WebMedia2012

XVIII SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB

Foto: Museu do Ipiranga - São Paulo-SP por Yuri Alexandre

MINICURSOS

Tópicos em Multimídia, Hipermídia e Web

XVIII Simpósio Brasileiro de Sistemas Multimídia e Web

COODENAÇÃO DE MINICURSOS //
Alessandra Alaniz Macedo (USP) e Itana Stiubiener (UFABC)

COORDENAÇÃO GERAL //
Graça Bressan (USP) e Regina Melo Silveira (USP)

EDIÇÃO //
Sociedade Brasileira de Computação

15 a 18 de outubro de 2012

Frei Caneca Shopping & Convention Center

São Paulo / SP

http://sws2012.ime.usp.br/webmedia

SIMPÓSIOS'12
SBBD / WebMedia / SBSC

MINICURSOS

XVIII Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia)

15 a 18 de Outubro de 2012

São Paulo - SP

TÓPICOS EM MULTIMÍDIA, HIPERMÍDIA E WEB

Coordenação de Minicursos

Alessandra Alaniz Macedo (USP)

Itana Stiubiener (UFABC)

Coordenação geral

Graça Bressan (USP)

Regina Melo Silveira (USP)

Edição

Sociedade Brasileira de Computação

XVIII SIMPOSIO BRASILEIRO DE SISTEMAS MULTIMIDIA E WEB (WebMedia)

15 a 18 de outubro de 2012
São Paulo – SP

Tópicos em Multimídia, Hipermídia e Web

Minicursos

Coordenação de Minicursos
Alessandra Alaniz Macedo (USP)
Itana Stiubiener (UFABC)

Coordenação Geral
Graça Bressan (USP)
Regina Melo Silveira (USP)

Sociedade Brasileira de Computação
São Paulo
2012

FICHA CATALOGRÁFICA

Tópicos em multimídia, hipermídia e Web : minicursos / coord. A.A. Macedo, I. Stiubiener ; coord. geral G. Bressan, R.M. Silveira. – São Paulo : Sociedade Brasileira de Computação, 2012.
p.

Minicursos realizados durante o XVIII Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia), 15 a 18 de outubro de 2012.
ISBN: 978-85-7669-265-2

1. Multimídia 2. Hipermídia 3. Web I. Macedo, Alessandra Alaniz II. Stiubiener, Itana III. Bressan, Graça IV. Silveira, Regina Melo V. t.

CDD 006.7

Índice

Capítulo 1

Capítulo 2

Prefácio

Este livro é uma coletânea de minicursos com resultados de atividades de pesquisa científica ou tecnológica. O conteúdo de cada minicurso foi fornecido pelos seus autores e apresentado durante o Simpósio Brasileiro de Bancos de Dados (SBBD 2012), o Simpósio Brasileiro de Sistemas Colaborativos (SBSC 2012) e o Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia 2012), de 15 a 18 de Outubro de 2012 em São Paulo/SP.

Os minicursos, eventos de curta duração (4 horas), visam apresentar tópicos atuais de pesquisa e/ou tecnologia de interesse da comunidade de Sistemas Multimídia e Web. Pretende-se oferecer oportunidades de aprendizagem e de utilização dos novos conhecimentos nas áreas de atuação ou pesquisa dos participantes dos minicursos.

Os minicursos selecionados foram submetidos à chamada pública, divulgada no site dos eventos, em emails e na lista eletrônica da SBC (Sociedade Brasileira de Computação). Foram recebidas diversas propostas de minicursos, as quais foram avaliadas pelo comitê científico de minicurso do WebMedia: Alessandra Alaniz Macedo (USP), Carlos Alberto Kamienski (UFABC), João Paulo Góis (UFABC), Itana Stiubiener (UFABC), Marcelo Zuffo (EPUSP), Romero Tori (EPUSP), Roseli Lopes (EPUSP), Tatiana Tavares (UFPB) e Thais Batista (UFRN). Quatro propostas foram selecionadas. Cada proposta de minicurso aceita é apresentada em um capítulo deste livro.

O Capítulo 1 do minicurso "**Introdução ao Desenvolvimento Colaborativo de Regras SWRL com o SWRL Editor**" apresenta uma introdução ao desenvolvimento de conjunto de regras em SWRL (Semantic Web Rule Language). A linguagem SWRL (um padrão do W3C) permite a combinação de regras e termos de ontologias (definidos em OWL - Web Ontology Language) para aumentar a expressividade de ambos. O capítulo apresenta uma breve introdução sobre web semântica, na qual é abordada a linguagem OWL, uma linguagem padrão do W3C para representar e compartilhar ontologias na Web. Em seguida, é feita uma introdução prática da linguagem SWRL com a construção de uma pequena ontologia com um conjunto de regras. Para a criação desse conjunto de regras é usado o SWRL Editor, um novo editor de regras SWRL integrado ao Web-Protégé (versão Web do editor de ontologias Protégé).

O Capítulo 2 do minicurso "**Software as a Service: Desenvolvendo Aplicações Multi-Tenancy com Alto Grau de Reúso**" apresenta software como serviço (SaaS) que representa um novo paradigma e um modelo de negócios onde as empresas não precisam comprar e manter sua própria infraestrutura de TI. Ao invés disso, elas adquirem um serviço de software de terceiros, obtendo assim consideráveis benefícios, principalmente no que tange a redução de custos de manutenção dessa infraestrutura. O objetivo desse minicurso é apresentar os principais conceitos relacionados à arquitetura multi-tenancy, uma das abordagem para implementação de Software como Serviço. Durante esse trabalho, são apresentadas as principais tecnologias associadas ao assunto e apresentar um exemplo prático da implementação de um aplicativo multi-tenancy utilizando o framework Grails e componentes reutilizáveis.

O Capítulo 3 do minicurso "**Desafios em Cloud Computing: Armazenamento, Banco de Dados e Big Data**" apresenta que com o aumento da velocidade de conectividade e evolução dos sistemas Web, começa a surgir os sistemas de Internet, que mais comumente são chamados de Computação nas Nuvens. Este termo designa uma plataforma de suporte a sistemas de software que provê aos seus usuários: gerenciamento, uso sob demanda, adequação às necessidades, racionalização do uso dos recursos e automação dos processos relacionados à criação de infra-estruturas. Neste contexto, surgem os sistemas de armazenamento de dados em nuvem, escalabilidade de banco de dados e busca e recuperação que hoje chamamos de BIGDATA. Este minicurso aborda estes temos exemplificando como implementar e utilizar tais plataformas.

O Capítulo 4 apresenta o texto do minicurso "**Análise de Informações Contextuais através de Técnicas de Aprendizagem de Máquinas**" que apresenta os conceitos introdutórios de técnicas de aprendizado de máquina como rede bayesiana, árvore de decisão e redes neurais e discuti como essas técnicas podem ser aplicadas na análise de informações contextuais. Neste minicurso são utilizadas as APIs da ferramenta Weka, desenvolvida na Universidade de Waikato na Nova Zelândia, que contempla, um conjunto de técnicas de aprendizagem de máquina implementado. Também são apresentados exemplos de uso de contexto na personalização e recomendação de conteúdo em ambientes Web, móveis e de TV Digital.

Alessandra Alaniz Macedo (USP)

Itana Stiubiener (UFABC)

Coordenadoras dos Minicursos do WebMedia 2012

TÓPICOS EM MULTIMÍDIA, HIPERMÍDIA E WEB

Capítulo

1

Introdução ao desenvolvimento colaborativo de regras SWRL com o SWRL Editor

João Paulo Orlando, Adriano Rívolli e Dilvan de Abreu Moreira
Dept. of Computer Science, ICMC Universidade de São Paulo – Campus de São Carlos, Caixa Postal 668, 13560-970, São Carlos-SP, Brazil
{orlando, rivolli, dilvan}@icmc.usp.br

Abstract

This chapter presents an introduction to the development of rule sets in SWRL (Semantic Web Rule Language). The SWRL language (a W3C standard) allows the combination of rules and ontology terms (defined in OWL Web Ontology Language) to increase the expressiveness of both. The chapter presents a brief introduction to the semantic web, in which we introduce the OWL language, a W3C standard for representing and sharing ontologies over the Web. After that, the chapter shows a hands-on introduction to SWRL where we build a small ontology with a rule set. This rule set will be created and tested using the SWRL Editor, a new SWRL rule editor integrated with Web-Protégé (the Web version of the Protégé ontology editor).

Resumo

Este capítulo apresenta uma introdução ao desenvolvimento de conjuntos de regras em SWRL (Semantic Web Rule Language). A linguagem SWRL (um padrão do W3C) permite a combinação de regras e termos de ontologias (definidos em OWL Web Ontology Language) para aumentar a expressividade de ambos. O capítulo apresenta uma breve introdução sobre web semântica, na qual será abordada a linguagem OWL, uma linguagem padrão do W3C para representar e compartilhar ontologias na Web. Em seguida, é feita uma introdução prática da linguagem SWRL com a construção de uma pequena ontologia com um conjunto de regras. Para a criação desse conjunto de regras, será usado o SWRL Editor, um novo editor de regras SWRL integrado ao Web-Protégé (versão Web do editor de ontologias Protégé).

1.1. Introdução

A Web Semântica é uma maneira de explorar a associação de significados explícitos aos conteúdos de documentos presentes na Web, para que esses possam ser processados diretamente ou indiretamente por máquinas [Berners-Lee e Fischetti 2008]. Para possibilitar esse processamento, os computadores necessitam ter acesso a coleções estruturadas de informações (dados e metadados) e a conjuntos de regras de inferência sobre esses conteúdos (que ajudem no processo de dedução automática) para que seja possível o raciocínio automatizado sobre os mesmos [Berners-Lee et all 2001].

A Web Semântica renovou e aumentou o interesse em sistemas baseados em regras e seu desenvolvimento [Zacharias 2008]. A SWRL é a linguagem padrão para regras da Web Semântica, muitas áreas de estudo na computação estão usando SWRL, entre elas podemos citar: Serviços sensíveis ao contexto [Wusheng et all 2011], gestão de energia em ambientes domésticos [Rossello-Busquet et all 2011], sistemas de *e-learning* [Vesin et all 2011], gerenciamento de SLA (*Service Level Agreement*) para serviços de IPTV (*Internet Protocol Television*) [Seo et all 2011], cálculos de redes de co-autoria em redes sociais [Ahmedi et all 2011], sensores em ambientes inteligentes [Sadoun et all 2011], etc. Na área da informática biomédica, é possível citar outras aplicações de regras SWRL: Atribuição de notas a tumores [Levy et all 2009] e para classificar fenótipos de Autismo [Hassanpour et all 2011]. Como pode ser visto muitas áreas tem usado regras SWRL para facilitar a modelagem de problemas.

1.2. Ontologia e a Web Semântica

1.2.1. Ontologia

O W3C [Heflin 2004] afirma que as ontologias são vistas como a tecnologia de consolidação para a construção da Web Semântica. O termo é emprestado da Filosofia, em que uma ontologia é um relato sistemático da existência [Gruber 1993].

[Studer et all 1998] define ontologia como uma especificação **formal** e **explícita** de uma **conceituação compartilhada**. **Conceituação** se refere a um modelo abstrato de algum fenômeno no mundo, identificando os conceitos relevantes daquele fenômeno. **Explícito** significa que os conceitos utilizados e as restrições sobre seu uso são explicitamente definidos. **Formal** refere-se ao fato de que a ontologia deve ser legível pelas máquinas. **Compartilhado** refere-se à noção de que uma ontologia captura o conhecimento consensual, isto é, não é privado de algum indivíduo, mas aceito por um grupo.

Discussões e estudos aprofundados sobre a definição e conceituação do termo ontologia podem ser encontrados em [Guarino and Giaretta 1995]; [Chandrasekaran et all 1999]; [Smith et all 2001]; [Almeida and Bax 2003] e [Corazzon 2010]. Mesmo sem um consenso sobre sua definição, ontologias compartilham características comuns e impulsionam o desenvolvimento de diversos trabalhos referentes a metodologias, ferramentas, linguagens e aplicações.

Uma ontologia é especificada por meio de componentes básicos que são as **classes**, **relações**, **axiomas** e **instâncias**, além de ser expressa por meio de uma linguagem de construção [Almeida and Bax 2003].

As **classes**, o foco da maioria das ontologias, são utilizadas para descrever os conceitos de um domínio, possibilitando a organização das classes em um sistema lógico e hierárquico contendo subclasses que representam conceitos mais específicos [Noy and McGuinness 2001].

As **relações** representam o tipo de interação entre os conceitos de um domínio e as propriedades presentes nas classes e indivíduos. Os termos *slot*, *role*, propriedade e até mesmo atributo podem ser empregados como sinônimo para as relações [Noy and McGuinness 2001] [Horridge et all 2004]. As relações podem ter características próprias como serem transitivas, simétricas, ou terem uma cardinalidade definida.

Os **axiomas** são utilizados para modelar regras assumidas como verdadeiras no domínio em questão, de modo que seja possível associar o relacionamento entre os indivíduos, além de fornecer características descritivas e lógicas para os conceitos. Para [Uschold and Grüninger 1996] os axiomas são especificados para definir a semântica e significado dos termos (classes e propriedades) e sugere que a fase de definição dos axiomas (especificação da ontologia) é a mais difícil na construção de ontologias.

Por fim, os **indivíduos**, ou instâncias das classes, são utilizados para representar elementos específicos, ou seja, os próprios dados, que juntamente com a definição de uma ontologia constituem a base de conhecimento [Noy and McGuinness 2001]. Os indivíduos representam objetos do domínio de interesse [Horridge et all 2004].

Os componentes básicos de uma ontologia são definidos por meio de uma linguagem de representação. [Horridge et all 2004] aponta que diferentes linguagens para ontologias proporcionam diferentes facilidades e recursos. Alguns exemplos de linguagens que se prestam à construção de ontologias são apresentados a seguir, na Tabela 1.1. Uma listagem mais completa de linguagens de representação de ontologias pode ser encontrada em [Su and Ilebrekke 2002] e [Almeida and Bax 2003].

Tabela 1.1 – Linguagens para construção de ontologias

Linguagens	Breve descrição
DAML	*DARP Agent Markup Language* (DAML) é precursora do DAM+OIL que originou a OWL. Sua sintaxe permite que o computador possa fazer as mesmas inferências simples que os seres humanos podem fazer [Ouellet and Ogbuji 2002].
OBO	*Open Biological Ontologies* (OBO) é uma linguagem de ontologia que tem sido frequentemente utilizada para a modelagem de ontologias nas áreas biomédicas. Faz uso de uma sintaxe textual simples que foi projetada para ser compacta, legível por humanos e fácil de ser processada [Golbreic et all 2007].
OIL	Ontology Interchange Language (OIL) é base para as linguagens da Web Semântica é precursora do DAM+OIL que originou o OWL. É uma linguagem baseada em frames juntamente com as características de lógica descritiva (*Description Logic* – DL) [Fensel et all 2001].
OWL	*Web Ontology Language* (OWL) projetada para atender a necessidade de uma linguagem de representação de ontologias para a Web. Utiliza RDF para a sintaxe e lógica descritiva para a definição de regras de inferência. É o padrão adotado no W3C como linguagem da Web Semântica [McGuinness and Harmelen 2004].
RDF	*Resource Description Framework* (RDF) é uma linguagem para definição de informação na Web. Descreve os dados (significado dos termos e conceitos) em um formato que máquinas podem processar. Utiliza o *eXtensible Markup Language* (XML) e *Uniform Resource Identifier* (URI) e conjuntos de triplas (sujeito, verbo e objeto) formam as sentenças elementares [Berners-Lee et all 2001].

Por fim, as ontologias são utilizadas para promover a interoperabilidade entre sistemas, ao representarem os dados compartilhados por diversas aplicações [Uschold

and Grüninger 2004]. Ontologias são amplamente utilizadas para fins diferentes e em diferentes comunidades.

1.2.2. Web Semântica

A Web Semântica, uma extensão da Web atual, é uma representação capaz de associar significados explícitos aos conteúdos dos documentos disponíveis na Internet, sendo que sua principal meta é possibilitar que programas processem e interpretem automaticamente esses documentos [Berners-Lee et all 2001]. Para Berners-Lee, a Web Semântica deve possibilitar que computadores sejam capazes de acessar dados estruturados e de definir regras de inferências, transformando grandes volumes de dados em informação.

Seu emprego é fortemente recomendado pelo W3C, que busca desenvolver padrões, arquiteturas de metadados e linguagens para ontologias que juntos possibilitem a integração e entendimento dos dados por computadores, agregando aos mesmos significados. Sua exploração é motivada pelo potencial que tem em transformar a Internet, vista hoje como um repositório de dados, em um repositório explícito de conhecimento, disponível tanto para pessoas como para máquinas. O papel do W3C no contexto da Web atual é o desenvolvimento de padrões, recomendações e orientações com o objetivo de levar a Web ao seu potencial máximo. Além dos avanços relacionados com as aplicações da Web, o W3C tem mobilizado grandes esforços e iniciativas para o desenvolvimento de uma Web para todos, em todos os dispositivos, baseada no conhecimento, com confiança e confiabilidade [Diniz and Cecconi 2008].

A adoção das tecnologias da Web Semântica e de ontologias na representação de dados, embora não tenha sido amplamente difundida e adotada até o momento [Shadbolt et all 2006], conta com o apoio de inúmeras empresas na divulgação e desenvolvimento de soluções que fazem o uso da Web Semântica [Feigenbaum et all 2007].

A tarefa de associar significados aos dados é possível pelo uso de tecnologias como RDF e OWL. O RDF utiliza XML e URI para proporcionar uma representação minimalista do conhecimento na Web e tem como característica principal ser simples. Por outro lado, a OWL é uma tecnologia complexa e voltada para a representação de objetos que requerem grande poder de expressividade. OWL usa RDF e possibilita a criação de ontologias para representação de conhecimento [Shadbolt et all 2006].

1.2.2.1. RDF

Resource Description Framework (RDF) é um esquema para a definição de informações na Web, provendo tecnologia para expressar o significado de termos e conceitos de modo que os computadores possam facilmente processá-los [Berners-Lee et all 2001]. Sua sintaxe pode ser definida com o uso de marcações XML e URI, empregadas para especificar entidades, conceitos, propriedades e relacionamentos. [Lassila and Swick 2004] define que RDF é a tecnologia base para o processamento de metadados, provendo assim interoperabilidade entre aplicações que trocam e processam informações na Web. Para os autores, um dos objetivos do RDF é possibilitar a criação de semântica para dados baseados em XML. Contudo, o principal objetivo do desenvolvimento do RDF é permitir a descrição de recursos de qualquer domínio específico.

Definida pelo W3C, o RDF organiza o conhecimento por meio da idéia de redes semânticas, e permite a representação de conceitos, taxonomia de conceitos e relações binárias [Almeida and Bax 2003]. Em um relato histórico, [Smith and Welty 2004] afirma que o RDF foi a primeira linguagem especificada pelo W3C para representar informações semânticas na Web.

O modelo de dados em RDF é uma forma neutra de representar as expressões utilizadas para atribuir significado aos dados. O modelo básico consiste em três tipos de objetos: recursos, propriedades e afirmações[1] que são chamadas respectivamente de sujeito, verbo e objeto, formando a tripla RDF [Berners-Lee et all 2001]. A codificação do conhecimento se concretiza em conjuntos de triplas, representadas graficamente na Figura 1.1.

Figura 1.1 – Tripla RDF [Klyne et all 2004].

O sujeito compreende qualquer coisa que pode ser expressa e descrita em RDF. Alguns exemplos de recursos são: um documento HTML; uma parte de uma página Web; um elemento XML específico; um conjunto de páginas; e objetos que não podem ser acessados diretamente na Web, como um livro impresso ou uma pessoa. Os recursos são sempre nomeados por URI, pois a expressividade deste recurso possibilita que qualquer entidade possa ter seu identificador [Lassila and Swick 2004].

O verbo consiste nas características, atributos ou relacionamentos utilizados para descrever um recurso. Cada propriedade tem um significado específico, define seus valores permitidos, os tipos de recursos que podem descrever, e sua relação com outras propriedades. O valor de uma propriedade atribuída a um recurso específico é o objeto. O valor da propriedade pode ser outro recurso, especificado por um URI, ou um valor literal (simples string ou outro tipo primitivo definido em XML). Tendo como base a tripla, a linguagem RDF possui uma sintaxe abstrata que pode ser expressa em XML, Notation 3 (N3) ou graficamente, de modo que a semântica formal é definida com base nesta sintaxe abstrata. Uma completa descrição desta sintaxe é apresentada em [Klyne et all 2004].

Uma poderosa extensão do RDF é o *RDF Schema*, usado para descrição de vocabulário utilizado para descrever os relacionamentos internos de propriedades e valores, sendo considerado um mecanismo de extensão semântica para o RDF [Brickley 2004]. O *RDF Schema* prove mecanismos para descrever grupos de recursos relacionados, além do relacionamento entre estes recursos. É possível utilizar o conceito de classes, indivíduos, propriedades e subpropriedades semelhantemente ao recurso de hierarquia de classes.

Para [Feigenbaum et all 2007], o RDF é o mais fundamental bloco de construção para a Web Semântica, pois, além de poder ser utilizado para criar dados semânticos, é também utilizado como base para as linguagens de ontologia da Web Semântica. Entretanto para [Staab et all 2001], os dados em RDF são fracamente interligados, de modo que a Web Semântica necessita de técnicas mais sofisticadas.

[1] Tradução livre do o termo *statement*

1.2.2.2. OWL

A linguagem de ontologia *Web Ontology Language* (OWL) é destinada para os que desejam grande expressividade na descrição dos objetos e seus relacionamentos [Shadbolt et all 2006]. O *RDF Schema* é uma linguagem ontológica considerada leve [Staab et all 2001], pois não contém a complexidade e riqueza de outras linguagens como OWL. Todavia, OWL usa as ligações RDF e trabalha uma camada acima do mesmo [Feigenbaum et all 2007].

O W3C publicou em 2004 a linguagem de marcação semântica OWL como recomendação para representar e compartilhar ontologias na Web [Smith and Welty 2004]. Contudo, OWL foi concebida de uma revisão da linguagem DAML+OIL incorporando as lições aprendidas durante o desenvolvimento e aplicação desta [McGuinness and Harmelen 2004].

A OWL 1 provê três sublinguagens com expressividade crescente: *OWL Lite*, *OWL DL* e *OWL Full*, sendo a primeira a mais simples e a última a mais complexa. A simplicidade e complexidade referidas aqui dizem respeito ao vocabulário, recursos e capacidades de expressividade de cada sublinguagem.

OWL Lite suporta necessidades primárias, como classificação de hierarquia e construções simples, entretanto possui restrições de cardinalidade e tem um número menor de propriedades. O uso desta sublinguagem é recomendado para migração de thesauri e outras taxonomias. *OWL DL* e *OWL Full* compartilham a máxima expressividade incluindo todas as construções da linguagem OWL. A diferença entre estas sublinguagens consiste que *OWL DL* garante que a ontologia seja sempre computável e decidível enquanto que, com *OWL Full*, não se pode ter esta garantia. Isso ocorre pelo fato de *OWL DL* respeitar as restrições de definição presentes na lógica *Description Logic*, daí o sufixo DL.

[Barder and Nutt 2003] define *Description Logic* como um formalismo matemático que representa o conhecimento de um domínio de aplicação (o "mundo") primeiramente definindo os conceitos relevantes desse domínio (sua terminologia) e depois usando esses conceitos para especificar propriedades de objetos e indivíduos que ocorrem nesse domínio (a descrição do mundo). Esse formalismo é que garante que ontologias em *OWL DL* vão ser computáveis e decidíveis. A finalidade de uma linguagem de representação de conhecimento que usa *Description Logic* vai além de armazenar e definir conceitos e afirmações. Suas definições podem ser validadas (checagem de consistência) e podem conter conhecimento implícito, obtido através de inferências lógicas.

A ferramenta capaz de gerar as inferências lógicas e checar a consistência em OWL é chamada de *reasoner*. Um dos principais testes que essa ferramenta pode realizar é verificar se uma classe é uma subclasse de outra classe, de modo que isso ajuda a manter uma hierarquia correta e identificar classes inconsistentes [Horridge et all 2004]. Em OWL, uma inconsistência acontece se uma classe não pode ter instâncias.

Para [Smith and Welty 2004], uma das vantagens do uso da linguagem OWL para a construção de ontologias é que existem ferramentas de classificação. Estes tipos de ferramenta proporcionam um suporte genérico, aplicável a todos os domínios de aplicação. Uma lista de classificadores é encontrada em [Grau et all 2008].

OWL trabalha com suposições de mundo aberto, considerando que as descrições de recursos podem estar presentes em mais de um único arquivo ou escopo. Isso implica na impossibilidade de assumir que algo não existe até que isso seja explicitamente definido, ou ainda, algo não pode ser definido como verdadeiro, pelo fato de não poder ser assumido como falso, pois o conhecimento pode existir sem ter sido adicionado à base de conhecimento [Horridge et all 2004].

Proporcionando uma visão geral da OWL, [McGuinness and Harmelen 2004] apresenta a sintaxe expressa em RDF/XML. Com isso, alguns termos possuem o prefixo rdf: ou rdfs: compreendendo os mesmos termos presentes em RDF ou *RDF Schema* respectivamente.

Recentemente o W3C agregou novos recursos à linguagem OWL dando origem a uma nova recomendação, chamada OWL 2, que é uma extensão e revisão da anterior [W3C OWL Working Group 2009].

A nova linguagem é consequência de um amadurecimento e evolução da Web Semântica, e com isso, novos recurso e possibilidades foram incorporados em OWL 2. O uso de outras linguagens além do RDF/XML para definição da sintaxe, novos recursos semânticos como redes de propriedades, definição de intervalo de dados, tipos de dados mais ricos e novos tipos de propriedades são exemplos dos novos recursos incorporados.

A justificativa encontrada por [Grau et all 2008] para justificar a necessidade da evolução da OWL, é o seu sucesso. Pois, devido ao sucesso da "OWL 1" foram identificados problemas quanto ao design da linguagem, sendo alguns de natureza crítica. Problemas como as limitações de expressividade juntamente com o fato da linguagem OWL ter sua sintaxe influenciada pelo paradigma de frames, são alguns dos problemas identificados pelo autor que justificam a evolução da linguagem.

1.3. Regras SWRL

Infelizmente a expressividade de OWL nem sempre é suficiente para modelar todos os tipos de problemas. Especialmente OWL não tem suporte a cláusulas no formato de Horn (representa uma sentença de implicação). Para suprir essa deficiência, a SWRL (*Semantic Web Rule Language*) foi criada.

A SWRL é baseada em uma combinação de *OWL DL* e *OWL Lite* [O'Connor et all 2005]. Essa combinação amplia o conjunto de axiomas da linguagem OWL, mais especificamente para poder incluir cláusulas no formato de Horn [O'Connor et all 2005]. Essas regras podem ser usadas para inferir novos conhecimentos a partir de bases de conhecimento em OWL. SWRL permite que os usuários escrevam regras para raciocínio sobre os indivíduos OWL (instâncias) que podem inferir novos conhecimentos sobre esses indivíduos.

Regras em SWRL são compostas de duas partes: o antecedente (*body*) e o consequente (*head*). Cada regra é uma implicação entre o antecedente e o consequente, que pode ser entendida como: quando as condições do antecedente são verdadeiras, então as condições do consequente também são verdadeiras. Ambas as partes consistem em uma conjunção de zero ou mais átomos, não permitindo disjunções ou negação.

Os átomos, por sua vez, são formados por um predicado e um ou mais argumentos (o número e o tipo destes argumentos são determinados pelo tipo do átomo,

que por sua vez, é definido pelo predicado utilizado). Embora a especificação W3C defina sete tipos de átomos, neste capítulo foi adotada a convenção usada em [Hassanpour et all 2009], que trabalha com seis tipos de átomos, pois, SameAs e DifferenteFrom embora possuam semânticas diferentes, sintaticamente são idênticos. Na Tabela 1.2 são apresentados os tipos de átomos e suas descrições.

Tabela 1.2 – Tipos de átomos SWRL [Hassanpour et all 2009]

Tipo de átomo	Descrição
Class	O predicado corresponde a uma classe definida na ontologia e recebe um indivíduo[2] como argumento.
Object property	O predicado corresponde a uma propriedade definida na ontologia e recebe dois indivíduos como argumentos e os relaciona entre si.
Datavalued property	O predicado corresponde a uma propriedade definida na ontologia e recebe dois argumentos: um indivíduo e um valor[3] relacionando-os.
Data range	O predicado corresponde a um tipo de dado definido na ontologia e recebe um valor como argumento.
Same/different	O predicado corresponde ao termo *"same as"* ou *"different from"* e recebe dois indivíduos como argumentos definindo que estes são iguais ou diferentes respectivamente.
Built-in	O predicado corresponde a um conjunto de funções pré-definidas (comparações, funções matemáticas, lógicas, ...) ou definidas pelo usuário que recebem um ou mais argumentos e retornam verdadeiro quando estes satisfazem o predicado.

Os tipos *Class*, *Object property* e *Datavalued property* correspondem a elementos (conceitos) definidos na ontologia. Data range correspondem aos tipos de dados utilizados na ontologia. O tipo *Same*/*different* é utilizado para explicitar a igualdade ou diferença entre dois indivíduos, exigida pelo fato de OWL trabalhar com o paradigma de mundo aberto. Por fim, os Built-ins são tipos que encapsulam as mais diversas funções, podendo inclusive ser estendidos pelos usuários. Cada tipo de átomo define o número e o tipo de argumentos suportados, conforme ilustrado na Tabela 1.3.

Tabela 1.3 – Números e tipos de argumentos suportados pelos tipos de átomos

Tipo de átomo	Número de argumentos	Tipo de argument
Class	1	i-object
Object property	2	i-object, i-object
Datavalued property	2	i-object, d-object
Same/Different	2	i-object, i-object

[2] Na Tabela "Tipos de átomos SWRL" o termo indivíduo é empregado para representar: 1) um indivíduo específico definido na ontologia; ou, 2) variáveis para indivíduos, que podem ser quaisquer indivíduos definidos na ontologia. Ou seja, corresponde ao tipo I-Object.

[3] O termo valor neste caso é uma livre tradução de *data value* e representa um dado de um tipo primitivo definido na ontologia (Ex: inteiro, real, *string*). Na Tabela "Tipos de átomos SWRL", o termo valor é empregado para representar valores constantes ou variáveis para valores. Ou seja, corresponde ao tipo D-Object.

Tipo de átomo	Número de argumentos	Tipo de argument
Built-in	2 ou mais	d-object ou i-object
Data range	1	d-object

As variáveis são tratadas como quantificadores universais e possuem o escopo limitado à regra a qual pertencem. Apenas variáveis que ocorrem no antecedente podem ocorrer no consequente.

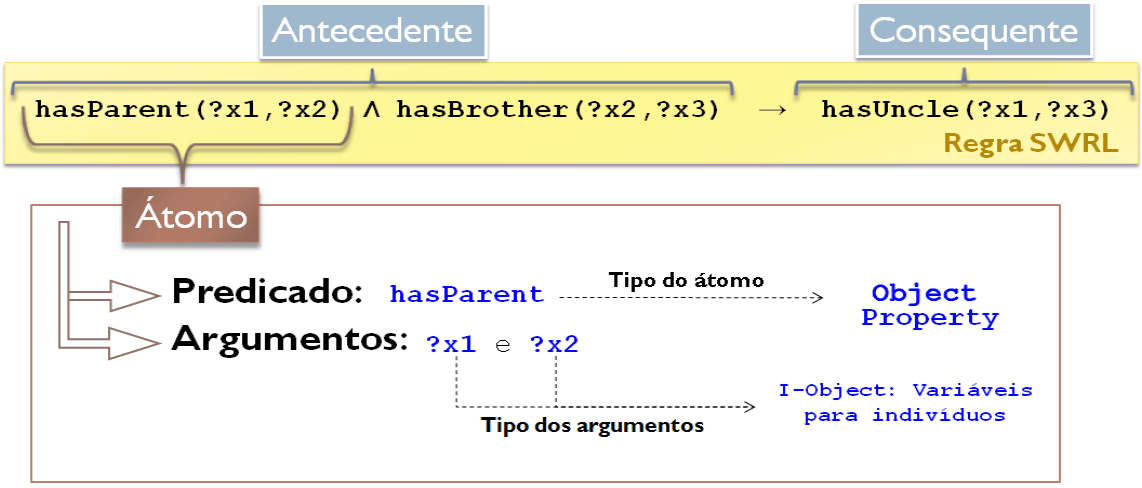


Figura 1.2 – Regra SWRL especificada no formato humano e suas partes.

Embora as regras SWRL possam ser representadas em mais de um formato, o formato de leitura humano é adotado neste capítulo. Neste formato, a seta (→) é usada para separar antecedente e consequente, o acento circunflexo (^) representa a conjunção entre os átomos e o sinal de interrogação (?) distingue as variáveis dos nomes de indivíduos. Na Figura 1.2 é apresentada uma regra SWRL, representada no formato humano, ressaltando suas partes e características.

Semanticamente, o exemplo anterior ilustra que **se** o pai/mãe de um indivíduo possui um irmão, **então** este irmão é tio do indivíduo em questão. A regra possui três átomos (todos do tipo *Object property*) sendo dois no antecedente e um no consequente (cujos predicados são: hasParent, hasBrother e hasUncle respectivamente) e faz uso de três variáveis para indivíduos: ?x1, ?x2 e ?x3. Outro exemplo, suma seria uma regra SWRL que expresse a seguinte afirmação: “uma pessoa qualquer que tenha qualquer carro será considerada um motorista”, ficaria assim:

pessoa(?x) ^ temCarro(?x, true) -> motorista(?x)

Ao executar-se a regra acima, o seu efeito seria classificar todos os indivíduos da classe pessoa que possuíssem carro como também pertencentes à classe motorista. As regras SWRL também permitem trabalhar com um indivíduo específico de uma ontologia. Por exemplo, é possível escrever uma regra diretamente para um indivíduo João, pertencente à classe pessoa, e fazer uma classificação direta deste indivíduo. Um exemplo pode ser visto na regra abaixo:

pessoa(João) ^ temCarro(João, true) -> motorista(João)

A regra acima apenas funciona para o indivíduo conhecido como João. Um dos recursos mais poderosos do SWRL é a capacidade de usuários usarem *built-ins* definidos [SWRLLanguage 2012]. *Built-in* é um predicado que pode ter um ou mais argumentos e realiza uma operação com os argumentos, avaliando e retornando verdadeiro ou falso. Por exemplo, uma regra que classifica um indivíduo da classe pessoa que seja maior de 18 anos como da classe adulto pode ser vista abaixo:

pessoa(?x) ^ temIdade(?x, ?idade) ^ swrlb:greaterThan(?idade, 18) -> adulto(?x)

SWRL permite o uso de novas bibliotecas de *built-ins* e os usuários podem definir suas próprias bibliotecas, o que torna SWRL uma linguagem muito rica em recursos de representação. Por exemplo, poderia ser criada uma biblioteca que tivesse operações de conversão de valores e busca de termos em taxonomias.

Um exemplo mais complexo do uso de SWRL é mostrado por [Hassanpour et all 2009], neste exemplo o objetivo é estabelecer relações entre duas ou mais entidade de uma ontologia. O exemplo cria uma regra que estabelece regras de condução no estado da Califórnia para menores de 18 anos:

Person(?p) ^ has_Driver_License(?p,?d) ^ issued_in_State_of(?d,?s) ^ swrlb:notEqual(?s,"CA") ^ has_Age(?p,?g) ^ swrlb:lessThan(?g,18) ^ number_of_Visiting_Days_in_CA(?p,?x) ^ swrlb:lessThan(?x,10) ^ Car(?c) ^ has_Weight_in_lbs(?c,?w) ^ swrlb:lessThan(?w,26000)

→ can_Drive(?p,?c)

Traduzindo essa regra para linguagem humana teríamos: “Qualquer indivíduo com idade inferior a 18 anos, mas que possua carta de motorista, sendo de fora da Califórnia, visitando o estado por menos de 10 dias e dirigindo um veiculo com menos de 26000 libras, pode dirigir normalmente”.

Cabe ressaltar que as regras são armazenadas na própria ontologia e o uso deste tipo de anotação permite a inferência de novos conhecimentos sobre indivíduos, uma vez que as regras correspondem a afirmações condicionais que, ao serem satisfeitas, agregam novas informações à base de conhecimento.

1.4. SWRL Editor

Nesta seção é descrita uma nova ferramenta, intitulada SWRL Editor, que foi desenvolvida nos trabalhos [Orlando 2012] e [Silva 2012]. O SWRL Editor foi desenvolvido como um plug-in para o Web-Protégé. O Web-Protégé é um editor open-source de ontologias, baseado no Protégé que funciona na Web [Tudorache et all 2008]. Ele foi desenvolvido usando o GWT (*Google Web Toolkit*). O principal motivo de se ter uma versão Web do Protégé é facilitar a colaboração entre desenvolvedores, já que não é necessário instalar programas locais. Nessa ferramenta, as ontologias estão disponíveis de forma centralizada e compartilhada. Como as ontologias estão se tornando cada vez maiores, é difícil que uma pessoa, ou um grupo pequeno de pessoas, consiga manter

grandes ontologias de forma eficaz [Tudorache et all 2008]. Daí a necessidade do desenvolvimento colaborativo de ontologias e regras.

O SWRL Editor foi integrado ao Web-Protégé e está dividido em Servidor e Cliente. O Servidor centraliza as operações com as regras, por exemplo, ontologias e regras são carregadas de seus arquivos apenas uma vez e ficam disponíveis para todos os clientes através de chamadas RPC. Além disso, a centralização das operações é um ponto chave para que a ferramenta possa funcionar colaborativamente, já que o servidor atua como nó central de distribuição das alterações feitas por cada usuário.

Já no Cliente estão as interfaces gráficas (as quais que são priorizadas nessa seção) são divididas em duas partes: Visualização (Seção 1.4.1), Filtros (Seção 1.4.2), Opções (Seção 1.4.3) e Composição (Seção 1.4.4). Para acessar o cliente do SWRL Editor é necessário escolher uma das ontologias. Na Figura 1.3 é mostrado o Web-Protégé após se ter acessado a Ontologia da Família. Nessa figura é possível ver a guia SWRL Editor (A), onde se encontra a nova ferramenta.

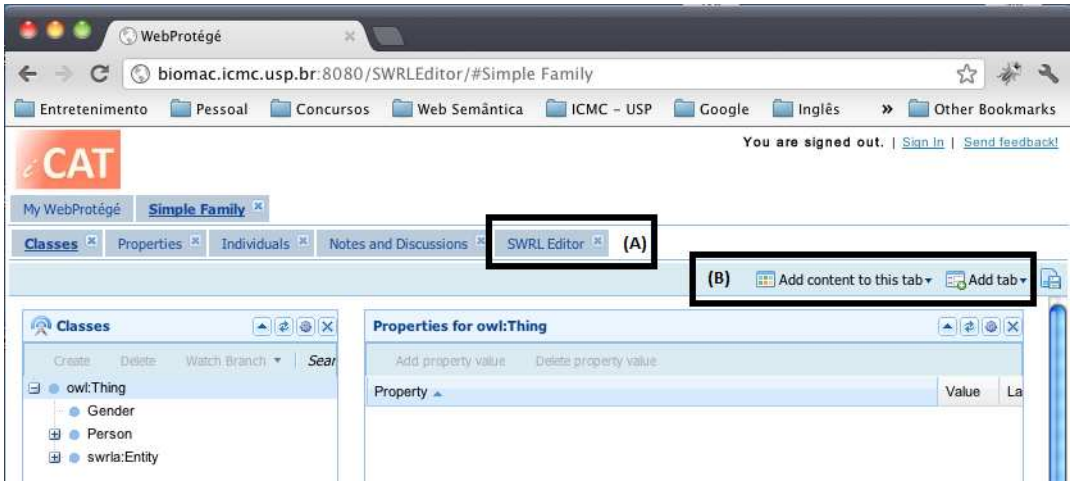


Figura 1.3 – Tela após o acesso a ontologia da família: (A) Acesso ao plug-in; (B) Acesso a todos os plug-ins não carregados automaticamente.

O Web-Protégé, após acessar a ontologia, carrega automaticamente os principais plug-ins (guias). Como não existe no Web-Protégé um plug-in para manipulação de SWRL, foi definido que o SWRL Editor é carregado automaticamente. Porém, caso seja fechado o plug-in, é possível acessá-lo novamente usando os botões Figura 1.3 (B).

1.4.1. Visualização

A página inicial do SWRL Editor é a *Visualizations* (Figura 1.4). Essa página é dividida em:

- Barra de Ferramentas (A) – Da esquerda para a direita: *Options* é o acesso as configurações do SWRL Editor; *Info* é o acesso as informações gerais do conjunto de regras; *New Rule* acessa a tela de composição para inserção de uma nova regra; *Run Rules* executa as regras no servidor. O botão Edit em *Rule Filter* acessa a tela para modificar o filtro das regras exibidas.
- Formas de Visualização dos Conjuntos de Regras (B) – Da esquerda para a direita: A guia *List* mostra as regras em uma visualização hierárquica (Seção 1.4.1.2). A guia *Text* mostra as regras usando Parafraseamento (Seção 1.4.1.3).

A guia SWRL mostra as regras usando SWRL com *Highlight* (1.4.1.1). A guia *Autism* mostra uma representação visual especial só para um conjunto especifico de regras para classificação de fenótipos de autismo [Silva 2012] e que não será abordada nesse capítulo. A guia *Groups* dispõe das técnicas para agrupamento (Seção 1.4.1.4). A guia *Decision Tree* contém as técnicas para transformar os conjuntos de regras para um formato de árvore (Seção 1.4.1.5).

- Botões para cada regra (C) – Para cada regra, nas guias *List*, *Text*, *SWRL* e *Autism*, existem botões para manipulação. Da esquerda para a direita são eles: O botão *Edit Rule* que coloca a regra em modo de edição. O botão *Duplicate and Edit* que faz uma cópia da regra e coloca a essa copia em modo de edição. O botão *Delete* excluí uma regra do conjunto. O botão *Similar Rules* mostra uma lista de regras similares usando um dos algoritmos de agrupamento.
- Barra de Status (D) – Serve como status do sistema, sendo utilizada para mostrar o numero de regras filtradas/número total de regras do conjunto.

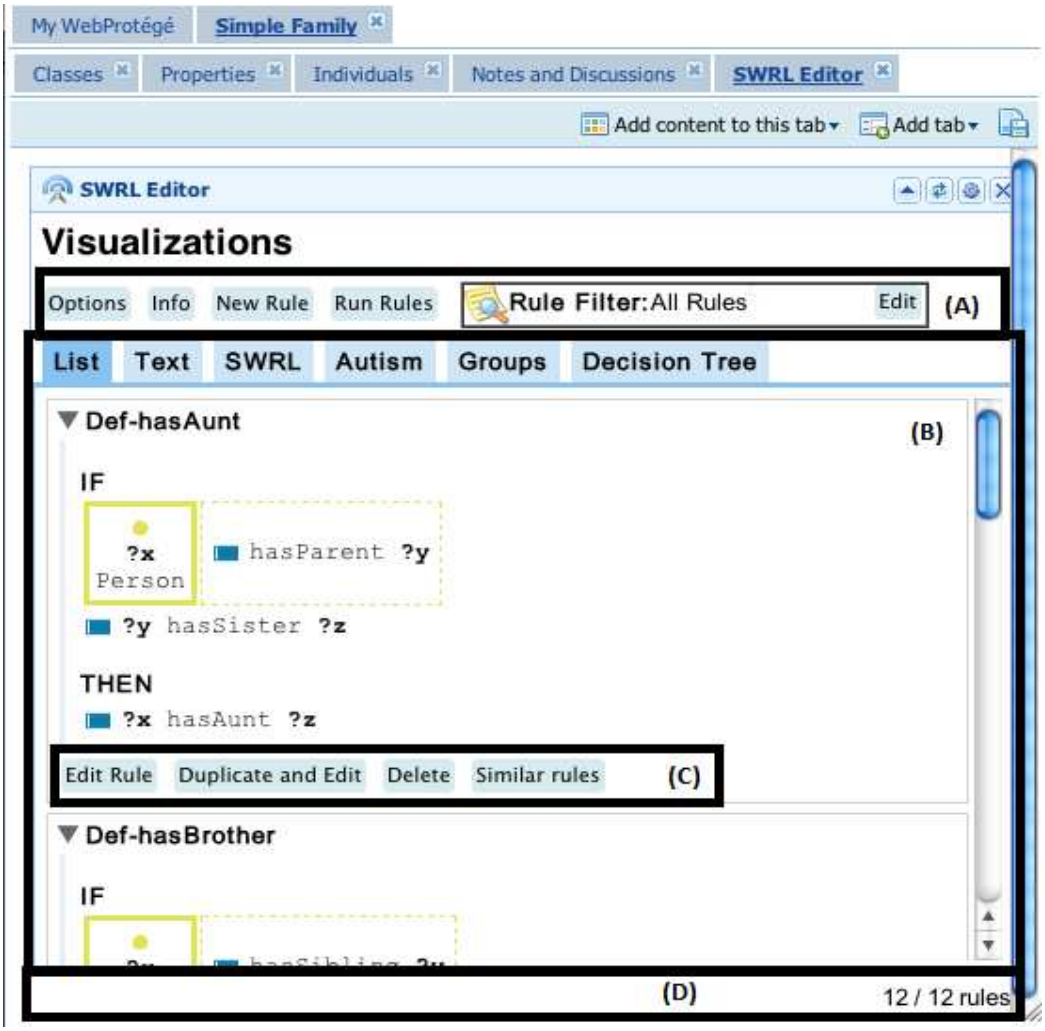


Figura 1.4 – SWRL Editor – Visualização: (A) Barra de Ferramentas; (B) Formas de Visualização dos Conjuntos de Regras; (C) Botões para cada regra; (D) Barra de Status do plug-in.

1.4.1.1. SWRL com Highlight

A representação visual *SWRL* exibe a regra sem ocultar os detalhes da sintaxe, contudo, agrega cores aos átomos e argumentos semelhantemente a recursos encontrados em alguns IDEs. Ao invés de utilizar a quebra automática no final da linha, cada átomo é apresentado em uma linha e o símbolo de implicação →, que divide o antecedente do consequente, é disposto em uma linha separada, o que torna a regra mais legível. Na Figura 1.5, a regra **def_hasAunt** é apresentada no formato original (A) e na representação com *highlight* (B).

