Web Services Mediation, Choreography
. and Discovery. **SWS Challenge**, 23 June 2008. Disponivel em: <http://sws-challenge.org/wiki/index.php/Main_Page>. Acesso em: 1 September 2008.

83 MILANOVIC, N.; MIROSLAW, M. Current solutions for Web service composition. **IEEE Internet**
. **Computing**, 8, n. 6, November 2004. 51 - 59.

84 RAJASEKARAN, P. et al. Enhancing Web Services Description and Discovery to Facilitate
. Composition. In: _____ **Semantic Web Services and Web Process Composition**. Berlin/Heidelberg: Springer-Verlag, v. 3387, 2005. p. 55–68. ISBN ISBN 978-3-540-24328-1. http://www.springerlink.com/content/kcwqtykxtu83ydal/.

85 O'SULLIVAN, J. **Towards a Precise, Understanding of Service Properties**. Queensland
. University of Technology. [S.l.], p. 237. 2006.

86 PATHAK, J.; BASU, S.; HONAVAR, V. Modeling Web Services by Iterative Reformulation of
. Functional and Non-Functional Requirements. **Lecture Notes in Computer Science**, Berlin / Heidelberg, 4294, 24 November 2006. http://www.springerlink.com/content/u230756033469533/.

87 W3C. **Web Service Semantics - WSDL-S**. Disponivel em:
. <http://www.w3.org/Submission/WSDL-S/>.

88 AKKIRAJU, R. et al. Web Service Semantics - WSDL-S. **W3C**, April 2005. Disponivel em:
. <http://www.w3.org/Submission/WSDL-S/>. Acesso em: 18 March 2011.

89 AKKIRAJU, R. et al. Web Service Semantics - WSLD-S. **W3C Workshop on Frameworks for**
. **Semantics in Web Services**, January 2006. http://domino.research.ibm.com/library/cyberdig.nsf/papers/EF9FE52551FB21DC8525710D005A8480/$File/rc23854.pdf.

90 STEVE, B. et al. World Wide Web Consortium (W3C). **Semantic Web Services Ontology (SWSO)**, 2005. Disponivel em: <http://www.w3.org/Submission/2005/SUBM-SWSF-SWSO-20050909/>. Acesso em: 2011 set. 06.

91 ESSI WSMO WORKING GROUP. Publications and Links. **Web Service Modeling Ontology**. Disponivel em: <http://www.wsmo.org/index.html>. Acesso em: 1 September 2008.

92 ROMAN, D. et al. Web Service Modeling Ontology. **Applied Ontology**, 1, n. 1, 2005. 77-106.

93 BATTLE, S. et al. Semantic Web Services Framework (SWSF) Overview. **World Wide Web Consortium**, 2005. Disponivel em: <http://www.daml.org/services/swsf/1.0/overview/>. Acesso em: 1 September 2008.

94 MOTTA, E. **Reusable components for knowledge modelling:** case studies in parametric design problem solving. Amsterdam, The Netherlands: IOS Press, 1999.

95 MOTTA, E. et al. IRS-II: A Framework and Infrastructure for Semantic Web Services. In: FENSEL, D.; SYCARA, K.; MYLOPOULOS, J. **The Semantic Web - ISWC 2003**. [S.l.]: Springer Berlin / Heidelberg, v. 2870, 2003. p. 306-318. URL: http://dx.doi.org/10.1007/978-3-540-39718-2_20.

96 CABRAL, L. et al. Approaches to Semantic Web Services: An Overview and Comparisons. **European Semantic Web Conference**, 2004. 225-239. http://kmi.open.ac.uk/projects/irs/cabralESWS04.pdf.

97 ALVES, A. et al. Web Services Business Process Execution Language Version 2.0 (WS-BPEL). **Organization for the Advancement of Structured Information Standards**, 11 April 2007. Disponivel em: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>. Acesso em: 1 September 2008.

98 BOLIE, J. et al. **BPEL Cookbook:** Best Practices for SOA-based integration and composite applications development. [S.l.]: Packt Publishing, 2006. 188 p. ISBN ISBN-10: 1904811337. http://www.oracle.com/technology/pub/articles/bpel_cookbook/carey.html.