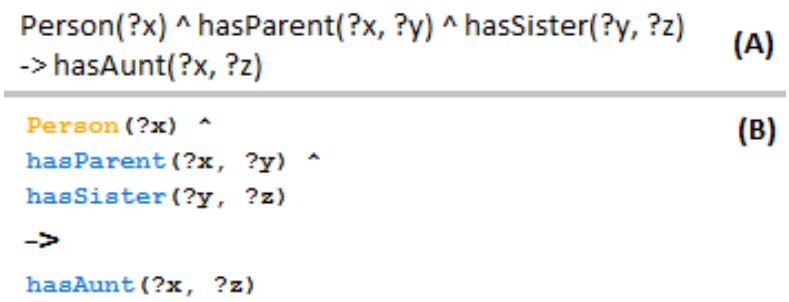


Figura 1.5 – Comparativo entre (A) uma regra apresentada no formato original e (B) a representação visual com *Highlight*.

Na Tabela 1.4 são apresentadas as cores utilizadas na geração dos *highlights* e seus significados. A representação visual destaca os tipos dos átomos, variáveis e valores literais sem sobrecarregar a apresentação da regra. Algumas das cores utilizadas foram extraídas dos símbolos utilizados pelo editor de ontologias Protégé, enquanto outras foram extraídas do editor Eclipse. O uso de uma fonte de tamanho fixo (monoespaçada) foi adotado seguindo o padrão encontrado nos editores de texto usados na programação, o que torna a leitura da regra mais organizada.

Tabela 1.4 – Cores utilizadas na representação *SWRL Highlight*

Cor	Exemplo	Descrição
Laranja	Person_Female	Representa predicados do tipo *Class*[4]
Azul	has_natural_father	Representa predicados do tipo *Object property*
Verde	has_current_age	Representa predicados do tipo *Datavalued property*
Vermelho	sqwrl:avg	Representa predicados do tipo *Built-in*
Cinza	differentFrom	Representa predicados do tipo *Same/different*
Preto em negrito	**?a**	Representa as variáveis
Roxo	32; "teste"	Representa os valores literais (Strings e números)

[4] Embora o Protégé utilize amarelo no símbolo que representa as classes, foi adotado para esta visualização o tom mais alaranjado pelo fato da cor amarela apresentar pouco contraste com fundos claros.

1.4.1.2. Visualização Hierárquica

A representação denominada visualização hierárquica busca abstrair a sintaxe da regra apresentando os átomos agrupados hierarquicamente a partir dos átomos do tipo *Class*. Assim, ao mesmo tempo em que torna a regra mais clara para apresentação, enriquece com cores e símbolos o seu significado. Os símbolos empregados na ferramenta Protégé foram reutilizados, contudo, para representar os demais elementos (*Same/Different* e *Built-ins*) foram empregados retângulos com as cores utilizadas na representação com *highlight* (Seção 1.4.1.1). Na Tabela 1.5 são apresentados exemplos dos símbolos utilizados na visualização.

Tabela 1.5 – Cores utilizadas na representação *SWRL Highlight*

Tipo	Símbolos e Cores	Exemplo
Class	Circulo amarelo	?b Person_Female
Object property	Retângulo azul	has_natural_father(?a,?f)
Datavalued property	Retângulo verde	has_current_age(?a,?b)
Built-in	Retângulo vermelho	?c > 18
Same/different	Retângulo cinza	?m same ?n ?m different ?n

O resultado dessa visualização pode ser visto na Figura 1.6, onde é apresentada a mesma regra (**def_hasAunt**) da representação com *highlight*. Embora esta representação oculte alguns detalhes da sintaxe SWRL e exiba os átomos organizadamente, seu uso se torna apropriado a desenvolvedores não técnicos em computação, apenas quando são utilizados na ontologia nomes apropriados nos labels (rdfs: Label) dos termos.

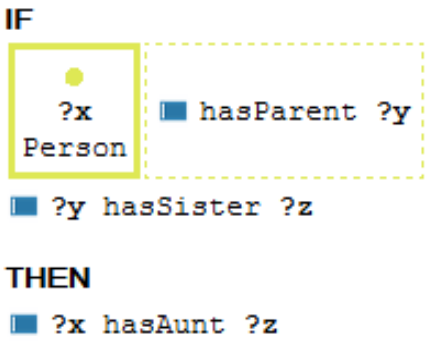


Figura 1.6 – Representação Hierárquica.

1.4.1.3. Parafraseamento

Essa representação fornece uma explicação em forma de texto da regra. É uma re-implementação de uma técnica do Axiomé [Hassanpour 2010] para gerar paráfrases em inglês. Apenas foi implementada uma mudança para essa técnica: a utilização de negrito em termos de ligação (IF, IS, AN, ...) entre os átomos de uma regra. A mudança é simples, porém torna mais fácil a leitura (Figura 1.7). Além disso, o SWRL Editor oferece a possibilidade de entrar em modo de edição a partir da representação em parafraseamento. Não edita o parafraseamento, mas acessa a edição desta regra.

```
IF
        "y" IS A Person
        AND "y" HAS Child "x"
        AND "y" HAS Child "z"

THEN
        "x" HAS Sibling "z"
```

Figura 1.7 – Parafraseamento.

1.4.1.4. Agrupamento

O agrupamento tem o objetivo de facilitar a visualização de grandes conjuntos de regras. Essa técnica facilita a localização de regras pelo usuário dentro de um conjunto de regras. O agrupamento é feito por meio de algoritmos que implementam interfaces Java e são carregados automaticamente no SWRL Editor. Atualmente, o SWRL Editor possui 3 algoritmos de agrupamento:

- *K-means* – Similaridade de átomos: Implementa o algoritmo de *K-means* considerando a similaridade dos átomos de cada regra [Silva 2012];
- *K-means* – Similaridade de predicados: Implementa o algoritmo de *K-means* considerando a similaridade dos predicados de cada átomo [Silva 2012];
- Estrutura da regra: Separa as regras em grupos por estrutura sintática das regras [Orlando 2012];

Além disso, o SWRL Editor possui sistemas de carregamento automático de algoritmos para o agrupamento. Esse sistema carrega os algoritmos de forma automática usando a classe Java: ServiceLoader. Para o ServiceLoader carregar os algoritmos é necessário que esses sejam implementados usando as interfaces Java que o SWRL Editor disponibiliza para agrupamento ou para árvore de decisão. Mais detalhes de como desenvolver novos algoritmos para o SWRL Editor estão disponíveis em [Orlando 2012].

O sistema de carregamento automático de algoritmos tornou-se uma ótima estratégia para usuários mais avançados que tenham interesse em desenvolver seus próprios algoritmos. Ele facilita essa operação já que para inserir um novo algoritmo é apenas necessário gerar um arquivo JAR com as implementações e o colocá-lo numa pasta do servidor, sem ser necessário alterar o código do SWRL Editor.

1.4.1.5. Árvore de Decisão

Da mesma forma que no agrupamento, árvores de decisão usam o sistema de carregamento automático de algoritmos e podem ser adicionados novos algoritmos. Atualmente, o SWRL Editor conta com 3 implementações de árvore de decisão:

- Ocorrência de átomos (Figura 1.8) – Essa técnica apenas cria uma árvore com base em um ranking de ocorrência de átomos, ou seja, átomos que mais ocorrem tendem a estar mais perto da raiz.
- Dependência de variáveis (Figura 1.9) – Essa técnica cria as árvores levando em conta a dependência das variáveis.

- Paráfrases (Figura 1.10) – Divide o parafraseamento (Seção 1.4.1.3) em pequenas paráfrases e com isso cria uma árvore mantendo a ordem original da paráfrase.

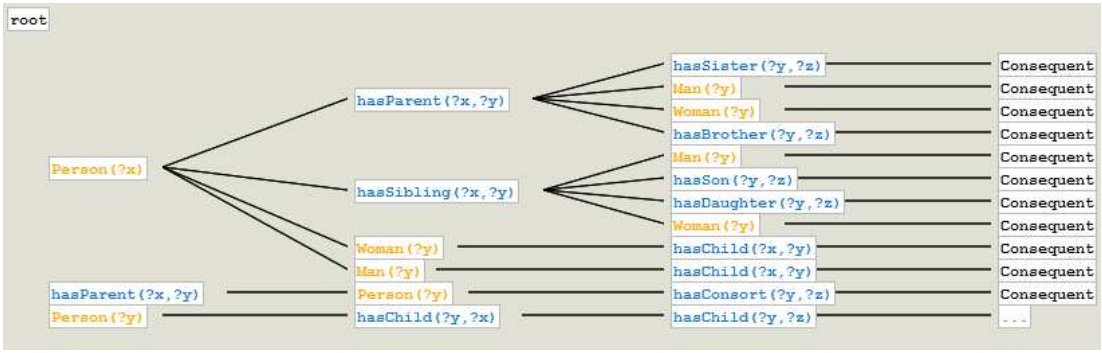


Figura 1.8 – Árvore de decisão: Ocorrência de átomos.

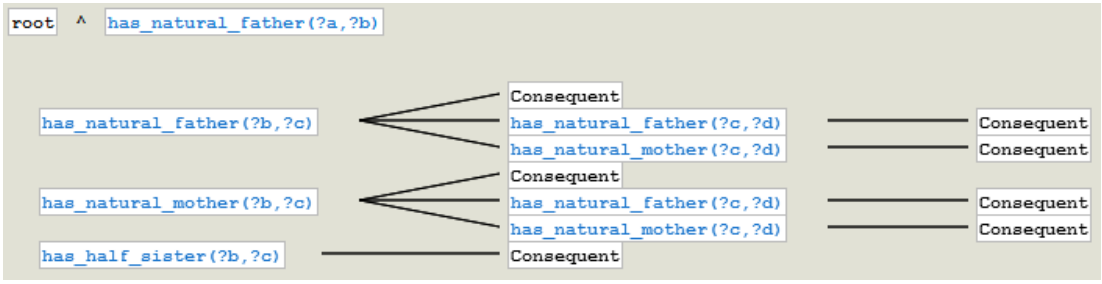


Figura 1.9 – Árvore de decisão: Dependência de variáveis.

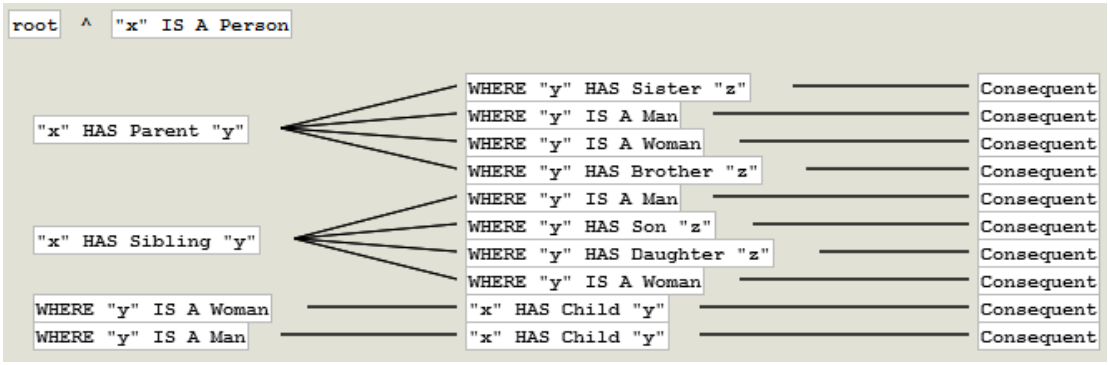


Figura 1.10 – Árvore de decisão: Paráfrases.

A funcionalidade das árvores de decisão é uma das principais contribuições do SWRL Editor, pois essa técnica nunca foi usada para regras SWRL e mostra-se como uma forma fácil de navegar em conjuntos de regras. Através dela é possível encontrar facilmente regras com antecedentes iguais ou semelhantes. Na Figura 1.11 é possível identificar que ocorrem antecedentes com mais de um consequente. De cima para baixo, na primeira ocorrência de um antecedente com dois consequentes, ao passar o mouse sobre os dois consequentes é possível perceber que eles são diferentes. Sendo assim, pode ser que não seja um erro, mas uma opção de projeto. Já na segunda ocorrência, quando são olhados os consequentes é possível identificar que eles são iguais (um erro). Como os algoritmos de árvore de decisão juntam átomos semelhantes é possível identificar os tipos de repetição que merecem atenção.

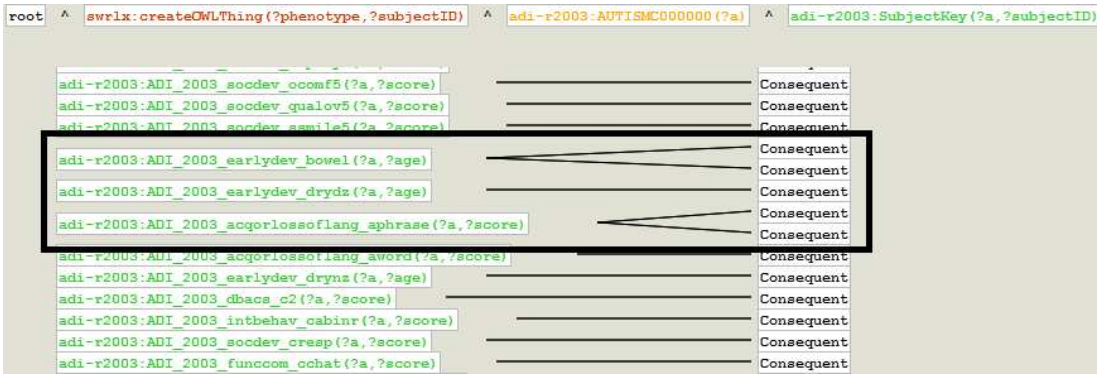


Figura 1.11 – Árvore de decisão: Identificação de regras com antecedentes iguais.

Muitas regras se diferenciam às vezes por um ou dois átomos. Então como ocorrem muitas repetições nos átomos das regras, a árvore de decisão pode ser usada para fazer o reaproveitamento de átomos. A Figura 1.12 mostra a tela da árvore de decisão, como pode ser visto na imagem, ao se clicar sobre um nodo da árvore são mostradas opções em um menu popup. Quando o nodo for um átomo do antecedente, o menu irá mostrar a opção para reaproveitar todos os átomos até a raiz, em uma nova regra. Já quando o nodo for um consequente, o menu mostrará a opção de editar essa regra.

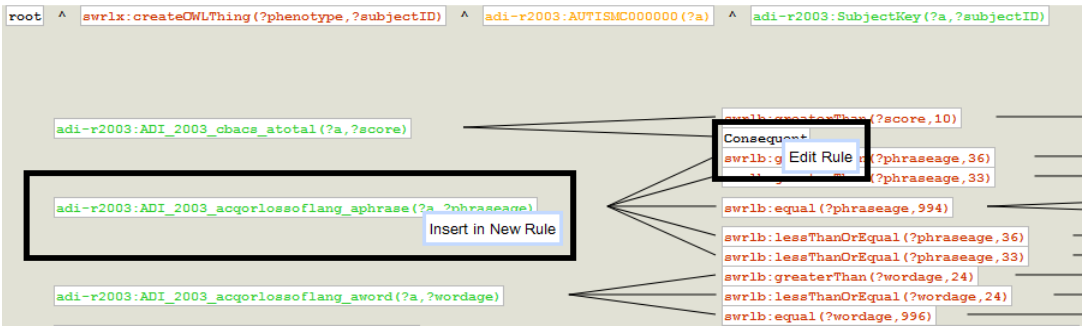


Figura 1.12 – Árvore de decisão: Botão direito sobre os nós.

1.4.2. Filtros

Como conjuntos de regras tendem a crescer, torna-se cada vez mais difícil localizar regras, então o SWRL Editor possui uma interface para filtrar regras SWRL. Nessa interface, o usuário pode usar os operadores lógicos AND, OR e NOT para criar uma expressão lógica para filtrar regras. O filtro é montado usando a interface do usuário na Figura 1.13, em que há um campo para cada operador lógico. Nesses campos, os valores de filtro são separados por espaços, expressões com espaços devem ser delimitadas por "" (exemplo "casa e carro").

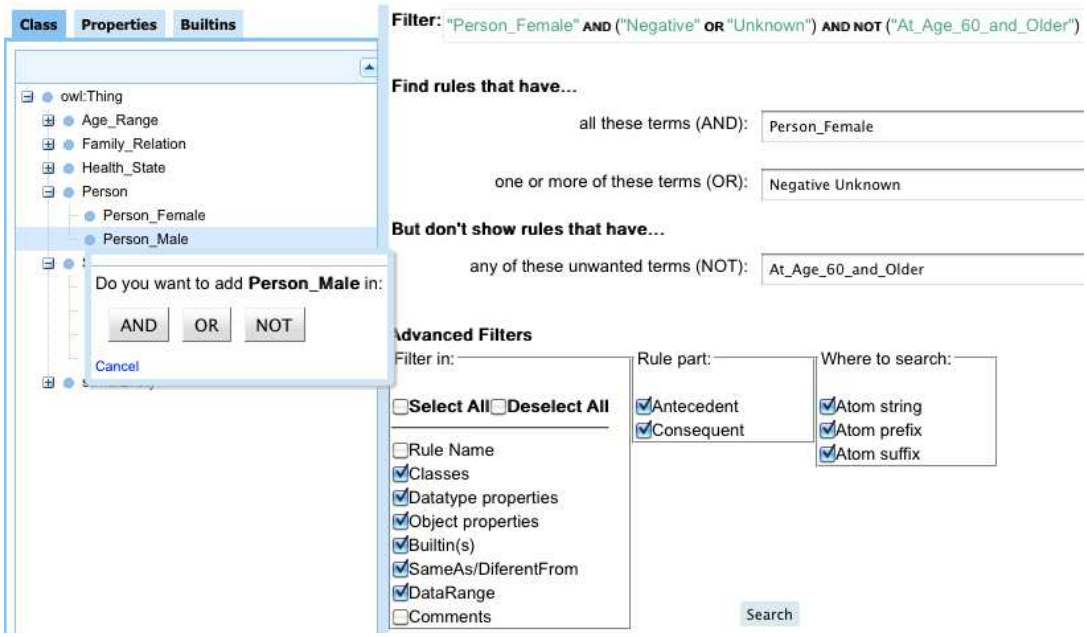


Figura 1.13 – Filtro das regras.

O filtro utiliza a expressão lógica montada (Figura 1.13, parte superior) para procurar automaticamente no rdf:ID e no rdfs:Label dos predicados de cada átomo. Além disso, a interface fornece opções avançadas para filtrar, sendo possível escolher partes de uma regra e tipos de átomos a serem filtrados. É possível escolher filtrar no Nome da regras, em *Class*, *Datatype properties*, *Object properties*, *Buitins*, *SameAs/DiferentFrom*, *DataRange*, *Comments* (Comentários dos predicados de cada átomo). Também é possível filtrar somente no antecedente ou no consequente. As opções para a filtragem ainda permitem os usuários realizem buscas nos prefixos, sufixos ou qualquer parte da string que representa cada átomo.

Na Figura 1.13, na lateral esquerda, encontram-se três guias (*Class*, *Properties* e *Bultins*) com os termos da ontologia. Essas guias são usadas para facilitar a inserção de termos da ontologia em uma busca. Como pode ser visto nessa figura, assim que o usuário clica na classe Person_Male a interface oferece a opção para escolher a qual operador lógico quer inserir.

Quando o usuário termina de montar o filtro, basta clicar no botão *Search* (localizado no lado direito parte inferior) que o filtro será executado e o seu resultado será mostrado na tela de visualização do SWRL Editor. Na Figura 1.14 é mostrada a tela de visualização com o filtro de regras modificado. Apenas as regras que obedecerem as condições desse filtro serão mostradas na interface e poderão ser usadas nas outras ferramentas.

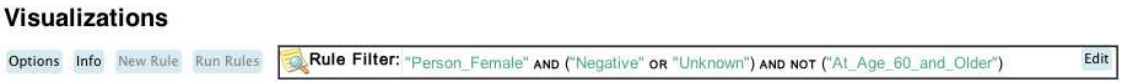


Figura 1.14 – Resultado do filtro de regras.

1.4.3. Opções

As configurações do SWRL Editor servem para tornar a ferramenta mais amigável e facilitar os processos de visualização e edição de regras. As configurações do SWRL

Editor foram divididas em três partes (Figura 1.15): *General*, *Composition* e *Visualization*. Só serão detalhadas as configurações gerais, as demais configurações são apenas detalhes de interface, já as configurações gerais servem para toda a ferramenta.

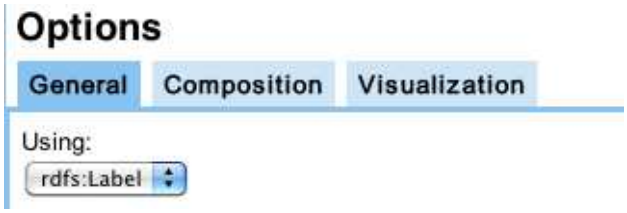


Figura 1.15 – Opções Gerais do SWRL Editor.

Na Figura 1.15 é apresentada a tela das configurações gerais, nela somente existe uma configuração para usar rdf:ID ou rdfs:Label. O rdf:ID é um identificador único para cada termo da ontologia, já o rdfs:Label é uma descrição para o termo da ontologia. Usando a ontologia do autismo [Hassanpour 2011], na Tabela 1.6 é mostrado alguns rdf:ID de termos da ontologia com o seu respectivo rdfs:Label. Fica evidente que os rdfs:Labels podem tornar mais fácil o entendimento e a edição das regras.

Tabela 1.6 – rdf:ID versus rdfs:Label

rdf:ID	rdfs:Label
adi-r2003:AUTISMC000000	*Autism Diagnostic Interview-Revised*
ados1:AUTISMC000004	*Autism Diagnostic Observation Schedule*
ados4:AUTISMC000001	*Autism Diagnostic Observation Schedule*
vabs_survey:AUTISMC000003	*Vineland Adaptive Behavior Scales*
Autism-core:AUTISMC100000	*Phenotype Record*
Autism-core:AUTISMC100002	*Quantitative value*
NIF-Invertigation:birnlex_2387	*Citation Record*
Autism-core:AUTISMC1000069	*Present*
Autism-core:AUTISMC1000070	*Repetitive stereotypical behavior*

Essa é uma importante técnica, pois permite ao usuário escolher a opção de visualizar e editar as regras usando rdfs:Labels. Porém, isso pode acarretar problemas na edição, pois um mesmo rdfs:Label pode estar em mais de um termo da ontologia. Um exemplo está na Tabela 1.6 em que os termos ados1:AUTISMC000004 e ados4:AUTISMC000001 compartilham um mesmo rdfs:Label. A ferramenta trata esse tipo de redundância, pois mesmo com a opção para usar rdfs:Label selecionada, o usuário poderá usar um rdf:ID durante a edição.

1.4.4. Composição

O processo de criação de regras é ativado no SWRL Editor quando o usuário seleciona a opção “New Rule” na visualização ou opta por editar uma regra específica. Na Figura 1.16 é apresentada a tela da ferramenta SWRL Editor no modo composição. Na lateral esquerda (A) dessa figura são exibidos os termos disponíveis na ontologia que podem ser selecionados pelos usuários (da mesma forma que nas Opções Seção 1.4.2). Quando

um deles é selecionado, uma janela é aberta com as opções de adicioná-lo como um átomo no antecedente ou consequente.

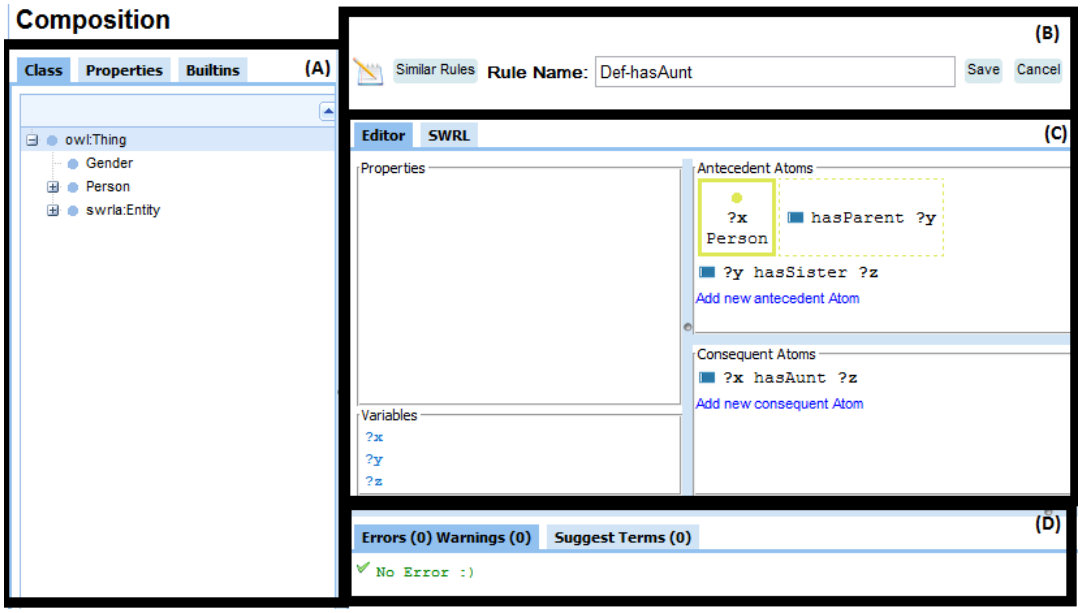


Figura 1.16 – SWRL Editor – Composição: (A) Termos disponíveis na ontologia; (B) Nome da regra e opção de salvar; (C) Editores da regra; (D) Avisos e sugestões de regras.

Na Figura 1.16 (B), é possível definir ou alterar o nome da regra. O botão salvar (*Save*) fica desativado enquanto são identificados erros na regra em desenvolvimento. Além disso, quando o desenvolvedor opta por voltar ao modo de visualização sem salvar (*Cancel*) a regra, a ferramenta exibe uma mensagem de confirmação. Ainda nessa figura em (C) é possível ver os dois modos de edição dos átomos de uma regra:

- *Editor* (ativo na Figura 1.16) – Formulário que utiliza a representação hierárquica para apresentar e organizar os átomos da regra. Divide o ambiente de desenvolvimento nas seguintes janelas de trabalho:
    - Caixa de propriedades (*Properties*), contendo as propriedades do átomo que o desenvolvedor selecionar;
    - Lista de variáveis (*Variables*) utilizadas na regra;
    - Átomos do antecedente (*Antecedent Atoms*), contendo os átomos presentes na parte antecedente da regra. Os átomos são apresentados conforme a representação hierárquica e podem ser alterados na caixa de propriedades;
    - Átomos do consequente (*Consequent Atoms*), contendo os átomos presentes na parte consequente da regra. Os átomos são apresentados conforme a representação hierárquica e podem ser alterados na caixa de propriedades;
- SWRL – Editor SWRL com *Highlight*, no qual o desenvolvedor escreve a regra utilizando a sintaxe SWRL. Os átomos e argumentos são apresentados conforme o padrão de cores adotado na visualização do SWRL com *Highlight*;

Na parte inferior da Figura 1.16 (D), são apresentadas as seguintes abas:

- Problemas (*Errors* e *Warnings*) – contem as possíveis notificações de erros e avisos para o desenvolvedor da regra;
- Termos sugeridos (*Suggest Terms*) – conterá uma lista com os termos sugeridos;

Um ponto importante, já citado, é o da ferramenta permitir que o usuário utilize os rdfs:Labels (mais fácies que os identificadores únicos rdf:ID). Porém para que não ocorram redundâncias são apresentados erros como o da Figura 1.17. Nesta Figura é mostrada a seguinte mensagem de erro: "The predicate label **Autism Diagnostic Observation Schedule** represents more than one ID: **ados4:AUTISMC000001**, **ados1:AUTISMC000004**". O usuário poderá então usar um dos dois rdf:IDs para representar o elemento e assim evitar o erro.

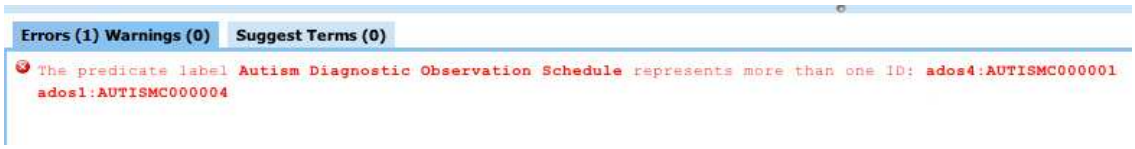


Figura 1.17 – Mensagem de erro de redundância de rdfs:Labels.

1.5. Construção de uma ontologia com regras SWRL

Essa seção descreve a criação de uma ontologia de relações familiares. A idéia de utilizar esse exemplo reside no fato de que esse domínio é de fácil entendimento por todos e pode gerar bons exemplos. Vamos dividir essa demonstração em duas etapas:

- Ontologia – Definir os termos da ontologia e suas relações:
  - Classes (Conceitos) O objetivo criar classes e subclasses, modificando a hierarquia de classes da ontologia;
  - Propriedades (Relações) As propriedades representam relacionamentos entre dois indivíduos;
- SWRL:
  - SWRL Regras de inferências;
  - Indivíduos da ontologia representam objetos no domínio de interesse (ou indivíduos de uma classe);
  - Execução – Demonstração dos resultados gerados após a execução de um conjunto de regras;

1.5.1. Construção da ontologia

Toda ontologia contém uma classe chamada owl:Thing. Conforme mencionado, as classes OWL são interpretadas como conjuntos de indivíduos (ou conjunto de objetos). A classe owl:Thing é a classe que representa o conjunto que contém todas as classes e os indivíduos, uma vez que todas as classes são subclasses de owl:Thing. Para exemplificar será apresentada nessa seção a criação de uma ontologia que representem relações parentesco entre indivíduos.

Na Figura 1.18 são definas as subclasses de owl:Thing que farão parte da ontologia de exemplo. Já nas Figuras 1.19 e 1.20 são definidas as outras classes que são subclasses de Man e Woman respectivamente.

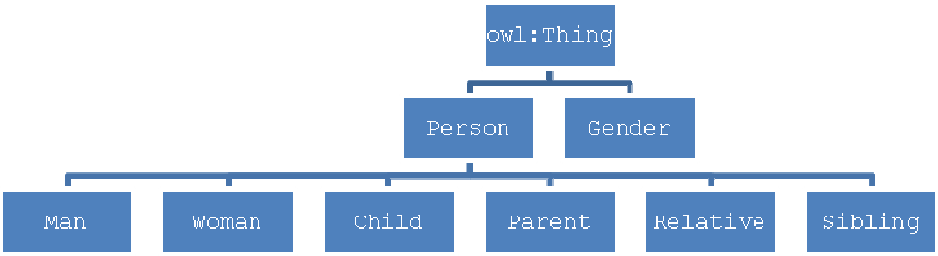


Figura 1.18 – Definição da ontologia: owl:Thing.

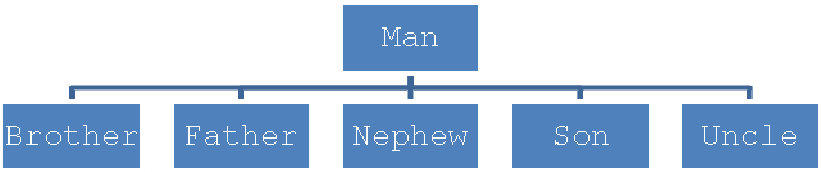


Figura 1.19 – Definição da ontologia: Man.

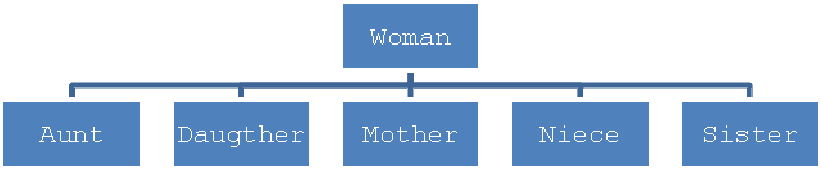


Figura 1.20 – Definição da ontologia: Woman.

Como apresentado anteriormente, esses são identificadores únicos (rdf:ID) da ontologia, ou seja, só podem existir uma vez. Porém muitas vezes, quando não existe um consenso sobre o nome ideal de um termo, é adotado um código para ser o rdf:ID e são usados rdfs:Labels para nome. Outra boa prática é usar o termo em inglês (ex: Person) e colocar rdfs:Labels para outras linguagens (seguindo o ex: Pessoa).

Com todos os termos definidos para a ontologia vamos demonstrar a criação de uma classe no Web-Protégé. Para pode editar é necessário estar logado no Web-Protégé. O Web-Protégé disponibiliza uma guia chamada “Classes” que edita as classes da ontologia. Essa guia possui por default 3 parte:

- *Classes* (Figura 1.21A): Usada exibir a hierarquia de classes;
- *Properties* (Figura 1.21B): Exibe as propriedades de uma classe;
- *Asserted Conditions* (Figura 1.21 C): Contém as restrições de uma classe;

Na Figura 1.21, para criar a classe Person é necessário selecionar a classe owl:Thing e clicar em *Create* (Figura 1.21A), irá aparecer um popup para informar o nome da nova classe. Após informar basta clicar em OK e a classe já fica disponível na hierarquia. Ao selecionar a nova classe, ela não possuirá nenhuma propriedade e possuirá uma restrição (a relação com owl:Thing). Ainda na Figura 1.21(B) foram criadas 3 propriedades de anotação para essa nova classe:

- Dois rdfs:Labels: Não tem limite de *labels*, no exemplo foi disponibilizada a tradução do termo em português e espanhol;
- Um rdfs:Comment: os comentário são interessantes para ter uma explicação do termo da ontologia;

Já nas restrições Figura 1.21(C) existem somente dois tipos que podem ser definidos:

- *Necessary*: Essa restrição pode ser lida assim: “se alguma coisa é um membro da classe então é necessário que preencha essas condições”. Na Figura 1.21 (C) é informado que para pertencer a Person o individuo deve estar em owl:Thing.
- *Necessary & Sufficient*: Essa restrição agrega mais esse fato: “se alguma coisa preenche essas condições então deve ser um membro dessa classe”. Ex: Qualquer individuo que pertença a Man ou Womem é uma Person.

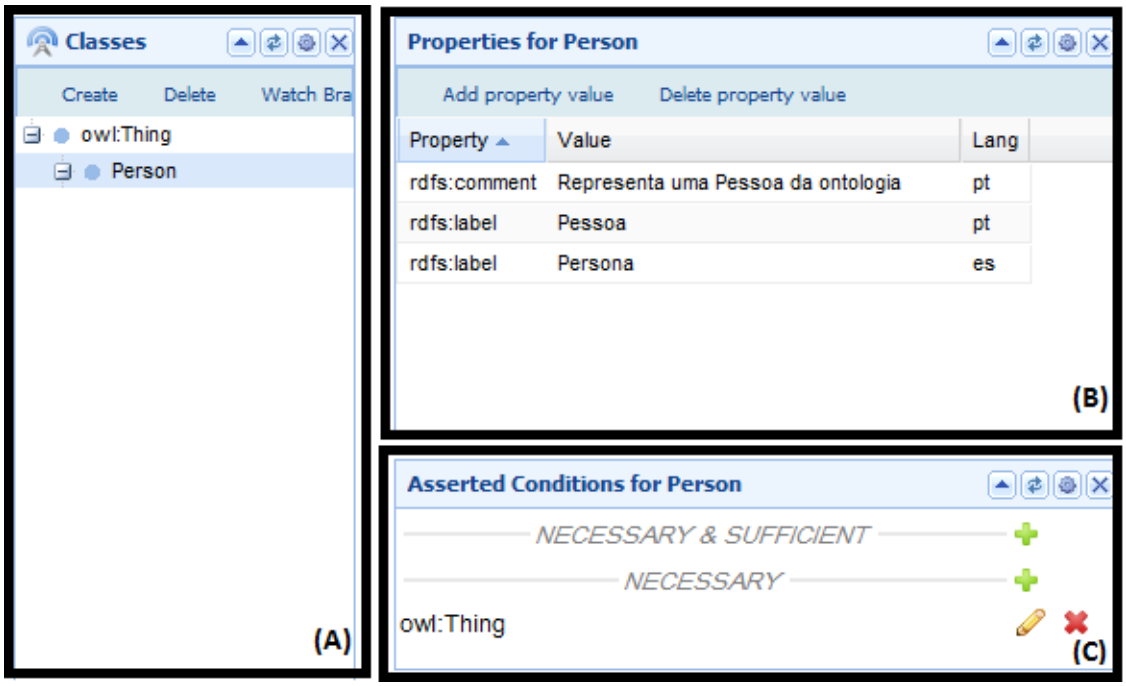


Figura 1.21 – Web-Protégé Interface para edição de classes: (A) Hierarquia de Classes; (B) Propriedades de uma classe; (C) Restrições de uma classe.

Se analisarem as Figuras 1.18, 1.19 e 1.20 pode-se perceber que, por exemplo, a classe Son criada para Man, também pertence a classe Child, porém como são identificadores únicos não podem ser inseridos duas vezes na hierarquia de classes. Então a solução é inserir uma restrição em *Necessary & Sufficient* para esses termos. No caso de Son ele já possui essa restrição para Man, então apenas basta inserir para Child também. Na tabela abaixo são apresentados os termos que necessitam de restrições:

Tabela 1.7 – Lista de restrições para estabelecer a hierarquia de classes

Classe	Tipo de Restrição	Restrição
Brother	*Necessary & Sufficient*	Sibling
Father	*Necessary & Sufficient*	Parent
Nephew	*Necessary*	Relative
Son	*Necessary & Sufficient*	Child
Uncle	*Necessary*	Relative
Aunt	*Necessary*	Relative

Classe	Tipo de Restrição	Restrição
Daugther	*Necessary & Sufficient*	Child
Mother	*Necessary & Sufficient*	Parent
Niece	*Necessary*	Relative
Sister	*Necessary & Sufficient*	Sibling

Com isso é possível montar toda a hierarquia de classes chegando ao que é apresentado na Figura 1.22.

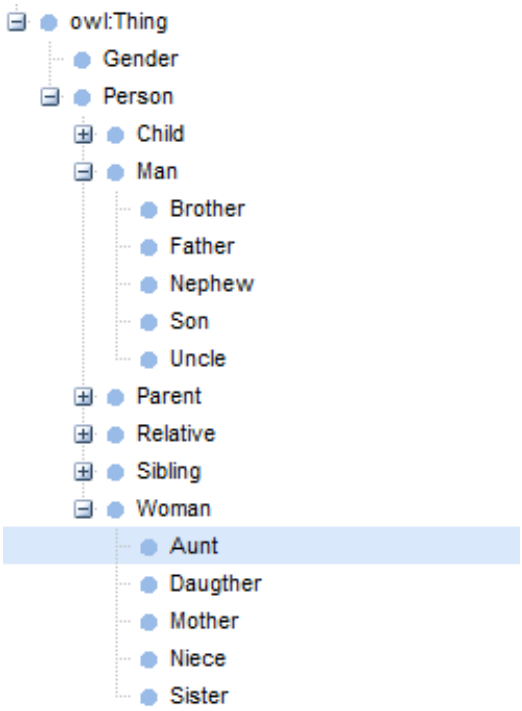


Figura 1.22 – Hierarquia Inicial de Classes.

O ideal agora é preencher todos os rdfs:Label da ontologia e disponibilizar, por exemplo, os termos da ontologia em português, porém não será apresentado neste capítulo, por não ser obrigatório. Antes de preencher as demais restrições é necessário criar as *Properties*. O Web-Protégé disponibiliza uma guia chamada “Properties” (Figura 1.23). Por default as *Properties* precisam de duas propriedades de anotação preenchidas:

- rdfs:domain: Na Figura 1.23 está selecionada a propriedade hasAunt, o domínio dessa propriedade é qualquer pessoa (Person), ou seja, qualquer pessoa pode ter uma tia;
- rdfs:range: O range já é quem pode ser classificada para aquela classe. Ex Figura 1.23: só pode ser uma tia quem for da classe Woman.

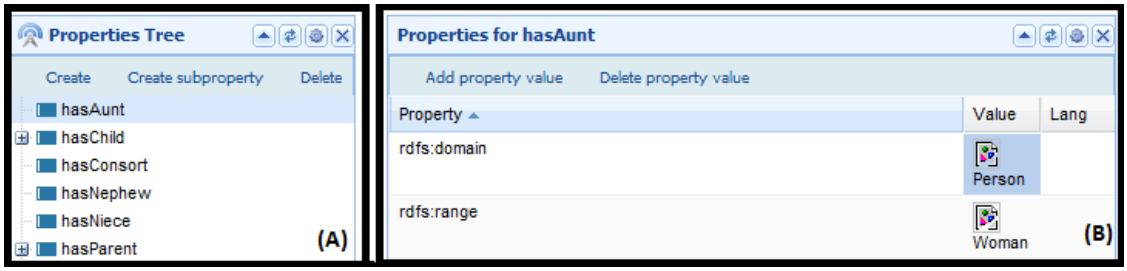


Figura 1.23 – Web-Protégé Interface para edição de propriedades: (A) Hierarquia de propriedades; (B) Propriedades de uma propriedade.

Na tabela abaixo seguem todas as propriedades da ontologia com os seus respectivos rdfs:domain e rdfs:range. Para inserir uma propriedade na Ontologia basta clicar em *Create* em (Figura 1.23 – A) e irá aparecer um popup para informar o nome da nova propriedade. Após informar basta clicar em OK e a propriedade já fica disponível. Caso queira inserir uma subpropriedade basta selecionar a propriedade pai e clicar em *Create subproperty*.

Tabela 1.8 – Lista de propriedades da ontologia

Propriedade pai	Propriedade a criar	rdfs:domain	rdfs:range
-	hasAunt	Person	Woman
-	hasChild	Person	Person
hasChild	hasDaughter	Person	Woman
hasChild	hasSon	Person	Man
-	hasConsort	Person	Person
-	hasNephew	Person	Man
-	hasNiece	Person	Woman
-	hasParent	Person	Person
hasParent	hasFather	Person	Man
hasParent	hasMother	Person	Woman
-	hasSex	Person	Gender
-	hasSibling	Person	Person
hasSibling	hasBrother	Person	Man
hasSibling	hasSister	Person	Woman
-	hasUncle	Person	Man

Ainda antes de inserir as demais restrições nas classes é necessário instanciar dois indivíduos (Female e Male) para classe Gender na guia "*Individuals*" do Web-Protégé (Figura 1.24). Para inserir um indivíduo na Ontologia é necessário primeiramente selecionar a classe referente (Figura 1.24 A). O próximo passo é clicar em *Create* em (Figura 1.24 – B) e irá aparecer um popup para informar o nome do novo indivíduo. Após informar basta clicar em OK e o indivíduo já fica disponível.

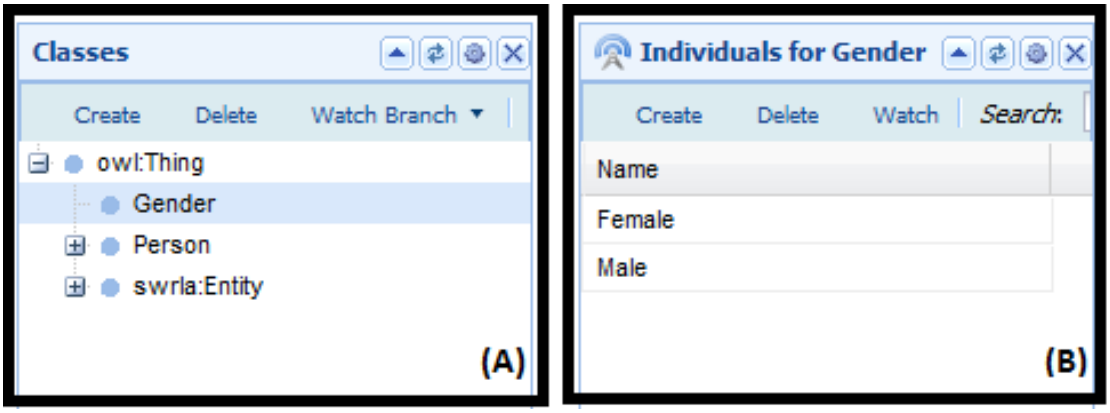


Figura 1.24 – Web-Protégé Interface para edição de indivíduos: (A) Hierarquia de classes; (B) Indivíduos de uma classe.

Após isso é possível inserir as restrições que envolvem propriedades ou indivíduos com em (Figura 1.21 C). Essas restrições podem ser vistas na Tabela abaixo.

Tabela 1.9 – Lista das demais restrições

Classe	Tipo de Restrição	Restrição
Gender	*Necessary & Sufficient*	{Female Male}
Person	*Necessary & Sufficient*	Man or Woman
Child	*Necessary & Sufficient*	Hasparent min 1
Man	*Necessary & Sufficient*	hasSex has Male
Nephew	*Necessary & Sufficient*	(hasUncle min 1) or (hasAunt min 1)
Uncle	*Necessary & Sufficient*	(hasNephew min 1) or (hasNiece min 1)
Parent	*Necessary & Sufficient*	hasChild min 1
Relative	*Necessary & Sufficient*	Child or Parent or Aunt or Nephew or Niece or Uncle or Sibling
Aunt	*Necessary & Sufficient*	(hasNephew min 1) or (hasNiece min 1)
Niece	*Necessary & Sufficient*	(hasUncle min 1) or (hasAunt min 1)
Sibling	*Necessary & Sufficient*	hasSibling min 1
Woman	*Necessary & Sufficient*	hasSex has Female

1.5.2. Construção e execução das regras SWRL

Nesta seção são apresentadas e explicadas 12 regras SWRL, as regras estão disponíveis nas Tabelas 1.10 e 1.11 abaixo:

Tabela 1.10 – Lista das demais restrições

Nº	Nome	Regra

Nº	Nome	Regra
1	Def-hasAunt	Person(?x) ∧ hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)
2	Def-hasBrother	Person(?x) ∧ hasSibling(?x, ?y) ∧ Man(?y) → hasBrother(?x, ?y)
3	Def-hasDaughter	Person(?x) ∧ hasChild(?x, ?y) ∧ Woman(?y) → hasDaughter(?x, ?y)
4	Def-hasFather	Person(?x) ∧ hasParent(?x, ?y) ∧ Man(?y) → hasFather(?x, ?y)
5	Def-hasMother	Person(?x) ∧ hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)
6	Def-hasNephew	Person(?x) ∧ hasSibling(?x, ?y) ∧ hasSon(?y, ?z) → hasNephew(?x, ?z)
7	Def-hasNiece	Person(?x) ∧ hasSibling(?x, ?y) ∧ hasDaughter(?y, ?z) → hasNiece(?x, ?z)
8	Def-hasParent	Person(?y) ∧ hasConsort(?y, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)
9	Def-hasSibling	Person(?y) ∧ hasChild(?y, ?x) ∧ hasChild(?y, ?z) ∧ differentFrom(?x, ?z) → hasSibling(?x, ?z)

Nº	Nome	Regra
10	Def-hasSister	Person(?x) ∧ hasSibling(?x, ?y) ∧ Woman(?y) → hasSister(?x, ?y)
11	Def-hasSon	Person(?x) ∧ hasChild(?x, ?y) ∧ Man(?y) → hasSon(?x, ?y)
12	Def-hasUncle	Person(?x) ∧ hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)

Tabela 1.11 – Explicação das Regras em forma de texto

Nº	Explicação da Regra
1	Uma regra em que uma Pessoa X que tem um Pai/Mãe Y e que Y tem uma Irmã Z pode-se concluir que X tem uma Tia Z.
2	Uma regra em que uma Pessoa X que tem um Irmão/Irmã Y e Y seja Homem pode-se concluir que X tem um irmão Y.
3	Uma regra em que uma Pessoa X que tem um Filho/Filha Y e Y seja Mulher pode-se concluir que X tem uma filha Y.
4	Uma regra em que uma Pessoa X que tem um Pai/Mãe Y e Y seja homem pode-se concluir que X tem um Pai Y.
5	Uma regra em que uma Pessoa X que tem um Pai/Mãe Y e Y seja mulher pode-se concluir que X tem uma Mãe Y.
6	Uma regra em que uma Pessoa X que tem um Irmão/Irmã Y e que Y tem um filho Z pode-se concluir que X tem um sobrinho Z.
7	Uma regra em que uma Pessoa X que tem um Irmão/Irmã Y e que Y tem uma filha Z pode-se concluir que X tem uma sobrinha Z.
8	Uma regra em que uma Pessoa Y que tem um Cônjuge Z e que Y é Pai/Mãe de X pode-se concluir que Z também é Pai/Mãe X.
9	Uma regra em que uma Pessoa Y que tem um Filho/Filha X e que tem outro Filho/Filha Z pode-se concluir X tem um Irmão/Irmã Z.
10	Uma regra em que uma Pessoa X que tem um Irmão/Irmã Y e Y seja Mulher pode-se concluir que X tem uma irmã Y.
11	Uma regra em que uma Pessoa X que tem um Filho/Filha Y e Y seja Homem pode-se concluir que X tem um filho Y.
12	Uma regra em que uma Pessoa X que tem um Pai/Mãe Y e que Y tem um Irmão Z pode-se concluir que X tem um Tio Z.

Com as regras definidas, vamos criar e executar a inferência de uma regra. Para criar uma nova regra no SWRL Editor basta clicar no botão *New Rule* da tela de visualização (Figura 1.4). Após clicar, a ferramenta acessa a tela de composição (Figura 1.25). Para adicionar átomos a uma regra é possível a partir da navegação nas classes, propriedades e built-ins. Quando achar o termo, basta clicar sobre ele que o SWRL Editor exibirá um popup que serve para selecionar se o predicado será inserido no antecedente ou no consequente (Figura 1.25 A). Assim que for selecionada a opção do popup, o novo predicado é carregado para edição (Figura 1.25 B).

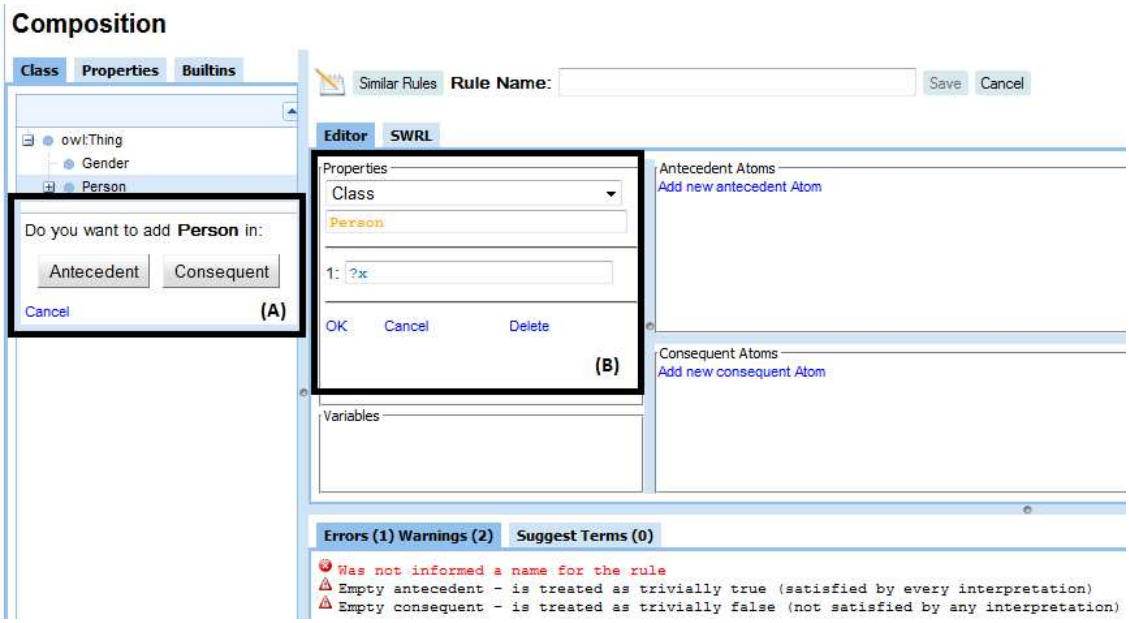


Figura 1.25 – SWRL Editor Tela de composição para nova regra: (A) Selecionar entre antecedente e consequente; (B) Átomo sendo editado.

Após informar o nome e todos os átomos da regra 1 se obtém a Figura 1.26. Na sequência é só necessário salvar a regra em "Save" e assim foi criada a primeira regra SWRL.

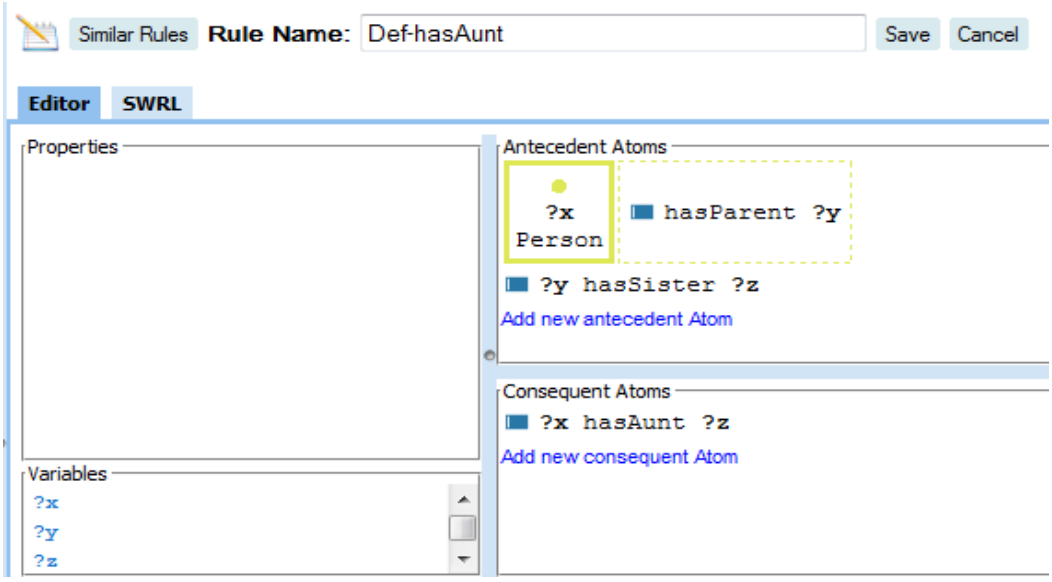


Figura 1.26 – SWRL Editor Tela de composição com uma regra preenchida.

A Regra 1 criada tem função de criar a relação de uma Pessoa ter uma Tia. Para testar regra 1 é necessário criar os seguintes indivíduos na ontologia:

- Criar o indivíduo "Filho" na classe Man.
    - Atribuir "Pai" a propriedade hasFather de "Filho";
- Criar o indivíduo "Pai" na classe Man.

    - Atribuir “Filho” a propriedade hasSon de “Pai”;
    - Atribuir “Tia” a propriedade hasSister de “Pai”;
- Criar o indivíduo “Tia” na classe Woman.
    - Atribuir “Pai” a propriedade hasBrother de “Tia”;

Agora ao executar as regras (*Run Rules* na Figura 1.4 B) será preenchida com “Tia” a propriedade hasAunt no “Filho”

1.6. Conclusão

A Web Semântica é uma maneira de explorar a associação de significados explícitos aos conteúdos de documentos presentes na Web, para que esses possam ser processados diretamente ou indiretamente por máquinas [Berners-Lee and Fischetti 2008]. Para possibilitar esse processamento, os computadores necessitam ter acesso a coleções estruturadas de informações (dados e metadados) e a conjuntos de regras de inferência sobre esses conteúdos (que ajudem no processo de dedução automática) para que seja possível o raciocínio automatizado sobre os mesmos [Berners-Lee et all 2001).

A linguagem recomendada pelo W3C para representação e compartilhamento de ontologias na Web Semântica é o OWL. Essa linguagem foi projetada para aplicações que necessitam processar o conteúdo da informação em vez de apenas apresentar informações em texto [McGuinness and Harmelen 2004]. Infelizmente a expressividade de OWL nem sempre é suficiente para modelar todos os tipos de problemas. Especialmente OWL não tem suporte a cláusulas no formato de Horn (se a → b). Para suprir essa deficiência, a SWRL (Semantic Web Rule Language) foi criada. A linguagem SWRL é muito útil para desenvolvedores de ontologias, pois faz inferências de novos conhecimentos sobre indivíduos da ontologia (OWL).

Este capítulo apresentou conceitos ligados a Web Semântica, especialmente os ligados a regras SWRL e o SWRL Editor, um editor de regras SWRL encontrado no Web-Protégé. Por último, foi mostrado um guia passo a passo para a criação de uma ontologia, com regras, que descreve relações de parentescos entre indivíduos.

1.7. Agradecimentos

Dilvan A. Moreira agradece o apoio recebido da Fapesp para apresentação do trabalho.

1.8. Referências

Ahmedi, L., Abazi-Bexheti, L. and Kadriu, A. (2011). “A Uniform Semantic Web Framework for Co-authorship Networks”. In: IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC), p. 958-965. doi: 10.1109/DASC.2011.159.

Almeida, M. B. and Bax, M. P. (2003). Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. Ci. Inf., Brasília, v. 32, n. 3. doi: 10.1590/S0100-19652003000300001.

Barder, F. and Nutt, W. (2003). Basic Description Logics. In: BARDER, F. et al. The Description Logic Handbook. Cambridge University Press. Capítulo 2, p. 43-90.

Berners-Lee, T. and Fischetti, M. (2008). Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web. HarperSanFrancisco. ISBN: 978-0062515872.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001). The Semantic Web. Scientific American, p. 34–43. Disponível em: http://www.scientificamerican.com/article.cfm?id=the-semantic-web, acesso em Jan. 2012.

Brickley, D., Guha, R. V. and Mcbride, B. (Editors). (2004). RDF Vocabulary Description Language 1.0: RDF Schema. W3C. Disponível em: http://www.w3.org/TR/rdf-schema/, acesso em Jan. 2010.

Chandrasekaran, B., Josephson, J. R. and Benjamins, V. R. (1999). What Are Ontologies, and Why Do We Need Them?. IEEE Intelligent Systems, Piscataway, NJ, USA, v. 14, n. 1, p. 20-26. doi: 10.1109/5254.747902.

Corazzon, R. (2010). Theory and History of Ontology. A Resource Guide for Philosophers. Disponível em: http://www.formalontology.it/, acesso em Jan. 2010.

Diniz, V. and Cecconi, C. (2008). Padrões Web: Passado, presente e futuro, V Conferência Latino Americana de Software Livre. Disponível em: http://www.w3c.br/palestras/internet-web-jun-jul-2008/internetWeb_Out08.html, acesso em Dez. 2009.

Feigenbaum, L., Herman, I., Hongsermeier, R, Neumann, E. and Stephens, S. (2007). The Semantic Web in action, Scientific American, p. 64-71. Disponível em: http://www.ncbi.nlm.nih.gov/pubmed/18237102, acesso em Jan. 2010.

Fensel, D., Van Harmelen, F., Horrocks, I., Mcguinness, D. L. and Patel-Schneider, P. F. (2001). OIL: An Ontology Infrastructure for the Semantic Web. IEEE Intelligent Systems, v. 16, n. 2, p. 38-45. doi: 10.1109/5254.920598.

Golbreich, C., Horridge, M., Horrocks, I., Motik, B. and Shearer, R. (2007). OBO and OWL: leveraging semantic web technologies for the life sciences. In: Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference. Busan, Korea, p. 169-182.

Grau, B. C., Horrocks, I., Motik, B., Parsia, B., Patel-Scjmeoder, P. and Sattler, U. (2008). OWL 2: The next step for OWL. Journal of Web Semantics: Science, Services and Agents on the World Wide Web, Amsterdam, Netherlands, v. 6, n. 4, p. 309–322. doi: 10.1016/j.websem.2008.05.001.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. Knowl. Acquis. London, UK, v. 5, n. 2, p. 199-220. doi: 10.1006/knac.1993.1008.

Guarino, N. and Giaretta, P. (1995). Ontologies and Knowledge Bases: Towards a Terminological Clarification. Journal Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. v. 1, n. 9, p. 25-32.

Hassanpour, S., O'connor, M. J. and Das, A. K. (2009). Exploration of SWRL Rule Bases through Visualization, Paraphrasing, and Categorization of Rules. In: Proceedings of the 2009 International Symposium on Rule Interchange and Applications (RuleML 2009), Las Vegas, Nevada, p. 246–261. doi: 10.1007/978-3-642-04985-9_23.

Hassanpour, S., O'connor, M. J. and Das, A. K. (2010). Visualizing Logical Dependencies in SWRL Rule Bases. The International RuleML Symposium on Rule Interchange and Applications, Washington, DC, p. 259-272.

Hassanpour, S., O'Connor, M. J. and Das, A. K. (2011). "Evaluation of Semantic-Based Information Retrieval Methods in the Autism Phenotype Domain". In: AMIA Annual Symposium.

Heflin, J. (2004). W3C Recommendation: OWL Web Ontology Language Use Cases and Requirements. Disponível em: www.w3.org/TR/2004/REC-webont-req-20040210. acesso Fev. 2012.

Horridge, M., Knublauch, H., Rector, A., Stevens, R. and Wroe, C. (2004). A practical guide to building OWL ontologies using the protégé-OWL plugin and CO-ODE tools edition 1.0. University Of Manchester. Disponível em: http://www.co-ode.org/resources/, acesso em Out. 2009.

Horrocks, I, Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B. and Dean, M. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C. Disponível em: http://www.w3.org/Submission/SWRL/, acesso em Jan. 2010.

Klyne, G., Carrol, J. J. and Mcbride, B. (Editors). (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C. Disponível em: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, acesso em Set. 2009.

Lassila, O. and Swick, R. (Editors). (2004). Resource Description Framework (RDF) model and syntax specification. W3C. Disponível em: http://www.w3.org/TR/REC-rdf-syntax/, acesso em Out. 2009.

Levy, M., O'Connor, M. J. and Rubin, D. L. (2009). "Semantic Reasoning with Image Annotations for Tumor Assessment". In: AMIA Annual Symposium.

McGuinness, D. L. and Harmelen, F. (Editors). (2004). OWL Web Ontology Language Overview. W3C. Disponível em: http://www.w3.org/TR/2004/REC-owl-features-20040210/, acesso em Jun. 2009.

Noy, N. F. and McGuinness, D. L. (2001). Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05.

O'connor, M., Knublauch, H., Tu, S., Grosof, B., Dean, M., Grosso, W. and Musen, M. (2005). Supporting Rule System Interoperability on the Semantic Web with SWRL Fourth International Semantic Web Conference (ISWC2005), Galway, Ireland. Disponível em: http://bmir.stanford.edu/file_asset/index.php/1157/BMIR-2005-1080.pdf, acesso em Maio de 2012.

Orlando, J. P. (2012). Usando aplicações ricas para internet na criação de um ambiente para visualização e edição de regras SWRL. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos. Recuperado em 2012-08-26, de http://www.teses.usp.br/teses/disponiveis/55/55134/tde-25072012-101810/

Ouellet, R. and Ogbuji, U. (2002). Introduction to DAML: Part I. Disponível em: http://www.xml.com/pub/a/2002/01/30/daml1.html, acesso em Jan. 2010.

Rossello-Busquet, A., Brewka, L. J., Soler, J. and Dittmann, L. (2011). “OWL Ontologies and SWRL Rules Applied to Energy Management”. In: International Conference on Computer Modelling and Simulation (UKSim), p. 446-450. doi: 10.1109/UKSIM.2011.91.

Sadoun, D., Dubois, C., Ghamri-Doudane, Y. and Grau, B. (2011). “An Ontology for the Conceptualization of an Intelligent Environment and Its Operation”. In: Mexican International Conference on Artificial Intelligence (MICAI), p. 16-22. doi: 10.1109/MICAI.2011.32.

Seo, S., Kwon, A., Kang, J. and Hong, J. W. (2011). “OSLAM: Towards ontology-based SLA management for IPTV”. In: IEEE International Symposium on Integrated Network Management (IM), p. 1228-1234. doi: 10.1109/INM.2011.5990570.

Shadbolt, N., Hall, W. and Berners-Lee, T. (2006). The Semantic Web Revisited. IEEE Intelligent Systems, v. 21, n. 3, p. 96-101. doi: 10.1109/MIS.2006.62.

Silva, A. R. (2012). Aprimorando a visualização e composição de regras SWRL na Web. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos. Recuperado em 2012-08-26, de http://www.teses.usp.br/teses/disponiveis/55/55134/tde-27022012-142801/

Smith, B. and Welty, C. (2001). Ontology: Towards a New Synthesis. In: FOIS’01: Proceedings of the international conference on Formal Ontology in Information Systems, Ogunquit, Maine, USA, p. 3-9. doi: 10.1145/505168.505201.

Smith, M. K., Welty, C. and McGuinness, D. L. (Editors). (2004). OWL Web Ontology Language Guide. W3C. Disponível em: http://www.w3.org/TR/owl-guide/, acesso em Nov. 2008.

Staab, S., Maedche, A. and Handschuh, S. (2001). An annotation framework for the semantic web. In: Proceedings of the First Workshop on Multimedia Annotation, Tokyo, Japan, p. 30-31. doi: 10.1.1.25.910.

Studer, R., Benjamins, R. and Fensel, D. (1998). Knowledge Engineering: Principles and Methods. IEEE Transactions on Data and Knowledge Engineering, v. 25(1-2), p. 161-197. doi: 10.1.1.41.1007.

Su, X. and Ilebrekke, L. (2002). A Comparative Study of Ontology Languages and Tools. In: CAiSE'02:Proceeding of the 14th Conference on Advanced Information Systems Engineering, Toronto, Canada, v. 2348, p. 761-765. doi: 10.1007/3-540-47961-9_62.

SWRLLanguage. (2012). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Disponível em: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ, acesso em Maio 2012.

Tudorache, T., Vendetti, J. and Noy, N. F. (2008). Web-Protégé: A Lightweight OWL Ontology Editor for the Web. In: OWLED 2008: OWL: Experiences and Directions, Karlsruhe, Germany. doi: 10.1.1.142.8568.

Uschold, M. and Grüninger, M. (1996). Ontologies: principles, methods and applications. The Knowledge Engineering Review, v. 11, n. 2, p. 93-155. doi: 10.1017/S0269888900007797.

Uschold, M. and Grüninger, M. (2004). Ontologies and semantics for seamless connectivity. SIGMOD Rec, NY, USA v. 3, n. 4, p. 58-64. doi: 10.1145/1041410.1041420.

Vesin, B., Ivanovic, M., Klasnja-Milicevic, A. and Budimac, Z. (2011). “Rule-based reasoning for altering pattern navigation in Programming Tutoring System”. In: International Conference on System Theory, Control, and Computing (ICSTCC), p. 1-6.

W3C OWL Working Group. (2009). OWL 2 Web Ontology Language Document Overview. W3C. Disponível em: http://www.w3.org/TR/owl2-overview/, acesso em Dez. 2009.

Wusheng, W., Weiping, L., Zhonghai, W., Weijie, C. and Tong, M. (2011). “An Ontology-Based Context Model for Building Context-Aware Services”. In: International Conference on Intelligent Systems, Modelling and Simulation (ISMS), p. 296-299. doi: 10.1109/ISMS.2011.52.

Zacharias, V. (2008). Development and verification of rule based systems – a survey of developers. In: Rule Representation, Interchange and Reasoning on the Web: International Symposium, Orlando, Florida, USA, v. 5321, p. 6-16. doi:10.1007/978-3-540-88808-6_4.

Capítulo

2

Software as a Service: Desenvolvendo Aplicações Multi-tenancy com Alto Grau de Reuso

Josino Rodrigues Neto, Vinicius Cardoso Garcia, Andrêza Leite de Alencar, Júlio César Damasceno, Rodrigo Elia Assad e Fernando Trinta

Abstract

Software as a Service (SaaS) represents a new paradigm and business model with witch companies don't need buy and maintain their own IT infrastructure. Instead, they acquire a software as service of third party, obtaining significant benefits, especially regarding the reduction of infrastructure maintenance costs. The aim of this work is to present the main concepts related to multi-tenancy architecture, and an approach to implementation of Software as a Service. During this work the key technologies will be presented associated with the subject and a practical example of implementing a multi-tenancy application using Grails Framework and reusable components.

Resumo

Software como serviço (SaaS) representa um novo paradigma e um modelo de negócios onde as empresas não precisam comprar e manter sua própria infraestrutura de TI. Ao invés disso, elas adquirem um serviço de software de terceiros, obtendo assim consideráveis benefícios, principalmente no que tange a redução de custos de manutenção dessa infraestrutura. O objetivo desse minicurso é apresentar os principais conceitos relacionados à arquitetura multi-tenancy, uma das abordagem para implementação de Software como Serviço. Durante esse trabalho serão apresentadas as principais tecnologias associadas ao assunto e um exemplo prático da implementação de um aplicativo multi-tenancy utilizando o framework Grails e componentes reutilizáveis.

2.1. Introdução

Em 1969, Leonard Keinrock, um dos cientistas chefe da ARPANET, precursora da internet, disse: “A partir de agora, redes de computadores estão ainda na sua infância, mas a medida que crescem e tornam-se sofisticadas, nós iremos provavelmente ver a disseminação do ‘computador utilitário’ que, como telefones e energia elétrica, irão servir casas e escritórios por todo país” [30]. Essa visão de computação utilitária baseada no modelo de fornecimento de serviços antecipa a transformação massiva de toda a indústria de computação nos últimos anos. Serviços de computação estarão prontamente disponíveis sob demanda, como os outros serviços utilitários disponíveis atualmente como água, energia e telefone. Similarmente, os usuários (consumidores) precisam pagar os provedores somente quando eles acessam os serviços de computação. Além disso, consumidores não precisam mais realizar altos investimentos ou enfrentar a dificuldade de construir e manter complexas infra-estruturas de TI para instalar um aplicativo em suas dependências.

Os profissionais de desenvolvimento de software estão enfrentando inúmeros novos desafios com o objetivo de criar serviços que atendam a milhões de consumidores ao invés de prover um software que seja executado em computadores pessoais. E ao longo dos anos, o surgimento de avanços tecnológicos como processadores multicore e ambientes de computação em rede como Cluster Computing [62], Grid Computing [25], computação P2P [57] e o mais recente Cloud Computing [42], tornam esse objetivo cada vez mais factível.

Os serviços de computação citados anteriormente precisam ser altamente confiáveis, escaláveis e dinâmicos para suportar acesso ubíquo e fornecer a possibilidade de composição de outros serviços [15]. Além disso, consumidores devem ter a capacidade de determinar o nível de serviço requerido através de parâmetros de QoS (Quality of Service) e SLA (Service Level Agreement) [7].

Cloud Computing foi o último desses paradigmas a emergir e promete entregar serviços confiáveis através da nova geração de datacenters que são construídos utilizando tecnologias de virtualização de processamento e armazenamento. Consumidores estarão habilitados a acessar aplicações e dados da “cloud” em qualquer lugar do mundo e sob demanda, como é o caso do Gmail[5], Google Docs[6], Office Web Apps[7], Dropbox[8], dentre outros.

[5] http://www.gmail.com
[6] http://docs.google.com
[7] http://office.microsoft.com/web-apps
[8] http://dropbox.com

2.2. Conceitos Fundamentais

2.2.1. Cloud Computing

Atualmente não existe uma definição oficial do que seja *Cloud Computing*. Nesse trabalho usaremos a definição proposta pelo National Institute of Standards and Technology(NIST), que define Cloud Computing como um modelo que permite, de forma conveniente, o acesso a um pool de recursos computacionais compartilhados (rede, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento ou interação do provedor de serviço.

Ainda segundo o NIST, *Cloud Computing* é composta por cinco características essenciais [42]:

- **Alocação de recursos sob demanda:** O usuário pode adquirir unilateralmente recursos computacionais, como tempo de processamento do servidor ou armazenamento, através da rede na medida em que necessite e sem precisar de interação humana com os provedores de cada serviço;

- **Amplo acesso a rede:** recursos estão disponíveis através da rede e podem ser acessados por meio de mecanismos que funcionem em plataformas heterogêneas (por exemplo, telefones celulares, laptops e PDA);

- ***Pooling* de recursos:** os recursos do provedor de computação são agrupados para atender vários consumidores através de um modelo *multi-tenancy*, com diferentes recursos físicos e virtuais atribuídos dinamicamente e novamente de acordo com a demanda do consumidor. Há um senso de independência local em que o cliente geralmente não tem nenhum controle ou conhecimento sobre a localização exata dos recursos disponibilizados, mas pode ser capaz de especificar o local em um nível maior de abstração (por exemplo, país, estado ou data center). Exemplos de recursos incluem o armazenamento, processamento, memória, largura de banda de rede e máquinas virtuais;

- **Elasticidade rápida:** recursos podem ser adquiridos de forma rápida e elástica, em alguns casos automaticamente, caso haja a necessidade de escalar com o aumento da demanda, e podem também ser liberados, na retração dessa demanda. Para os usuários, os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento; e

- **Serviço medido:** sistemas em nuvem automaticamente controlam e otimizam a utilização dos recursos, alavancando a capacidade de medição em algum nível de abstração adequado para o tipo de serviço (por exemplo, armazenamento,

processamento, largura de banda, e contas de usuários ativos). Uso de recursos pode ser monitorado, controlado e relatado a existência de transparência para o fornecedor e o consumidor do serviço utilizado.

Ainda segundo o NIST [42], Cloud Computing é dividido em três modelos de serviço (Figura 2.1):

• **Software como Serviço (SaaS):** A capacidade fornecida ao consumidor é a de usar as aplicações do fornecedor em uma infraestrutura da nuvem. As aplicações são acessíveis de vários dispositivos cliente através de uma interface thin client, como por exemplo um browser web. O consumidor não administra ou controla a infraestrutura básica, incluindo rede, servidores, sistemas operacionais, armazenamento, ou mesmo capacidades individuais da aplicação. Mesmo assim ainda é possível definir algumas configurações específicas para o usuário na aplicação;

• **Plataforma como Serviço (PaaS):** A capacidade fornecida ao consumidor é a de realizar deploy de uma aplicação em uma infraestrutura pré-definida ou adquirir aplicações criadas usando linguagens de programação e as ferramentas suportadas pelo provedor de PaaS. O consumidor não administra ou controla a infraestrutura básica como rede, servidores, sistemas operacionais, ou armazenamento, mas tem controle sobre os aplicativos utilizados e, eventualmente, hospedagem de aplicativos e configurações de ambiente; e

• **Infraestrutura como Serviço (IaaS) :** A capacidade prevista para o consumidor é a de processamento, armazenamento, redes e outros recursos computacionais fundamentais para que o consumidor seja capaz de implantar e executar programas arbitrários. Esses recursos podem incluir sistemas operacionais e aplicativos. O consumidor não administra ou controla a infraestrutura de nuvem subjacente, mas tem controle sobre os sistemas operacionais, armazenamento, aplicativos implantados, e, eventualmente, o controle limitado de componentes de rede (por exemplo, firewall).

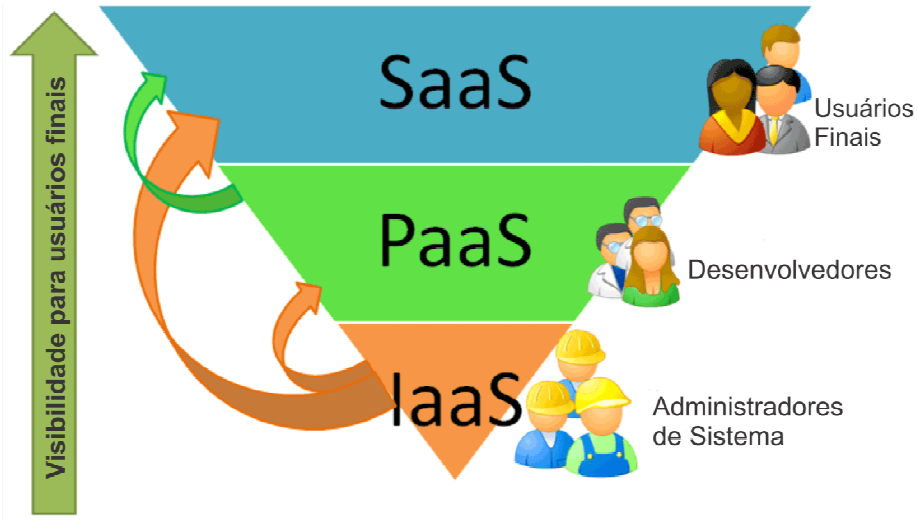


Figura 2.1. Modelos de serviço de *Cloud Computing*

Nesse trabalho teremos como foco principal os tópicos relacionados a SaaS, embora sejam mencionados alguns conceitos associados à PaaS e IaaS. O motivo disso é que multitenancy é um conceito existente dentro do contexto de Software como serviço.

2.2.2. Software como Serviço (SaaS)

Nos últimos anos muitas empresas têm saído do modelo de entrega de software empacotado para o modelo de fornecimento de software na web [23]. Essas aplicações entregues através da web vão desde emails a calendários, sistemas colaborativos, publicações online, processadores de texto simples, aplicações para negócios e aplicações para uso pessoal.

Salesforce.com [35], por exemplo, criou um aplicativo para CRM (*Customer Relationship Management*) e o configurou não como um software empacotado, mas como software rodando sobre servidores acessível através do browser. Quando fez isso, criou a própria plataforma in-house para entregar o software como serviço a seus consumidores.

Logo em seguida Salesforce.com criou o *AppExchange*, uma plataforma de integração aberta para outras empresas de software construírem produtos utilizando algumas funcionalidades do CRM do *salesforce*. Pouco tempo depois *Salesforce* estendeu o conceito de plataforma aberta como force.com, um ambiente de desenvolvimento e deployment usando a infraestrutura de SaaS do *Salesforce*. Posteriormente, algumas outras empresas ingressaram nesse mercado, a Amazon com o EC2 [2] e Google com o Google AppEngine[9], abrindo suas infraestruturas de cloud para hospedarem aplicações de terceiros, além de produzirem serviços online.

Amazon, em especial, vem se tornando bastante atraente porque possui uma rica infraestrutura para suportar operações de varejo online e tem oferecido vários serviços para usuários de cloud como: *data storage*, processamento, fila de mensagens, *billing* e etc. O artigo *"Amazon S3's Amazing Growth"* [24] menciona o crescimento vertiginoso no uso do serviço S3 da Amazon [2] nos últimos anos, é possível ver o gráfico de crescimento desse serviço na Figura 2.2.

Cloud Computing e SaaS também parecem ser eficientes tanto para usuários quanto para fornecedores de software. Vários consumidores podem usar a mesma instalação do software e consequentemente isso melhora as taxas de utilização de hardware e rede. Por exemplo, Amazon[10] e Google[11] têm enormes *datacenters* que eles não utilizam completamente o tempo todo. Eles podem executar seus próprios produtos enquanto hospedam aplicações de outras empresas. Empresas como Amazon, Google e

[9] https://appengine.google.com
[10] http://aws.amazon.com
[11] http://google.com

Salesforce geralmente garantem a qualidade de serviço para todos os seus consumidores de cloud através de SLAs (*Service Level Agreements*) detalhadas.

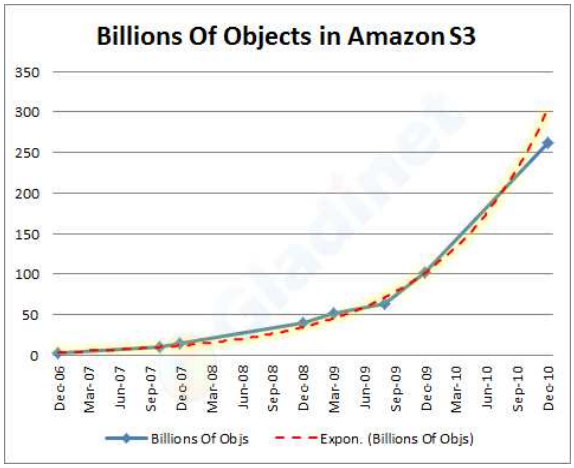


Figura 2.2. Amazon S3's Growth (Fonte: [24])

Os serviços providos por esses grandes players do mercado permitem que empresas possam desenvolver software no modelo SaaS e hospedar em algum de seus datacenters pagando apenas pelo que usarem. Isso torna-se um atrativo principalmente para Startups e empresas com poucos recursos financeiros.

Segundo Harris et al.[21] existem três tipos diferentes de aplicações SaaS:

- **Single-instance:** as aplicações proveem serviços para um único cliente e executam em um servidor exclusivo. Nesse caso cada servidor web possui uma única instância da aplicação. Essa pode considerada a abordagem com maior desperdício de recursos, dado que um servidor pode, por exemplo, executar com apenas 10% de sua capacidade;

- **Multi-instance:** essas aplicações executam em ambientes onde o servidor web é compartilhado. Nessa abordagem uma cópia da aplicação é inicialmente configurada para cada cliente, e então cada cópia é implantada como um contexto no mesmo servidor web; e

- **Multi-tenancy:** provê uma única aplicação compartilhada por vários clientes. Nessa abordagem várias aplicações "virtuais" são criadas na mesma instância.

Implementar o conceito de SaaS nem sempre é tão simples como parece. Chong[12] propõe 4 níveis de maturidade para aplicações que utilizam o modelo de SaaS:

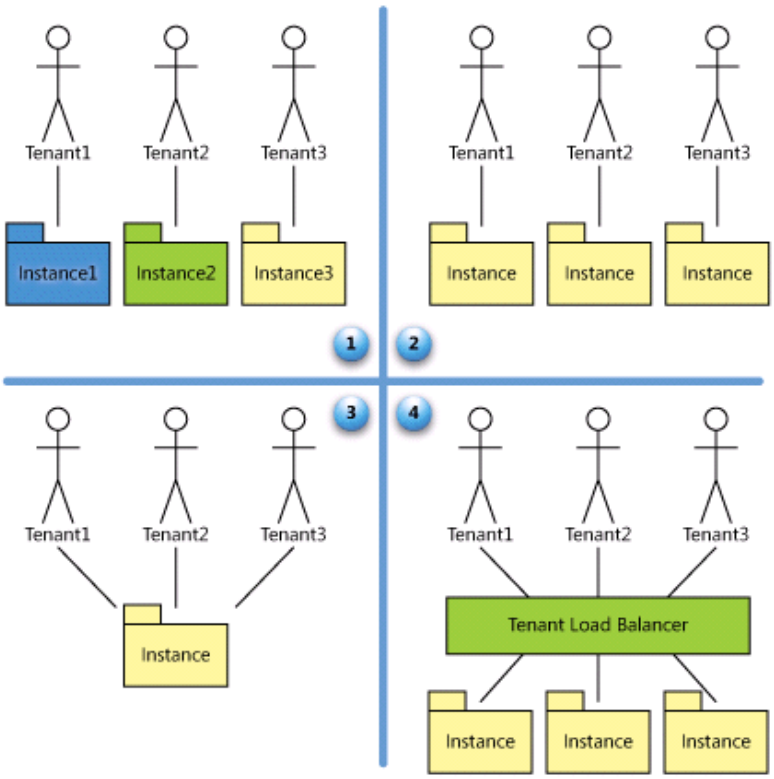


Figura 2.3. Níveis de Maturidade SaaS (Fonte: [12])

- **Nível 1 Ad-Hoc/Personalizado:** O primeiro nível de maturidade é semelhante ao modelo de entrega de software do provedor de serviços de aplicativos (ASP Application Service Provider) tradicional, que data da década de 1990. Nesse nível, cada cliente tem a sua própria versão personalizada do aplicativo hospedado e executando nos servidores do provedor. Pensando em arquitetura, software nesse nível de maturidade é muito semelhante aos softwares corporativos vendidos tradicionalmente, em que diferentes usuários de uma organização conectam a uma instância quaisquer outras instâncias ou processos que o host esteja executando para os seus outros clientes;

- **Nível 2 Configurável:** No segundo nível de maturidade, o fornecedor hospeda uma instância separada do aplicativo para cada tenant. Enquanto no primeiro nível cada instância é personalizada individualmente para o tenant, neste nível todas as instâncias utilizam a mesma implementação de código e o fornecedor atende as necessidades dos clientes fornecendo opções de configuração detalhadas que permitem ao cliente alterar a aparência e o comportamento do aplicativo para os seus usuários. Apesar de serem idênticas a nível do código, cada instância permanece totalmente isolada de todas as demais;

- **Nível 3 Configurável e eficiente para vários tenants:** No terceiro nível de maturidade, o fornecedor executa uma única instância que serve a todos os clientes. Metadados configuráveis são usados para fornecer uma experiência de usuário e um conjunto de recursos exclusivos para cada instância. Políticas de

autorização e de segurança garantem que os dados de cada cliente sejam mantidos separados dos dados de outros clientes e que, da perspectiva do usuário final, não exista qualquer indicação de que a instância do aplicativo esteja sendo compartilhada entre vários tenants; e

- **Nível 4 Escalonável, configurável e eficiente para vários tenants:** No quarto e último nível de maturidade, o fornecedor hospeda vários clientes em um ambiente com balanceamento de carga. Os dados de cada cliente são mantidos separados e com metadados configuráveis fornecendo uma experiência do usuário e um conjunto de recursos exclusivos para cada cliente. Um sistema de SaaS é escalonável para um número de clientes arbitrariamente grande, uma vez que a quantidade de servidores e instâncias no lado do fornecedor pode ser aumentada ou diminuída conforme necessário para corresponder à demanda sem a necessidade de remodelar a arquitetura aplicativo, além disso as alterações ou correções podem ser transmitidas para milhares de *tenants* tão facilmente quanto para um único *tenant*.

Normalmente se esperaria que o quarto nível fosse a meta definitiva para qualquer aplicativo de SaaS, mas não é sempre assim. É necessário verificar as necessidades operacionais, arquiteturais e de negócio relacionadas à aplicação. Uma abordagem *singletenant* faz sentido financeiramente? O seu aplicativo pode ser feito para executar em uma única instância lógica? Você pode garantir que a qualidade de serviço desejada por cada cliente seja atendida? Essas são questões que devem ser respondidas quando se pretende adotar o modelo SaaS.

2.2.3. Multi-tenancy

Multi-tenancy é uma abordagem organizacional para aplicações SaaS. Bezemer e Zaidman [8] definem multi-tenancy como aplicações que permitem a otimização no uso dos recursos de hardware, através do compartilhamento de instâncias da aplicação e da instância do banco de dados, enquanto permite configurar a aplicação para atender às necessidades do cliente como se estivesse executando em um ambiente dedicado. Tenant é uma entidade organizacional que aluga uma aplicação multi-tenancy. Normalmente, um tenant agrupa um número de usuários que são os stakeholders da organização.

A definição anterior foca no que nós consideramos aspectos importantes em aplicações multi-tenancy:

- Possibilidade de compartilhamento de recursos de hardware, permitindo a redução de custos [58];

- Alto grau de configurabilidade, permitindo que cada consumidor customize sua própria interface e seu workflow na aplicação [47, 26]; e

- Uma abordagem arquitetural na qual os tenants fazem uso de uma única aplicação e banco de dados [34].

É possível que algumas pessoas confundam multi-tenancy com o conceito de multi-usuário. Multi-tenancy não é multi-usuário. Em uma aplicação multi-usuário nós assumimos que os usuários estão usando a mesma aplicação com opções de acesso limitadas e que essa instância da aplicação é utilizada por apenas um único cliente. Nesse caso podemos ter vários usuários com o perfil "gerente", com o perfil "vendedor", ou com qualquer perfil de usuário necessário para o funcionamento da aplicação. Em uma aplicação multi-tenancy nós assumimos que existe uma instância da aplicação que atende a vários clientes (tenants) e possui um alto grau de configuração. Dependendo da definição dessas configurações, dois tenants podem possuir aparência e workflows diferentes. Um argumento adicional para essa distinção é que o SLA para cada tenant pode ser diferente [39].

Uma abordagem mais profunda sobre o tema multi-tenancy será apresentado na seção seguinte.

2.3. Propostas de Arquitetura Multi-tenancy

Um ponto importante durante o desenvolvimento de software em qualquer segmento é conhecer implementações semelhantes já realizadas por outros desenvolvedores. Partindo desse princípio essa seção apresenta propostas de arquitetura de software que auxiliem no desenvolvimento de aplicações multi-tenancy. Foram encontradas propostas de arquitetura para aplicações multi-tenancy [31, 9, 8, 49, 56, 63, 28, 11, 32] e para plataformas de suporte a multi-tenancy [60, 48, 5].

Dentre essas propostas algumas já foram aplicadas à indústria, como é o caso do Force.com [60] e EXACT [9]. Force.com é uma plataforma de desenvolvimento de software que utiliza arquitetura dirigida à metadados para construção de aplicações multitenancy. Nessa arquitetura tudo que é exposto para os desenvolvedores e usuários da aplicação é internamente representado como metadado. Formulários, relatórios, workflows, priviégios de acesso, customizações específicas do tenant e lógica de negócio, tudo é armazenado como metadados. Outro exemplo de arquitetura dirigida à metadados pode ser encontrada em [48]. Bezemer et al. [9] apresentam uma arquitetura utilizada no desenvolvimento de aplicações na empresa EXACT, uma empresa de desenvolvimento de software especializada em ERP, CRM e aplicações financeiras. Em sua arquitetura os autores apresentam 3 componentes básicos para a implementação de aplicações multitenancy: componente de autenticação, customização e banco de dados. Além disso os autores apresentam um estudo de caso e uma lista de lições aprendidas.

Outros autores apresentam Arquiteturas Orientadas a Serviço (SOA ServiceOriented Architecture) que implementam requisitos de multi-tenancy como é o caso de Jing and Zhang [28],Azeez et al. [5] e Pervez et al. [49]. Jing and Zhang [28] apresentam OSaaS, uma arquitetura que utiliza tecnologias SOA para desenvolvimento de aplicações SaaS. Azeez et al. [5] apresentam uma arquitetura para implementar

multi-tenancy a nível de SOA, que permite a usuários executar seus serviços e outros artefatos SOA em um framework multi-tenancy SOA. Já Pervez et al. [49] apresentam uma arquitetura para SaaS que foca nos aspectos de segurança e balanceamento de carga em ambiente de cloud computing.

Além das propostas de arquitetura citadas anteriormente, foi encontrado também uma proposta de estilo arquitetural, o SPOSAD (*Shared, Polymorphic, Scalable Application and Data*), que é descrito em [31] e [32]. Um estilo arquitetural define os tipos de elementos que podem aparecer em uma arquitetura e as regras que regem a sua interconexão. O SPOSAD descreve componentes, conectores e elementos de dados de uma arquitetura multi-tenancy, bem como restrições impostas nesses elementos. Os autores também apresentam os benefícios do uso dessa arquitetura e informações que podem auxiliar em decisões de projeto.

Tsai et al. [56] apresentam uma arquitetura de duas camadas que foca em escalabilidade, que trabalha a nível de serviço e aplicação para economizar o uso de recursos, e a idéia chave é aumentar os recursos somente onde houver gargalos. Várias técnicas de duplicação de recursos computacionais são propostas, incluindo duplicação preguiçosa e duplicação pro-ativa para alcançar a melhor performance do sistema. Além da arquitetura esse trabalho apresenta um algoritmo para alocação de recursos para cluster em ambientes de cloud. Já Yuanyuan et al. [63] apresentam uma arquitetura multi-tenancy para sistemas de suporte a negócios (BSS Business Support System) e discutem brevemente uma abordagem para alcançar configurabilidade, seguranca e escalabilidade na arquitetura. Focando em escalabilidade de dados, os autores propõem um particionamento horizontal baseado em grupos de entidades pela análise das relações entre as entidade de negócio do sistema.

Calero et al. [10] apresentam a arquitetura de um sistema de autorização multitenancy apropriado para um serviço de middleware para PaaS. Cada empresa pode provê vários serviços de cloud que podem colaborar com outros serviços tanto da mesma organização quanto de organizações diferentes. Esse sistema de autorização suporta acordos de colaboração entre empresas (também conhecidos como federações).