99 OBJECT MANAGEMENT GROUP, INC. (OMG). OMG's MetaObject Facility. **Object Management**

. **Group**, 11 jan. 2008. Disponivel em: <http://www.omg.org/mof/>. Acesso em: 22 out. 2008.

10 OBJECT MANAGEMENT GROUP, INC. (OMG). OMG Model Driven Architecture. **Object**
0. **Management Group**, 17 abr. 2008. Disponivel em: <http://www.omg.org/mda/>. Acesso em: 21 out. 2008.

10 MOTIK, B.; F. PATEL-SCHNEIDER, P.; HORROCKS, I. OWL 1.1 Web Ontology Language Structural
1. Specification and Functional-Style Syntax. **World Wide Web Consortium (W3C)**, 2006. Disponivel em: <http://www.w3.org/Submission/2006/SUBM-owl11-owl_specification-20061219/>. Acesso em: 09 August 2011.

10 F. PATEL-SCHNEIDER, P.; HORROCKS, I. OWL Web Ontology Language Semantics and Abstract
2. Syntax Section 2. Abstract Syntax. **World Wide Web Consortium (W3C)**, 2004. Disponivel em: <http://www.w3.org/TR/owl-semantics/syntax.html>. Acesso em: 08 August 2011.

10 HORROCKS, I.; PATEL-SCHNEIDER, P. F.; HARMELEN, F. V. From SHIQ and RDF to OWL: The
3. making of a web ontology language. **Journal of Web Semantics**, v. 1, p. 7-23, 2003. http://www.websemanticsjournal.org/index.php/ps/article/download/24/22.

10 SYCARA, K. et al. Automated discovery, interaction and composition of Semantic Web services.
4. **Web Semantics: Science, Services and Agents on the World Wide Web**, 1, n. 1, December 2003. 27-46.

10 PAOLUCCI, M. et al. Semantic Matching of Web Services Capabilities. **Proceedings of the First**
5. **International Semantic Web Conference on The Semantic Web**, London, UK, 2002. 333-347. http://dl.acm.org/citation.cfm?id=646996.711287.

10 SILVA, L. M. D.; BRAGA, R.; CAMPOS, F. A semantic service based framework for workflow
6. composition in e-Science projects. **Information Sciences**, v. 186, n. 1, p. 186-208, March 2010. ISSN doi:10.1016/j.ins.2011.10.010.

10 KLUSCH, M.; FRIESB, B.; SYCARAC, K. OWLS-MX: A hybrid Semantic Web service matchmaker
7. for OWL-S services. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 7, n. 2, p. 121-133, April 2009.

http://www.sciencedirect.com/science/article/pii/S1570826808000838.

10 TRAN, B. D.; TAN, P. S.; GOH, A. **Composing OWL-S Web Services**. 21st International
8. Conference on Distributed Computing Systems. [S.l.]: IEEE Computer Society. 25-28 September 2007. p. 322-329.

10 OUNDHAKAR, S. et al. Discovery of Web Services in a Multi-Ontology and Federated Registry
9. Environment. **International Journal of Web Services Research**, 3, 2005.

11 GÖKER, A.; DAVIES, J. **Information Retrieval:** Searching in the 21st Century. [S.l.]: John Wiley
0. and Sons, 2009. 295 p.

11 DONG, X. et al. Similarity Search for Web Services. **VLDB '04 Proceedings of the Thirtieth**
1. **international conference on Very large data bases**, Toronto, Canada, 2004. 372-383. URL: http://dl.acm.org/citation.cfm?id=1316689.1316723.

11 COST, S.; SALZBERG, S. A Weighted Nearest Neighbor Algorithm for Learning with Symbolic
2. Features. **Journal Machine Learning**, Hingham, MA, USA, v. 10, n. 1, p. 57-78, January 1993. ISSN ISSN: 0885-6125. URL: http://dx.doi.org/10.1023/A:1022664626993.