2.4. Componentes Básicos de uma Aplicação Multi-tenancy

Multi-tenancy afeta quase todas as camadas de uma aplicação típica e possui um grande potencial para ser implementado como interesse transversal [43]. Para reduzir a complexidade do código, a implementação de requisitos multi-tenancy deve ser separada da lógica de negócio o máximo possível. Caso contrário, a manutenção pode se tornar um pesadelo, porque:

- Implementar código de requisitos da arquitetura multi-tenancy juntamente com a lógica de negócio dos tenants, exige que todos os desenvolvedores sejam reeducados para entenderem os conceitos de multi-tenancy; e

- Misturar multi-tenancy com código de lógica de negócio dos tenants leva ao aumento da complexidade da implementação, pois é mais difícil manter o controle de onde o código multi-tenancy é introduzido.

Estes problemas podem ser superados integrando cuidadosamente multi-tenancy na arquitetura. No restante desta seção, descrevemos os componentes da arquitetura abordada por Bezemer et al. [9] para implementação de multi-tenancy como um interesse transversal. A Figura 2.4 apresenta a descrição dessa aplicação e as subseções seguintes descrevem cada componente da aplicação.

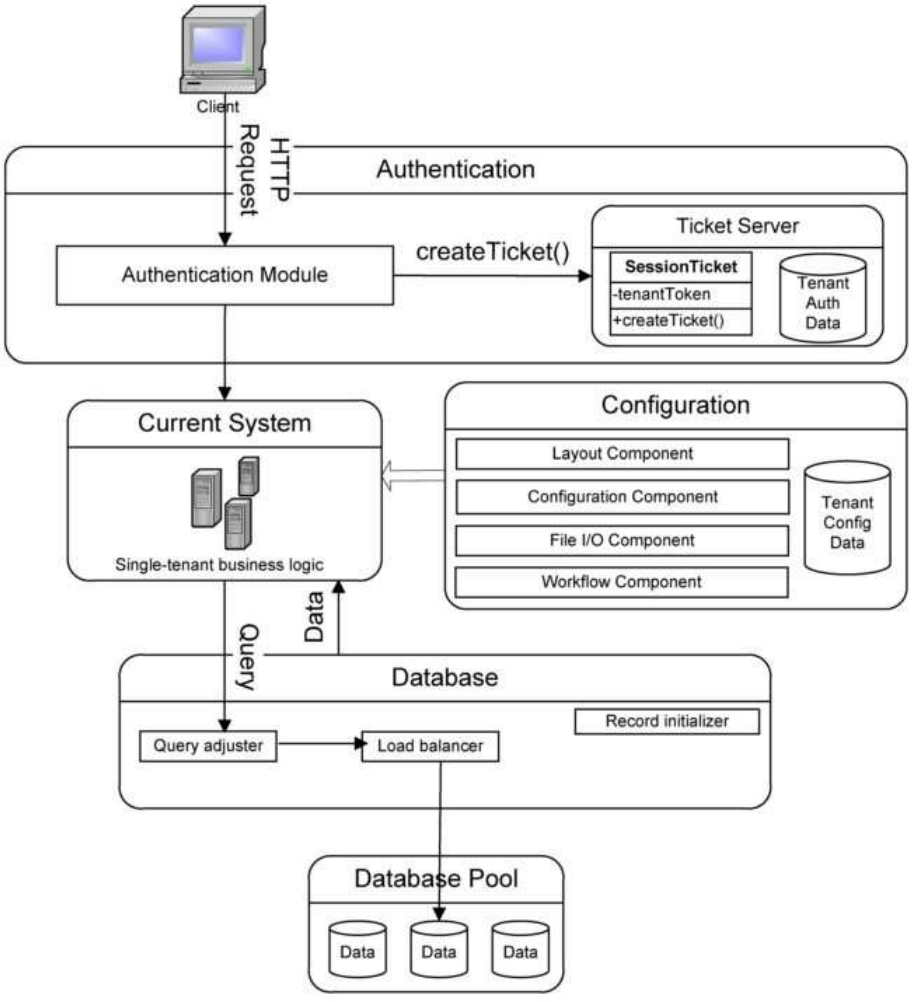


Figura 2.4 Arquitetura de Referência adotada (Fonte: [9])

2.4.1. Autenticação

Devido a uma aplicação multi-tenancy ter apenas uma instância da aplicação e do banco de dados, todos os tenants usam o mesmo ambiente físico. A fim de oferecer a customização do ambiente e ter certeza de que os tenants podem acessar somente os seus próprios dados, tenants devem ser autenticados. Enquanto autenticação de usuário é, possivelmente, já presente na aplicação de destino, um mecanismo separado de

autenticação de tenants específicos pode ser necessário, por duas razões: (1) geralmente é muito mais fácil introduzir um mecanismo de autenticação adicional do que mudar um já existente, e (2) autenticação de tenants permite que um único usuário faça parte de mais do que uma organização lógica, o que estende a idéia de autenticação de usuários com “grupos”.

2.4.2. Configuração

Em uma aplicação multi-tenancy a customização deve ser possível através de configuração. A fim de permitir que o usuário tenha uma experiência como se ele estivesse trabalhando em um ambiente dedicado, é necessário permitir pelo menos os seguintes tipos de configuração:

- **Estilo de Layout (Layout Style):** O componente de configuração de estilo de layout permite o uso de temas e estilos específicos;
- **Configuração Geral (General Configuration):** O componente de configuração geral permite a especificação de configurações específicas, como configurações de chave de criptografia e detalhes do perfil pessoal;
- **Entrada e saída de arquivo (File I/O):** O componente de configuração de I/O de arquivo permite a especificação de caminhos de arquivos, que podem ser usados para, por exemplo, geração de relatório; e
- **Fluxo de trabalho (Workflow):** O componente de configuração de fluxo de trabalho permite a configuração de fluxos específicos. Por exemplo, configuração de fluxos é necessária em uma aplicação de planejamento de recursos empresariais ERP, em que os passos para se realizar uma mesma tarefa pode variar significativamente entre diferentes companhias.

2.4.3. Banco de dados (Database)

Em uma aplicação multi-tenancy há uma grande exigência pelo isolamento dos dados. Pelo fato de os tenants usarem a mesma instância de um banco de dados é necessário ter certeza de que eles podem acessar somente seus próprios dados. Atualmente sistemas de gerenciamento de dados (Data Base Management Systems DBMS) de prateleira não são capazes de lidar com multi-tenancy de forma nativa, isso deve ser feito em uma camada entre a camada lógica de negócios e o pool de banco de dados da aplicação. As principais tarefas dessa camada são as seguintes:

- **Criação de novos tenants no banco de dados:** Se a aplicação armazena ou recupera dados que podem ser de tenants específicos, é tarefa da camada de banco de dados criar os registros do banco de dados correspondente quando um novo tenant se inscreveu para a aplicação;

- **Adaptação de consulta:** A fim de prover um isolamento de dados adequado, a camada de banco de dados deve ter certeza que consultas são ajustadas de forma que cada tenant possa acessar somente seus próprios registros; e

- **Balanceamento de carga:** Para melhorar o desempenho de uma aplicação multi-tenancy é necessário um balanceamento de carga eficiente para o pool de banco de dados. Note que qualquer acordo feito no SLA de um tenant e quaisquer restrições impostas pela legislação do país onde o tenant está localizado deve ser satisfeita. Por outro lado, nossa expectativa é a de que é possível criar algoritmos de balanceamento de carga mais eficientes usando as informações coletadas sobre as características de funcionamento dos tenants.

2.5. Implementando um Protótipo de Aplicação Multi-tenancy

2.5.1. Tecnologias

Após a escolha de uma arquitetura de referência para ser adotada por nossa aplicação, tem-se a necessidade de escolher as tecnologias que serão adotadas para a implementação. Durante a escolha dessas tecnologias avaliou-se a possibilidade do uso de JSF (Java Server Faces)[12], Struts 2[13], Spring Framework[14] e Grails[15]. Para a implementação do protótipo descrito nesse trabalho optou-se por utilizar o Grails Framework, pois apresenta um conjunto de recursos que podem dar produtividade para a equipe de desenvolvimento como geração de CRUD (Create, Read, Update e Delete); arquitetura baseada em plugins, que permite a criação e reuso de componentes; e total integração com frameworks e APIs Java. É possível desenvolver aplicações multi-tenancy em outras linguagens de programação como PHP, C#, Python, Ruby, etc. O que pode variar de uma linguagem para outra é o esforço necessário para desenvolver esse tipo de aplicação e algumas variações de performance.

Grails é um framework web open-source que utiliza a linguagem Groovy[16], e outros frameworks consagrados como Hibernate[17], Spring Framework e Sitemesh[18]. Uma descrição visual da arquitetura do Grails é apresentada na Figura 2.5. Grails foi projetado para desenvolver aplicações CRUD de forma simples e ágil, utilizando o modelo de “escrever código por convenção” introduzido pelo Ruby on Rails. O Grails propõe trazer a produtividade do Ruby on Rails para a plataforma Java, porém ele possui uma grande vantagem, já que é baseado na linguagem Groovy. Groovy (padronizado pela JSR-241) é uma linguagem dinâmica e ágil para a plataforma Java, que possui muitas características de linguagens de script como Ruby, Python e Smalltalk; e, além disso, aplicações Groovy podem utilizar classes Java facilmente.

[12] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[13] http://struts.apache.org/2.3.1.2/index.html
[14] http://www.springsource.org/spring-framework
[15] http://grails.org
[16] http://groovy.codehaus.org
[17] http://hibernate.org
[18] http://sitemesh.org

Linguagens de script estão ganhando cada vez mais popularidade, devido a quantidade reduzida de código fonte necessário para implementar determinadas funcionalidades, se comparado com uma implementação em Java.

O protótipo multi-tenancy descrito nesse trabalho, em particular, se beneficiará da capacidade de definir métodos e propriedades em tempo de execução, disponibilizado pela linguagem Groovy. Essa característica da linguagem Groovy é chamada de metaprogramação. Em uma linguagem estática como Java, o acesso a uma propriedade ou invocação de um método é resolvido em tempo de compilação. Em comparação, Groovy não resolve o acessos à propriedades ou invocação de métodos até que a aplicação seja executada [50]. Uma aplicação que utiliza essa linguagem pode dinamicamente definir métodos ou propriedades em tempo de execução, isso vai de encontro à necessidade de customização das aplicações, dado que, com a utilização desse recurso, pode-se adicionar atributos e customizar comportamentos de um tenant específico futuramente.

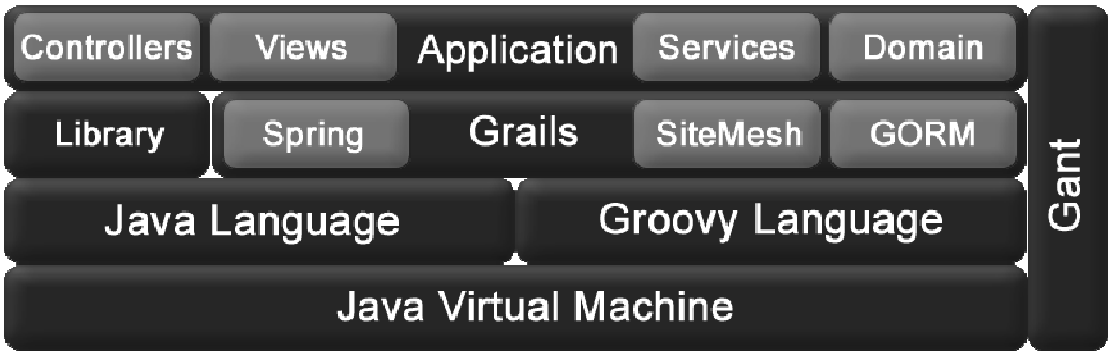


Figura 2.5. Arquitetura do Grails Framework (Fonte: [29])

Outro ponto que influenciou bastante na escolha de Grails foi sua arquitetura baseada em plugins. Grails não se propõe a ter todas as respostas e soluções para o desenvolvimento de aplicações web. Ao invés disso, ele provê uma arquitetura baseada em plugins e um repositório mantido pela comunidade de desenvolvedores onde é possível encontrar plugins que implementam as mais diversas funcionalidades como segurança, teste, busca, geração de relatórios, REST, web services, etc. Essa abordagem proporciona o reuso e permite que funcionalidades de difícil implementação possam ser adicionadas na aplicação de forma bastante simples.

Além das tecnologias relacionadas à programação, foi necessário escolher a tecnologia utilizada para armazenamento de dados. Como já existiam dados legados cadastrados em um banco de dados relacional, decidiu-se adotar o SGBD (Sistema de Gerenciamento de Banco de Dados) PostgreSQL. A escolha desse SGBD deu-se pelo fato do mesmo ser open-source, bastante consolidado no mercado e possuir integração com várias ferramentas de relatórios, o que poderia facilitar futuras extrações de dados para os clientes.

2.5.2. Prototipagem

Para exemplificar uma aplicação real que implementa o conceito de multi-tenancy será descrita a implementação de uma aplicação desenvolvida utilizando Grails Framework. A Arquitetura dessa aplicação é apresentada na Figura 2.6. Em nossa proposta os módulos associado à lógica de negócio da aplicação são implementados o mais independente possível dos requisitos multi-tenancy. Juntamente com o Framework Grails foram adotados alguns plugins existentes em seu repositório. A implementação do protótipo tem o objetivo de apresentar como esses conceitos podem ser aplicados na prática.

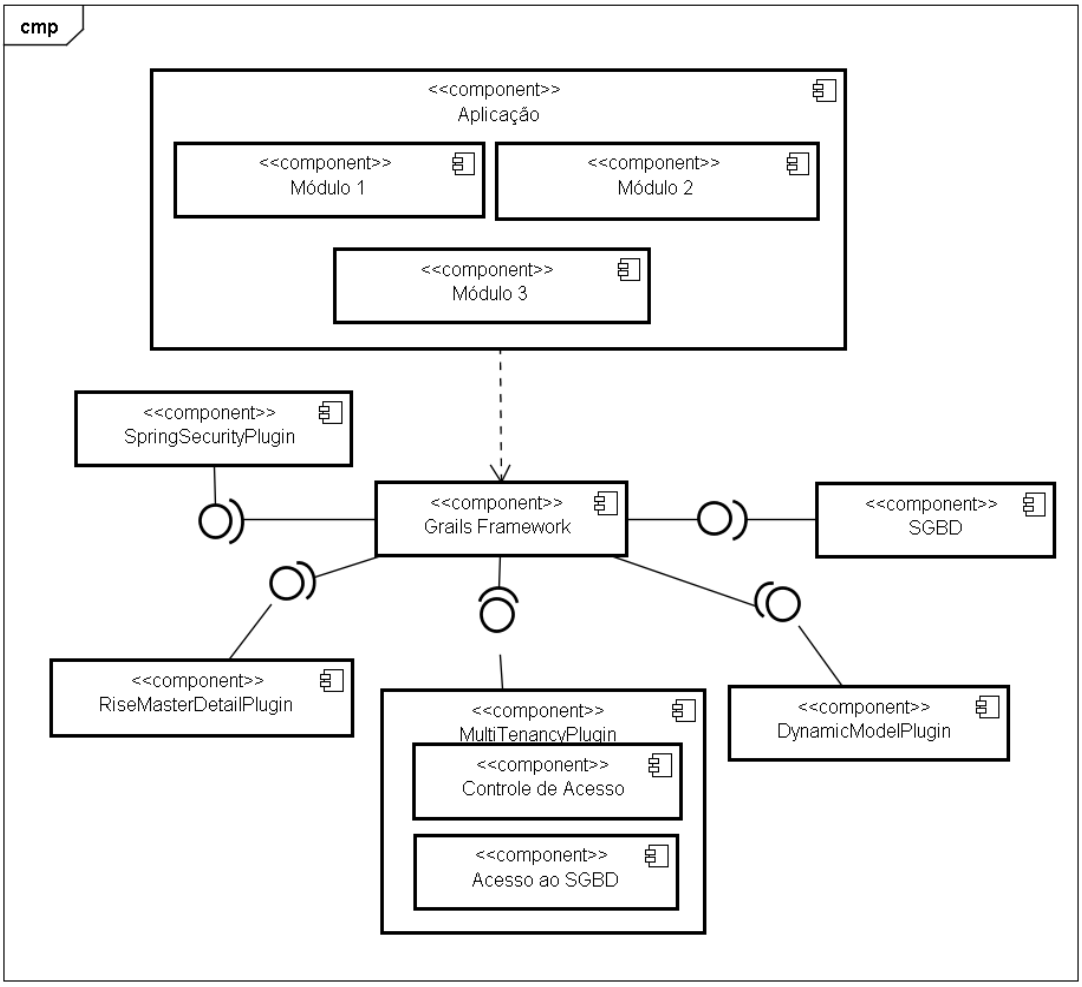


Figura 2.6. Arquitetura da Aplicação

Como mencionado anteriormente, existe uma grande necessidade de separar o código de lógica de negócio do código relacionado aos requisitos multi-tenancy. Com o objetivo de implementar os requisitos multi-tenancy de forma reutilizável, a solução foi implementar esses requisitos como um plugin do Grails, dessa forma aplicações futuras poderiam se beneficiar do código implementado. Uma vantagem do uso dessa abordagem é que dentro de um plugin grails é possível adicionar não só ter classes

implementadas em Groovy ou Java, mas também páginas da camada de visão e outros tipos de arquivo. Antes de implementar os requisitos multi-tenancy como plugin, foi realizado uma pesquisa no repositório de plugins do Grails para verificar se já existia algo semelhante ao que se pretendia implementar.

Durante a pesquisa foi encontrado o plugin *Multi-tenancy Core*[19] que implementa as funcionalidades de dois dos componentes mencionados em nossa arquitetura de referência[9]: o componente de autenticação e o componente de acesso ao banco de dados. Esse plugin implementa a funcionalidade de associar um usuário da aplicação a um tenant específico e além disso realiza de forma dinâmica o filtro dos dados durante uma consulta ao banco de dados. Dessa forma um usuário da aplicação tem acesso apenas aos dados vinculados ao seu tenant. Para utilizar esse plugin é necessário apenas instalá-lo na aplicação e adicionar uma anotação *@MultiTenant* nas classes de domínio da aplicação. Essa anotação indica que todos os objetos dessa classe deverão ser gerenciados pelo plugin e sempre que um registro for salvo no banco deverá ser associado ao tenant do usuário que estiver logado no momento. Um exemplo simplificado de uma classe Groovy com essa anotação é apresentada na Figura 2.7.

```groovy
package br.com.rise.alexandria.fur

import grails.plugin.multitenant.core.groovy.compiler.MultiTenant;

/**
* The Product entity.
*
* @author Josino Rodrigues
*/
@MultiTenant
class Product {

    String name
    String description

    static hasMany = [modules : Module]


    static constraints = {
        project(blank:false)
        name(size:1..100, blank:false, unique:['tenantId','project'])
        description()
    }

    //MÉTODOS ...

}
```

Figura 2.7. Exemplo de classe escrita em linguagem Groovy que utiliza o plugin Multi-Tenancy Core

A autenticação e controle de acesso pode ser realizado através da integração desse plugin com o plugin Spring Security[20], um framework de controle de acesso bastante utilizado em aplicação Java. A Figura 2.8 apresenta a tela de autenticação gerada pelo próprio plugin do Spring Security. De forma integrada com o plugin Multi-tenancy Core, ele associa cada usuário logado ao tenant ao qual ele pertence.

[19] http://multi-tenancy.github.com/grails-multi-tenancy-core
[20] http://static.springsource.org/spring-security/site/index.html

O terceiro componente mencionado em nossa arquitetura de referência é o componente de configuração. O objetivo desse componente é gerenciar as configurações relacionadas a cada tenant e prover funcionalidades de customização da aplicação para estes.

Figura 2.8. Tela de autenticação gerada pelo Spring Security Plugin

Essa customização pode ir desde alterações em interface com o usuário até alteração nas classes de negócio, tudo isso realizado de forma dinâmica. Durante a pesquisa foi encontrado o plugin *Dynamic Domain Class*[21] que proporciona a criação de classes de domínio de forma dinâmica, esse plugin ajudou a validar a idéia de usar Groovy e Grails para customizar as classes de domínio em tempo de execução. A tela apresentada na Figura 2.9 apresenta o formulário utilizado na aplicação para criação de classes dinâmicas. Para cada classe dinâmica criada o Grails Framework cria automaticamente o CRUD para essa entidade.

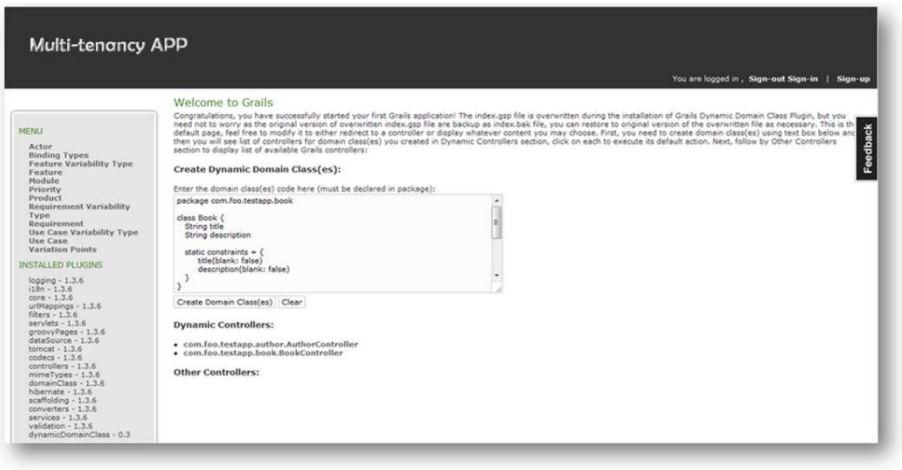


Figura 2.9. Tela com Formulário para criação de classes dinâmicas

Durante a implementação de um projeto real, detectamos que o padrão de geração de telas do Grails não satisfazia aos anseios dos usuários quanto à forma de interação com o aplicativo. Apartir dessa necessidade foi desenvolvido um plugin para implementar o

[21] http://code.google.com/p/grails-dynamic-domain-class-plugin/

padrão de tela Mestre/Detalhe [45], não implementado nativamente pelo gerador de telas padrão do Grails. Durante a implementação desse plugin identificou-se que poderia ser viável implementar módulos completos da aplicação como plugins Grails, desde classes de negócio à camada de visão.

Após a implementação do protótipo foi realizado a avaliação do protótipo para verificar alguns atributos de qualidade atendidos pela proposta. A Tabela 2.1 apresenta para cada atributo de qualidade uma breve descrição de como ele foi atendido.

Tabela 2.1. Resultado da aplicação dos critérios de exclusão (Fonte: Elaboração própria)

ATRIBUTO	AVALIAÇÃO
Disponibilidade	Como foi adotado um SGBD *PostgreSQL*, é possivel agendar tarefas automáticas no servidor para executar *backups* periódicos.
Integridade Conceitual	Tanto as entidades de negócio quanto as classes utilitárias da aplicação foram modularizadas em *plugins* de forma que os interesses da aplicação foram bem definidos e separados.
Flexibilidade	A flexibilidade pode ser alcançada através do uso das característica de linguagem dinâmica existente na linguagem *Groovy*. Dessa forma é possivel criar Entidade de domínio e alterar entidades já existentes em tempo de execução, caso seja necessário.
Interoperabilidade	O uso de *plugins Grails* permite que os dados da aplicação possam ser expostos através de REST ou *Web Services* para os usuários.
Manutenibilidade	A arquitetura de *plugins* do *Grails* facilita a manutenção dos código já que cada plugin pode ser mantido separadamente, reduzindo as dependências entre os componentes da aplicação.
Reusabilidade	As partes da aplicação implementadas como *plugins* podem ser reutilizadas em aplicações futuras facilmente, sem a necessidade de qualquer adaptação no *plugin*.
Segurança	A autenticação e controle de acesso puderam ser garantidos pelo uso do *Framework Spring Security*, que pôde ser facilmente integrado com o *plugin* de *multi-tenancy* no atendimento do requisito de isolamento de acesso entre dados dos *tenants*.
Testabilidade	O *Grails Framework* provê uma infraestrutura para auxiliar na implementação de testes unitários que utiliza *JUnit*, um *framework* de teste para aplicações *Java*. Durante a criação de uma classe o *Grails Framework* já cria uma classe de testes correspondente para otimizar o tempo de desenvolvimento.
Usabilidade	*Grails* trabalha com templates para geração de telas. Embora já possua um template padrão, o desenvolvedor pode implementar um template próprio que atenda à necessidades específicas de interação com o usuário. O template padrão do *Grails Framework* já implementa alguns padrões de usabilidade bem estabelecidos no mercado.

Esse protótipo foi baseado em uma experiência adquirida durante o desenvolvimento de uma aplicação real. A aplicação desenvolvida entrou em produção e atendeu satisfatoriamente aos requisitos do cliente. Durante o desenvolvimento do software em questão foi utilizado Grails porque era, na época, a opção open source mais madura no tocante a multi-tenancy. Atualmente já existem outros frameworks que

dão suporte ao desenvolvimento de aplicações multi-tenancy como Atena Framework[22], Hibernate 4[23], Rails[24], etc.

2.6. Vantagens e desvantagens

Durante uma vasta pesquisa bibliográfica foram identificados 12 trabalhos que citam vantagens ou desvantagens da adoção de multi-tenancy. Apartir da leitura dos mesmos identificamos as seguintes vantagens:

- O provedor de serviço pode usar a mesma versão da aplicação (com o único código base) para prover serviços para várias organizações [55];
- Consumidores obtêm acesso à capacidade de processamento elástico que pode ser aumentada ou diminuída de acordo com a demanda sem a necessidade de realizar manutenção em servidores [55];
- Dois benefícios de uma abordagem baseada em uma plataforma multi-tenancy são colaboração e integração. Pelo fato de todas as aplicações executarem em um único espaço, torna-se mais fácil permitir a um usuário de qualquer aplicação acesso à uma coleção de dados de outra aplicação semelhante. Essa capacidade simplifica o esforço necessário para integrar aplicações relacionadas e os dados que elas gerenciam [60];
- Atualização do software de uma só vez para todos os tenants [44];
- Economia nos custos com infra-estrutura de hardware [53, 33, 4];
- Economia nos custos de gerenciamento de infra-estrutura [53, 33, 4];
- Aumento da margem de lucro para o provedor de serviço através da redução dos custos de entrega e diminuição dos custos de assinatura do serviço para os clientes [36];
- Possibilidade de reusar regras de negócio com o mínimo de adaptação [9];
- Organização dos usuários em vários níveis de acordo com suas necessidades e melhor gerenciamento de recursos [27];
- O usuário pode customizar sob-demanda os serviços providos pelo fornecedor do software [67]; e
- Redução dos custos de venda e manutenção do software por porte do provedor do software [67].

As desvantagens da adoção de multi-tenancy foram pouco mencionadas nos trabalhos encontrados. Em geral os trabalhos mencionavam mais vantagens do que

[22] http://athenasource.org
[23] http://docs.jboss.org/hibernate/orm/4.1/devguide/en-US/html/ch16.html
[24] http://rubyonrails.org

desvantagens. A seguir são listados alguns pontos considerados desvantagens por alguns autores:

- É difícil calcular os recursos requeridos por cada novo tenant e ao mesmo tempo garantir que as restrições de todos os outros tenants da mesma instância sejam atendidas[33];
- Fatores limitantes e gargalos nos recursos computacionais exigidos pelas várias instâncias devem ser identificados [33], e isso não é trivial nesse tipo de ambiente;
- Dificuldade de comparar e otimizar a redução de custos das diferentes formas de distribuição dos tenants entre os servidores, pelo fato de existirem várias variáveis envolvidas [33];
- Preocupação das empresas com o custo inicial de reestruturas suas aplicações legadas para multi-tenancy, [8]; e
- Preocupação dos mantenedores de software com a possibilidade de multi-tenancy introduzir problemas adicionais de manutenção decorrentes do fato desses novos sistemas serem altamente customizáveis [8].

2.7. Desafios da área

2.7.1. Alocação de Recursos

Para que se possa desfrutar dos benefícios que uma aplicação multi-tenancy traz, um conjunto de desafios devem ser solucionados [33]:

1. Calcular os recursos computacionais necessários para o funcionamento de cada novo tenant, e ao mesmo tempo atender às restrições de todos os tenants em instância da aplicação compartilhada;
2. Identificar fatores limitantes ou gargalos nos recursos computacionais requeridos para as várias instâncias, cada uma com vários tenants contendo diferentes restrições que devem ser atendidas;
3. Durante a distribuição dos tenants é necessário indicar a melhor localização para que nenhuma restrição de SLA seja violada;
4. A distribuição dos tenants e instâncias em um ambiente de computação distribuído deve ser automatizada; e
5. A economia alcançada entre as diferentes formas de distribuições de tenants e instâncias deve ser comparada e otimizada, de forma que seja encontrada a melhor distribuição possível.

O cálculo do número máximo de usuários e tenants em uma instância compartilhada em um servidor, sem que haja violação de restrições definidas no

contrato de SLA de cada tenant é um desafio não trivial. Kwok e Mohindra [33] propõem uma abordagem para o cálculo de recursos requeridos para o bom funcionamento de vários tenants em uma instância de aplicação compartilhada. Nesse trabalho também é descrito um método para otimizar a distribuição de tenants e instâncias de uma aplicação em um conjunto de servidores sem violar qualquer requisito de SLA dos tenants. Por fim os autores apresentam uma ferramenta que tem o objetivo de auxiliar o deployment de aplicações multi-tenancy usando o número mínimo de servidores, fornecendo portanto o máximo de economia em um ambiente de computação distribuído.

Fehling et al. [18] também realizam um estudo sobre as oportunidades de otimização do uso de recursos, mas nesse caso o cenário considerado é um ambiente de computação heterogêneo onde os recursos utilizados são oriundos de clouds públicas e privadas. Nesse trabalhos os autores utilizam um algoritmo Smarter Simulated Annealing para auxiliar na busca de uma distribuição otimizada dos recursos computacionais.

Outra questão associada à alocação de recursos é a priorização das requisições recebidas por uma instância de uma aplicação multi-tenancy. Em um cenário multi-tenancy é comum que cada tenant necessite priorizar as requisições de seus consumidores de tal maneira que um consumidor com prioridade alta poderá ser atendido mais rapidamente que outro, e que a instância da aplicação também priorize os tenants, de forma que as requisições de um tenant tenham maior prioridade de atendimento que as de outro. Diante desse cenário, Tsai et al.e [55] propõem um modelo para priorizar requisições de vários tenants enquanto preserva as prioridades locais das requisições de um tenant específico. Os autores propõem um algoritmo chamado Crystalline Mapping que mapeia prioridades internas de um tenant específico para prioridades globais.

2.7.2. Banco de Dados

Uma das preocupações quando pretende-se implementar uma aplicação multi-tenancy é planejar como os dados da aplicação serão armazenados. Atualmente, boa parte das aplicações existentes no mercado utilizam bancos de dados relacionais. Existem várias estratégias para implementar um esquema de banco de dados relacional de forma que ele permita o armazenamento de dados de vários tenants. Aulbach et al. [3] apresenta várias dessas técnicas, que são mostradas na Figura 2.10 e brevemente descritas a seguir:

- ***Basic Layout*** **-** a técnica mais básica para implementar multi-tenancy é adicionar uma coluna TENANTID em cada tabela para armazenar um valor que identifica a qual tenant um registro pertence. Essa abordagem provê uma boa consolidação mas não provê uma boa extensibilidade, já que não possui nenhum mecanismo para customização de armazenamento de dados para um tenant específico;

- ***Private Table*** **-** é a forma mais básica para suportar extensibilidade, dado que cada tenant possui sua próprias tabelas privadas. Nessa abordagem, é necessário uma camada de transformação de queries para ajustar o nome das tabelas nas queries e substituir pelo nome da tabela privada;
- ***Extension Table*** **-** essa técnica é a combinação das duas técnicas anteriores, as tabelas de extensão bem como as tabelas base devem possuir uma coluna para armazenar os dados de identificação do tenant. Outra coluna também precisa ser adicionada para armazenar a tabela lógica para que os dados possam ser reconstruídos. Essa abordagem é utilizada para mapear esquemas orientados a objeto com herança nos modelos relacionais atualmente;
- ***Universal Table*** **-** estruturas genéricas permitem a criação de um número arbitrário de tabelas com formas arbitrárias. Universal Table é uma estrutura genérica com uma coluna Tenant, uma coluna Table e um grande número de colunas de dados genéricas. A coluna de dados tem um tipo flexível, como VARCHAR, no qual outro tipo pode ser convertido;
- ***Pivot Table*** **-** nessa técnica as entidades de domínio são representadas por tabelas lógicas, cujas informações são montadas dinâmicamente. Uma Pivot Table possui as seguintes colunas: Tenant (identifica o tenant), Table (identifica a tabela à qual o dado está associado), Row (identificar a linha à qual o dado está associado) Col (identifica o campo que a linha representa) e uma coluna para armazenar o dado em si, que normalmente é de um tipo flexível como VARCHAR. Essa abordagem proporciona uma alta flexibilidade mas possui muitas colunas de metadados que podem impactar negativamente na performance da aplicação, durante a manipulação dos dados;
- ***Chunk Table*** **-** essa técnica é uma extensão da técnica anterior e trabalha com o conceito de Chunk Table. Uma Chunk Table é semelhante a uma Pivot Table exceto pelo fato de possuir um conjunto de colunas de dados de vários tipos, e a coluna Col é substituida pela coluna Chunk. Em comparação com a técnica Pivot Table, essa técnica armazena uma quantidade menor de metadados, o que diminui o tempo de reconstrução da tabela lógica; e
- ***Chunk Folding*** **-** as tabelas lógicas são verticalmente particionadas em chunks, os quais permitem junção quando necessário.

Essas são as técnicas básicas para a implementação de modelos de dados para aplicações multi-tenancy, outras abordagens foram criadas a partir delas [19, 61, 59]. Aulbach et al. [4] realizam um estudo experimental para comparar cinco técnicas de implementação de aplicações multi-tenancy a nível de banco de dados. Os autores concluíram que ainda não existe um sistema de gerenciamento de banco de dados (SGBD) ideal para esse tipo de aplicação e que um SGBD para SaaS deveria ser baseado na técnica Private Table.

Schiller et al. [52] apresentam em seu trabalho as primeiras funcionalidades para que um SGBD relacional possa suportar múltiplos tenants nativamente. Em sua

proposta tenants são introduzidos como objetos de primeira classe e é proposto o conceito de “contexto” para isolar um tenant de outro. Além disso, o conceito de herança permite compartilhar o esquema da aplicação entre os tenants, ao mesmo tempo em que permite que o esquema seja extendido com estruturas de dados adicionais. Ao final do trabalho o autor realiza uma avaliação da implementação de sua proposta através de experimentos.

2.7.3. Customização

Um ponto importante em uma aplicação multi-tenancy é a customização. Em aplicações web customizáveis o código torna-se mais complexo, problemas de performance impactam todos os tenants da aplicação e planejar segurança e robustez tornam-se pontos importantes. Segundo Jansen et al. [26], existem três áreas de pesquisa que são

$Account_{17}$

Aid	Name	Hospital	Beds
1	*Acme*	*St. Mary*	135
2	*Gump*	*State*	1042

$Account_{42}$

Aid	Name	Dealers
1	*Big*	65

diretamente relacionadas a Técnicas de Realização de Customização (Customization Realization Techniques CRT) em aplicações multi-tenancy: variabilidade em linhas de produtos de software, personalização do usuário final em aplicações web e arquiteturas multi-tenancy. Pesquisadores que pretendem atacar esse problema devem direcionar seus estudos nessas três áreas.

a) Private Table Layout

$Account_{Ext}$

Tenant	Row	Aid	Name
17	0	1	*Acme*
17	1	2	*Gump*
35	0	1	*Ball*
42	0	1	*Big*

$Healthcare_{Account}$

Tenant	Row	Hospital	Beds
17	0	*St. Mary*	135
17	1	*State*	1042

$Automotive_{Account}$

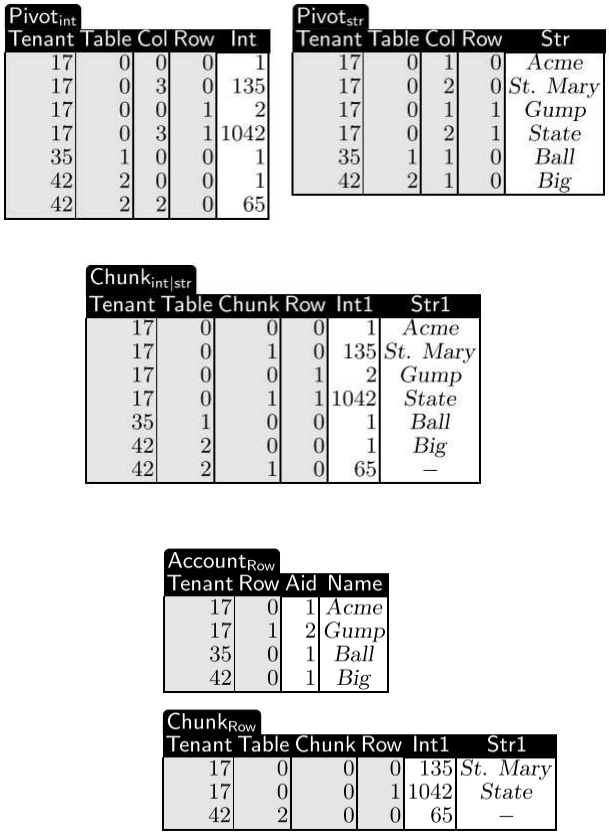
Tenant	Row	Dealers
42	0	65

b) Extension Table Layout

Universal

Tenant	Table	Col1	Col2	Col3	Col4	Col5	Col6
17	0	*1*	*Acme*	*St. Mary*	*135*	–	–
17	0	*2*	*Gump*	*State*	*1042*	–	–
35	1	*1*	*Ball*	–	–	–	–
42	2	*1*	*Big*	*65*	–	–	–

c) Universal Table Layout



Pivot$_{int}$

Tenant	Table	Col	Row	Int
17	0	0	0	1
17	0	3	0	135
17	0	0	1	2
17	0	3	1	1042
35	1	0	0	1
42	2	0	0	1
42	2	2	0	65

Pivot$_{str}$

Tenant	Table	Col	Row	Str
17	0	1	0	*Acme*
17	0	2	0	*St. Mary*
17	0	1	1	*Gump*
17	0	2	1	*State*
35	1	1	0	*Ball*
42	2	1	0	*Big*

Chunk$_{int|str}$

Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	1	*Acme*
17	0	1	0	135	*St. Mary*
17	0	0	1	2	*Gump*
17	0	1	1	1042	*State*
35	1	0	0	1	*Ball*
42	2	0	0	1	*Big*
42	2	1	0	65	–

Account$_{Row}$

Tenant	Row	Aid	Name
17	0	1	*Acme*
17	1	2	*Gump*
35	0	1	*Ball*
42	0	1	*Big*

Chunk$_{Row}$

Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	135	*St. Mary*
17	0	0	1	1042	*State*
42	2	0	0	65	–

d) Pivot Table Layout

e) Chunk Table Layout

f) Chunk Folding

Figura 2.10. Tecnicas de mapeamento de esquema (Fonte: [3])

Outro ponto importante é que os tenants de uma aplicação multi-tenancy não somente têm diferentes requisitos com respeito a propriedades funcionais, mas também podem exigir diferentes propriedades de qualidade de serviço como privacidade e performance. Alguns tenants exigem que a aplicação possua alta disponibilidade e estão dispostos a pagar um alto preço pelo uso desse serviço. Outros tenants não estão interessados em alta disponibilidade mas preocupam-se mais com um baixo preço. Em casos como esse é necessário que exista na aplicação algum mecanismo para ajustar a aplicação às reais necessidades do usuário, no tocante aos casos citados anteriormente.

2.7.4. Escalabilidade

Esse é um dos problemas mais críticos para serem resolvidos em um cenário real. Dado um número fixo de servidores, é necessário otimizar a distribuição dos tenants de forma a maximizar o número total de tenants possíveis sem violar seus requisitos de SLA e ainda estar preparado para o crescimento do volume de dados e de requisições.

Yuanyuan et al. [63] apresentam uma arquitetura focada na escalabilidade de dados. Eles propõem uma entidade com base em grupos de particionamento horizontal, através da análise das relações entre as entidades de uma aplicação multi-tenancy. Nessa abordagem, entidades de negócio altamente coesas formam um grupo de entidades e o particionamento ocorre com base nesse grupo. Dessa forma cada operação acessa um único grupo de instâncias, podendo obter os dados necessários através do acesso a um único banco de dados e evitando a necessidade de transações distribuídas.

Koziolek [31, 32] apresenta um estilo arquitetural focado em escalabilidade de dados e ilustra como os conceitos apresentados nesses trabalhos podem ajudar a fazer as Plataformas como Serviço (PaaS) atuais, como Force.com, Windows Azure, e Google App Engine, escaláveis e customizáveis. Já Zhang et al. [66] focam no problema de localização de tenants, propõem um algoritmo heurístico chamado Tenant Dispatch Heuristic (TDH) e realizam um conjunto de simulações e comparações onde é avaliado como o uso desse algoritmo impacta na escalabilidade e economia de recursos.

2.7.5. Migração

Migrar aplicações web legadas que trabalham com um único tenant (Isolated Tenancy Hosted Applications ITHA) para multi-tenancy (multi-tenancy Enabled Application MTEA) não é uma tarefa trivial, devido a quantidade de ferramentas e técnicas de migração apropriadas. De acordo com Zhang et al. [65] os métodos existentes para migração são muito abstratos ou muito específicos quando tentamos aplicá-los como guias para migrar uma aplicação que trabalha com um tenant isolado para uma aplicação multi-tenancy. Nesse mesmo trabalho os autores propõem um método sistemático que provê diretrizes para migrar aplicações ITHA para MTEA, levando em conta custo, risco e efetividade do método de migração.

Dado a característica de demanda sazonal existente em vários tipos de aplicação, uma funcionalidade essencial para um ambiente multi-tenancy é a funcionalidade que permite a migração de tenants entre hosts, uma técnica chamada live migration [13, 40] é apresentada por alguns autores como uma possível solução.

Elmore et al. [17] apresentam Zephyr, uma técnica para live migration que tem o objetivo de minimizar a interrupção de serviço do tenant migrado. Através da introdução de uma sincronização dupla que permite a ambos os nós, o nó origem dos dados e o nó destino, executarem simultaneamente transações para o tenant. A migração inicia-se com a transferência dos metadados do tenant para o nó destino, que pode realizar novas transações enquanto o nó origem completa as transações que foram iniciadas quando a migração se iniciou. De acordo com os autores, live migration é uma importante característica para habilitar elasticidade em bancos de dados multi-tenancy para plataformas de cloud.

2.7.6. Monitoramento

Para obter uma visão geral do funcionamento de seus serviços, provedores precisam de informações sobre todas as camada de seu sistema, incluindo a aplicação, máquinas virtuais e uso de rede. Por razões de simplicidade, todas essas informações devem idealmente ser entregues através de uma única interface de monitoramento. Hasselmeyer e D'Heureuse [22] apresentam alguns requisitos básicos de uma infraestrutura de monitoramento para ambientes multi-tenancy:

- ***Multi-tenancy:*** a infraestrutura de monitoramento precisa naturalmente ser capaz de lidar com informações de monitoramento provinientes de diferentes clientes (tenants);
- ***Escalabilidade:*** uma solução de monitoramento precisa ser escalável em vários aspectos. Ela precisa escalar para um grande número de agentes de monitoramento, de notificação de eventos, de tenants, de recursos (virtuais e físicos, servidores, elementos de rede e aplicações), e de tipos de informação de monitoramento;

- ***Dinamismo:*** sistemas de monitoramento multi-tenancy precisam suportar o dinamismo que é inerente a um ambiente multi-tenancy. Esse dinamismo decorre da rápida e frequente adição e remoção de tenants no datacenter. Além disso, a alocação de recursos para os tenants também está em constante mudança e o conjunto de recursos que está sendo monitorado também pode mudar;

- ***Simplicidade:*** o sistema de monitoramento deveria ser simples em dois aspectos: primeiro, a interface para monitoramento do sistema precisa ser fácil de entender e usar. Segundo, o sistema precisar ser fácil de instalar e manter, tanto para quem gerencia o datacenter quanto para os consumidores. Uma API simples é desejável, uma vez que facilita o trabalho de desenvolvedores que criam soluções de controle e monitoramento; e

- ***Abrangência:*** esse requisito aborda a necessidade de uma infraestrutura única de monitoramento que deve ser utilizável para todos os tipos de informação de monitoramento, não importa o que está relacionado ao recurso ou qual o significado que ele transmite. Essa abrangência aplica-se a múltiplos aspectos: tipos de dados, fontes de notificação e tenants.

Já Cheng et al. [11] propõem um mecanismo de controle que monitora a qualidade de serviço por tenant, detecta situações anormais e dinamicamente ajusta o uso de recursos baseado nos parâmetros de SLA definidos para o tenant. Nesse trabalho os autores apresentam sua proposta e realizam um estudo experimental para avaliar os resultados.

2.7.7. Performance

Esse assunto está diretamente ligado à monitoramento de aplicações multi-tenancy, isso porque aplicações são monitoradas para que se possa verificar se a mesma está executando em um nível de performance compatível com a SLA definida para o cliente. Desde 2008 temos um bom número de trabalhos que estudam esse problema, dentre eles estão propostas de arquitetura, frameworks, métodos, técnicas, ferramentas e experimentos, que utilizam as mais variadas abordagens. O primeiro trabalho encontrado relacionado à performance foi o de Wang et al. [58], onde os autores apresentam os principais padrões de implementação de aplicações multi-tenancy que tratam dos aspectos de isolamento e segurança, e avaliam a performance desse padrões através de uma série de experimentos e simulações.

Para permitir que um provedor de serviço ofereça diferentes níveis de performance baseado no nível de SLA definido para um tenant específico, Lin et al.[38] propõe uma solução que utiliza um regulador de performance baseado no controle de feedback. O regulador possui uma estrutura hierárquica, na qual um controlador de alto nível gerencia a taxa de admissão de requisições para prevenir sobrecarga e um controlador de baixo nível que gerencia os recursos alocados pelas requisições admitidas para alcançar um nível específico de diferenciação de serviço entre os tenants que compartilham os mesmos recursos. Um protótipo dessa abordagem é implementado utilizando Tomcat e MySQL, e ao final são realizados alguns experimentos para validar a proposta.

Outro ponto importante é o isolamento de performance para prevenir potenciais anomalias de comportamento de um tenant, que possam afetar a performance de outros tenants que compartilham os mesmos recursos. Li et al. [36] propõem o SPIN (Service Performance Isolation Infrastructure) que visa permitir um abrangente compartilhamento de recursos em ambientes multi-tenancy. Uma vez que alguns tenants agressivos podem interferir na performance de outros, SPIN fornece um relatório de anomalias, identifica os tenants agressivos, e fornece um mecanismo de moderação para eliminar o impacto negativo em outros tenants. Durante sua pesquisa os autores implementaram um protótipo do SPIN e realizaram alguns experimentos para demonstrar sua eficiência.

Um desafio interessante na área de gerenciamento de performance é entender e predizer performance de sistemas. Durante a realização desse mapeamento foram encontradas duas abordagens relacionadas a esse assunto. Schaffner et al. [51] apresentam um estudo sobre a automação de tarefas operacionais em clusters multi-tenancy de bancos de dados em memória orientados a coluna. Foi desenvolvido um modelo para predizer se a alocação de um tenant em um servidor no cluster, levaria a uma violação no tempo de resposta esperado. Já Ahmad e Bowman [1] realizaram um estudo experimental para entender e predizer a performance de sistemas, nesse trabalho

os autores sugerem uma abordagem de máquina de aprendizado que usa dados monitorados para entender a performance do sistema.

Guo et al. [20] apresentam um framework que possui um componente específico para isolamento de performance. Segundo os autores os objetivos principais do isolamento de performance incluem dois aspectos. Primeiro, evitar que o comportamento (potencialmente ruim) de um tenant afete o comportamento de outro, de maneira imprevista. Segundo, evitar a injustiça entre tenants em termos de performance de uso. Para alcançar esse objetivo os autores sugerem um padrão de isolamento de performance híbrido, baseado em várias técnicas apresentadas em seu trabalho.

2.7.8. Segurança

Confiança é um dos maiores desafios que influenciam a ampla aceitação de SaaS. Na ausência de confiança em SaaS, segurança e privacidade de dados figuram entre os principais e mais importantes assuntos relacionados a arquitetura multi-tenancy. Foram encontrados durante esta pesquisa alguns trabalhos publicados nos últimos 2 anos que estão relacionados a esse assunto.

Um assunto bastante relevante é a avaliação de credibilidade de tenants, que indica quais tenants têm um comportamento que possa prejudicar o funcionamento dos demais. Zhiqiang [46] apresenta um algoritmo de avaliação de credibilidade de tenants baseado na experiência. Essa abordagem visa realizar a detecção e gerenciamento de tenants maliciosos a um baixo custo. De acordo com o histórico de acesso dos tenants, ele pode calcular a credibilidade de tenants, atribuir privilégios de acesso e determinar estratégias de detecção e monitoramento.

De acordo com Li et al. [37], separação de responsabilidade de segurança entre provedores SaaS e consumidores precisa ser suportada em um ambiente de cloud. Arquitetura multi-tenancy baseada em modelo de controle de acesso (MTACM multi-tenancy based access control model) foi projetada para inserir o princípio de separação de responsabilidade de segurança em cloud; essa arquitetura define dois níveis de granularidade no mecanismo de controle de acesso. O primeiro nível está relacionado ao provedor de serviço, que compartilha sua infraestrutura entre vários clientes. O segundo nível é o nível da aplicação onde uma mesma aplicação hospeda informações de vários tenants.

Zhang et al. [64], Calero et al. [10] e Tsai et al. [54] apresentam três abordagens diferentes para implementação de mecanismos de segurança e controle de acesso para aplicações multi-tenancy. Zhang et al. [64] apresentam uma abordagem de controle de acesso baseado em restrições de privacidade customizáveis. Essa abordagem combina criptografia de dados e separação de informação e define três tipos de restrições de privacidade baseado na funcionalidade de customização em aplicações SaaS. Calero et al. [10] descrevem um sistema de autorização para um serviço de middleware em uma PaaS, que suporta controle de acesso baseado em hierarquia de

funções, hierarquia de objetos baseada em caminho e federações. Os autores apresentam também uma arquitetura de sistema de autorização para implementação do modelo.

De acordo com Tsai et al. [54] as abordagens atuais para controle de acesso em cloud não escalam bem para requisitos multi-tenancy porque eles são baseados principalmente em identificadores (IDs) de usuário individual em diferentes níveis de granularidade. No entanto, o número de usuários pode ser enorme e causar um overread significativo no gerenciamento de segurança. RBAC (Role-Based Access Control) tornase atrativo nesse caso pelo fato do número de papéis de usuário em uma aplicação ser significativamento menor, e usuários podem ser classificados de acordo com seus papéis. Os autores propõem um RBAC usando uma ontologia de papéis para arquitetura multitenancy em clouds.

2.7.9. Integração com outros sistemas

De acordo com o cenário apresentado até aqui, é possível perceber que seria muito difícil e caro criar uma aplicação multi-tenancy que fosse tão configurável a ponto de permitir que todos os requisitos do usuário pudessem ser atendidos através de configurações. Mesmo nesse cenário é possível permitir que os próprios usuários implementem parte de suas soluções e as integrem à plataforma multi-tenancy utilizada. Para solucionar esse problema, alguns autores propuseram abordagens para implementar arquitetura multi-tenancy com SOA.

Azeez et al. [5] apresentam uma arquitetura para implementar multi-tenancy no nível de SOA, que habilita usuários a executar seus serviços e outros artefatos SOA em um framework multi-tenancy SOA. Em seu trabalho os autores discutem arquitetura, decisões de projeto, e problemas encontrados, juntamente com potenciais soluções. Diferentes aspectos de arquitetura de sistemas no que se refere a multi-tenancy para SOA também são considerados como: deployment de serviços, envio de mensagens, segurança, execução de serviços e finalmente acesso a dados.

2.8. Considerações Finais

Grandes empresas de TI (Tecnologia da Informação), como HP e IBM têm investido bastante nessa área, isso é comprovado com o fato de vários artigos encontrados durante esse estudo serem escritos por membros dessas empresas. Além disso, algumas empresas também vendem aplicativos que podem ser configurados para executar como SaaS em nuvens privadas; alguns sistemas da SAP por exemplo, podem ser usados como um SaaS oferecido dentro das empresas.

A primeira dificuldade encontrada durante as pesquisas foi identificar quais os requisitos de uma aplicação multi-tenancy que são essenciais para sua implementação. Durante a tentativa de identificá-los, foi possível perceber que duas características chave de multi-tenancy são o auto grau de automação nas atividades de manutenção da aplicação e uma interface amigável para que o usuário possa realizar suas

customizações, reduzindo ao máximo a necessidade de intervenções por parte de desenvolvedores e administradores de sistema.

Durante a elaboração desse trabalho foi possível notar também a existência de muitos trabalhos associados à multi-tenancy e armazenamento de dados. Propostas de métodos, técnicas e até uma proposta de SGBD multi-tenancy foi encontrada. Embora esse seja o campo de multi-tenancy onde mais se encontrou publicações, ainda é um campo onde existem muitos pontos de melhoria, principalmente pelo fato do armazenamento de dados influenciar diretamente no tempo de resposta de todas as operações que manipulam dados neste tipo de ambiente.

Embora vários experimentos já tenham sido realizados com as abordagens existentes na literatura para armazenamento de dados em aplicações multi-tenancy, não foi encontrado um consenso sobre qual abordagem é a melhor. Migração de dados, modelagem dos dados, tempo de resposta, armazenamento distribuído, integração com ferramentas de BI (Business Inteligence), todos esses fatores podem influenciar na escolha de uma abordagem para armazenamento de dados.

Atualmente, novas formas de armazenamento de dados estão surgindo como: HBase[25], Cassandra[26], Hipertable[27], MongoDB[28], CouchDB[29], etc. Esses sistemas armazenam dados de uma forma diferente do usado nos modelos relacionais e compõem um movimento chamado NoSQL. Uma lacuna pouco explorada foi a utilização dessas novas tecnologias no desenvolvimento de aplicações multi-tenancy, dado que bancos de dados NoSQL surgiram como uma solução para a questão da escalabilidade no armazenamento e processamento de grandes volumes de dados[14].

A necessidade de uma nova tecnologia de banco de dados surgiu como consequência da ineficiência dos bancos de dados relacionais em lidar com a tarefa de manipulação de grandes volumes de dados. Vale ressaltar que o modelo relacional foi proposto na década de 70, quando as aplicações de banco de dados caracterizavam-se por lidar com dados estruturados, ou seja, dados que possuem uma estrutura fixa e bem definida, e esse não é o caso de aplicações multi-tenancy que podem ser customizadas [41].

O crescente surgimento de APIs [16] web mostra que a utilização de APIs para permitir que usuários possam criar extensões de aplicações faz muito sentido no contexto de aplicações web e SaaS. Um exemplo de sucesso é o Tweetdeck, uma aplicação cliente do twitter que foi desenvolvida utilizando a API disponibilizada pelo twitter aos desenvolvedores e teve grande aceitação pelos usuários do twitter.com. O valor real da aquisição não foi declarado, mas especula-se algo em torno de 40 milhões de dólares [6]. Uma API web bem definida pode permitir que usuários de uma

[25] http://hbase.apache.org
[26] http://cassandra.apache.org
[27] http://hypertable.com
[28] http://www.mongodb.org
[29] http://couchdb.apache.org

aplicação multi-tenancy possam, eles mesmos, criarem extensões para a aplicação e até fazer disso um negócio.

Outro exemplo de sucesso é o marketplace desenvolvido pela Salesforce, o AppExchange, que possui mais de 1 milhão de aplicações cadastradas, desde extensões da aplicação de CRM do Salesforce até aplicações para outros propósitos. Exemplos como esse mostram quão vantajoso é trazer desenvolvedores independentes para o mercado de SaaS.

A realidade é que para entender o campo de customização de aplicações multitenancy é necessário entender o contexto no qual essas aplicações estão surgindo. Dado que não é possível atender com uma única aplicação a necessidade de todos os clientes, como já foi mencionado anteriormente, enfrenta-se em aplicações multi-tenancy o desafio de permitir de alguma forma que o cliente adicione novas regras e customize a aplicação ou faça extensões que auxiliem na solução de seus problemas específicos. Durante esse trabalho foram encontrados vários estudos relacionados ao tema, muitos deles utilizando metadados, orientação a aspectos, SOA, etc.

Customização de aplicações já é um tema bastante explorado pelos pesquisadores da área de linha de produtos de software. Basicamente podemos considerar que a customização de aplicações multi-tenancy é equivalente a variabilidade de linha de produtos de software em tempo de execução. Uma iniciativa de customização de aplicações multi-tenancy tomando como base os conhecimentos de linha de produtos de software é apresentado por Jansen et al. [26].

Dado que SaaS tem como objetivo prover software para uma grande massa de consumidores, o atendimento aos requisitos de alocação de recursos, monitoramento, escalabilidade e performance são requisitos não funcionais que podem até ser ignorados em uma aplicação comum, mas que são de vital importância em aplicações multi-tenancy, principalmente se a aplicação estiver hospedada em um ambiente de cloud onde a tarifação (pagamento) é realizada por consumo. Nesses ambientes uma aplicação mal projetada pode levar a custos operacionais tão altos que podem inviabilizar o funcionamento da aplicação, dado o valor que deverá ser pago para manter a infraestrutura de servidores funcionando.

Por outro lado, em uma contexto onde a alocação de recursos é feita de forma inteligente, o monitoramento indica quais recursos estão ociosos e quais os gargalos da aplicação. Nesse caso, se a elasticidade (capacidade de aumentar e diminuir os recursos de hardware) é realizada de forma automática e inteligente, os custos operacionais tendem a ser diminuidos drasticamente se a aplicação consumir somente o que é estritamente necessário para atender aos requisitos de SLA definidos aos clientes.

Nos últimos anos muitas empresas têm saído do modelo de entrega de software empacotado para o modelo de fornecimento de software na web. Essas aplicações entregues através da web vão desde emails a calendários, sistemas colaborativos, publicações online, processadores de texto simples, aplicações para negócios e

aplicações para uso pessoal. Com o aumento da popularidade de marketing baseado em web, e-commerce e muitos outros serviços baseados em web, mesmo os pequenos negócios têm sido obrigados oferecer seus produtos e serviços na internet para serem competitivos no mercado. A característica de compatilhamento de recursos e diminuição de custos trazidos por multitenancy, têm feito dessa abordagem uma excelente alternativa para prover software de qualidade e com um baixo custo para pequenas e médias empresas.

Por fim, consideramos que a implementação de multi-tenancy não só é viável do ponto de vista de negócios como do ponto de vista técnico, embora ainda existam problemas e pontos de melhoria que devem ser tratados.

2.9. Referências

[1] M. Ahmad and I. T. Bowman. Predicting system performance for multi-tenant database workloads. In Proceedings of the Fourth International Workshop on Testing Database Systems, DBTest '11, pages 6:1–6:6, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0655-3. doi: http://doi.acm.org/10.1145/1988842.1988848. URL http://doi.acm.org/10.1145/1988842.1988848.

[2] Amazon. What is AWS? Amazon Company, 2011. URL http://aws.amazon.com/pt/what-is-aws. Disponível em: <http://aws.amazon.com/pt/what-is-aws>. Acesso em: 10 dez. 2011.

[3] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, SIGMOD '08, pages 1195–1206, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6.doi: http://doi.acm.org/10.1145/1376616.1376736. URL http://doi.acm.org/10.1145/1376616.1376736.

[4] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold. A comparison of flexible schemas for software as a service. In Proceedings of the 35th SIGMOD international conference on Management of data, SIGMOD '09, pages 881–888, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-551-2. doi: http://doi.acm.org/10.1145/1559845.1559941. URL http://doi.acm.org/10.1145/1559845.1559941.

[5] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle. Multi-tenant soa middleware for cloud computing. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pages 458 – 465, july 2010. doi: 10.1109/CLOUD.2010.50.

[6] BBC News. Twitter acquires Tweetdeck app for undisclosed fee. BBC News, 2011. URL http://www.bbc.co.uk/news/technology-13518382. Disponível em: <http://www.bbc.co.uk/news/technology-13518382>. Acesso em: 10 dez. 2011.

[7] D. Berberova and B. Bontchev. Design of service level agreements for software services. In Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing, CompSysTech'09, pages 26:1–26:6, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-986-2. doi: http://doi.acm.org/10.1145/1731740.1731769. URL http://doi.acm.org/10.1145/1731740.1731769.

[8] C.-P. Bezemer and A. Zaidman. Multi-tenant saas applications: maintenance dream or nightmare? In Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), IWPSE-EVOL '10, pages 88–92, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0128-2. doi: http://doi.acm.org/10.1145/1862372.1862393. URL http://doi.acm.org/10.1145/1862372.1862393.

[9] C.-P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. 't Hart. Enabling multi-tenancy: An industrial experience report. In Software Maintenance (ICSM), 2010 IEEE International Conference on, pages 1 –8, sept. 2010. doi: 10.1109/ICSM.2010.5609735.

[10] J. Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. Toward a multitenancy authorization system for cloud services. Security Privacy, IEEE, 8(6):48–55, nov.-dec. 2010. ISSN 1540-7993. doi: 10.1109/MSP.2010.194.

[11] X. Cheng, Y. Shi, and Q. Li. A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on sla. In Pervasive Computing (JCPC), 2009 Joint Conferences on, pages 599 –604, dec. 2009. doi: 10.1109/JCPC.2009.5420114.

[12] F. Chong and G. Carraro. Architecture strategies for catching the long tail. 2006. URL http://msdn2.microsoft.com/en-us/library/aa479069.aspx. Disponível em: <http://msdn2.microsoft.com/en-us/library/aa479069.aspx>. Acesso em: 10 dez. 2011.

[13] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation Volume 2, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1251203.1251223.

[14] D. Diana, Mauricio, Gerosa, and M. Aurélio. Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0. Communications, 2010.

[15] S. Dustdar and W. Schreiner. A survey on web services composition. Int. J. Web Grid Serv., 1:1–30, August 2005. ISSN 1741-1106. doi: 10.1504/IJWGS.2005.007545. URL http://dl.acm.org/citation.cfm?id=1358537.1358538.

[16] A. DuVander. Api growth doubles in 2010, social and mobile are trends. 2011. URL http://blog.programmableweb.com/2011/01/03/api-growth-doubles-in-2010-social-and-mobile-are-trends/.

[17] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi. Zephyr: live migration in shared nothing databases for elastic cloud platforms. In Proceedings of the 2011 international conference on Management of data, SIGMOD '11, pages 301–312, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4. doi: http://doi.acm.org/10.1145/1989323.1989356.

[18] C. Fehling, F. Leymann, and R. Mietzner. A framework for optimized distribution of tenants in cloud applications. In Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD '10, pages 252–259, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4130-3. URL http://dx.doi.org/10.1109/CLOUD.2010.33.

[19] F. Foping, I. Dokas, J. Feehan, and S. Imran. A new hybrid schema-sharing technique for multitenant applications. In Digital Information Management, 2009. ICDIM 2009. Fourth International Conference on, pages 1 –6, nov. 2009. doi:10.1109/ICDIM.2009.5356775.

[20] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao. A framework for native multitenancy application development and management. In E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on, pages 551 –558, july 2007. doi: 10.1109/CEC-EEE.2007.4.

[21] I. S. Harris and Z. Ahmed. An open multi-tenant architecture to leverage smes. European Journal of Scientific Research, 65(4):601–610, 2011.

[22] P. Hasselmeyer and N. d'Heureuse. Towards holistic multi-tenant monitoring for virtual data centers. In Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP, pages 350 –356, april 2010. doi: 10.1109/NOMSW.2010.5486528.

[23] J. Huang. Gartner says worldwide software as a service revenue is forecast to grow 21 percent in 2011. 2011. URL http://www.gartner.com/it/page.jsp?id=1739214. Disponível em: <http://www.gartner.com/it/page.jsp?id=1739214>. Acesso em: 10 dez. 2011.

[24] J. Huang. Amazon s3's amazing growth. Cloud Computing Journal, 2011. URL http://cloudcomputing.sys-con.com/node/1699373. Disponível em: <http://cloudcomputing.sys-con.com/node/1699373>. Acesso em: 10 dez. 2011.

[25] B. Jacob. Introduction to Grid Computing. IBM redbooks. IBM, International Technical Support Organization, 2005. ISBN 9780738494005. URL http://books.google.com.br/books?id=4GSUtgAACAAJ.

[26] S. Jansen, G.-J. Houben, and S. Brinkkemper. Customization realization in multitenant web applications: case studies from the library sector. In Proceedings of the 10th international conference on Web engineering, ICWE'10, pages 445–459, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-13910-8, 978-3-642-13910-9. URL http://dl.acm.org/citation.cfm?id=1884110.1884147.

[27] A. Jasti, P. Shah, R. Nagaraj, and R. Pendse. Security in multi-tenancy cloud. In Security Technology (ICCST), 2010 IEEE International Carnahan Conference on, pages 35 –41, oct. 2010. doi: 10.1109/CCST.2010.5678682.

[28] J. Jing and J. Zhang. Research on open saas software architecture based on soa. In Computational Intelligence and Design (ISCID), 2010 International Symposium on, volume 2, pages 144 –147, oct. 2010. doi: 10.1109/ISCID.2010.125.

[29] C. M. Judd, J. F. Nusairat, J. Shingler, M. Moodie, J. Ottinger, J. Pepper, F. Pohlmann, B. Renow-clarke, D. Shakeshaft, M. Wade, and et al. Beginning groovy and grails from novice to professional. Specialist, page 440, 2008. URL http://books.google.com/books?hl=en&lr=&id=9AxFrcvawvAC&oi=fnd&pg=PR15&dq=Beginning+Groovy+and+Grails:+From+Novice+to+Professional&ots=AWQ-JVaL8&sig= BzgKgJMF9jI7E27n9UwfxSRT4OY.

[30] L. Kleinrock. An internet vision: the invisible global infrastructure. Ad Hoc Networks, 1(1):3–11, 2003. URL http://linkinghub.elsevier.com/retrieve/pii/S157087050300012X.

[31] H. Koziolek. Towards an architectural style for multi-tenant software applications. Proc Software Engineering 2010 SE10 Fachtagung des GIFachbereichs Softwaretechnik, To appear:81–92, 2010.nURL http://www.koziolek.de/docs/Koziolek2010-SE.pdf.

[32] H. Koziolek. The sposad architectural style for multi-tenant software applications. In Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on, pages 320 –327, june 2011. doi: 10.1109/WICSA.2011.50.

[33] T. Kwok and A. Mohindra. Resource calculations with constraints, and placement of tenants and instances for multi-tenant saas applications. In Proceedings of the 6th International Conference on Service-Oriented Computing, ICSOC '08, pages 633–648, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-89647-0. URL http://dx.doi.org/10.1007/978-3-540-89652-4_57.

[34] T. Kwok, T. Nguyen, and L. Lam. A software as a service with multi-tenancy support for an electronic contract management application. In Services Computing,

2008. SCC ’08. IEEE International Conference on, volume 2, pages 179 –186, july 2008. doi: 10.1109/SCC.2008.138.

[35] S. Lehnen. Understanding the basic building blocks of sales force crm. 2011. URL http://www.salesforce.com/customer-resources/learning-center/details/best-practices/understanding-the-basic-building.jsp. Disponível em: <http://www.salesforce.com/customer-resources/learning-center/details/best-practices/understanding-the-basic-building.jsp>. Acesso em: 28 dez. 2011.

[36] X. H. Li, T. C. Liu, Y. Li, and Y. Chen. Spin: Service performance isolation infrastructure in multi-tenancy environment. In Proceedings of the 6th International Conference on Service-Oriented Computing, ICSOC ’08, pages 649–663, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-89647-0. doi: http://dx.doi.org/10.1007/978-3-540-89652-4_58. URL http://dx.doi.org/10.1007/978-3-540-89652-4_58.

[37] X.-Y. Li, Y. Shi, Y. Guo, and W. Ma. Multi-tenancy based access control in cloud. In Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on, pages 1 –4, dec. 2010. doi: 10.1109/CISE.2010.5677061.

[38] H. Lin, K. Sun, S. Zhao, and Y. Han. Feedback-control-based performance regulation for multi-tenant applications. In Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on, pages 134 –141, dec. 2009. doi: 10.1109/ICPADS.2009.22.

[39] H. Lin, K. Sun, S. Zhao, and Y. Han. Feedback-control-based performance regulation for multi-tenant applications. In Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on, pages 134 –141, dec. 2009. doi: 10.1109/ICPADS.2009.22.

[40] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu. Live migration of virtual machine based on full system trace and replay. In Proceedings of the 18th ACM international symposium on High performance distributed computing, HPDC ’09, pages 101–110, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-587-1. URL http://doi.acm.org/10.1145/1551609.1551630.

[41] B. F. Lóscio, H. R. d. Oliveira, and J. C. d. S. Pontes. Nosql no desenvolvimento de aplicações web colaborativas. VIII Simpósio Brasileiro de Sistemas Colaborativos, 2011.

[42] P. Mell and T. Grance. The nist definition of cloud computing. National Institute of Standards and Technology, 53(6):50, 2009. URL http://www.nist.gov/ itl/cloud/upload/cloud-def-v15.pdf.

[43] A. Mesbah and A. van Deursen. Crosscutting concerns in j2ee applications. In Web Site Evolution, 2005. (WSE 2005). Seventh IEEE International Symposium on, pages 14 – 21, sept. 2005. doi: 10.1109/WSE.2005.4.

[44] R. Mietzner, T. Unger, R. Titze, and F. Leymann. Combining different multitenancy patterns in service-oriented applications. In Proceedings of the 2009 IEEE International Enterprise Distributed Object Computing Conference (edoc 2009), EDOC '09, pages 131–140, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3785-6. doi: http://dx.doi.org/10.1109/EDOC.2009.13. URL http://dx.doi.org/10.1109/EDOC.2009.13.

[45] T. Neil. 12 standard screen patterns, 2010. URL http://designingwebinterfaces.com/designing-web-interfaces-12-screen-patterns. Disponível em:<http://designingwebinterfaces.com/designing-web-interfaces-12-screen-patterns>. Acesso em: 10 dez. 2011.

[46] Z. Nie. Credibility evaluation of saas tenants. In Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on, volume 4, pages V4–488 –V4–491, aug. 2010. doi: 10.1109/ICACTE.2010.5579322.

[47] Nitu. Configurability in saas (software as a service) applications. In Proceedings of the 2nd India software engineering conference, ISEC '09, pages 19–26, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-426-3. doi: http://doi.acm.org/10.1145/1506216.1506221. URL http://doi.acm.org/10.1145/1506216.1506221.

[48] B.-T. Oh, H.-S. Won, and S.-J. Hur. Multi-tenant supporting online application service system based on metadata model. In Advanced Communication Technology (ICACT), 2011 13th International Conference on, pages 1173 –1176, feb. 2011.

[49] Z. Pervez, S. Lee, and Y.-K. Lee. Multi-tenant, secure, load disseminated saas architecture. In Advanced Communication Technology (ICACT), 2010 The 12th International Conference on, volume 1, pages 214 –219, feb. 2010.

[50] C. Richardson. Orm in dynamic languages. Commun. ACM, 52(4):48–55, Apr.2009. ISSN 0001-0782. doi: 10.1145/1498765.1498783. URL http://doi.acm.org/10.1145/1498765.1498783.

[51] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier. Predicting in-memory database performance for automating cluster management tasks. In Data Engineering (ICDE), 2011 IEEE 27th International Conference on, pages 1264–1275, april 2011. doi: 10.1109/ICDE.2011.5767936.

[52] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang. Native support of multitenancy in rdbms for software as a service. In Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT '11, pages

117–128, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0528-0. doi: http://doi.acm.org/10.1145/1951365.1951382. URL: http://doi.acm.org/10.1145/1951365.1951382.

[53] L. Shwartz, Y. Diao, and G. Grabarnik. Multi-tenant solution for it service management: A quantitative study of benefits. In Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on, pages 721 –731, june 2009. doi: 10.1109/INM.2009.5188879.

[54] W.-T. Tsai and Q. Shao. Role-based access-control using reference ontology in clouds. In Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on, pages 121 –128, march 2011. doi: 10.1109/ISADS.2011.21.

[55] W.-T. Tsai, Q. Shao, and J. Elston. Prioritizing service requests on cloud with multitenancy. In e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on, pages 117 –124, nov. 2010. doi: 10.1109/ICEBE.2010.38.

[56] W.-T. Tsai, X. Sun, Q. Shao, and G. Qi. Two-tier multi-tenancy scaling and load balancing. In e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on, pages 484 –489, nov. 2010. doi: 10.1109/ICEBE.2010.103.

[57] Q. H. Vu, M. Lupu, and B. C. Ooi. Peer-to-peer computing. Media, 2010. URL http://www.springerlink.com/index/10.1007/978-3-642-03514-2.

[58] Z. H. Wang, C. J. Guo, B. Gao, W. Sun, Z. Zhang, and W. H. An. A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing. In Proceedings of the 2008 IEEE International Conference on e-Business Engineering, pages 94–101, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3395-7. doi: 10.1109/ICEBE.2008.60. URL http://dl.acm.org/citation.cfm?id=1471605.1472222.

[59] C. Weiliang, Z. Shidong, and K. Lanju. A multiple sparse tables approach for multitenant data storage in saas. In Industrial and Information Systems (IIS), 2010 2nd International Conference on, volume 1, pages 413 –416, july 2010. doi: 10.1109/ INDUSIS.2010.5565824.

[60] C. D. Weissman and S. Bobrowski. The design of the force.com multitenant internet application development platform. In Proceedings of the 35th SIGMOD international conference on Management of data, SIGMOD '09, pages 889–896, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-551-2. URL http://doi.acm.org/10.1145/1559845.1559942.

[61] W. Xue, L. Qingzhong, and K. Lanju. Multiple sparse tables based on pivot table for multi-tenant data storage in saas. In Information and Automation (ICIA), 2011 IEEE International Conference on, pages 634 –637, june 2011. doi: 10.1109/ICINFA.2011.5949071.

[62] C. S. Yeo, R. Buyya, H. Pourreza, R. Eskicioglu, P. Graham, F. Sommers, and G. Computing. Cluster computing: High-performance, high-availability, and highthroughput processing on a network of computers. Communication, pages 1–24, 2006. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.66.1453.

[63] D. Yuanyuan, N. Hong, W. Bingfei, and L. Lei. Scaling the data in multi-tenant business support system. In Knowledge Engineering and Software Engineering, 2009. KESE '09. Pacific-Asia Conference on, pages 43 –46, dec. 2009. doi: 10.1109/KESE.2009.20.

[64] K. Zhang, Y. Shi, Q. Li, and J. Bian. Data privacy preserving mechanism based on tenant customization for saas. In Multimedia Information Networking and Security, 2009. MINES '09. International Conference on, volume 1, pages 599 –603, nov. 2009. doi: 10.1109/MINES.2009.256.

[65] X. Zhang, B. Shen, X. Tang, and W. Chen. From isolated tenancy hosted application to multi-tenancy: Toward a systematic migration method for web application. In Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on, pages 209 –212, july 2010. doi: 10.1109/ICSESS.2010.5552407.

[66] Y. Zhang, Z. Wang, B. Gao, C. Guo, W. Sun, and X. Li. An effective heuristic for on-line tenant placement problem in saas. In Web Services (ICWS), 2010 IEEE International Conference on, pages 425 –432, july 2010. doi: 10.1109/ICWS.2010. 65.

[67] J. Zheng, X. Li, X. Zhao, X. Zhang, and S. Hu. The research and application of a saas-based teaching resources management platform. In Computer Science and Education (ICCSE), 2010 5th International Conference on, pages 765 –770, aug. 2010. doi: 10.1109/ICCSE.2010.5593500.

Capítulo

3

Desafios em Cloud computing: Armazenamento, Banco de Dados e BIG Data

Rodrigo Elia Assad[1], Marco André Santos Machado[1,2], Paulo Fernando Almeida Soares[1,2], Anderson Fonseca e Silva[1] , Thiago Jamir e Silva[1,2], Vinicius Cardoso Garcia[2] Fernando Mota Trinta[3], Silvio Romeiro Lemos Meira[2]

[1]USTO.RE {assad@usto.re, marco@usto.re, paulo@usto.re, anderson@usto.re, tjamir@usto.re}
[2]Universidade Federal de Pernambuco { vcg@cin.ufpe.br, srlm@cin.ufpe.br }
[3]Universidade Federal do Ceará { fernando.trinta@lia.ufc.br }

Abstract

With increasing connectivity speed and Web systems evolution, emerges the Internet systems, which are more commonly called cloud computing. It´s designates a support platform that provides: management, on-demand use, fitness requirements, rational use of resources and automation of processes related to creation of infrastructure. On this context emerge systems of data storage in the cloud, database scalability and data search and retrieval now called as BIGDATA. This short course addresses these exemplifying how we implement and use such platforms.

Resumo

Com o aumento da velocidade de conectividade e evolução dos sistemas WEB, começa a surgir os sistemas de Internet, que mais comumente são chamados de Computação nas Nuvens. Este termo designa uma plataforma de suporte a sistemas de software que provê aos seus usuários: gerenciamento, uso sob demanda, adequação às necessidades, racionalização do uso dos recursos e automação dos processos relacionados a criação de infra-estruturas. Neste contexto surge os sistemas de armazenamento de dados em nuvem, escalabilidade de banco de dados e busca e recuperação que hoje chamamos de BIGDATA. Este minicurso aborda estes temos exemplificando como implementar e utilizar tais plataformas.

3.1 Introdução

Em 1990, Tim Berners-Lee deu início à implementação das linguagens, dos protocolos e dos artefatos de software necessários à construção da *World Wide Web*, ou simplesmente web, como conhecemos hoje [W3C, 2000]. Duas décadas depois, a web ganhou proporções mundiais, atingindo mais de um bilhão e meio de pessoas em todos os continentes[30]. Ao longo desta história, a mesma passou por constantes evoluções, permitindo uma variedade de aplicações não antes previstas por Tim Berners-Lee, as quais compõem o momento atual da web. A **Erro! Fonte de referência não encontrada.** apresenta a distribuição dos usuários de internet por continente em 2009.

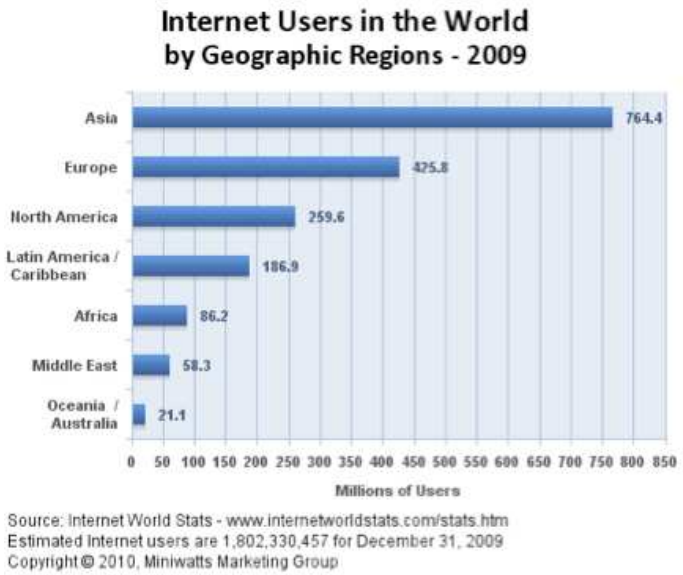


Figura 3.1 - Distribuição dos usuários de internet por continente em 2009

Adicionalmente, de acordo com um estudo do *The Economist*[31] no início de 2010 estimava-se que a quantidade de informação gerada no planeta, no ano de 2010, seria duas vezes maior do que o armazenamento disponível para guardá-la, caracterizando assim um "*dilúvio informacional*". Isto faz com que haja um grande aumento na demanda de volume de recursos para o apoio as tarefas de elicitação, compartilhamento, manipulação e exploração de grandes conjuntos de dados, bem como para o desenvolvimento e execução de serviços de apoio à Pesquisa.

A *McKinsey Quartely*[32] [Bughin et al., 2010], jornal de negócios da *McKinsey & Company*, divulgou artigo em que relaciona 10 tendências de tecnologia que devem estar no radar dos estrategistas das empresas para geração de novos modelos de negócios. O artigo faz um alerta aos altos executivos das empresas sugerindo que eles devem pensar estrategicamente sobre como preparar as organizações para o novo ambiente desafiador que se aproxima com o crescimento de tecnologias como computação em nuvem, internet das coisas, colaboração em escala entre outras.

[30] URL: http://bit.ly/MM7XnF, Acessado em 05/07/2012
[31] URL: http://econ.st/MM7YrC, Acessado em 05/07/2012
[32] URL: http://bit.ly/9stJjH, Acessado em 05/07/2012

Ainda segundo o artigo da *McKinsey*, a tecnologia continua a evoluir rapidamente. *Facebook*[33], em pouco mais de dois curtos anos, quintuplicou de tamanho a uma rede que atinge mais de 500 milhões de usuários. Mais de 4 bilhões de pessoas ao redor do mundo já utilizam telefones celulares, e para 450 milhões de pessoas a Web é uma experiência completamente móvel. A maneira como utilizamos tecnologia, como Computação em Nuvem (*Cloud Computing*) e virtualização, redistribuem os custos de tecnologia, criam novas formas para os indivíduos consumirem produtos e serviços e torna possível que empresários e empresas pensem em novos modelos de negócios que muitos consideram impossíveis como, por exemplo, marketing contextual.

A *McKinsey* também alerta que as empresas não devem se limitar a compreender as 10 tendências abaixo descritas. Elas precisam também pensar estrategicamente sobre como adaptar a gestão e estruturas organizacionais para atender as novas demandas proporcionadas por essas 10 tendências, pois muitas dessas tendências exigirão a necessidade de atravessar fronteiras organizacionais tradicionais fazendo com que líderes criem conexões entre as equipes em diferentes e espalhados cantos da organização, buscando maior interdisciplinaridade de forma que toda a empresa compreenda a necessidade de explorar plenamente essas tendências. Isso implica em transformar as organizações em pequenos laboratórios capazes de testar mais rapidamente e aprender mais rapidamente em pequena escala e depois expandir o sucesso rapidamente.

As 10 tendências citadas pela *McKinsey* [Bughin et al., 2010] são:

1. **Distribuição estimula a Co-Criação** (*Distributed cocreation moves into the mainstream*): Inspirados pelo pioneirismo da *Wikipedia* e uma comunidade de desenvolvedores de software de código aberto, a capacidade de organizar as comunidades da Web para desenvolver , comercializar produtos e serviços de apoio passou das margens da prática empresarial para o *mainstream*, onde nos dias de hoje diversas empresas tem pautado seu negócio (ou o relacionamento com seus clientes, no mínimo) que tem como objetivo redes abertas de compartilhamento de conhecimento;
2. **Transformar a organização em uma rede** (*Making the network the organization*): Muitas empresas estão indo além fronteiras estaduais, nacionais ou até continentais, construindo redes flexíveis que se estendem além das fronteiras internas e externas da própria organização. Barreiras culturais, burocráticas ou até processuais em torno de determinados silos organizacionais podem impedir que os gestores acessem os melhores talentos em toda a organização para resolver problemas críticos com soluções criativas. Utilizando tecnologias da Web, por exemplo, para expandir o acesso a especialistas de todo o mundo, as empresas podem montar comunidades de inovação entre as unidades de negócios em silos, acelerando a prestação de serviços e melhorando simultaneamente a qualidade destes serviços e do relacionamento com todos os seus *stakeholders*;
3. **Colaboração em escala** (*Collaboration at scale*): Com o advento da Internet e da WEB 2.0, o número de pessoas que realizam trabalho, muitas vezes colaborativamente, de conhecimento (voltado para inovação e criação de riqueza em cada setor da economia) tem crescido muito mais rapidamente do que os

[33] URL: http://www.facebook.com, Acessado em 05/07/2012

trabalhadores de produção de bens. Como resultado, é crescente o interesse em tecnologias de colaboração que prometem melhorar o rendimento e a eficácia desses trabalhadores;

4. **A crescente "Internet das Coisas"** (*The growing 'Internet of Things'*): Dispositivos, eletrodomésticos, embalagens, *containers*, entre outros objetos (ou genericamente "*coisas*"), embutidos de sensores, atuadores e capacidades de comunicações, em breve serão capazes de absorver e transmitir informações em grande escala e, em alguns casos, adaptar-se e reagir às mudanças no ambiente automaticamente. Estes ativos "inteligentes" podem tornar os processos mais eficientes, fornecendo novos recursos e novos modelos de negócio das empresas;
5. **Experimentação e Grandes Volumes de Dados** (*Experimentation and big data*): O volume de dados vem crescendo a taxas nunca antes vistas, e tecnologias para a captura e análise de informações estão amplamente disponíveis a preços cada vez mais baixos. Muitas empresas estão levando o uso desses dados para novos níveis, utilizando a Tecnologia da Informação e Comunicação (TIC) para suportar a experimentação constante de negócios que orientem as decisões e para testar novos produtos, modelos de negócios e inovações na experiência do cliente. Esta tendência tem o potencial de conduzir a uma transformação radical na pesquisa, inovação e na forma como as empresas conduzem seus negócios. Isso exigirá que TIC e Negócios não sejam mais compreendidos como áreas estanques e separadas, e sim como um único problema, ou seja, TIC não é mais parte do negócio, na verdade, é necessário um modelo de negócio que utilize a TIC como instrumento do negócio;
6. **Cabeamento para um mundo sustentável** (*Wiring for a sustainable world*): Mesmo com os marcos regulatórios continuando a evoluir, gestão ambiental e sustentabilidade são temas prioritários da agenda empresarial. Além do mais, a sustentabilidade está se tornando rapidamente uma importante métrica de desempenho corporativo, que os *stakeholders*, incluindo até mesmo os mercados financeiros começaram a dar mais atenção. TIC desempenha um duplo papel neste debate: ela é uma fonte significativa de emissões para o ambiente e um fator essencial de muitas estratégias para mitigar os danos ambientais. A pesquisa da *McKinsey* mostra, no entanto , que o uso da TIC em áreas como as redes de energia inteligentes, edifícios eficientes e melhor planejamento de logística poderia eliminar cinco vezes as emissões de carbono que a indústria produz;
7. **Imagine TUDO como Serviço** (*Imagining anything as a service*): Cada vez mais podemos afirmar que TIC é serviço e não produto. Consumidores desejam pagar apenas pelo uso e, dessa forma, evitar grandes gastos e futuras dificuldades de compra e manutenção de um produto. Na indústria de TIC, o crescimento de "*cloud computing*" [Armbrust et al., 2009; Galán & Sampaio, 2009; Leavitt, 2009; Smith et al., 2009; Sun, 2009] exemplifica esta mudança, além de software como serviço (SaaS) [Turner et al., 2003], que permite às organizações acesso a serviços como, por exemplo, a gestão de relacionamento com clientes. A aceitação pelos consumidores dos serviços de *cloud* baseada na Web para tudo, desde e-mail até armazenamento de vídeos e fotos, tende naturalmente a tornar-se universal, e as companhias estão seguindo essa tendência;

8. **A era do modelo de negócios multilaterais** (*The age of the multisided business model*): O maior exemplo desse modelo de negócios é o Google, que vende publicidade oferecendo serviços de TIC que permitem a busca e organizam a informação na Web. Outro exemplo, citado no artigo da *McKinsey*, é a empresa *Spiceworks*[34] oferece gerenciamento de aplicativos de TIC para 950 mil usuários, sem qualquer custo, enquanto vende espaço publicitário para empresas de B2B que querem acesso a profissionais de TIC. Quanto mais pessoas migrarem para atividades on-line, o valor dos modelos de negócio multilaterais são ampliados por estes efeitos proporcionados pela rede, sendo que quanto maior o número de usuários, utilizando os serviços gratuitamente, mais valioso o serviço torna-se para todos os clientes;
9. **Inovando na base da pirâmide** (*Innovating from the bottom of the pyramid*): A adoção de tecnologia é um fenômeno global, e a intensidade de seu uso é particularmente impressionante nos mercados emergentes. A pesquisa da *McKinsey* mostrou que os modelos de negócios disruptivos surgem quando a tecnologia se combina com condições extremas de mercado, tais como a demanda dos clientes por preços muito baixos, pouca infraestrutura, fornecedores de difícil acesso e curvas de baixo custo para o talento;
10. **Expandir serviços públicos** (*Producing public good on the grid*): É preciso entender que o custo de transação das empresas, como o trânsito caótico para movimentação das pessoas, cresce cada vez mais tornando-se inviável do ponto de vista macroeconômico e isso tende a se agravar, uma vez que cerca de metade da população mundial vive em áreas urbanas, e essa parte é projetada para atingir 70 por cento em 2050. Políticas públicas criativas, que incorporem as novas tecnologias, poderiam ajudar a aliviar as tensões econômicas e sociais da densidade populacional. Londres, Singapura e Estocolmo têm utilizado recursos inteligentes para gerenciar o congestionamento do tráfego em seus núcleos urbanos, e muitas cidades em todo o mundo estão implementando essas tecnologias para melhorar a confiabilidade e a previsibilidade dos sistemas de transporte de massa. Da mesma forma, redes de água inteligentes (*networked water smart grids)* serão fundamentais para responder à necessidade de água limpa [Baumgarten & Chui, 2009]. A plena exploração do potencial da tecnologia na esfera pública significa re-imaginar a forma como os serviços públicos são criados , entregues e gerenciados;

Torna-se imperativo que as instituições considerem adotar tais tendências de tecnologia para sua indústria de Tecnologia de Informação e Comunicação, tanto para atender de forma efetiva sua demanda interna, quanto para tornar-se uma potência exportadora de sistemas de software que, com as características citadas, teriam qualidade e rapidez superior aos seus concorrentes.

Como uma plataforma chave de fornecimento de serviço na área de computação de serviços [Zhang et al., 2007], Computação em Nuvem oferece ambientes que permitem o compartilhamento de recursos em termos de infraestruturas escaláveis, middleware e plataformas de desenvolvimento de aplicações, algumas com alto valor agregado ao negócio. Os modelos de operação podem incluir modelos de utilidade de

[34] URL: http://www.spiceworks.com/, acessado em 28/08/2010

pagamento pelo uso (*pay-as-go*), serviços de infraestrutura grátis com plataformas de serviço com valor agregado, serviços de infraestruturas baseados em taxas composto por serviços de aplicações com valor agregado ou de serviços gratuitos para os fornecedores, mas compartilhando as receitas geradas pelos consumidores.

Tipicamente, existem quarto tipos de recursos que podem ser provisionados e consumidos através da Internet [Zhang et al., 2008]. Estes recursos podem ser compartilhados entre os usuários por meio da economia de escala. O provisionamento é uma forma de compartilhar recursos com solicitantes através da rede. Um dos objetivos principais da computação em nuvem é aproveitar a Internet ou uma Intranet para provisão de recursos para os usuários.

O primeiro tipo de recursos são os recursos de infraestrutura, que incluem o poder de computação, armazenamento e fornecimento de máquinas. Por exemplo, o Amazon EC2 oferece interface de serviço web para facilmente solicitar e configurar capacidades online [Amazon EC2, 2012]. O serviço *Xdrive Box* oferece armazenamento online para os usuários [Xdrive, 2012]. O *Microsoft SkyDrive* oferece serviço de armazenamento gratuito, com um modelo integrado *offline* e *online* que mantém a privacidade relacionada com arquivos em discos rígidos, e permite às pessoas acessar esses arquivos remotamente [Microsoft, 2012]. Na área de compartilhamento de poder computacional, a computação em Grid tomou como seu foco principal a utilização de clusterização e tecnologias de computação paralela para compartilhar o poder computacional com os outros usuários (consumidores do serviço), com base no agendamento de tarefas quando os computadores estão ociosos.

O segundo tipo de recursos em computação em nuvem são os recursos de software, incluindo middlewares e recursos de desenvolvimento. O middleware consiste em sistemas operacionais centrados na nuvem, servidores de aplicação, banco de dados, entre outros. Os recursos de desenvolvimento compreendem plataformas de projeto, ferramentas de desenvolvimento, teste e implantação além de projetos de referência baseados em outros projetos de código aberto [Zhang and Zhou, 2009].