11 LARKEY, L. S. Automatic essay grading using text categorization techniques. **Proceedings of the**
3. **21st annual international ACM SIGIR conference on Research and development in information retrieval**, Melbourne, Australia, p. 90-95, 1998. ISSN ISBN:1-58113-015-5. URL: http://doi.acm.org/10.1145/290941.290965.

11 A. MILLER, G. WordNet - A lexical database for the English language. **WordNet - A lexical**
4. **database for the English language**, 2006. Disponivel em: <http://wordnet.princeton.edu/>. Acesso em: 08 September 2008.

11 CHAPMAN, S. Sam's String Metrics. **Sam Chapman**, 2006. Disponivel em:
5. <http://www.dcs.shef.ac.uk/~sam/stringmetrics.html#jaro>. Acesso em: 08 September 2008.

11 TRASTOUR, D.; BARTOLINI, C.; GONZALEZ-CASTILLO, J. A Semantic Web Approach to Service
6. Description for Matchmaking of Services. **Proceedings of the International Semantic Web**

Working Symposium (SWWS), July 2001.

11 LUIS AMBITE, J.; KAPOOR, D. Automatic Generation of Data Processing Workflows for
7. Transportation Modeling. **Proceedings of the 8th annual international conference on Digital government research: bridging disciplines & domains**, Philadelphia, Pennsylvania, 228, n. Digital Government Society of North America, May 20-23 2007. 82 - 91.

11 MCCARTHY, J.; J. HAYES, P. Some philosophical problems from the standpoint of artificial
8. intelligence. In: _____ **Machine Intelligence**. [S.l.]: Edinburgh University Press, 1969. p. 463-502. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.85.5082.

11 J. LEVESQUE, H. et al. GOLOG: A Logic Programming Language for Dynamic Domains. **Journal of**
9. **Logic Programming**, v. 31, 1997. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.5032.

12 A. MCILRAITH, S.; CAO SON, T.; ZENG, H. Semantic Web Services. **IEEE Intelligent Systems**, v.
0. 16, p. 46-53, March/April 2001. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.61.

12 PEER, J. A PDDL Based Tool for Automatic Web Service Composition. **Lecture Notes in**
1. **Computer Science**, Principles and Practice of Semantic Web Reasoning, September 2004. 149-163.

12 SIMMONS, R.; VELOSO, M.; SMITH, S. **The Fourth International Conference on Artificial**
2. **Intelligence Planning Systems 1998 (AIPS '98)**, June 1998. Disponivel em: <http://www.cs.cmu.edu/~aips98/>.

12 CLAßEN, J.; HU, Y.; LAKEMEYER, G. A Situation-Calculus Semantics for an Expressive Fragment
3. of PDDL. **Proceedings of AAAI-07**, 2007.

12 GHALLAB, M. et al. **PDDL - The Planning Domain Definition Language - Version 1.2**. Yale
4. Center for Computational Vision and Control. [S.l.], p. 27. October, 1998. (CVC TR-98-003/DCS TR-1165). http://www.informatik.uni-ulm.de/ki/Edu/Vorlesungen/GdKI/WS0203/pddl.pdf.

12 NAU, D. et al. SHOP2: An HTN planning system. **Journal of Artificial Intelligence Research**, v.

5. 20, p. 379-404, 2003.

12 GIACOMO, G. D.; LESPÉRANCE, Y.; LEVESQUE, H. J. Congolog, a concurrent programming
6. language based on the situation calculus. **Artificial Intelligence**, 1-2, August 2000. 109-169. http://www.sciencedirect.com/science/article/pii/S000437020000031X.

12 PRAZERES, C. V. S. **Serviços Web Semânticos: da modelagem à composição**. Universidade de
7. São Paulo. São Carlos, p. 189. 2009. Tese de doutorado.

12 SHIN, D.-H.; LEEA, K.-H.; SUDAB, T. Automated generation of composite web services based on
8. functional semantics. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 7, n. 4, p. 332-343, December 2009. ISSN DOI: 10.1016/j.websem.2009.05.001.