O terceiro tipo de recursos em computação em nuvem são os recursos de aplicação. As empresas de TIC estão migrando gradualmente aplicações e dados para Internet (nuvem). As aplicações de software são entregues através do modelo de Software como Serviço (*SaaS – Software as a Service*) ou *mashups*[35] de aplicações de valor agregado. Por exemplo, o Google tem utilizado computação em nuvem para oferecer aplicações Web para comunicação e colaboração [Google, 2012]. O pacote de escritório do Google, o Google Docs, moveu gradativamente para a Web as aplicações de produtividade anteriormente existentes apenas no modelo de licenciamento por máquina. Desta forma, fica claro que desenvolvendo uma arquitetura aberta de computação em nuvem reutilizável e customizável possibilitando um efetivo ambiente de desenvolvimento de aplicações como serviço é a chave para o compartilhamento (ou distribuição) de aplicações por meio da Internet.

O quarto tipo de recursos na computação em nuvem são os processos de negócio. Algumas aplicações podem ser compartilhadas (publicadas) como serviços públicos (*utilities*), ou seja, por meio de sub-processos com baixo acoplamento ou tarefas dentro de processos de negócio de clientes. O compartilhamento de processos de

[35] URL: http://pt.wikipedia.org/wiki/Mashup, Acessado em 09/07/2012

negócios é a terceirização de aplicativos orientados a negócios que dá suporte ao reuso, composição e provisionamento de aplicações.

Neste contexto, está claro o porque de grandes empresas estarem investindo recursos de esforço neste novo modelo computacional. Principalmente as fornecedoras de serviço como Google, Amazon e Microsoft que possuem hoje suas plataformas de serviços que permitem o aluguel de sua infraestrutura de máquinas para que terceiros possam hospedar aplicações, repassando questões como administração de máquinas, controle de escalabilidade, dentre outras, para as empresas contratadas.

Atualmente, boa parte da economia mundial já é baseada em sistemas de TIC, e a demanda por software vai aumentar ainda mais nas próximas décadas [Osterweil, 2007]. Estas mudanças se darão devido a fatores como o uso de software em larga escala por meio de infraestrutura de serviços, novos tipos de aplicações de integração de negócios e novas plataformas, além de combinação de serviços oferecidos via Web e dispositivos móveis.

3.1.1 Sistemas de Informação e os modelos de computação em nuvem

Sistemas de informação compreendem pessoas, componentes físicos (hardware) e lógicos (software) que, ligados por redes de comunicação, permitem a transformação de dados brutos em informação e conhecimento. A quantidade e diversidade dos dados requerem cuidados adicionais sobre seu armazenamento, como por exemplo, segurança, confiabilidade, entre outros, além de suporte a procedimentos comuns como: backups, salvamento de arquivos e sua recuperação.

Um sistema ao utilizar o novo paradigma de computação em nuvem, o faz por meio de três modelos básicos:

a) Software como Serviço (*SaaS* - do inglês, *Software as a Service*): Neste modelo, o provedor da infraestrutura de computação em nuvem cuida de todos os aspectos relacionados à disponibilidade, manutenção e adição de novas funcionalidades para possíveis clientes. Exemplos são *Google Docs*, *OfficeLive (+)* e o *SalesForce* que fornecem versões online de editores de texto, planilhas de cálculo e demais programas de escritório;
b) Plataforma como Serviço (*PaaS* - do inglês, *Platform as a Service*): Nesta outra abordagem, o provedor oferta uma plataforma que será utilizada pela aplicação do usuários final. Exemplos: *Amazon S3*, *AllMyData*, *Windows Azure* que fornecem sistemas de armazenamento;
c) Infraestrutura como Serviço (*IaaS*, do inglês, *Infrastructure as a Service*): O provedor oferece um ambiente virtualizado e gerenciável pelo usuário. Exemplos: *Amazon EC2*, *CloudBR* que fornecem sistemas virtualizados para processamento sobre demanda.

Em qualquer um dos modelos apresentados, um propósito básico é a racionalização da utilização dos recursos computacionais por uma organização ou um conjunto de organizações. A ideia é que clientes em um modelo de computação em nuvem possam melhor administrar o dimensionamento de seus recursos de infraestrutura, software, aplicação e/ou de negócio. Neste modelo, aplicações tornam-se escaláveis de forma automática, onde os encargos relacionados ao seu correto funcionamento são cobrados de acordo com a demanda das próprias aplicações.

No caso especifico de uma plataforma oferecida como serviço no paradigma de computação em nuvem, deve ser entendida com uma infraestrutura que permita a seus usuários a utilização de serviços por meio de uma interface pública, estabelecida através de uma interface de programação - *API* (do inglês, *Application Programming Interface*). Ao utilizar esta *API*, cada participante pode utilizar e oferecer recursos para os demais integrantes da nuvem, permitindo uma melhor utilização dos recursos compartilhados, otimização de processamento, armazenamento, transferência, dentre outros fatores.

Os dados de uma plataforma em modelo de computação em nuvem podem ser distribuídos entre os vários participantes da nuvem, aumentando a sua disponibilidade, assim como otimizando o uso de recursos subutilizados. Porém, há problemas a serem enfrentados, tais como escalabilidade horizontal segurança, disponibilidade.

Em geral, a infraestrutura de uma plataforma de computação em nuvem é construída sob uma tradicional forma da topologia cliente/servidor, onde o lado servidor da nuvem é composto por um alto número de máquinas que criam a abstração de um único provedor de serviços em cluster. Apesar de ser um modelo consolidado, existem problemas associados ao custo da manutenção desta infraestrutura, bem como do consumo de banda, energia, dentre outros fatores. Como alternativa para este modelo, ao longo dos anos tem se visto o crescimento de soluções baseados em topologia ponto a ponto - P2P (do inglês, *Peer to Peer*), onde não há a figura de um host ou conjunto de hosts que assume o papel de servidor ou coordenador. Os exemplos mais conhecidos de aplicações P2P são as redes de compartilhamento de arquivos, como *gnutella* ou *e-donkey*.

Aplicações P2P lidam melhor com questões de escalabilidade e tolerância a falhas do que aquelas no modelo cliente/servidor, que também contam com um custo adicional devido à complexidade necessária para o gerenciamento das buscas e roteamento dos dados entre seus participantes.

Unir os conceitos de redes P2P e computação em nuvem cria um cenário muito interessante de pesquisa para novas aplicações. Busca-se neste casamento trazer o melhor dos dois mundos. De computação em nuvem, temos **(i)** a alta disponibilidade de informações, **(ii)** a elasticidade das aplicações, ou seja, a alocação ou liberação de recursos de forma automática, **(iii)** a garantia de serviços e **(iv)** o acesso ubíquo através da rede. De redes P2P, tem-se a descentralização e a possibilidade de serviços mais tolerantes a falhas.

3.2 Fundamentação teórica

Este projeto tem por objetivo explorar os conceitos apresentados anteriormente, principalmente os ligados a abordagens P2P, para compartilhar recursos de armazenamento que estejam ociosos e disponíveis pelos nós (*peers*) que compõem uma rede para formar uma solução de backup, sincronização e compartilhamento de dados por meio de uma plataforma de computação em nuvem. Esta solução se baseia ainda no paradigma orientado a serviço e tem por princípio os principais benefícios e práticas de reutilização de software. Doravante chamaremos de USTO.RE a plataforma confiável em computação em nuvem para a realização de backups, sincronização e compartilhamento de dados que seja eficiente, segura e totalmente distribuída.

3.2.1 Reutilização e Arquitetura Orientada a Serviços

A ideia de reutilização de software não é nova. Em 1968, durante a conferência da NATO de engenharia de software, Mcllroy [McIlroy, 1968], em seu revolucionário artigo, intitulado: "*Mass Produced Software Components*", apresentou a tese que dizia: "*a indústria de software é fracamente fundada e um aspecto dessa fraqueza é a ausência de uma indústria de componentes de software*". O autor propunha investigar técnicas de produção em massa de software, conforme algumas ideias da construção industrial.

Mcllroy idealizava, com a indústria de componentes, encontrar catálogos de rotinas padronizadas, classificadas por precisão, robustez, performance; aplicar rotinas existentes no catálogo para uma variedade de máquinas; e, ter confidência na qualidade das rotinas.

Durante cerca de quatro décadas, diversas pesquisas têm sido apresentadas e discutidas em conferências, como: *International Conference on Software Engineering* (ICSE), *International Conference on Software Reuse* (ICSR) e, em periódicos, na área de engenharia de software, tentando seguir as direções apontadas por Mcllroy.

Dentre as principais razões para a distância entre as ideias de Mcllroy e o atual estado da arte, está o atraso da indústria de software, comparado a indústria de hardware em termos de princípios de manufatura e catálogo de padrões; a mudança cultural dos desenvolvedores, que, normalmente, utilizam o verbo desenvolver, ao contrário do verbo reutilizar; o fato dos atuais sistemas de repositórios raramente considerarem processos de Engenharia de Domínio ou Linha de Produtos para o desenvolvimento dos artefatos, além de processos de garantia de qualidade antes de publicar os artefatos para reutilização.

Há muito vem se formando uma consciência na comunidade de engenharia de software de que, a fim de se obter produtos com alta qualidade e que sejam economicamente viáveis, torna-se extremamente necessário um conjunto sistemático de processos, técnicas e ferramentas. Nesse conjunto, a reutilização está entre as técnicas mais relevantes. Ao longo dos últimos anos, muitas técnicas têm sido propostas para favorecer o reuso. Dentre estas técnicas, destacam-se: a engenharia de domínio, frameworks, padrões, o Desenvolvimento Baseado em Componentes (DBC) e, atualmente, Linhas de Produto de Software que tem atraído o maior foco de pesquisa e desenvolvimento [Clements & Northrop, 2001].

Segundo Papazoglou et al. [Papazoglou et al., 2007], o paradigma de desenvolvimento orientado a serviços utiliza serviços para o suportar o desenvolvimento efetivo, com baixo custo e alta interoperabilidade de aplicações distribuídas. Serviços são entidades autônomas e independentes de plataforma que podem ser descritas, desenvolvidas, publicadas e descobertas. Os serviços realizam funções que podem variar de simples requisições, como serviços meteorológicos, até executar processos de negócios sofisticados, que requerem relacionamentos ponto a ponto entre camadas de consumidores e serviços [Erl, 2008]. Quando tratamos de serviços, o processo natural de criação dos serviços passa por uma fase inicial conhecida como modelagem de serviços.

Segundo Bell [Bell, 2008], a disciplina de modelagem é um campo de conhecimento que oferece boas práticas, padrões e procedimentos para facilitar as

atividades de desenvolvimento orientado a serviços durante o ciclo de vida de um serviço. Em geral, as atividades de modelagem orientada a serviços são realizadas antes da construção e implantação de serviços. As disciplinas de modelagem orientada a serviços identificam os processos chave nos quais o negócio e as pessoas envolvidas com o desenvolvimento devem estar engajados para produzir os artefatos de projeto e implementação.

Anos atrás, com o objetivo de analisar o mercado de engenharia de software baseada em componentes, o *Software Engineering Institute* (SEI) analisou a indústria de software, com ênfase em componentes. O estudo [Bass et al., 2000], conduzido durante o período de setembro de 1999 até fevereiro de 2000, examinou componentes de software sob a perspectiva técnica e de negócios.

A partir dessa pesquisa, um distinto conjunto de inibidores para adoção de componentes de software foi identificado. Com base nestes dados e de entrevistas realizadas, o SEI identificou os seguintes inibidores para a adoção de componentes, apresentados aqui em ordem decrescente de importância:

1. Carência de componentes disponíveis;
2. Carência de padrões estáveis para a tecnologia de componentes;
3. Carência de componentes certificados; e
4. Carência de um método efetivo para construção de sistemas baseados em componentes.

Com o crescimento do paradigma de desenvolvimento de software orientados a serviços, podemos perfeitamente fazer uma associação destes inibidores para o reuso de componentes, com a reutilização de serviços, uma vez que os próprios serviços podem ser vistos como componentes de software (ativos) reutilizáveis.

Vale ressaltar também, que em uma recente pesquisa envolvendo o estado da arte e direções de pesquisa na área de desenvolvimento orientado a serviços, Michael Papazoglou, principal investigador da área, em conjunto com outros pesquisadores, constataram que um dos grandes desafios está na definição de processos voltados para o desenvolvimento de software orientado a serviço [Papazoglou et al., 2007]. Assim, analisando as visões das duas comunidades, tanto de reuso de software e linhas de produto, quanto de desenvolvimento orientado a serviços, pode-se identificar a relevância do tema e a oportunidade para o projeto.

Quando tratamos do desenvolvimento de software, principalmente em relação à reutilização (seja por meio do DBC ou de linha de produtos de software) um ponto chave é a arquitetura de software. No caso do paradigma orientado a serviços, novos desafios são identificados por meio das arquiteturas de software orientadas a serviços.

Uma arquitetura orientada a serviço estabelece um modelo arquitetural que visa aumentar a eficiência, agilidade e produtividade das empresas com a utilização de serviços representando os seus processos de negócio. Como uma tecnologia de arquitetura de software, uma implementação de uma arquitetura orientada a serviço pode consistir da combinação de tecnologias, produtos e *APIs* [Erl, 2008]. Web Services [Stal, 2002], uma opção tecnológica para se implementar SOA, tem sido utilizado na indústria nos últimos anos com o objetivo de construir aplicações que reutilizam serviços disponíveis na rede [Stal, 2002]. Esta tecnologia permite disponibilizar funcionalidades através de serviços, utilizando padrões conhecidos da Web, tais como, SOAP, XML e HTTP, provendo uma maior interoperabilidade entre as aplicações.

Desta maneira, o desenvolvimento de aplicações distribuídas através de serviços se torna mais rápido, aumentando a qualidade e diminuindo o custo e tempo de entrega do produto.

3.2.2 Arquiteturas Peer-to-Peer (P2P)

Os grandes responsáveis pelo impulso e popularização dos sistemas distribuídos P2P, foram o Napster [Napster, 2012] e Gnutella [Gnutella, 2012]. Além de protagonizar inúmeras questões judiciais quanto a direitos autorais e pirataria, foi através destas ferramentas que se evidenciou o potencial da tecnologia no que diz respeito a compartilhamento de recursos sem desprender maiores investimentos em hardware. Desde a época dos precursores a tecnologia sofreu diversas mudanças.

Pesquisas mostram que os sistemas tendem a sofrer um processo de descentralização contínuo, onde o primeiro grande avanço se deu com o modelo cliente-servidor. Este consiste em uma máquina central no qual disponibiliza algum serviço que é consumido por máquinas com um hardware com bem menos recursos. A segunda forma de descentralização mostra o surgimento das aplicações P2P, onde podem ser desenvolvidas sobre sua forma completamente descentralizada, denominada de pura, ou um modelo híbrido, onde se desenvolvem estruturas de controle centralizadas e a utilização de recursos descentralizados. Cada um dos modelos de descentralização possui suas vantagens e desvantagens conforme se pode acompanhar na sessão seguinte.

3.2.3 Arquiteturas Puras

As redes denominadas puras possuem como principal característica a não existência de nenhum tipo de controle central. Todo o funcionamento se dá com uso de um algoritmo descentralizado onde é possível localizar peers e/ou serviços [Schollmeier & Rudiger, 2002].

Esta localização é feita fazendo uso da técnica de enchente (flooding) [Schollmeier & Rudiger, 2002], onde a mensagem é enviada a todos os computadores diretamente ligados ao emissor e cada máquina que recebe a mensagem faz o mesmo. Para evitar que a rede entre em colapso (loop), um tempo de vida é atribuído a mensagem para que caso esta não chegue a seu destino seja descartada. Os pontos negativos aqui são:

- Algumas máquinas podem não receber a mensagem, negando assim um serviço que estaria disponível em tese;
- Há um grande volume de mensagens enviadas até que a mesma encontre a máquina de destino.

A técnica mais utilizada na implementação de arquiteturas puras que gerou um avanço significativo na área é o *Distributed Hash Tables* (DHT) [Balakrishnan et al., 2003]. Utilizada nas seguintes ferramentas: *pStore* [Oliveira, 2007], *Pastiche* [Mattos, 2005] e *Oceanstore* [Kubiatowicz et al., 2000], *PeerStore* [Landers et al., 2004] e *BitTorrent* [BitTorrent, 2012]. As DHTs pertencem à classe de sistemas distribuídos descentralizados e oferecem recursos de localização similar às *hash tables* (chave, valor). Um par de chave e valor é armazenado na DHT e qualquer participante da rede pode acessar o valor desejado apenas informando a chave associada. As primeiras

quatro especificações de *DHTs* (*Pastry* [Rowstron & Druschel, 2001], *Chord* [Stoica et al., 2001], *CAN* [Ratnasamy et al., 2001] e *Tapestry* [Zhao et al., 2004]) surgiram quase simultaneamente no ano 2001, depois disso, sua utilização se popularizou em aplicações destinadas ao compartilhamento de arquivos na internet. Um estudo mais aprofundado sobre as diferentes implementações de DHT pode ser encontrado em [Balakrishnan et al., 2003]. As *DHTs* possuem como principais características:

- Descentralização: os próprios nós criam e mantêm o sistema, sem a necessidade de um servidor;
- Escalabilidade: o sistema suporta a participação de um crescente número de nós simultaneamente;
- Tolerância a erros: o sistema deve ser confiável, mesmo com nós entrando e saindo continuamente da rede.

Para alcançar os objetivos supracitados, as redes *DHTs* utilizam a técnica de que um nó na rede deve estar em contato direto com apenas uma fração de todos os nós participantes. Isso reduz o custo de manutenção quando um nó entra ou sai do sistema.

Para armazenar um arquivo numa *DHT*, primeiro se calcula uma chave (geralmente o código *hash SHA-1* [FIPS, 1995] do seu nome ou do seu conteúdo), em seguida esse arquivo é enviado para a rede até ser encontrado o conjunto de nós responsáveis por seu armazenado. Para recuperá-lo, uma mensagem é enviada informando a chave do arquivo desejado, essa mensagem é encaminhada até um nó que possua o conteúdo desejado e então o mesmo é enviado como resposta. A seguir descreveremos uma das implementações de *DHT* mais utilizadas, o *Chord* [Stoica et al., 2001].

3.2.4 Arquitetura Chord

A implementação de *DHT* utilizando *Chord* se destaca pela sua simplicidade em oferecer uma única operação em que dada uma determinada chave, ela será mapeada para um nó na rede. Segundo Stoica [Stoica et al., 2001], sua arquitetura foi projetada para se adaptar facilmente a entrada e saída de novos *peers* na rede. Embora seja uma implementação que possui apenas uma tarefa (associar chaves a nós da rede), o *Chord* possibilita ao desenvolvimento de aplicações p2p uma série de benefícios:

- Balanceamento de carga, as chaves são distribuídas igualmente entre os nós da rede;
- Descentralização, nenhum nó é considerado mais importante que outro;
- Escalabilidade, o uso do *Chord* é viável mesmo com uma grande quantidade de nós;
- Disponibilidade, ajuste de sistema automático a entrada e saída de novos nós, fazendo que um nó sempre esteja visível na rede; e,
- Flexibilidade, não é necessário seguir nenhuma regra para o nome das chaves.

Para atribuir chaves aos nós, o *Chord* usa uma variação de *consistent hashing* [Karger et al., 1997] que cuida do balanceamento de carga, uma vez que cada nó tende a receber naturalmente um mesmo número de chaves. Em trabalhos anteriores ao *Chord* usando *consistent hashing* se assumia que cada nó possuísse conhecimento de todos os

outros, diferentemente, no *Chord* cada nó precisa ter conhecimento de apenas uma fração dos outros nós na rede.

Supondo uma rede de n nós, um *peer* mantém informações sobre *O (log n)* nós, e para encontrar um determinado nó na rede basta que ele possua apenas uma referência válida.

3.2.5 Arquiteturas Híbridas

Em alguns sistemas P2P é necessária a identificação dos peers conectados na rede. Para tal, sistemas como o *OurBackup* [Oliveira, 2007], que fazem uso de redes sociais para backup, é utilizado em sua arquitetura um servidor responsável pela autenticação dos usuários, manutenção da rede e dos *metadados* onde as cópias estão armazenadas. Pode-se ressaltar que a utilização de servidores não é obrigatória para a localização dos *peers* e dos *metadados*, podendo essa ser feita utilizando as DHT mencionadas anteriormente.

Nesses sistemas, o papel do servidor está em oferecer uma interface aos *peers* da rede com diversas operações tais como: autenticação do usuário; manipulação dos dados armazenados por outros *peers*, adicionar, remover, excluir, atualizar; manipulação dos usuários cadastrados no sistema; localização dos *peers* e relacionamento entre eles; dentre outras tarefas que venham a atender os requisitos do sistema. Em todos os casos geralmente é verificado se o cliente possui permissão para executar tais operações.

Essa centralização de informações pode trazer prejuízos de escalonamento no sentido de que sempre vai existir uma exigência maior do servidor à medida que se aumenta o número de requisições, usuários, *metadados* ou quaisquer outras informações delegadas ao serviço. Porém, sistemas como o *eDonkey* [Aidouni et al., 2009] se mostraram bastante eficientes quanto ao gerenciamento centralizado de informações dessa natureza. Se necessário esse escalonamento também pode ser resolvido com a utilização de clusters ou grids no lado do servidor, fazendo-se mais importante a disponibilização da interface de comunicação com a máquina cliente.

3.2.6 Detalhamento de ferramentas existentes

Em um estudo conduzido pela equipe deste projeto de pesquisa, foi constatado que existem diversas ferramentas implementadas e funcionais, ou seja, que fazem backup, sincronização, compartilhamento e recuperação em um ambiente P2P. Entretanto, foi constatado que existem apenas mecanismos que aumentam a disponibilidade. Em nenhuma das ferramentas investigadas foi identificado um o mecanismo que quantifique e garanta, em termos percentuais, a disponibilidade da recuperação. As alternativas que possuíam essa características só atendiam a esta necessidade por meio da adição de um componente extra (i.e. ferramentas, suítes ou ambientes).

Vignatti [Vignatti et al., 2009] faz um levantamento das soluções existentes e propõe um mecanismo de armazenamento digital a longo prazo baseado na escolha de repositórios confiáveis. Em essência a contribuição do trabalho é a preservação do dado, ou seja, que o mesmo não se corrompa durante sua transmissão, armazenamento, falha de hardware, entre outros. Segundo o autor muito pouca ênfase foi dada à recuperação dos itens.

O *BackupIT* [Loest et al., 2009] é uma ferramenta completa e funcional que faz uso de um mecanismo denominado de *Byzantine Quorum Systems* [Malkhi & Reiter, 1997] para controlar integridade e disponibilidade. No entanto, apenas há um aumento das chances de uma recuperação bem sucedida.

Colaço et al. [Colaço et al., 2008] especificou e desenvolveu um mecanismo de priorização dos arquivos mais usados por um usuário para diminuir o tempo que o sistema fica indisponível para o mesmo. A ideia central é que os arquivos mais importantes sempre estejam com uma disponibilidade maior que os demais.

O *pStore* [Batten et al., 2002] combina um algoritmo de roteamento chamado *Distributed Hash Table* (*DHT*) com quebra de arquivos em tamanhos fixos e uma técnica de controle de versão. A ideia é bastante completa, há suporte para encriptação de arquivos mantendo assim a privacidade nas máquinas clientes, bem como manter versões de arquivos já salvos, reaproveitando os pedaços de arquivos que se repetem em versões anteriores.

Samsara estimula a troca de recurso de maneira igualitária sem uso de uma terceira entidade no sistema para intermediar a comunicação [Cox & Noble, 2003]. O mecanismo utilizado para garantir a troca justa de recursos é um esquema punitivo para o *peer* da rede que por ventura venha a perder dados, atividade esta que é típica de usuários que querem utilizar e não disponibilizar recursos.

CleverSafe Dispersed Storange [Cleversafe, 2012] é uma aplicação *open source* desenvolvida para a plataforma *CentOS*. Faz uso do algoritmo de dispersão (*IDA*) e recuperação da informação injetada na rede. A ferramenta garante maior escalabilidade por não ter servidores centralizados e uma garantia de disponibilidade de acesso aos dados de 99,9999 (12 noves) por cento. Porém, a ferramenta faz uso de máquinas em ambientes controlados.

Dibs [Martinian, 2012] é uma ferramenta *open source* programada em *Python* que faz *backup* incremental, ou seja, de maneira inteligente o arquivo não é salvo novamente, apenas as modificações feitas no mesmo. Sua interação acontece mediante fechamento de contrato entre as partes, onde é comparado espaço mínimo disponível em disco.

Resilia [Meira, 2005] é um protótipo de sistema de backup que combina p2p com compartilhamento secreto e seguro de arquivos, faz uso de um mecanismo de replicação para aumentar a disponibilidade e permite a reconstrução de backups perdidos por falhas de nodos que compõe a rede.

Dropbox [Dropbox, 2012] é uma ferramenta de armazenamento de arquivos no qual faz uso da ideia de computação na nuvem, mantém um conjunto de servidores ligados em rede, com ambiente controlado. O salvamento dos arquivos dos usuários é feito por intermédio de um software instalado na máquina do cliente. Assim como o *CleverSafe,* o *Dropbox* também possui toda sua infraestrutura em ambiente controlado.

Disco virtual RNP [GTAR, 2012] é uma ferramenta Web que utiliza de servidores conectados montando um *Grid* de dados. O salvamento dos arquivos dos usuários é feito por intermédio de um software instalado na máquina do cliente ou Web.

Conforme pode ser constatado nesta sessão, diversos trabalhos vêm sendo desenvolvidos na área de salvamento e backup de dados distribuídos, A tabela abaixo

contextualiza de maneira mais detalhada os trabalhos levantados no estado da arte, onde foram tabuladas as principais características dos sistemas implementados para esse tipo de problema.

1. Garantia de disponibilidade: um mecanismo que quantifique e forneça estatisticamente que a restauração vai estar disponível em determinado horário.
2. Aumento de disponibilidade: mecanismo de redundância no qual aumente as chances de uma restauração bem sucedida.
3. Segurança: mecanismo de segurança dos dados salvos na rede de backup e compartilhamento.

   Inspeção dos dados: mecanismo de checagem que garante que os dados copiados em um *peer* não foram perdidos.
4. *Backup* Incremental: mecanismo no qual a similaridade entre diferentes versões é usada para salvar recursos de rede.
5. Ponto Central de controle: se o sistema teve sua arquitetura projetada para ser um sistema P2P híbrido.
6. Controle de espaço de armazenagem: Mecanismo no qual controla o espaço utilizado de *backup* e compartilhamento por usuário, evitando que o usuário utilize mais do que necessário e doe seus recursos.
7. Baseado em redes sociais: Método usado em busca de atingir maior índice de confiabilidade na rede. A proposta baseia-se em formar uma rede social de amigos (*friend-to-friend* [Li & Dabek, 2006]) e com isso evitar que seus arquivos sejam apagados.
8. Sistema de Armazenamento: Se o referido sistema tem em seu conceito ser utilizado como armazenamento ou backup.

3.2.7 Demais trabalhos relacionados

Além das ferramentas já apresentadas, várias são as soluções existentes para armazenamento de dados em nuvem no modelo de *DaaS*. Em geral, tais soluções são proprietárias e fechadas, o que dificulta uma análise técnica mais minuciosa sobre cada proposta. Analisando algumas das principais soluções existentes, como a *Amazon S3*, o *Megastore*, o *MSFSS* e o *Hadoop Distributed File System* (HDFS), nota-se que invariavelmente, todas utilizam uma estratégia comum baseada na replicação dos dados em diversos servidores, onde o número de servidores envolvidos na replicação de dados, varia de 3 a 7 vezes dependendo da solução. Esta replicação forçada é feita porque não há automação nos processos de replicação que possa garantir 99,9999999\% de disponibilidade [DeCandia et al., 2007].

O *Amazon S3* (*Simple Storage Service*) [Amazon, 2012] é o sistema de armazenamento por trás de muitos dos serviços da Amazon.com como o *Dynamo* [DeCandia et al., 2007], que recentemente teve seu banco de dados No-SQL lançado com o nome de *DynamoDB*[36]. O *DynamoDB* possui uma política de *backup* onde são feitas no mínimo três cópias do mesmo dado, onde duas são feitas na mesma zona e

[36] URL: http://bit.ly/xBc8mD, Acessado em 05/07/2012

uma terceira em uma zona externa, de modo a aumentar a disponibilidade da informação. Segundo dados de 2009, o sistema armazenava cerca de 40 bilhões de arquivos de 400.000 clientes. Seus desafios incluíam garantia de disponibilidade e o gerenciamento de falhas.

Outra plataforma, o *Megastore*, é um sistema desenvolvido com foco nos serviços *online* interativos [Baker et al., 2011]. Ele foi desenvolvido e é usado pela Google há muito tempo e manipula mais de 3 bilhões de escritas e 20 bilhões de transações de leitura diariamente, além de também armazenar um valor próximo a um *petabytes* de dados primários nos seus *datacenters* espalhados globalmente.

Já o *MSFSS* é um sistema de arquivos distribuído de alta escalabilidade e flexível, desenvolvido para armazenar uma grande quantidade de pequenos arquivos [Yu et al., 2007]. A sua arquitetura é dividida em três componentes principais: *single master*, *storage nodes* e os *metadata servers*. Todo arquivo armazenado no sistema recebe um identificador de 128 bits (*FID - File ID*) gerado a partir da data de criação do arquivo. Esses arquivos são armazenados no sistema de arquivos local de cada *storage node*. Dentre seus requisitos, o *MSFSS* dá suporte à replicação de dados, assegura a consistência entre as réplicas e executa rápida sincronização para identificar réplicas obsoletas. O *FSI* (*File System Interface*), biblioteca que dá acesso ao sistema para os clientes externos, aloca uma grande quantidade de *FIDs* em processos *batch* e os deixa armazenados em memória local, para uso posterior [Yu et al., 2007]. Por padrão, o *MSFSS* armazena duas réplicas por arquivo, mas esse valor pode ser configurado para garantir maior disponibilidade e confiabilidade. Ele usa qualquer uma das réplicas no processo de leitura. Já no processo de escrita, todas as réplicas devem ser atualizadas automaticamente. Para minimizar a latência, os dados são armazenados próximo ao usuário e as réplicas próximas umas das outras. Ele separa os grupos por regiões e cria 3 ou 5 réplicas para os *datacenters*.

O *HDFS* é um sistema de arquivos utilizado pelo *Hadoop* [Shvachko et al., 2010] e seus projetos relacionados. *Hadoop* é um *framework* para análise e transformação de grande quantidade de dados usando *MapReduce* [Dean & Ghemawat, 2008]. Uma das principais características do *Hadoop* é o particionamento de dados e a computação dos mesmos utilizando milhares de *hosts*. O *cluster* do *Hadoop* no *Yahoo!*, por exemplo, alcançou 25.000 servidores (com *cluster* de até 3.500 servidores) e armazenavam 25 *Petabytes* de dados [Shvachko et al., 2010].

Os trabalhos aqui apresentados apresentam como característica comum a replicação excessiva de informação. Este fato reside na necessidade de se garantir uma alta disponibilidade dos dados, aumentar a confiabilidade e desempenho da recuperação de informações. Porém, o uso de réplicas não só traz benefícios. Elas criam um tráfego extra de dados na rede, tráfego este que pode ser tão excessivo ao ponto de se tornar o principal gargalo de uma aplicação, além de gerar um custo da banda tão alto que pode tornar a aplicação inviável [Yang et al., 2006].

Nestas soluções, infere-se que as mesmas exigem a aquisição de infraestrutura dedicada para prover o serviço e para garantir a réplica dos dados. De forma a melhorar este cenário, o objetivo do usto.re é permitir a criação de uma nuvem para armazenamento de dados utilizando tecnologia P2P que se baseia na disponibilidade de cada *peer* para, dinamicamente, criar federações e definir a quantidade de replicações de

cada *chunk*[37] de cada arquivo. Esta abordagem permite que em ambientes onde os *peers* tenham maior disponibilidade (taxa menor de falhas), se tenha uma replicação menor e, no caso de um ambiente mais dinâmico como a Intranet de uma empresa, se tenha uma replicação maior. A partir da próxima seção, este artigo detalha nossa proposta.

Tabela 3.1 - Tabela comparativa entre as soluções de backup/armazenamento p2p

	pStore	Pastiche	OurBackup	OceanStore	PAST	BackupIT	Samsara	Resilia	Clever Safe	Dibs	Drop box	Disco RNP
Garantia de disponibilidade												
Aumento de disponibilidade	x	x	x	x	x	x	x	x	x	x	x	x
Segurança	x	x	x	x	x	x	x	x	x	x	x	x
Inspeção dos dados				x		x	x	x	x			
Incremental	x	x	x				x	x		x	x	
Ponto Central de controle			x									x
Controle de espaço de armazenagem		x	x	x	x		x		x		x	x
Baseado em redes sociais			x			x						
Sistema de Armazenamento				x	x				x		x	x

3.3 USTO.RE: Um Sistema P2P confiável para Data Cloud

Como vemos, o principal desafio em se implementar um sistema que realize o armazenamento e backup remoto é garantir que o os dados estarão disponíveis, pois em todas as soluções analisadas existe a possibilidade de falhas acontecerem e os servidores estarem indisponíveis, tornando a replicação e a montagem de infraestruturas redundantes e consideravelmente custosas como a única solução. Neste caso, se

[37] *Chunk* é um fragmento de informação, ou seja, uma pequena "parte" de um arquivo armazenado. Arquivos são divididos em *chunks*, que por sua vez, são replicados em pontos de armazenamento na infraestrutura da nuvem.

utilizarmos os conceitos trazidos por sistemas P2P e computação em nuvem podemos implementar um sistema de backup menos custoso desde que consigamos resolver o problema dos dados estarem disponíveis quando os usuários solicitarem.

Sendo assim, uma abordagem que permite solucionar este problema é replicar os dados em outros *peers* de forma a utilizar seus recursos ociosos sem haver a necessidade de se adquirir novos recursos.

Consequentemente temos que definir a quantidade de *peers* em que devemos replicar os dados de forma que os mesmos estejam disponíveis quando for necessário realizar a sua recuperação. Se conseguirmos prever a probabilidade de falha/indisponibilidade de um *peer* na rede poderemos calcular a quantidade utilizada de *peers* para a replicação de dados. A formalização matemática desta solução está descrita abaixo:

$$f(t)=\lambda e^{-\lambda t}$$

Equação 1: Equação exponencial

$$\lambda = \frac{1}{MTTF}$$

Equação 2: Taxa de falhas

Onde λ é a taxa de falhas e MTTF é o tempo médio entre falhas. A probabilidade de falha para função exponencial no intervalo de 0 a t é dado pela equação F(t):

$$F(t) = \int_0^t \lambda e^{-\lambda t}\, dt = 1 - e^{-\lambda t}$$

Equação 3: Probabilidade de Falha

A **confiabilidade**, ou seja, probabilidade do sistema não falhar num instante de tempo t pode ser obtido pela função R(t), fazendo o complemento da probabilidade de falha:

$$R(t) = 1 - F(t)$$

Equação 4: Formulação geral da confiabilidade

Substituindo a equação 4 em 5, se tem:

$$R(t) = 1 - \left(1 - e^{-\lambda t}\right) = e^{-\lambda t} \quad (1.5)$$

Equação 5: Confiabilidade no modelo exponencial

Uma vez que se têm as taxas de falhas contabilizadas nos módulos, *Avaibility* e *SeftAvaibility* pode-se, a partir da equação 6, saber a probabilidade de um *Peer* não falhar em determinada hora. É sabido ainda que várias máquinas podem estar ligadas e que há um número mínimo de máquinas configurável no qual um pedaço tem que ser distribuído. Para se obter um resultado geral do algoritmo, é necessário fazer as somas das probabilidades. Meyer [Meyer, 1995] define a soma das probabilidades como sendo:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Equação 6: Soma das probabilidades

Por indução matemática em 7, se chega à equação geral de soma de probabilidade:

$$P(A_1 \cup A_2 \cup \ldots \cup A_K) = \sum_{i=0}^{K} P(Ai) - \sum_{i<j=2}^{k} P(Ai \cap Aj) + \sum_{i<j<r=3}^{k} P(Ai \cap Aj \cap Ar) + \cdots + (-1)^{k-1} P(A_1 \cap A_2 \cap \ldots \cap A_K)$$

Aplicando em conjunto as equações 6 e 7 o algoritmo consegue calcular a probabilidade de um conjunto de máquina não falhar em determinado horário e com base neste calculo definir com um percentual de confiança de que a mesma esteja ligada e funcionando.

Uma vez que já foram abordadas as entradas, saídas e a fundamentação matemática do algoritmo, na sessão seguinte será apresentado o processamento de toda a informação. Foi optado por deixar o código da forma como foi implementado por ter sido usado assim na prova de conceito.

3.3.1 Arquitetura do sistema

A arquitetura do usto.re foi especificada tendo como meta atingir um conjunto de atributos de qualidade peculiares aos sistemas de armazenamento distribuído e que atendesse aos principais benefícios oferecidos pelas aplicações P2P, a saber:

- **Escalabilidade**: atendendo a possibilidade de explorar recursos de hardware de um grande número de máquinas (*hosts*) conectados à rede, principalmente por meio do uso racional de recursos ociosos em grandes corporações.
- **Otimização** de iterações (troca de mensagens): a "distância" entre os *peers* que interagem no sistema tem impacto no desempenho geral na latência das interações individuais, desta forma a carga do tráfego da rede também sofre um impacto negativo por esta latência. Neste contexto, a escolha pela estratégia do uso de federações visa agrupar os *peers* relacionados e reduzir esta latência.
- **Disponibilidade**: sistemas P2P são baseados em computadores livres para se "unir" ao sistema a qualquer momento, bem como "sair" dele. Além disso, as conexões não são gerenciadas pelo sistema ou alguma autoridade que garanta a conectividade e a qualidade do serviço, neste contexto, uma estratégia de garantia de disponibilidade do serviço deve ser implementada considerando as características básicas de um sistema de armazenamento com alta disponibilidade, bem como as restrições que uma rede P2P impõe.
- **Segurança dos dados**: em se tratando de armazenamento de dados, políticas de segurança e proteção devem ser efetivamente adotadas de modo a garantir a privacidade e autenticidade dos usuários e dados do sistema.

Para detalhar o funcionamento do usto.re, esta seção apresenta uma visão geral da arquitetura do USTO.RE, seus os principais componentes, dependências e relacionamentos, conforme ilustrado na Figura 2. O USTO.RE é composto por um conjunto de cinco componentes, dispostos em uma arquitetura P2P híbrida estruturada e em três camadas. São eles: **(i)** *Super Peers Rendezou Relay* ou simplesmente *Super Peers*, **(ii)** Servidores, **(iii)** *Proxies*, **(iv)** Bancos de Dados Relacionais e Não-SQL e **(v)** *Simple Peers*. Estes componentes possuem funcionalidades distintas e interagem como um sistema distribuído descentralizado híbrido, de modo semelhante a uma rede P2P, onde cada nó realiza tanto funções de servidor quanto de cliente. Os componentes agrupam-se dinamicamente como federações de dados, onde os grupos são montados de modo a minimizar a troca de mensagens no sistema.

A Figura 3.2 apresenta o diagrama de sequencia do USTO.RE, onde vemos de maneira geral o processo de registro de *peer*, localização de serviços e como um *simple peer* obtém os dados.

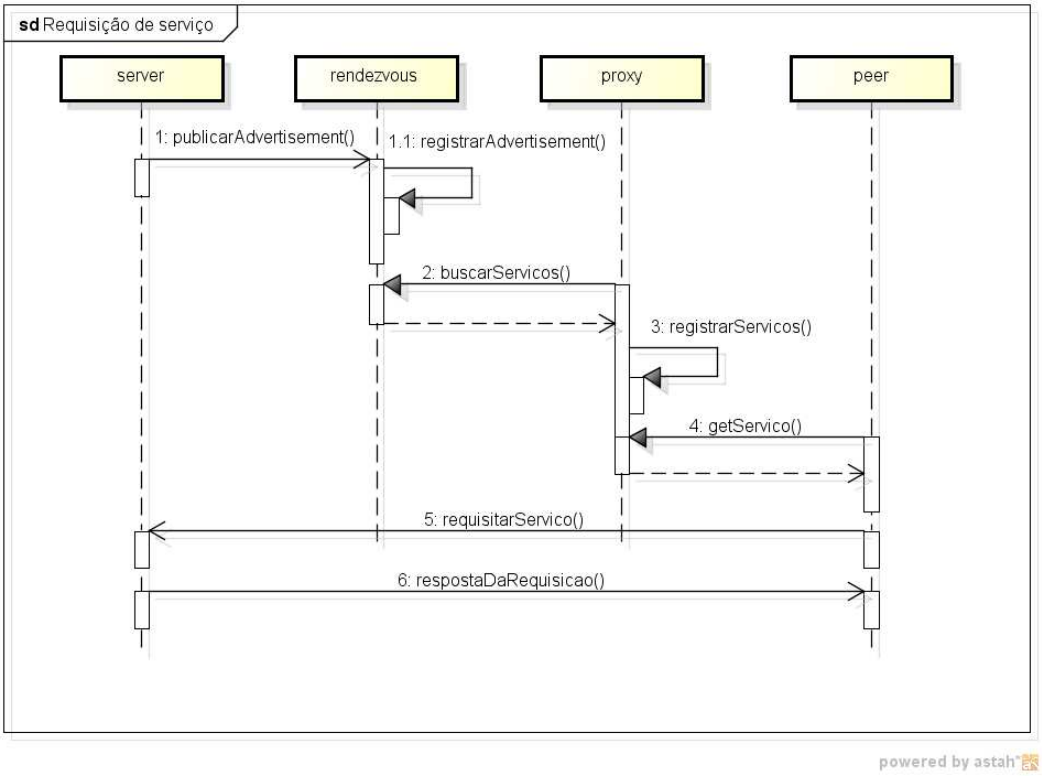


Figura 3.2 - Diagrama de sequencia USTO.RE

Na **Erro! Fonte de referência não encontrada.**.3 é apresentado este mesmo processo porém com a visão de processos isolados rodando em servidores com papeis específicos.

A organização do sistema nesta arquitetura multicamadas possibilita a distribuição do processamento, uma vez que os componentes estão fisicamente distribuídos. Entretanto, por se tratar de uma arquitetura híbrida P2P estruturada e multicamadas, o sistema possui uma distribuição dita horizontal. Nesta distribuição horizontal, em uma rede P2P, um cliente ou um servidor podem estar fisicamente divididos em partes logicamente equivalentes, onde cada um opera sobre a sua própria porção dos dados, o que propicia um balanceamento da carga.

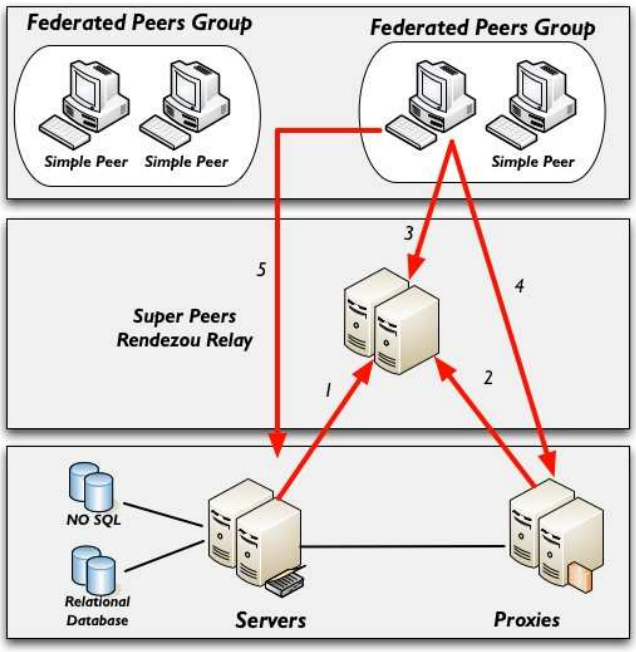


Figura 3.3 - Arquitetura do USTO.RE

Na Figura 3.4, é apresentado em detalhes cada um dos componentes de arquitetura do USTO.RE, separando cada um deles entre interfaces, bibliotecas compartilhadas, componentes da rede P2P e interfaces de consumo de serviços. Nas subseções a seguir, maiores detalhes sobre cada componente são fornecidos e para os componentes principais será acrescido o detalhamento da arquitetura.

3.3.2 Super peers

Os *super peers* funcionam como elementos de referência para os demais componentes da arquitetura, sendo a porta de entrada para a participação de servidores, *proxies* e *simples peers* no sistema. O papel do *super peer* é definir as federações de dados quando cada *peer* solicita conexão à rede. Para isso, os *super peers* devem ter sua localização previamente conhecida por todos os demais *peers* por meio de uma pré-configuração. Consequentemente, eles são os primeiros componentes a serem inicializados para o correto funcionamento do usto.re. Também como consequência, um *super peer* guarda informações sobre todos os servidores, *proxies* e *simple peers*, agrupando-os dinamicamente de acordo com o perfil de cada *peer*, a ser explicado adiante.

Também é papel deste tipo de *peer*, escolher dinamicamente os *peers* e servidores das federações baseando-se em um algoritmo de proximidade [Duarte, 2010]. O agrupamento em federações permite o crescimento elástico e garante a escalabilidade do sistema, pois não existe um limite para a quantidade de federações que podem ser criadas. Crescimento elástico é a característica do sistema de crescer ou diminuir, em termos de capacidade e consumo de recursos, de maneira dinâmica e não intrusiva. Os *peers* se comunicam com a rede P2P através do protocolo *JXTA* [JXTA, 2012] e opcionalmente podem ofertar uma interface de serviço *REST* [Webber et al., 2010] para permitir a interoperabilidade com outras aplicações.

Figura 3.4 - Visão geral da arquitetura do sistema

3.3.3 Servidores

Os *peers* servidores são aqueles que oferecem um conjunto (ou subconjunto) de uma lista existente de serviços. Na ordem de configuração do usto.re, os servidores são os componentes que devem ser executados logo após a inicialização dos *super peers* (Passo 1 na Figura 3.2). Os *Super peers* estabelecem um esquema de sincronização fazendo com que a lista de servidores em cada um deles seja atualizada quando da entrada ou saída de um *peer* servidor.

A definição de *peers* com funcionalidades especificas na rede opõem-se à algumas propostas para sistemas P2P, onde cada *peer* deveria ser capaz de desempenhar todos os papéis no sistema, promovendo a ideia de uma *DHT* (*Distributed Hash Table*) completa [Oliveira, 2007; Loest et al., 2009]. No entanto, a implementação utilizando uma DHT em sua essência é bastante custosa e de difícil escalabilidade. Sendo assim, a proposta adotada no usto.re é a criação de níveis hierárquicos que implementam serviços bem definidos e que podem crescer horizontalmente. Dentre os serviços disponibilizados pelos servidores, pode-se citar:

- ***Autenticação***: usado para que cada *peer* se autentique;
- ***Disponibilidade***: permite checar a disponibilidade de cada *peer*;
- ***Chunk***: usado para o controle de *chunks*;
- ***Erro***: permite o controle de erros como, por exemplo, um serviço indisponível;
- ***Controle de Saída***: controla a saída voluntária de um *peer*, quando este desconecta-se voluntariamente da rede;
- ***Gerência de Diretórios***: utilizado para armazenamento e recuperação de diretórios inteiros;
- ***Gerencia de Arquivos***: utilizado para armazenamento e recuperação de arquivos;
- ***Busca***: procura por um conjunto de *peers* que obedeçam ao acordo de nível de serviço (do inglês, *Service-Level Agreement - SLA*), no caso do usto.re fortemente relacionado à disponibilidade do *peer*, para o salvamento de um arquivo;
- ***Árvore de Diretórios***: utilizado para visualização de diretórios inteiros;
- ***Segurança de Acesso***: controla a permissão de acesso aos *chunks*;
- ***Rastreabilidade***: mantém uma lista de usuários e arquivos a ser consultada quando a recuperação de um arquivo é solicitada.

Como se pode observar na Figura 2, os *peers* servidores utilizam dois tipos de banco de dados para manter a consistência do sistema. Um banco de dados tradicional relacional contém dados dos usuários do sistema, e um banco de dados No-SQL [Chang et al., 2006] que permite um crescimento horizontal e a recuperação mais rápida das informações relacionadas aos arquivos e *chunks* salvos. A escolha desta separação se dá pelas questões relacionadas ao desempenho do sistema, visto que, com o aumento do volume de arquivos e *chunks* salvos, o sistema de gerenciamento de banco de dados tende tornar-se um ponto de gargalo. A utilização de um sistema nativamente distribuído torna a solução viável e escalável [Chang et al., 2006].

Todas as informações referentes à autenticação e SLA dos *peers* são salvas em um banco de dados relacional devido à a garantia da integridade provida por este tipo de banco. Já o serviço do *File Tracker*, que permite a identificação de qual *peer* possui partes do arquivo a ser recuperado, é utilizado um banco de dados No-SQL, o que permite o seu crescimento horizontal. As instâncias dos bancos, quer seja este relacional ou Não-SQL, podem ser compartilhadas entre mais de um servidor.

Um *peer* servidor pode prover um ou mais serviços da rede, sendo assim, da mesma forma que na criação de federações de dados, pode-se iniciar *peers* servidores e *proxies* sobre demanda aumentando a escalabilidade e elasticidade do sistema. No atual estágio do projeto, este aumento deve ser feito manualmente por um humano, de acordo com o monitoramento do desempenho do sistema.

O relacionamento entre cada componente da solução descrita acima com suas interfaces e conectores está apresentada na Figura 3.5 abaixo. Nela se pode observar que um dos elementos centrais do sistema são as filas de envio e recepção de mensagens. No USTO.RE toda troca de mensagem passa por uma fila que é consumida sobre demanda e de acordo com os recursos disponíveis no sistema operacional. A quantidade de filas pode ser configurada, sendo assim, o USTO.RE pode ser utilizado em equipamentos com características diferentes, garantindo que as operações sempre ocorrerão. Quando se deseja mais performance aumenta-se a quantidade de filas desde que haja recursos disponíveis no sistema operacional.

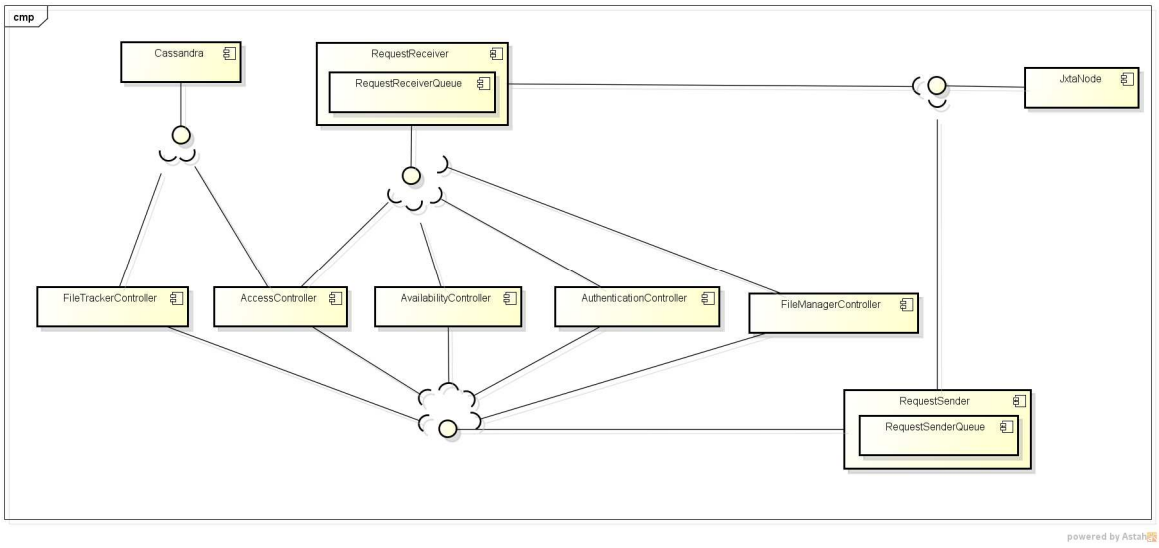


Figura 3.5 - Visão componente-conector do servidor

3.3.4 Proxies

Após a inicialização de *super peers* e servidores, o terceiro componente que precisa ser executado é o *proxy*. Cada *proxy* atua como um catálogo, um serviço de localização para serviços em execução nos diferentes servidores do usto.re. Um *proxy* ao se anunciar a um *super peer* (Passo 2 da Figura 3.2), recebe a lista de servidores registrados. Além disso, um proxy obtém a informação de quais serviços estão disponíveis em cada servidor. Desta forma, quando um *peer* deseja a informação sobre um determinado serviço, um *proxy* pode fornecer a referência para um servidor que atenda tal requisição. Desta forma, um *proxy* estabelece uma ponte de ligação entre

consumidores de serviços, tipicamente os *simple peers*, e o provedor de um serviço, no caso, os *peers* servidores.

3.3.5 Simple peers

Simple peers são aqueles responsáveis por armazenarem os *chunks* dos arquivos. Na visão alto nível da proposta, estes *peers* representam máquinas de usuários comuns que podem oferecer espaço de armazenamento para ser compartilhado entre vários usuários. Cada *simple peer* possui um perfil que define a sua disponibilidade e que lhe é atribuído quando de sua conexão com o sistema. Esta disponibilidade é relacionada ao período de tempo em que o *peer* esteja disponível para compartilhar dados. Como exemplo, um *peer* que esteja em uma *Intranet* de uma empresa pode ter em seu perfil, a disponibilidade atribuída como "***8:00 às 12:00 e 14:00 às 18:00***". Quando um *simple peer* se conecta ao sistema, ele recebe de um *super peer* a lista de *proxies* disponíveis (Passo 3 da Figura 3.2). Desta lista, o proxy busca por um serviço específico e obtém a referência de quais servidores ofertam determinado serviço (Passo 4 da Figura 3.2). Um servidor aleatório é escolhido, e o *simple peer* solicita o serviço desejado (Passo 5 da Figura 3.2). Caso um serviço não seja atendido por algum motivo, como um *timeout*, o *proxy* pode fornecer um novo servidor para o *simple peer*.

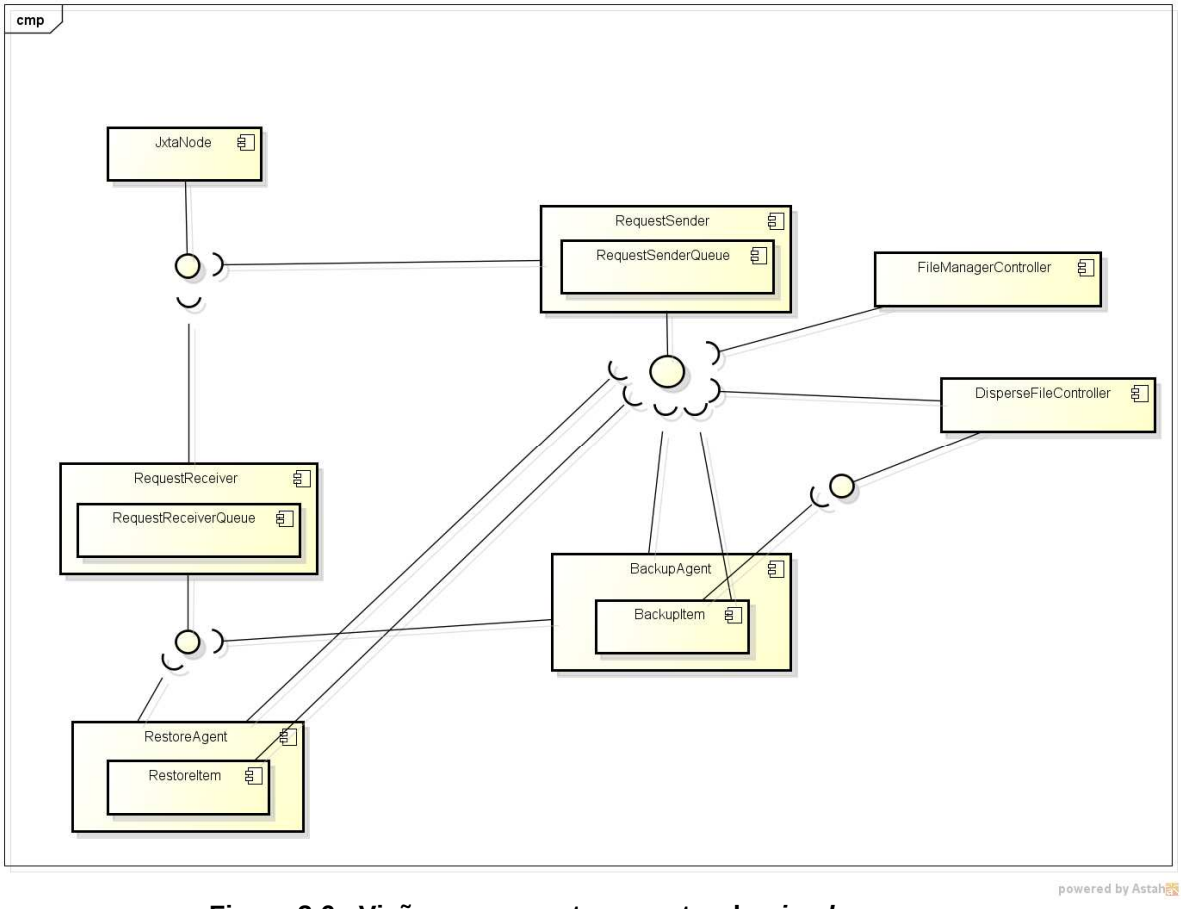


Figura 3.6 - Visão componente-conector do *simple peer*

Observando-se Figura 3.6, também se vê como elemento central do sistema o gerenciador de filas que neste caso também pode ser configurado trazendo os mesmos benefícios descritos para o servidor. Ainda como elementos centrais do sistema temos o

BackupAgent e o *RestoreAgent*. Cada componente deste representa uma ação recuperação (*restore*) ou *backup* (salvamento). Quando uma operação é solicitada uma instancia deste componente é criada e o componente é responsável por garantir que a operação solicitada seja executada. Para evitar que haja consumo elevado de memória ou recursos do sistema, apenas duas instancias de *BeckupAgent* e *RestoreAgent* pode ser executada por vez em um *simple peer.*

Como citado anteriormente, *simple peer* são agrupados em federações de dados pelos *super peers*. O objetivo de agrupá-los desta forma é minimizar a sobrecarga na rede e em cada *peer*, além de diminuir a quantidade de mensagens trocadas e permitir que uma federação desempenhe o papel de *backup* de outra federação. O agrupamento dos *peers* em federações obedece os seguintes critérios por ordem de relevância: proximidade; perfil de cada *simple peer*; latência de rede; latência da federação; georeferenciação; capacidade de cada *peer*; e, capacidade final da federação, que é definido pelos *super peers*.

Cada *peer* possui uma interface de serviço *REST* [Webber et al., 2010] que permite a autenticação do usuário, armazenamento, recuperação e remoção dos dados salvos. Tal característica apresenta como principal vantagem a possibilidade de compatibilizar o sistema com outras interfaces existentes, como *S3* da *Amazon*. Na atual arquitetura, o serviço de armazenamento dos dados pode ser modificado para outras alternativas (i.e. *Megastore*, *MSFSS* ou *Amazon S3*).

Para garantir o espalhamento de cada *chunk* de forma a garantir os níveis de serviço adequados, cada *peer* precisa periodicamente relatar seu estado atual com o objetivo de manter o SLA sempre atualizado. A **Erro! Fonte de referência não encontrada. (a)** apresenta o fluxo de funcionamento de um *peer* (*PL_1*), desde o momento em que ele anuncia o seu perfil à comunicação estabelecida com os demais pares por meio do servidor (*PS_1*). Periodicamente cada *peer* envia para um dos servidores uma mensagem de "*keep alive*" informando que está *online*. Desta forma, o servidor sabe se o *peer* está cumprindo com o perfil acordado (SLA) e o torna elegível para receber *chunks* no horário definido. Também periodicamente cada *peer* verifica com os demais *peers* existentes se os *chunks* que ele possui estão replicados na quantidade mínima de *peers* obedecendo o critério de disponibilidade exigida [Duarte, 2010]. Caso não esteja, ele replica o *chunk* em outro *peer*. Quando o *peer* que possui este *chunk* voltar a se conectar a rede, ele irá verificar que existe um excesso deste *chunk* e irá excluí-lo.

A **Erro! Fonte de referência não encontrada. (b)** apresenta o fluxo de funcionamento entre *peers* e servidores desde a conexão do *peer* local à rede, ao salvamento dos arquivos solicitados. Após o *peer* (*PL_1*) ao se conectar à rede *P2P_1*, o *super peer* indica um *peer servidor* para ele se autenticar. Com a autenticação bem sucedida, um processo de identificação dos pares para formar as federações é executado. Com a federação (agrupamento de *peer*) estabelecida, o sistema está apto a receber arquivos. Ao receber um arquivo ("*arq1.zip*"), o *peer PL_1* informa a necessidade de armazená-lo no sistema, para isto é feita uma segmentação do arquivo (em *chunks*) e estes segmentos são enviados para serem salvos na rede *P2P_1*. Em seguida, para efetuar o salvamento, uma rotina para medir a confiabilidade analisa o estado dos *peer* e, consequentemente, da disponibilidade dos segmentos do arquivo na rede e caso exista uma combinação de *peer* que atenda ao SLA para o armazenamento, os segmentos do

arquivo são enviados para estes *peer*. Se não houverem mais segmentos de arquivos a serem salvos, o *peer servidor PS_1* comunica ao *peer PL_1*, que solicitou o salvamento do arquivo, que o mesmo foi salvo com sucesso.

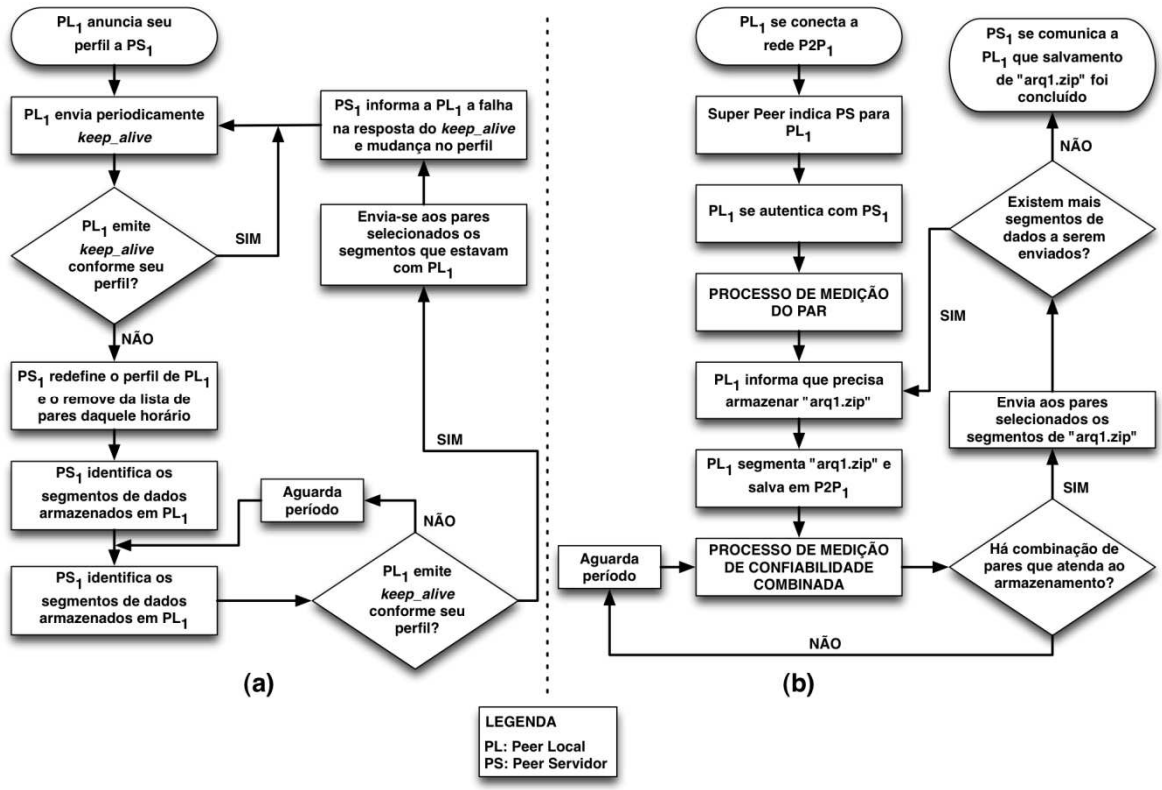


Figura 3.7 - a) Funcionamento de um *peer* b) Funcionamento *peer Server proxy*

3.3.6 Plataforma USTO.RE S3

Ainda como resultado deste projeto, utilizando os conceitos trazidos pelo reuso de software e conforme foi apresentado, pode-se enxergar a plataforma USTO.RE como sendo um sistema que permite a utilização dos recursos ociosos nos discos nos computadores que possuem o software instalado. Este sistema também pode ser visto como uma plataforma de armazenamento em nuvem. Sendo assim, se pode ofertar o uso desta plataforma como sendo um serviço (*PaaS*) na nuvem para pesquisadores que necessitam de muitos *Gigabaytes* para armazenar os dados ou backup dos seus projetos .

Neste caso será provido aos pesquisadores uma interface *SOAP/REST* compatível com as interfaces providas por outros serviços como *Allmydata*, *Amazon S3*, entre outros, de forma que seja possível utilizar os *peers* que compõe a rede como um sistema para armazenamento de dados como *PaaS*. Isto se torna possível porque a plataforma USTO.RE provê a garantia da recuperação dos dados. Neste sentido, será investigado a possibilidade de permitir aos usuários que utilizam plataforma de armazenamento a um custo mais elevado, utilizar o sistema USTO.RE S3 através de um custo mais acessível.

Pode-se afirmar que existe a diminuição de custo porque ao analisar as soluções descritas anteriormente, em todas elas existe a necessidade de se prover infraestrutura dedicada, mesmo que a baixo custo, para suportar o sistema. Neste caso, o custo diminui bastante visto que será utilizado parte do espaço disponível no disco dos usuários.

3.4 Banco de dados

Seguindo este mesmo principio a escalabilidade de banco de dados pode ser dada através da alocação de múltiplos sistemas de banco de dados em paralelo e s associando aos conceitos de sistemas multi-tenant.

Neste contexto a proposta da criação de sistemas multi-database, como adotado facebook se mostra extremamente viável.

Os objetivos deste modelo de funcionamento são:

Redistribuir e crescer sistemas de banco de dados dinamicamente sem a necessidade de alteração no código da aplicação.

a) Diminuição da carga no SGBD's
b) Crescimento elástico
c) Sistema de provisionamento baseado nos princípios de cloud computing

Como funciona:

I. A aplicação ao iniciar consulta a lista de banco de dados existentes
   a. Esta lista contém a URL de conexão do BD e qual informação ele possui
II. O pool de conexão conecta nestes bancos
III. Baseado no identificador de segmentação o sistema ao realizar o getConnection para o pool de conexão informa o identificador e baseado nesta informação o pool seleciona o DB
IV. Quando administrador do SGBD encontrar a necessidade de dividir o banco ele envia o comando através da interface de gerenciamento dos banco de dados provida pela aplicação
V. O banco de dados e colocado em read-only mode e um snapshot e feito. As consultas continuam porém novas transações serão rejeitadas
   a. Um mecanismo de tratamento de exceção precisa ser implementado/alterado para suportar o erro gerado
VI. Um novo servidor de SGBD é solicitado para o gerenciador de cloud
VII. O snapshot do BD em read-only é copiado para este novo banco
VIII. Queries de limpeza do Banco rodam para limpar os dois bancos novos.
IX. O sistema de particionamento atualiza o banco de dados com os hashes
X. O sistema de particionamento notifica a aplicação para atualizar seus hasehs.
XI. O pool de conexão referente ao servidor que foi particionado é ressetado

a. Assim as conexões existentes na aplicação gerarão um erro forçando outro getConnection a acontecer

A representação deste fluxo esta descrito na Figura 3.8

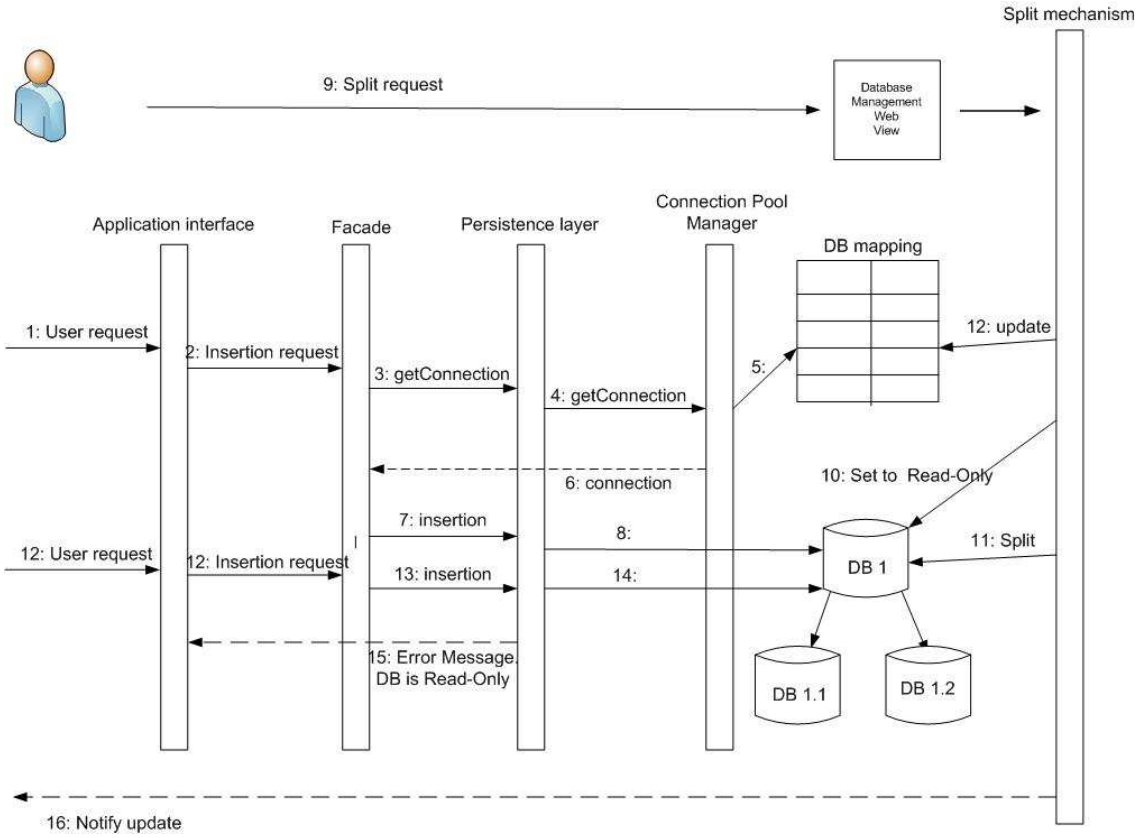


Figura 3.8 - Sistemas Multi-Database

3.5 Bigdata

Os conceitos relacionados a BIGDATA estão intrinsicamente ligados as necessidades de busca, recuperação e recomendação em uma base de dados desestruturados. Este tipo de dado, normalmente visto como um arquivo, agora esta salvo em sistemas de cloud storages como descrito nas sessões anteriormente.

Sendo assim é necessário que os mesmos conceitos ofertados por sistemas de arquivos tradicionais, como EXT4 do Linux e NTFS do Windows sejam portados para os sistemas de nuvem e a eles acrescidos informações extras que permitam a realização das operações desejadas nas plataformas de BIGDATA.

O começo da solução deste problema, esta na definição de metadados que contenham as informações mínimas de sistemas de arquivo, acrescidas de informações para a busca e recuperação, como índices e palavras chave.

3.5.1 Trabalhos Relacionados

Tanenbaum (2007) lista alguns atributos que podem compor os metadados de um sistema de arquivos para ajudar a superar os desafios existentes. Segundo o autor

supracitado, nenhum sistema existente dispõe de todos os atributos, porém cada um deles está presente em algum sistema. Na lista aparecem atributos ligados à proteção do arquivo e indicam quem poderá acessá-lo, bem como a senha necessária para isso, caso seja necessário. Há ainda flags para indicar se o arquivo é somente para leitura, oculto, se já foi feita cópia de segurança, além disso há campos para indicar datas de criação, último acesso e tamanho do arquivo.

O Microsoft Windows 8 utiliza como sistema de arquivos o ReFS[38](Resilient File System) que foi desenvolvido utilizando como base o NTFS. O ReFS utiliza árvores B+ para armazenar todos os dados e metadados do mesmo. A reutilização de código foi amplamente utilizada para manter uma alta compatibilidade com um subconjunto de funcionalidades do NTFS, como alguns atributos dos metadados (padrão de informações, nome do arquivo, valores de algumas flags, etc), interface do sistema de arquivos (leitura, escrita, abrir, fechar, modificar, notificar, etc), manter arquivos na memória, imposições de segurança e memória cache. Muitos dos atributos dos metadados utilizados no ReFS vem do NTFS, como os atributos para nome, dados, lista de controle de acesso, informações padrões (datas de criação, modificações e último acesso). Mas há atributos exclusivos do novo sistema de arquivos como o INTEGRITY_STREAM e o NO_SCRUB_DATA. O atributo INTEGRITY_STREAM é criado unicamente para gerenciar a integridade de diretórios e arquivos pertencentes a aplicativos que preferem gerenciar o armazenamento de arquivos cuidadosamente e, para isso, dependem de um determinado layout de arquivo no disco. Como os fluxos de dados realocam blocos sempre que o conteúdo de um arquivo é alterado, o layout do arquivo fica muito imprevisível para esses aplicativos. Para combater os casos de corrupções de disco e falhas no armazenamento, o ReFS periodicamente depura todos os metadados e dados em um volume do ReFS. O atributo de arquivo NO_SCRUB_DATA é utilizado para indicar que o depurador deve ignorar o arquivo. Esse atributo é útil para os aplicativos que mantém suas próprias informações de integridade[39].

O EXT4 é um sistema de arquivos de journaling para Linux desenvolvido como uma extensão para o EXT3 com o objetivo de revolver problemas de escalabilidade, desempenho e confiabilidade. O EXT3 possui usa 32 bits para representar blocos de números e tem por padrão um bloco de 4K, isso faz com que o sistema de arquivos fique limitado a 16 TB, o que hoje, devido à enorme massa de dados crescente existente, é uma capacidade de armazenamento pequena. O EXT4 ampliou o bloco de número para 48 bits, o que, em teoria, permite que o sistema de arquivos suporte até 1 Exabytes (1 milhão de TB) (Mathura, 2007). Além disso, o mesmo possui uma alta compatibilidade com versões futuras e antigas do sistema de arquivos, como o EXT3, e aprimoramentos na resolução e no intervalo do registro de data e hora[40]. Os metadados do EXT4 são compostos pelo número do inodes que o diretório aponta, tamanho do diretório, nome, tipo de dado armazenado no inode (arquivo, diretório, *socket*, *link* simbólico), além da lista de controle de acesso (que é guardada separadamente, em um *inode* especial) e atributos para a configuração do sistemas de arquivos, como o

[38]http://bit.ly/KzTYdZ
[39]http://bit.ly/KMEEQV
[40]http://ibm.co/KQQBB3

EXT4_INDEX_FL que indica se os diretórios serão armazenados em árvores B ou em um *array* linear[41].

O Hadoop Distributed File System (HDFS) é um sistema de arquivos distribuídos projetado para rodar sob hardwares comuns. Diferentes de alguns outros sistemas de arquivos, o HDFS é altamente tolerante a falhas e foi projetado para ser executado em hardwares de baixo custo. Todos os servidores são totalmente conectados e a comunicação acontece através de protocolos baseados em TCP.

Diferentemente de outras abordagens de sistemas de arquivos distribuídos, o armazenamento e processamento do HDFS é feito em cada nó do sistema. Assim como outros sistemas de arquivos, o HDFS guarda os metadados separados dos arquivos da aplicação. O metadados são armazenados em um servidor dedicado chamado NameNode e os dados da aplicação são armazenados em outros servidores, chamados DataNodes. O NameNode possui inodes, que representam os arquivos e diretórios. Além disso, os inodes guardam atributos como permissões, data de acesso e modificações, namespace onde os arquivos estão e espaço disponível em cada disco (Shvachko, 2011).

O XtreemFS[42] é um sistema de arquivos globalmente distribuído e replicado. Ele é desenvolvido como parte do projeto XtreemOS EU, o qual tem o objetivo de criar um sistema operacional open source para ser utilizado em ambientes de grid. XtreemFS é baseado em objetos, com metadados e dados armazenados em diferentes tipos de nós. Além disso, ele é compatível com o padrão POSIX, e possui sistema de tratamento de falhas. Ele replica os objetos para tolerância a falhas e faz cache dos dados e metadados para melhorar o desempenho no caso de alta latência (Hupfeld, 2007 e 2008). O XtreemFS faz a replicação a partir do arquivo completo, ao invés de utilizar as partes já divididas do arquivo. Isso faz com que haja um aumento considerável na comunicação interna para manter as réplicas sincronizadas. O XtreemFS utiliza o BabuDB como banco de dados para armazenar os metadados do sistema de arquivos. Os atributos utilizados nos metadados incluem uma identificação única para arquivos e diretórios, nome, tipo, além de atributos relacionados a data de criação e modificação, tamanho do arquivo, localização do conteúdos do arquivos, localização das réplicas[43], dono, controle de acesso e atributos estendidos.

O Google File System (GFS) é um sistema de arquivos distribuídos escalável para grandes aplicações distribuídas com uma grande quantidade de dados. Ele provê tolerância a falha e um acesso confiável, eficiente ao dados utilizando um conjunto de hardwares comuns. Este sistema de arquivos é otimizado para trabalhar com grandes arquivos, normalmente 100MB ou mais, sendo muito utilizado para arquivo com tamanho superior a 1GB. Sua arquitetura consiste de múltiplos nós, sendo eles de dois tipos: um nó Master e um grande ChunkServer. Cada arquivo armazenado é divido em chunks de tamanho fixo, os quais são armazenados no ChunkServer. Cada chunk criado é replicado para outros servidores do mesmo tipo, no mínimo, três vezes. O nó Master armazena todos os metadados associado com os chunks, como uma tabela onde há o mapeamento da localização dos chunks com os arquivos, a localização das cópias dos chunks, quais processos estão acessando os chunks (Ghemawat, 2003).

[41] http://bit.ly/LrsF6p

[42] http://www.xtreemfs.org

[43]http://www.xtreemfs.org/arch.php

O Amazon S3[44](*Simple Storage Systems*) é um de armazenamento oferecido pela Amazon Web Service que funciona através de interfaces *web services* baseadas em *REST*, *SOAP* e *BitTorrent*.

O Amazon S3 é projetado para fornecer escalabilidade, alta disponibilidade e baixa latência. Ele é formado por *buckets*, que é similar a um diretório or um *container*. Cada objeto armazenado é composto pelo nome, um *blob* com o conteúdo (até 5 Gb) e os metadados do mesmo, composto por atributos que representam o tipo de dado armazenado, associação com usuários e permissões de acesso, datas de criação, acesso e modificação, localização de objetos relacionados, classificação e comentários feitos pelo usuário, rótulos de área, assunto e geolocalização[45].

O Amazon S3 só permite buscas limitadas a consultas simples baseadas no nome dos objetos e dentro de um único *bucket*, não permite buscas baseadas em metadados e nem no conteúdo dos objetos (Palankar, 2008).

Existem outros sistemas focados em backup nas nuvens como Dropbox, SugarSync, Google Drive, mas os mesmos são soluções proprietárias e não possuem informações relacionadas com metadados disponíveis, o que dificulta uma análise técnica aprofundada sobre os mesmos, mas é possível dizer que os mesmos não possuem uma indexação que permita que consultas sejam realizadas baseadas no conteúdo de cada arquivo armazenado.

Os diversos sistemas mostrados acima são utilizados para diferentes fins: alguns são utilizados em ambientes tradicionais (disco local), outros são focados em ambientes de rede ou distribuídos e outros são direcionados para serem utilizados em ambientes de computação nas nuvens, por isso, eles possuem uma estrutura de metadados que varia de acordo com a finalidade desejada. Os metadados descritos possuem atributos utilizados para diversos fins, como: identificar os arquivos e diretórios, controlar o acesso aos mesmos, indicar a forma de armazenar os dados, localizar os *chunks* e réplicas, armazenar as datas de criação, acesso e modificação, controlar a integridade. Como pode ser observado, não há uma estrutura definida para metadados voltada para sistemas de *cloud storage* que atenda aos requisitos apresentados anteriomente na seção 2 (motivacao), principalmente, no que se refere a indexação do conteúdo armazenado.

3.5.2 Solução Proposta

Um sistema de arquivos é um conjunto de estruturas lógicas e de rotinas, que permitem ao sistema operacional controlar o acesso ao disco rígido[46] e, para isso, Tanembaum (2007) define que é necessário que estas característas estejam presentes:

- Segurança ou permissões
- Listas de controle de acesso (ACLs, em inglês, *Access Control Lists*)
- Mecanismo para evitar a fragmentação
- Capacidade de enlaces simbólicos (*symbolic links*) ou duros (*hard links*)

[44]http://aws.amazon.com/s3/
[45]http://bit.ly/JeaAZg
[46]http://bit.ly/JebDZe

- Integridade do sistema de arquivos (*Journaling*)
- Suporte para arquivos dispersos
- Suporte para quotas de discos
- Suporte de crescimento do sistema de arquivos nativo

Já para o desenvolvimento de um sistema de data cloud, como o usto.re, outras característas diferentes das mencionadas acima foram identificadas. Um sistema de *cloud storage* deve ser preocupar em como armazenar e recuperar os dados, além de identificar qual a máquina utilizada, indexação do conteúdo, como provisionamento sob demanda (habilidade de alocar espaço de acordo com a necessidade), snapshots (criar uma imagem instantânea do disco sem criar uma cópia completa), replicação e clonagem[47].

Portanto, o usto.re, deve ser visto como um sistema de *cloud storage* que implementa metadados como forma de facilitar o gerenciamento e acesso aos arquivos armazenados.

3.5.3 METASTORE

Assim como no modelo tradicional, onde um disco local possui o sistema de arquivo, é necessário que seja definido um conjunto de metadados associados ao arquivo que contém as informações sintáticas.

No usto.re os metadados foram descritos utilizando JSON (*JavaScript Object Notation*) devido a facilidade de representação de objetos e ao fato de consumirem menos recursos para serem processados, favorecendo assim a troca de dados, quanto comparado com o XML. Os atributos dos metadados foram definidos tendo como base os requisitos definidos anteriormente na seção motivacao, os atributos levantados na seção de trabalhos relacionados, além de atributos ligados às necessidades da computação nas nuvens para identificação e recuperação de informações.

A partir da análise dos sistemas de arquivos na seção de trabalhos relacionados observa-se que para o usto.re atender aos requisitos funcionais especificados são necessários atributos para identificação do dono do arquivo, caminho relativo do arquivo para o diretório monitorado, código hash para identificar se modificações foram feitas, a data e hora da última modificação. Além disso, também foi preciso criar um perfil para identificar a máquina a partir da qual o usuário acessa o sistema no momento do backup, para que o sistema pudesse identificar se o usuário está autenticado a partir de uma máquina diferente. Isso é necessário devido ao requisito de monitoramento de diretórios, já que uma máquina diferente, possivelmente, não possui a mesma organização hierárquica de arquivos e diretórios, podendo, assim, fazer com que o sistema tentasse monitorar um diretório que não existe. O perfil da máquina é composto pelo nome da máquina, sistema operacional e arquitetura do mesmo.

Também é necessário incluir atributos para identificar quais as opções de segurança e compartilhamento relacionadas ao arquivo. Estas opções de segurança

[47]http://bit.ly/KVuGsU

dizem respeito à forma como os outros usuários verão este arquivo (público ou privado) e como interagirão com ele (somente leitura, escrita, controle total). Através desse atributo é possível também determinar que o arquivo fique acessível apenas para o grupo ao qual o usuário pertença.

Além das informações relacionadas ao arquivo e sobre a maquina que o usuário está autenticado, outra característica identificada como necessária é a indexação do arquivo de forma a facilitar a localização de arquivos. Esta funcionalidade de indexação de dados em sistemas de data cloud é uma das funcionalidaes mais desejadas (Buyya, 2009), (Leung, 2009) e (Grossman,2008), porém de difícil implementação, visto que os arquivos não são salvos integralmente, mas sim quebrados em pedaços menores (*chunks*) e cada pedaço salvo em um servidor da nuvem de dados diferente.

Na implementação do usto.re, os *peers* que recebem os arquivos e os salvam na *data cloud*, são responsáveis também por indexá-los, antes que os mesmos sejam quebrados em partes menores para serem armazenados, com o objetivo de extrair o conteúdo dos mesmos e salvá-lo em um arquivo separado, formando, assim, um índice onde as buscas serão realizadas.

Após a geração dos arquivos necessários para armazenar o índice, um serviço provido pelo sistema é responsável por copiar esta informação para o servidor, onde esses dados ficarão disponíveis para a realização de buscas.

Para a indexação de arquivos foi utilizado o Apache Lucene[48], por ser um motor de buscas de alto desempenho escrito em Java. O Lucene contém apenas o núcleo do "motor" de busca. Por isso, ele não inclui um *Web crawler* ou um *parser* para diferentes formatos de documentos. Para resolver o problema do \textit{parser} foi utilizado o Apache Tika[49], que é um detector e extrator de conteúdo de metadados e texto estruturado, podendo ser utilizado com arquivos de diversos formatos como: HTML, XML, OLE2 e OOXML do Microsoft Office, OpenDocument Format, PDF, ePub, RTF, arquivos compactados e empacotados.

Sendo assim, quando um usuário realizar uma busca pelo conteúdo de um arquivo no usto.re ele poderá consultar todos os arquivos do sistema, porém:

- O usuário somente poderá acessar os arquivos pertencentes ou compartilhados com o mesmo;
- Somente poderá visualizar que um ou mais arquivos tem a informação procurada, desde que no metadado do mesmo seja informado que o arquivo poderá ser indexado;
- Caso um usuário deseja acessar um arquivo que não lhe pertence deverá solicitar ao dono que o compartilhe;
- Poderá existir um número N de arquivos que o usuário não tem acesso que contém esta informação;

[48]http://lucene.apache.org/
[49]http://tika.apache.org/

Assim, a estrutura dos metadados para o usto.re é composta pelos seguintes atributos:

- Nome do arquivo;
- Código *Hash*;
- Data da última modificação do arquivo;
- *Flag* para identificar se o arquivo deve ser indexado ou não;
- Atributos de segurança e compartilhamento;
- Dono do arquivo;
- Grupo ao qual o dono pertence;

Detalhes da Implementação do METASTORE

A Figura 3.3 representa o processo de backup, geração de metadados e indexação. No usto.re quando a confirmação de um backup chega ao *appClient*, é executado um método para gerar os metadados do arquivo salvo. Com a posse desses dados, o *appClient* os adiciona ao arquivo que representa o mapeamento do diretório do qual faz parte e onde se encontram todos os metadados dos arquivos cujo *backup* foi concluído com êxito. Após isso, o *appClient* envia esse mapeamento para que seja salvo no banco de dados juntamente com as informações de qual é o caminho completo para o diretório monitorado, caminho relativo e código *hash* do mapeamento.

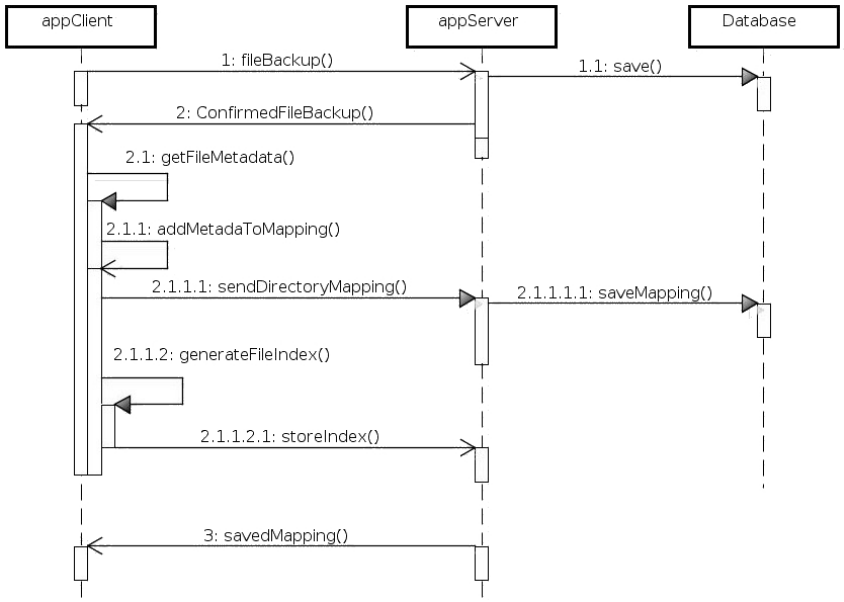


Figura 3.9 - Diagrama de sequencia metastore

Paralelamente à atividade descrita anteriormente é feita a indexação dos dados do arquivo e os dados gerados são enviados para o diretório *index* no servidor. Esta indexação extrai o conteúdo dos arquivos utilizando o Apache Tika e junta isso com a informação, repassada pelo appClient, de qual o nome do usuário que está fazendo backups no momento, o qual representa o dono dos arquivos e o nome junto com o caminho relativo para cada arquivo.

Na Figura 3.9 é apresentada a sequência executada quando é necessário efetuar uma busca por um arquivo no usto.re. A requisição feita ao *appClient* é repassada para o *appServer*. Nele o método *searcher* executa a busca utilizando a indexação gerada pelo Apache Lucene para isso. A busca pode ser realizada através do nome do arquivo, parte do conteúdo ou nome do dono. Se a busca for efetuada utilizando o nome do arquivo ou parte do conteúdo como parâmetro, o método retornará uma lista de arquivos contendo o nome dos mesmo e a quem pertence cada um deles. A depender das permissões de acesso do usuário, alguns arquivos podem deixar de ser exibidos na listagem ou podem aparecer arquivos aos quais o usuário não poderá acessar sem a permissão do dono.

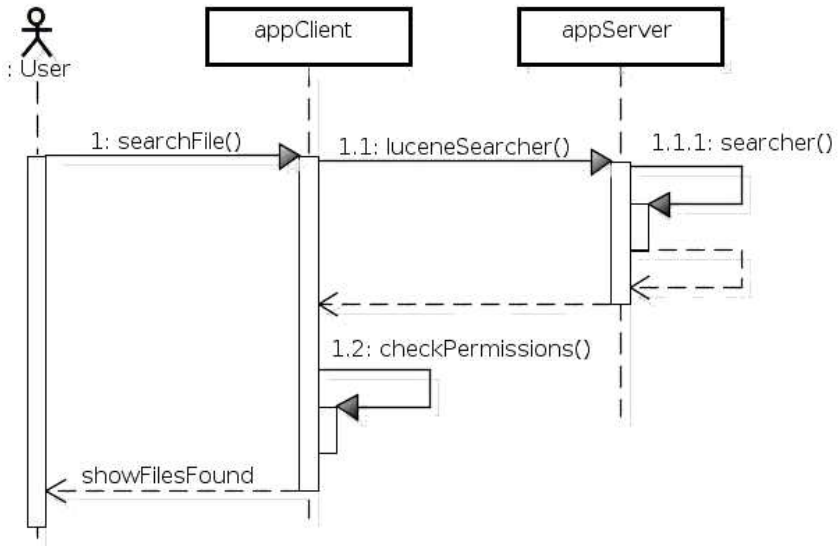


Figura 3.10 - Sincronização de diretorio

A sincronização de diretório só é possível porque o banco de dados armazena o caminho completo para os diretórios monitorados, isso permite chegar até os diretórios e, a partir disso, passar a monitorá-los. Quando um usuário loga, o usto.re faz uma consulta ao banco de dados pelos arquivos que o mesmo possui e quais são os diretórios monitorados. O usto.re traz o mapeamento armazenado no banco de dados e compara com os arquivos existentes nos diretórios monitorado, caso haja alguma diferença entre eles, será feita uma sincronização de arquivos, o que pode incluir adicionar e, até mesmo, remover arquivos.