12 WU, Z. et al. **Automatic Composition of Semantic Web Services using Process and Data**
9. **Mediation**. LSDIS lab, University of Georgia. [S.l.], p. 390-393. 2007. http://knoesis.wright.edu/library/publications/download/SWSChallenge-TR-METEOR-S-Feb2007.pdf.

13 HASHEMIAN, S. V.; MAVADDAT, F. A Graph-Based Approach to Web Services Composition.
0. **Proceedings of the The 2005 Symposium on Applications and the Internet**, 2005. 183-189. URL http://dl.acm.org/citation.cfm?id=1042446.1043662.

13 CHEN, K.; XU, J.; REIFF-MARGANIEC, S. Markov-HTN Planning Approach to Enhance Flexibility
1. of Automatic Web Services Composition. **IEEE International Conference on Web Services (ICWS)**, Los Angeles, CA, USA, 6-10 July 2009. 9-16. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=5175786&arnumber=5175801&count=143&index=1.

13 KLUSCH, M. et al. **OWLS-TC3**, September 2010. Disponivel em:
2. <projects.semwebcentral.org/projects/owls-tc/>.

13 OBJECT MANAGEMENT GROUP, INC. Business Process Model and Notation. **Object**
3. **Management Group**, 2012. Disponivel em: <http://www.bpmn.org/>. Acesso em: 03 April 2012.

13 ROBERTS, F. S.; TESMAN, B. **Applied combinatorics**. 2, reimpressão, ilustrada. ed. [S.l.]: CRC
4. Press, 2009. 860 p. ISBN ISBN 1420099825.

13 APPLEGATE, D. L. **The traveling salesman problem:** a computational study. Princeton:
5. Princeton University Press., 2006.

13 SIRIN, E. **Interface OWLType**, 2004. Disponivel em: <http://www.mindswap.org/2004/owl-
6. s/api/doc/javadoc/org/mindswap/owl/OWLType.html>.

13 SEMANTIC WEB RESEARCH GROUP. OWL-S API. **Mindswap _ Maryland Information and**
7. **Network Dynamics Lab Semantic Web Agents Project**. Disponivel em: <http://www.mindswap.org/2004/owl-s/api/doc/>. Acesso em: 29 October 2008.

13 APACHE SOFTWARE FOUNDATION. **Apache Tomcat**, 1999-2009. Disponivel em:
8. <http://tomcat.apache.org/index.html>.

13 CENAPAD - SÃO PAULO. Centro Nacional de Processamento de Alto Desempenho em São
9. Paulo. **Centro Nacional de Processamento de Alto Desempenho em São Paulo**, 2011. Disponivel em: <http://www.cenapad.unicamp.br/home/maim.shtml>. Acesso em: 11 July 2011.

14 JAIN, R. **The Art of Computer Systems Performance Analysis:** Techniques for Experimental
0. Design, Measurement, Simulation, and Modeling. New York, NY: Wiley- Interscience, April 1991.

14 PEDRYCZ, W.; GOMIDE, F. **Fuzzy Systems Engineering:** Toward Human-Centric Computing. 1.
1. ed. [S.l.]: IEEE/Wiley Interscience, v. 1, August 13, 2007. ISBN ISBN-10: 0471788570.

14 SILVA, L. N. C. **Engenharia imunologica: desenvolvimento e aplicação de ferramentas**
2. **computacionais inspiradas em sistemas imunologicos artificiais**. Unicamp. Campinas. 2001. (Código: vtls000220201). Tese de doutorado. http://www.bibliotecadigital.unicamp.br/document/?code=vtls000220201.

14 DASGUPTA, D. (Ed.). **Artificial Immune Systems and Their Applications**. 1. ed. [S.l.]: Springer, v.

3. 1, December 11, 1998. 306 p. ISBN ISBN-10: 3540643907.

14 ADLEMA, L. M. Computing with DNA: The manipulation of DNA to solve mathematical
4. problems is redefining what is meant by “computation”. **Scientific American**, p. 54-61, 1998. http://www.cs.virginia.edu/~robins/Computing_with_DNA.pdf.