Caso o usuário esteja em um cliente que não é sua máquina de trabalho (ou seja, que não possui seus arquivos) o usto.re criará uma pasta chamada "usto.re" dentro da pasta do usuário logado e realizará a recuperação de todos os arquivos do usuário. Isso permite que o mesmo tenha acesso a todos seus arquivos e diretórios independente do cliente utilizado.

3.6 Conclusão

Computação nas nuvens (Cloud Computing), consiste de um conjunto de tecnologias que agrupadas trazem vantagens significativas para a gerenciamento dos ambientes de Tecnologia da Informação, consequentemente a diminuição dos custos.

No entanto, a idéia de que qualquer aplicação pode ser migrada para a nuvem sem a necessidade de alterações, consiste de uma visão simplista do problema. Esta afirmativa, é verdadeira desde que os gestores de tecnologia abdiquem das questões relacionadas a performance (no caso de computação nas nuvens o crescimento elástico) e segurança, caso estejam em execução em ambientes públicos.

Este capítulo de livro apresentou como se define e implementa sistemas de armazenamento de massa em nuvens, cloud storages, como se pode definir e implementar sistemas de banco de dados escaláveis e uma abordagem para a incorporação dos conceitos de BIGDATA em sistemas de armazenamento de dados em nuvens.

3.7 Referências

[Aidouni et al., 2009] F. Aidouni, M. Latapy, and C. Magnien, "Ten weeks in the life of an eDonkey server," Proceedings of HotP2P'09, 2009, pp. 1-5.

[Armbrust et al., 2009] M. Armbrust et al. Above the Clouds: A Berkeley View of Cloud Computing. EECS Department, University of California, Berkeley. Technical Report No. UCB/EECS-2009-28. February 10, 2009.

[Amazon EC2, 2012] Amazon Elastic Compute Cloud (Amazon EC2), URL: http://aws.amazon.com/ec2/, Acessado em 05/07/2012.

[Amazon, 2012] Amazon. Amazon Simple Storage Service (Amazon S3), URL: http://aws.amazon.com/pt/s3/, last access 05/07/2012.

[Baker et al., 2011] J. Baker, C. Bond, J. Corbett, J. J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In CIDR'11, pages 223–234, 2011.

[Balakrishnan et al., 2003] Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R., and Stoica, I. Looking up data in P2P systems. Communications of the ACM 46, 2 (2003), 43.

[Bass et al., 2000] L. Bass, C. Buhman, S. Dorda, F. Long, J. Robert, R. Seacord, K. Wallnau, Market Assessment of Component-Based Software Engineering, Software Engineering Institute (SEI), Technical Report, Vol. 01, May, 2000, pp. 41.

[Batten et al., 2002] Christopher Batten, Kenneth Barr, Arvind Saraf, Stanley Trepetin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCSTM-632, Massachusetts Institute of Technology Laboratory for Computer Science, October 2002.

[Baumgarten & Chuí, 2009] Jason Baumgarten and Michael Chui, E-government 2.0, mckinseyquarterly.com, July 2009.

[Bell, 2008] M. Bell, Service-Oriented Modeling, 2008, pp. 366.

[BitTorrent, 2012] BitTorrent. Site Ofical. http://www.bittorrent.com/. Acessado em 09/07/2012.

[Bughin et al., 2010] Jacques Bughin, Michael Chui, and James Manyika, Clouds, big data, and smart assets: Ten tech-enabled business trends to watch, August 2010.

[Chang et al., 2006] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7, pages 15–15. USENIX Association, 2006.

[Clements & Northrop, 2001] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2001, pp. 608.

[Cleversafe, 2012] CLEVERSAFE. Site Oficial. Cleversafe dispersed storage project. http://www.cleversafe.org/dispersed-storage. Acessado em 09/07/2012.

[Colaço et al., 2008] COLAÇO, Eduardo José Moreira ; OLIVEIRA, Marcelo Iury ; SOARES, A.; BRASILEIRO, F ; GUERRERO, D. . Using a file working set model to speed up the recovery of Peer-to-Peer backup systems. ACM SIGOPS Operating Systems Review, v. 42, p. 64-70, 2008.

[Cox & Noble, 2003] Cox, L. P. and Noble, B. D. 2003. Samsara: honor among thieves in peer-to-peer storage. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (Bolton Landing, NY, USA, October 19 - 22, 2003). SOSP '03. ACM, New York, NY, 120-132.

[Dean & Ghemawat, 2008] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. Commun. ACM, 51:107–113, Jan. 2008.

[DeCandia et al., 2007] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon’s highly available key-value store. SIGOPS Oper. Syst. Rev., 41:205–220, Oct. 2007.

[Dropbox, 2012] Dropbox, URL: https://www.dropbox.com/. Acessado em 09/07/2012.

[Duarte, 2010]M. DUARTE. Um algoritmo de disponibilidade em sistemas de backup distribuído seguro usando a plataforma peer-to-peer. Dissertação de mestrado, Centro de Informática, Universidade Federal de Pernambuco, Recife-PE, Brazil, 2010.

[Erl, 2008] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, 2008, pp. 760.

[FIPS, 1995] FIPS 180-1. Secure Hash Standard. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, Apr. 1995.

[Galán & Sampaio, 2009] F. Galán, A. Sampaio et al. Service Specification in Cloud Environments Based on Extensions to Open Standards. In 4th International Conference on Communication System Software and Middleware (COMSWARE 2009), 2009.

[Grobauer et al., 2011] B. Grobauer, T. Walloschek, and E. Stocker. Understanding cloud computing vulnerabilities. IEEE Security and Privacy, 9:50–57, 2011.

[Gnutella, 2012] Gnutella, Site Oficial, http://rfc-gnutella.sourceforge.net/, Acessado em 09/07/2012.

[Google, 2012] Google Web Applications for Communication and Collaborations. http://www.google.com/apps. Acessado em 09/07/2012.

[GTAR, 2010]GTAR, DVN: Disco Virtual Nacional. URL: http://bit.ly/PCP3kj, Acessado em 09/07/2012.

[JXTA, 2012] JXTA. Jxta protocol, 2012. URL: http://java.net/projects/jxta/, Acessado em 05/07/2012.

[Karger et al., 1997] Karger, D., Lehman, E., Leighton, F., Levine, M., Lewin, D., and Panigrahy, R. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. 29th Annual ACM Symposium on Theory of Computing, (1997), 654-663.

[Kubiatowicz et al., 2000] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. 2000. OceanStore: an architecture for global-scale persistent storage. SIGOPS Oper. Syst. Rev. 34, 5 (November 2000), 190-201.

[Landers et al., 2004] Landers, M., Zhang, H., and Tan, K. PeerStore: better performance by relaxing in peer-to-peer backup. Proceedings of the Fourth International Conference on Peer-to-Peer Computing, (2004), 72-79.

[Leavitt, 2009] N. Leavitt, Is Cloud Computing Really Ready for Prime Time? IEEE Computer, 2009. 42(1): p. 15-20.

[Li & Dabek, 2006] J. Li and F. Dabek. F2F: Reliable storage in open networks.In Intl Workshop on Peer-to-Peer Systems, Santa BarbaraCA, Feb. 2006.

[Loest et al., 2009] S. R. Loest, M. C. Madruga, C. A. Maziero, and L. C. Lung. Backupit: An intrusion-tolerant cooperative backup system. In Proceedings of the 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science, pages 724–729. IEEE Computer Society, 2009.

[Malkhi & Reiter, 1997] D. Malkhi and M. Reiter. Byzantine quorum systems. In ACM Symposium on Theory of Computing, 1997.

[Martinian, 2012] Emin Martinian. Site Oficial. Distributed internet backup system (dibs). http://www.mit.edu/~emin/source_code/dibs/index.html. Acessado em 09/07/2012.

[Mattos, 2005] Mattos, Antonio Carlos M. Sistemas de Informação: uma visão executiva. São Paulo. Saraiva. 2005.

[McIlroy, 1969] M. D. McIlroy, "Mass Produced Software Components," in Software Engineering: Report on a conference sponsored by the NATO Science Committee, 1969, pp. 138--155.

[Meira, 2005] Meira, Fernando. Resilia: A safe & secure backup-system. Final year project, Engineerng Faculty of the University of Porto, May 2005

[Meyer, 1995] Meyer, Poul L. (1995) “ PROBABILIDADE Aplicações à Estatística”. Livros Técnicos e Científicos Editora. 2 Edição. Rio de Janeiro.

[Microsoft, 2012] Microsoft Skydrive service, http://skydrive.live.com/, 2012.

[Napster, 2012] Napster, Site Oficial, http://www.napster.com, Acessado em 09/07/2012.

[Oliveira, 2007] M. Oliveira. OurBackup: A P2P backup solution based on social networks, MSc Thesis, Universidade Federal de Campina Grande, Brazil, 2007

[Osterweil, 2007] Osterweil, L. J. 2007. A Future for Software Engineering?. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 1-11

[Papazoglou et al., 2007] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. 2007. Service-Oriented Computing: State of the Art and Research Challenges. Computer 40, 11 (November 2007), 38-45.

[Ratnasamy et al., 2001] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. 2001. A scalable content-addressable network. In Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications (San Diego, California, United States). SIGCOMM '01. ACM, New York, NY, 161-172.

[Rowstron & Druschel, 2001] Rowstron, A. and Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science, November 2001 (2001), 329-350.

[Schollmeier & Rudiger, 2002] Schollmeier, Rudiger. “A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications”. In: IEEE Internet Computing, 2002.

[Shvachko et al., 2010] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pages 1–10. IEEE Computer Society, 2010.

[Smith et al., 2009] D. M. Smith et al. Gartner Report ID Number G00163522, Predicts 2009: Cloud Computing Beckons, 2009.

[Stal, 2002] M. Stal, Web Services: Beyond Component-Based Computing, Communications of the ACM, Vol. 45, No. 10, October, 2002, pp. 71-76.

[Stoica et al., 2001] Stoica, I., Morris, R., Karger, D., Kaashoek, M., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, (2001), 160.

[Sun, 2009] Sun Microsystems, Introdution to cloud computing architecture whitepaper, June 2009.

[Turner et al., 2003] M. Turner, P. Brereton, and D. Budgen, "Turning Software into a Service," Computer, vol. 36, 2003, pp. 38-44.

[Vignatti et al., 2009] VIGNATTI, T. ; BONA, L. C. E. ; VIGNATTI, A. ; SUNYE, M. . Long-term Digital Archiving Based on Selection of Repositories Over P2P Networks. In: Ninth International Conference on Peer-to-Peer Computing (P2P'09), 2009, Seattle. Proceedings of Eight IEEE International Conference on Peer-to-Peer Computing, 2009.

[XDriver, 2012] XDriver Box service, http://www.box.net/xdrive, Acessado em 05/07/2012

[W3C, 2000] W3C, A Little History of the World Wide Web, 2000. URL: http://www.w3.org/History.html, Acessado em 05/07/2012.

[Webber et al., 2010] J. Webber, S. Parastatidis, and I. Robinson. REST in Practice: Hypermedia and Systems Architecture. O'Reilly Media, 2010.

[Yang et al., 2006] Q. Yang, W. Xiao, and J. Ren. Prins: Optimizing performance of reliable internet storages. In Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, pages 32–. IEEE Computer Society, 2006.

[Yu et al., 2007] L. Yu, G. Chen, W. Wang, and J. Dong. Msfss: A storage system for mass small files. In W. Shen, Y. Yang, J. Yong, I. Hawryszkiewycz, Z. Lin, J.-P. A. Barthes, M. L. Maher, Q. Hao, and M. H. Tran, editors, 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pages 1087–1092, Los Alamitos, CA, USA, April 2007. IEEE Computer Society Press.

[Zhang et al., 2007] Liang-Jie Zhang, Jia Zhang, Hong Cai, Services Computing, Springer and Tsinghua University Press, 2007, ISBN: 978-3-540-38281-2, July 2007

[Zhang et al., 2008] Liang-Jie Zhang, Carl K Chang, Ephraim Feig, Robert Grossman,Keynote Panel, Business Cloud: Bringing The Power of SOA and Cloud Computing, pp.xix, 2008 IEEE International Conference on Services Computing (SCC 2008), July 2008.

[Zhang and Zhou, 2009] Liang-Jie Zhang and Qun Zhou. 2009. CCOA: Cloud Computing Open Architecture. In Proceedings of the 2009 IEEE International Conference on Web Services (ICWS '09). IEEE Computer Society, Washington, DC, USA, 607-616.

[Zao et al., 2004] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 22, NO. 1, JANUARY 2004 41

[Grossman 2008] R. Grossman and Y. Gu. Data mining using high performance data clouds: experimental studies using sector and sphere. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '08, pages 920–927, New York, NY, USA, 2008. ACM.

[Ghemawat 2003] ,S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In ACM SIGOPS Operating Systems Review, volume 37, pages 29–43. ACM, 2003.

[Buyya 2009] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25(6):599 – 616, 2009.

[Palankar 2008] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon s3 for science grids: a viable solution? In Proceedings of the 2008 international workshop on Data-aware distributed computing, DADC '08, pages 55–64, New York, NY, USA, 2008. ACM.

[Ghemawat 2003] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In ACM SIGOPS Operating Systems Review, volume 37, pages 29–43. ACM, 2003.

[Mathur 2007] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier. The new ext4 filesystem current status and future plans. 2007.

[Steinacker 2001] A. Steinacker, A. Ghavam, and R. Steinmetz. Metadata standards for web-based resources. IEEE Multimedia, 8:70–76, 2001.

[Shvachko 2011] K. V. Shvachko. Apache Hadoop: The Scalability Update. Pages 7–13, 2011.

[XtreemFS 2008]F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario. The XtreemFS architecture: a case for object-based file systems. Concurrency and computation: Practice and experience, 20(17):2049–2060, 2008.

Capítulo 4

Análise de Informações Contextuais através de Técnicas de Aprendizagem de Máquina

Fábio Santos da Silva, Kátia Cilene Neles da Silva e Graça Bressan

Abstract

This chapter discusses how to employ some techniques from machine learning to facilitate the task of analyzing contextual information on context-sensitive systems. In addition, the chapter presents a proposed approach to this task. Finally, describes how to employ the techniques of machine learning in the proposed approach through the use of the Weka API.

Resumo

Este capítulo aborda como empregar algumas técnicas de aprendizagem de máquina para viabilizar a tarefa de análise de informações contextuais em sistemas sensíveis ao contexto. Além disso, o capítulo apresenta a proposta de uma abordagem para realização desta tarefa. Finalmente, descreve como empregar as técnicas de aprendizagem de máquina na abordagem proposta a através do uso da API Weka.

4.1. Introdução

O estudo da exploração das informações implícitas presentes na interação entre o ser humano e sistemas computacionais com o objetivo melhorar esta interação é uma linha de pesquisa que tem despertado cada vez mais atenção de pesquisadores (Dey e Abowd, 2001;Vieira et al., 2009; Alves, 2009; Silva, 2012). Uma categoria de informações abstraídas deste processo de interação é denominada de informações contextuais. Tais informações podem ser utilizadas por um sistema para automaticamente, em contextos diversos, adaptar seu serviço ou conteúdo, ou fornecer informações relevantes e personalizadas aos usuários. Este tipo de sistema está sendo referenciado na literatura como Sistema Sensível ao Contexto (do inglês, *Context-Sensitive System*) (Vieira et al., 2009) ou Sistema Ciente de Contexto (do inglês, *Context-Aware System*) (Dey e Abowd, 2001).

Segundo Vieira et al. (2009) as áreas da Computação Ubíqua e Inteligência Artificial foram às pioneiras nos estudos da exploração das informações contextuais em sistemas computacionais. Pesquisas recentes (Vieira et al., 2009; Silva, 2012; Neles, 2012) indicam que as informações contextuais estão sendo utilizadas em outras áreas como: *Hipermídia Adaptativa*, *Interface Humano-Computador*, *Sistemas de Recomendação* e *aplicações médicas.*

No entanto, o desenvolvimento de um sistema sensível ao contexto não é uma tarefa trivial, exige a superação de vários desafios técnicos como compreender, coletar, representar, analisar ou processar as informações contextuais. Desta forma, vem aumentando o interesse na concepção de metodologias de engenharia de software (Vieira et al., 2009), e no desenvolvimento de ferramentas para suporte a construção de sistemas sensíveis ao contexto tais como *toolkits* (Dey e Abowd, 2001), infraestruturas (Silva, 2011; Alves, 2009), *middlewares* (Gu; Pung; Zhang, 2005), entre outros.

Dentre os principais desafios técnicos se destaca a análise ou processamento de informações contextuais que consiste no uso de um conjunto de técnicas que permitem a partir de uma base de informações contextuais coletadas, ou um histórico de contexto, a inferência ou a geração de novas informações mais refinadas e relevantes que serão posteriormente empregadas para prover serviços como personalização e adaptação de conteúdo, recomendação de conteúdo, adaptação de interfaces, entre outros.

Atualmente, diversas técnicas vêm sendo utilizadas na implementação da tarefa de análise de informações contextuais tais como raciocínio baseado em regras, ou regras contextuais, raciocínio baseado em casos, tarefa de classificação, técnicas de aprendizagem de máquina, entre outras (Vieira et al., 2009). Em determinados tipos de sistemas sensíveis ao contexto (Silva, 2012; Alves, 2009) as técnicas de aprendizagem de máquina, estão se destacando como uma abordagem promissora para análise de informações contextuais.

Assim, este capítulo propõe-se apresentar os conceitos introdutórios de técnicas de aprendizado de máquina como classificador Bayesiano, Árvore de Decisão, Rede Neural Artificial e Máquina de Vetores de Suporte (Witten; Franck, 2005). Além disso, discutir uma abordagem para análise de informações contextuais baseada em tais técnicas, e finalmente demonstrar a tarefa de análise contextual por meio de exemplos práticos.

Neste capítulo, será utilizada a API da ferramenta Weka (Hall et al., 2009), desenvolvida na Universidade de Waikato na Nova Zelândia, que contempla, um conjunto de técnicas de aprendizagem de máquina implementado. Também será discutido um exemplo de uso da análise de contexto para recomendação personalizada de conteúdo em ambientes como Web, plataformas móveis e TV Digital.

O restante deste trabalho está estruturado como segue. A seção 4.2 apresenta uma visão geral sobre Computação Sensível ao Contexto, e discute os conceitos básicos sobre contexto e sistemas sensíveis ao contexto. A seção 4.3 apresenta uma visão geral sobre aprendizagem de máquina, e destaca algumas técnicas tradicionais empregadas nesta tarefa. A seção 4.4 descreve a abordagem de analise de informações contextuais baseada em aprendizagem de máquina. Na seção 4.5 um exemplo prático de implementação e emprego da abordagem é apresentado. Finalmente, a seção 4.6 apresenta as considerações finais deste capítulo.

4.2. Computação Sensível ao Contexto

A Computação Sensível ao Contexto ou Computação Ciente de Contexto é uma área de estudo que engloba pesquisas em diversas áreas tais como Inteligência Artificial, Computação Ubíqua, Redes Sensíveis a Serviços, Rede de Sensores Sem fio, TV Digital entre outros. Para todas essas áreas, o estudo dos aspectos contextuais envolvidos possibilita o aumento e melhoria da interação homem-máquina e máquina-máquina, com o objetivo de prover uma experiência de interação mais transparente e natural.

A Computação Sensível ao Contexto começou a ser discutida em meados de 1994 por Schilit e Theimer quando estes afirmaram que esta diz respeito à capacidade de um produto de software "se adaptar ao usuário de acordo com a sua localização, as pessoas e os objetos próximos e às mudanças destes ao longo do tempo". Schilit e Theimer expandiram o conhecimento acerca das aplicações sensíveis ao contexto afirmando que estas além de fornecerem informações sobre o contexto também tem a capacidade de se adaptar a este (Schilit e Theimer 1994 Apud Dey 2001).

Ainda Dey e Abowd (2000) apresentam uma definição mais abrangente e mais genérica que afirma que *"Um sistema é sensível ao contexto quando usa o contexto para fornecer informações e/ou serviços importantes para o usuário e essa importância depende da atividade ou da tarefa do usuário"*. Essa afirmação inclui todas as aplicações sensíveis ao contexto desde aquelas que se adaptam ao contexto quanto àquelas que o apresenta ao usuário final.

A Computação Sensível ao Contexto é um tema em expansão e seu estudo fornece uma base importante na construção de sistemas que envolvem a interação personalizada com o usuário e um comportamento inteligente. As próximas subseções destacam os principais conceitos relacionados à Computação Sensível ao Contexto.

4.2.1. Definição de Contexto

O contexto é um conceito que está presente de forma intrínseca nas atividades diárias das pessoas e durante o processo de interação humana este é empregado de maneira natural e implícita para melhorar a qualidade da comunicação (Silva, 2011). Em sistemas computacionais o conceito de contexto possibilita que haja maior aproximação deste com o usuário. Na literatura é possível encontrar diversas definições oriundas de diferentes áreas para o conceito de contexto. A seguir serão apresentadas algumas importantes definições deste conceito frequentemente referenciadas na literatura.

Uma definição clássica é apresentada em Dey (2001) em que "*Contexto é qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar, ou objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo o usuário e a aplicação em si*".

Zimmer (2004) e Ranganathan (2003) afirmam que o termo "Contexto" se refere a uma coleção de informações que caracterizam a interação entre o usuário e a aplicação. Tempo, localização, temperatura, luz, som e atividade são exemplos de informação contextual ou contexto. Também se relaciona com qualquer informação que pode ser usada para caracterizar circunstâncias, objetos ou condições pelos quais o usuário é rodeado.

A partir dessas definições tem-se que o contexto assume maior importância uma vez que este é característico de sistemas computacionais que visam interagir com o usuário de maneira inteligente e proativa.

4.2.2. Dimensões Semânticas – 5Ws+1H

As informações que constituem um contexto podem ser identificadas por meio de um conjunto de categorias de informações denominadas de dimensões semânticas. Tais informações contextuais são abstraídas durante o processo de modelagem do contexto de uma aplicação.

Dey (2001) e Abowd (2000) Apud Silva (2011) destacam que a identificação das informações contextuais pode ser facilitada por meio da utilização das dimensões semânticas ou 5Ws+1H: *when* (tempo), *where* (localização), *who* (identificação), *what* (atividade) e *why* (utilizada para se indicar o porquê da execução de uma determinada atividade, qual a motivação) (Dey, 2001; Abowd, 2000). Resumidamente essas dimensões semânticas podem ser observadas na Tabela 4.1.

Tabela 4.1 Dimensões semânticas ou 5Ws+1H

Dimensão	Descrição
Who	Quem realiza uma determinada atividade, quem pode alterar o contexto ou quem pode ser notificado caso o contexto seja alterado.
Where	Onde a pessoa está. Esta é uma das dimensões mais usadas devido ao grande interesse de sistemas baseados em localização.
When	A informação temporal para determinar quanto tempo uma entidade está dentro de um contexto. Esta dimensão associada com a dimensão *where* permite rastrear os caminhos que uma entidade tomou durante um período.
What	O que o usuário está fazendo neste momento. Geralmente necessita de sensores para determinar qual é a atividade, o que torna isso uma tarefa complexa.
Why	Determinar o porquê o usuário está realizando determinada atividade, essa é umas das tarefas mais difíceis por envolver questões de inteligência artificial.
How	Determina a forma como as informações contextuais serão capturadas.

Em Troung (2001), observa-se que devido à dificuldade na obtenção de informações relativas à dimensão *Why*, geralmente associa-se a dimensão *What* com uma sexta dimensão *How* (como) para inferir informações sobre *Why*. Para aplicações específicas, como captura e acesso, a dimensão *How* é importante e pode definir como os dados foram capturados e como eles podem ser acessados.

4.2.3. Sistemas Sensíveis ao Contexto

Sistemas sensíveis ao contexto são sistemas que reagem e se adaptam às mudanças, podendo fornecer serviços e informação de acordo com o contexto do usuário. Além disso, tais sistemas agem automaticamente, reduzindo o excessivo envolvimento do usuário e provendo assistência inteligente e proativa. Para Dey (2001) um sistema é dito sensível ao contexto se este utiliza o contexto para apresentar informação e serviço relevantes para o usuário dependendo da tarefa desempenhada.

A Figura 4.1 Silva (2011) destaca uma comparação entre os sistemas tradicionais e sistemas sensíveis ao contexto. Nos sistemas tradicionais há uma entrada explícita e clara que após processamento fornece uma saída que segue um padrão. Já nos sistemas sensíveis ao contexto há além da entrada explícita, a percepção do

contexto, e a inferência de conhecimento a partir de uma base de conhecimento para então fornecer serviços adaptados ao usuário. Este processo de inferência pode ser implementado, por meio de técnicas de aprendizagem de máquina.

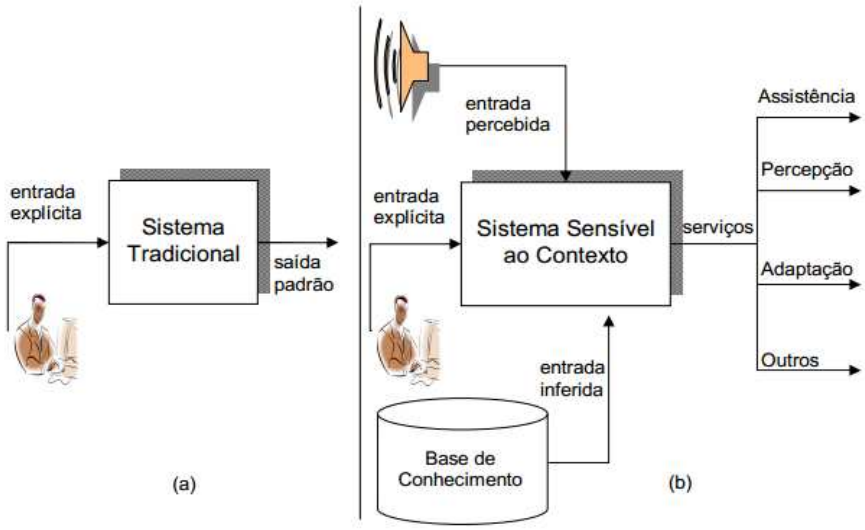


Figura 4.1. Sistema tradicional (a) vs. Sistema sensível ao contexto (b). (Vieira et al, 2009 Apud Silva 2011)

4.2.4. Exemplos de Sistemas Sensíveis ao Contexto

Em Pessoa (2006) é possível encontrar diversas referências de sistemas sensíveis ao contexto, conforme destacadas abaixo:

- Solar system: é uma arquitetura distribuída em componentes, organizada de maneira semelhante ao sistema solar, que oferece apoio à realização da Computação Sensível ao Contexto por parte das aplicações. Foi desenvolvido em Dartmouth Colleg por (Chen et al., 2002);
- Socam (*Service-Oriented Context-Aware Midleware):* é uma arquitetura orientada a serviços que utiliza um modelo forma de contexto com o objetivo de prover prototipação rápida de serviços e aplicações sensíveis ao contexto em ambientes de Computação Pervasiva. Essa arquitetura foi desenvolvida na Computing School of Singapure National University (Guet al., 2005);
- GaiaOS: trata-se de uma plataforma que introduz um ambiente de computação composto por um espaço físico qualquer enriquecido com dispositivos eletroeletrônicos capazes de realizar algum tipo de computação útil aos frequentadores do ambiente (Roman, 2002);
- Aura: é uma plataforma resultante de trabalhos desenvolvidos na Carnegie Mellon University, tem como intuito prover a usuários móveis uma infraestrutura de computação virtual, particular e repleta e serviços capazes de se adaptar em ambientes propensos a falhas e com variabilidade de recursos.
- CoBrA: arquitetura com distribuição híbrida, composta de um componente central denominado ContextBroker. Além do componente central (*broker*), essa

arquitetura também incorpora outros agentes configuráveis para notificar as aplicações acerca de mudanças de contexto.Foi desenvolvida na University of Maryland (Chen, 2004);

- Wasp (Web *Architectures for service plataforms*): arquitetura que tem como característica particular o uso da tecnologia de distribuição de serviços Web sendo executada sobre uma rede 3G. Foi desenvolvida na Universtity of Twente, na Holanda (Dockhorn, 2003; Santos, 2004).

Em Vieira et al (2009) outros exemplos de sistemas sensíveis ao contexto que usam pares de chave-valor para representar o contexto incluem o Context Toolkit (Dey et al., 2001) e o CXMS (Context Management System) (Zimmermann et al., 2005).

4.2.4.1. Personalização e Adaptação de Conteúdo baseadas em Contexto

O conceito de sensibilidade ao contexto tem uma estreita relação com a capacidade das aplicações serem personalizáveis ou adaptáveis. Esta capacidade relaciona-se com o comportamento, suas interfaces de apresentação ao usuário, a interação que esta pode ter com sistemas externos, ou ainda em seu conteúdo.

Neste sentido propostas de têm sido apresentadas buscando empregar as vantagens da sensibilidade ao contexto para a melhoria da comunicação entre dos sistemas para com o usuário final. Um exemplo de estudo voltado para personalização e adaptação de conteúdo pode ser observado no trabalho de Goularte (2003) no qual propõe técnicas com suporte à sensibilidade a contexto para adaptação e personalização de conteúdo em TV Interativa. As técnicas desenvolvidas são baseadas em esquemas de descrição compatíveis com o padrão MPEG-7, e na segmentação de conteúdo em objetos MPEG-4. Também foi definida e implantada uma infraestrutura para produção, distribuição, e consumo de programas de TV.

Já o trabalho de Zimmermann (2005) destaca o sistema *Listen* que rastreia o usuário com uma solução que permite identificar se o usuário está olhado para um detalhe de uma obra de arte em uma galeria ou apenas ao lado da obra de arte.

4.2.4.2. Recomendação Sensível ao Contexto de Conteúdo

Os sistemas de recomendação são desenvolvidos como forma de auxiliar os usuários na identificação de itens de seu interesse. Com o contínuo aumento da quantidade de serviços e informações disponíveis e por conseguinte a necessidade de acesso, torna-se cada vez mais necessária a tarefa de recomendação de conteúdos. Conceitualmente, um sistema de recomendação é um tipo de sistema que apresenta ao usuário de maneira personalizada os objetos úteis ou de seu interesse a partir de diversas opções possíveis. É também descrito como uma técnica inteligente para lidar com o problema da sobrecarga de informação (Vozalis e Margaritis 2003, Burke, 2002, Apud Silva 2011).

Silva (2011) destaca que a qualidade das recomendações geradas pelos tradicionais sistemas de recomendações pode ser melhorada por meio da exploração do contexto. O autor ressalta que informações contextuais podem ser empregadas para inferir preferências contextuais do usuário posteriormente utilizadas no processo de recomendação visando retornar itens que estão mais próximos dos interesses do usuário. Para isso torna-se necessária a utilização de técnicas cada vez mais complexas que

viabilizem as tarefas de aquisição, representação, análise de informações contextuais e filtragem sensível ao contexto.

No trabalho de Silva (2011) é proposta uma infraestutura de software alinhada aos padrões de TV Digital Interativa para suporte ao desenvolvimento e execução de sistemas recomendação sensíveis ao contexto para TV Digital Interativa. Foram implementadas esquemas para representação de contexto e conteúdo compatíveis com os padrões MPEG-7 e TV-Anytime e técnicas para análise de contexto e recomendação de conteúdo. Assim, os desenvolvedores de sistemas de recomendação concentram esforços na lógica de apresentação de seus sistemas, deixando questões de baixo nível para infraestrutura gerenciar.

4.2.4.3. Monitoramento remoto e sensível ao contexto da saúde humana

Nas aplicações na área de saúde, a sensibilidade ao contexto tem sido o foco em diversas pesquisas, especialmente naquelas relacionadas à mobilidade dos pacientes e dos profissionais de saúde.

Este tipo de aplicação emprega o conceito de sensibilidade ao contexto a partir da interação entre os sensores empregados por meio de uma rede de sensores. Uma vez que estes ocorrem à distância é importante que os especialistas médicos considerem o contexto em que o paciente está inserido para então direcionar tratamento e diagnóstico. As informações contextuais são capazes de alterar inclusive as informações fisiológicas dos pacientes monitorados (Neles, 2012).

Neste tipo de aplicação, os contextos podem ser classificados como de baixo e de alto nível (Abowd, 2000).

- Contexto de baixo nível: obtido diretamente de sensores ou através de processamento simples;
- Contexto de alto nível: obtido através da composição de informações contextuais de baixo nível ou a partir da análise de informações contextuais de baixo nível por meio de técnicas de Inteligência Artificial. Exemplo: uso de informação de tempo, localização e agenda do usuário como auxiliares da definição de sua situação social: “em reunião”, “recebendo medicação”, “cozinhando”.

Nesta relação, o contexto de baixo nível fornece dados oriundos de sensores e perfis ao contexto de alto nível que através de um processador provê a informação contextual (onde, que, quando, por que) que será usada de maneira ativa ou passiva pela aplicação, possibilitando assim a adaptação do seu comportamento

4.2.5. Requisitos para Implementação de Sistemas Sensíveis ao Contexto

Na literatura e possível encontrar um conjunto de requisitos que são bastante comuns na implementação de sistemas sensíveis ao contexto (Raatikainen et al. 2002), (Strang; Linnhoff, 2004), (Henricksen et al. 2005), e (Vieira et al., 2006). Este conjunto de requisitos é genérico, e visa orientar a implementação deste tipo de sistema. De acordo com o domínio e aplicação os requisitos possuem graus de relevância diferentes. A seguir serão descritos alguns desses principais requisitos.

4.2.5.1. Abstração de Informações Contextuais

Este requisito consiste em identificar quais informações contextuais serão exploradas pela aplicação. Para facilitar esta tarefa, as informações contextuais podem ser identificadas por meio do emprego das dimensões semânticas básicas, referenciadas como 5Ws+1H conforme descrito na seção 4.2.2. O desenvolvedor pode utilizar de modelos formais que viabilizem a representação em alto nível das principais entidades, suas respectivas informações contextuais e seus relacionamentos. Por meio desta representação se torna possível a utilização e a compreensão semântica das informações contextuais por usuários e sistemas computacionais.

Segundo Vieira et al. (2009), na escolha de um modelo formal também se deve levar em consideração fatores como interoperabilidade, extensibilidade, compartilhamento e reusabilidade. Na literatura é possível encontrar alguns trabalhos que apresentam propostas de técnicas para representação de contexto como *par chave-valor*, *linguagem de marcação*, *orientação a objetos*, *mapas de tópicos* e *ontologias* (Strang; Linnhoff-Popien, 2004; Vieira et al. 2009).

4.2.5.2. Separar Aquisição da Utilização das Informações Contextuais

A separação destas tarefas, viabiliza que os sistemas utilizem as informações contextuais sem se preocupar com os detalhes de como tais informações foram adquiridas e tratadas.O objetivo é tornar cada tarefa o mais independente possível, pois as informações contextuais podem ser obtidas a partir de fontes heterogêneas por meio de sensores físicos ou lógicos, de bases de dados, agentes inteligentes, ou até mesmo em último caso fornecidas explicitamente pelo próprio usuário.

Além disso, a independência dos componentes de aquisição em relação às aplicações permite que diversos sistemas possam fazer uso destes, de forma compartilhada. Deste modo, por meio desta separação de tarefas, a complexidade da aquisição e o tratamento da qualidade das informações contextuais são abstraídos para as aplicações.

4.2.5.3. Análise das Informações Contextuais

A análise, processamento ou interpretação das informações de contexto é um dos requisitos mais importantes de sistemas sensíveis ao contexto. Consiste no emprego de um conjunto de métodos e processos que viabilizam a aprendizagem, o raciocínio, a derivação, a predição de tendências, padrões ou preferências sobre as informações contextuais. O objetivo é obter informações de alto nível que permitam a melhorar a compreensão de um determinado contexto pela aplicação de modo a modificar o seu comportamento ou auxiliá-la na tomada de decisões.

Desta forma, é possível obter informações contextuais de alto nível que serão posteriormente exploradas pelas aplicações para diversas finalidades. Por exemplo, predizer a melhor rota de ambulâncias para hospitais através de informações contextuais sobre as condições do trânsito; descobrir qual gênero de música poderá ser recomendado para um usuário de acordo com seu o humor e ambiente.

Na literatura é possível encontrar a utilização de diversas técnicas na implementação do requisito de análise de informações contextuais tais como raciocínio baseado em regras, ou regras contextuais, raciocínio baseado em casos, tarefa de

classificação, técnicas de aprendizagem de máquina, entre outras (Vieira et al., 2009). Em determinados trabalhos (Silva, 2012; Alves, 2009) as técnicas de aprendizagem de máquina, se destacaram como uma abordagem promissora para análise de informações contextuais.

4.2.5.4. Comunicação Distribuída e Transparente

Em termos de arquitetura, os sistemas sensíveis ao contexto são modulares e distribuídos. Assim, a comunicação entre seus componentes dever ser implementada de forma flexível e padronizada. Para isso, protocolos de comunicação, padrões para integração de componentes devem ser utilizados em sistemas sensíveis ao contexto. Além disso, questões também relacionadas à segurança, privacidade e escalabilidade também devem ser consideradas.

Podem ser adotadas tecnologias como Serviços Web (do inglês, *Web Services*) (W3C, 2002). Serviço Web oferece um conjunto de padrões que propiciam uma comunicação padronizada entre diferentes aplicações, promovendo a integração entre elas. A descrição dos principais serviços é feita por meio da WSDL (*Web Service Definition Language*). Já a comunicação é viabilizada por meio da troca de mensagens XML baseada em SOAP (*Simple Object Acess Protocol*) (W3C, 2003). Tais mensagens XML são transportadas por meio de protocolos da família TCP/IP (W3C, 2002).

4.2.5.5. Aquisição Continua de Informações Contextuais

A aquisição de informações contextuais deve ser realizada de forma continua e não intrusiva, sem que usuário tenha que ser questionado insistentemente sobre o contexto em que se encontra. Para isso, sensores eletrônicos e lógicos podem ser empregados para o monitoramento do contexto do usuário. A disponibilidade constante destes componentes viabiliza a aquisição de informações em diversos contextos.

Algumas informações contextuais não podem ser obtidas diretamente por meio de componentes de aquisição. Torna-se necessária a utilização de técnicas de raciocínio para derivar novas informações a partir da composição de informações contextuais primárias ou de baixo de nível. Por exemplo, a informação sobre qual atividade o usuário está realizando “estudando” ou “trabalhando” pode ser derivada a partir de condições de teste de uma árvore de decisão sobre as informações contextuais de tempo e localização.

4.2.5.6. Armazenamento de Informações Contextuais

A necessidade da disponibilidade constante das informações contextuais; o compartilhamento destas informações entre os sistemas; e a tarefa de análise ou raciocínio sobre tais informações demanda o armazenamento das informações contextuais em repositórios contextuais. Torna-se necessário armazenar de forma eficiente um grande volume de informações contextuais levando em consideração o dinamismo do contexto. Para isso, as abordagens tradicionais para armazenamento de dados devem ser analisadas com o objetivo de identificar as abordagens mais adequadas para armazenamento e recuperação de tais informações.

4.2.5.7. Descoberta de Componentes ou Recursos

Em determinados sistemas pode ser necessário descobrir quais componentes estão disponíveis para apoiar a realização de determinadas tarefas. Por exemplo, se um componente que fornece determinada informação não estiver funcionando, torna-se necessário encontrar automaticamente outros componentes similares, ou até mesmo efetuar uma composição de componentes de forma a fornecer uma resposta adequada. Assim, automatizar a descoberta de recursos é um requisito importante em sistemas sensíveis ao contexto.

Devido ao aumento do número de componentes disponibilizados, surge um problema para os serviços de descobertas, que é o de localizar o componente mais apropriado para determinada operação (Forstadius et al. 2005). A descoberta de recursos pode ser implementada por meio da *computação orientada a serviços* (Papazoglou, 2003), o qual utiliza os serviços Web para composição de componentes considerados elementos fundamentais na construção de aplicações distribuídas.

4.2.6. Gerenciamento de Contexto

Conforme já foi destacado, os sistemas sensíveis ao contexto são aqueles que utilizam contexto para execução de uma tarefa como prover informações personalizadas e adaptadas ou serviços relevantes. No entanto, compreender e identificar o contexto e executar tarefas de acordo com as mudanças deste requer a superação de requisitos técnicos. Torna-se preciso lidar com questões como: que tipo de informação considerar como contexto, em qual formato representá-las, como podem ser adquiridas e analisadas ou processadas, e como serão efetivamente utilizadas para alterar o comportamento do sistema.

Diante deste cenário, o gerenciamento de contexto ocupa um papel muito importante em sistemas sensíveis ao contexto, pois envolve conforme ilustrado na Figura 4.2 proposta por (Vieira et al., 2009) as tarefas de aquisição, processamento ou análise de informações contextuais, e disseminação.

Figura 4.2. Principais elementos envolvidos no gerenciamento do contexto. (Vieira et al., 2009)

A tarefa de aquisição refere-se ao processo de captura de informações contextuais, que podem ser obtidas de diferentes fontes de contexto. A tarefa de processamento emprega um conjunto de técnicas para raciocínio, aprendizagem, predição com o objetivo de inferir preferências, produzir informações refinadas, derivar novas informações contextuais de alto nível. Por fim, é na disseminação que o resultado do processamento é empregado.

Em um sistema de recomendação, por exemplo, o resultado pode ser uma classe de conteúdo que deverá ser recomendado para o usuário (Silva, 2011). Segundo Samaras (1996) quando uma aplicação emprega a gerência de contexto essa gerência mantém controle sobre vários contextos, permite a aplicação criar um conjunto de contextos para trabalho e permite que essa aplicação altere o contexto quando apropriado. A gerência de contexto compartilha a noção de contexto com a gerência de recursos e a aplicação.

4.2.7. Técnicas de Representação de Contexto

A representação de contexto visa fornecer uma visão de alto nível, para que seja possível o compartilhamento, a utilização e a compreensão semântica das informações contextuais por usuários e sistemas computacionais. Para isso, uma representação de contexto apropriada deve especificar as principais entidades e informações contextuais que deverão ser processadas pelo sistema.

No trabalho de Vieira et al. (2009), é apresentada uma análise comparativa das técnicas mais empregadas para representação de contexto, onde são resumidas as vantagens e desvantagens de cada uma delas, assim como, a dos métodos utilizados para processamento e recuperação de contexto de cada uma das abordagens. Algumas dessas principais técnicas são sumarizadas na Tabela 4.2.

Além das técnicas apresentadas na Tabela 4.2, pode ser adicionada a técnica baseada em Orientação a Objetos, que oferece como vantagens: encapsulamento, herança, e reusabilidade. No cenário da TV Digital, em (Goularte, 2003), foi utilizada a linguagem de marcação (XML) e Esquema XML para criar uma biblioteca extensível de elementos contextuais estruturados. Tal biblioteca foi utilizada para o desenvolvimento de um serviço de adaptação e personalização de conteúdo para TV Digital.

Cada técnica de representação possui recursos próprios que podem facilitar a representação de contexto. Deste modo, não há uma técnica que seja considerada a ideal para todos os sistemas sensíveis ao contexto, uma vez que diferentes sistemas impõem diferentes restrições.

Tabela 4.2. Técnicas para representação de contexto.

Técnica	Vantagens	Desvantagens	Processamento e Recuperação
Par chave-valor	Estrutura simples, de fácil implementação e uso.	Não considera hierarquia. Inadequado para aplicações com estruturas complexas.	Busca linear com casamento exato de nomes.
Linguagem de marcação	Baseado em XML. Prevê hierarquia. Esquema de marcação implementa o próprio modelo. Utilização típica em *perfis*.	Incompletude e ambiguidade na informação devem ser tratadas pelo sistema. Inadequado para representar estruturas complexas.	Linguagem de consulta baseada em marcação.
Mapas de tópicos	Facilita a navegação entre os contextos. Facilita a modelagem por humanos.	Estágio inicial. Tecnologia imatura. Faltam exemplos reais.	Navegação por redes semânticas.
Ontologias	Contextos modelados como conceitos e fatos. Viabiliza formalização, compreensão e compartilhamento por humanos e computadores.	Tecnologia de manipulação imatura.	Motor de inferência, linguagem de consulta baseada em OWL
Modelos Gráficos	Facilita a especificação dos conceitos e definição do comportamento do CSS.	Não permite processar os conceitos: mapeamento para estruturas de dados.	Pode ser traduzido para XML e usa processamento em XML.

4.3. Aprendizagem de Máquina

A Aprendizagem de Máquina (do inglês, *Machine Learning*) é uma subárea da Inteligência Artificial que se concentra no desenvolvimento de técnicas que permitem sistemas computacionais aprender e predizer padrões ou tendências a partir de um grande volume de dados (Witten e Franck, 2005; Han e Kamber, 2006; Tan et al., 2009). Suas técnicas são exploradas, principalmente, pelas áreas de Recuperação de Informação, Mineração de Dados, Reconhecimento de Padrões, Robótica e Jogos.

Segundo Han e Kamber (2006) as principais abordagens de aprendizagem de máquina são:

- **Aprendizagem supervisionada**: consiste na aprendizagem de uma função a partir de um conjunto de registros com rótulos de classes conhecidos. Tais rótulos correspondem ao valor da classe que se espera ser descoberta pela função da técnica de aprendizagem sempre que receber registros com rótulos desconhecidos;
- **Aprendizagem não-supervisionada**: se refere à aprendizagem de padrões a partir de um conjunto de registros em que não são fornecidos os rótulos de classes.

Para a abordagem geral de análise de informações contextuais discutida neste trabalho foi empregada a aprendizagem supervisionada. O objetivo desta abordagem de análise é predizer os rótulos de classes de registros de testes a partir de registros de treinamento compostos por atributos preditivos e de classe correspondentes as dimensões semânticas 5Ws+1H. A implementação desta abordagem tem como base a tarefa de classificação discutida na próxima seção.

4.3.1. Tarefa de Classificação

A tarefa de classificar objetos em uma dentre as diversas classes pré-definidas, é um problema amplamente discutido na literatura (Witten e Franck, 2005; Han e Kamber, 2006; Tan et al., 2009). Por meio da classificação é possível analisar um conjunto de dados para obter modelos que podem ser usados para predizer classes, padrões ou tendências futuras e apoiar a tomada de decisão.

Por exemplo, um gerente de *marketing* de uma loja virtual pode utilizar um algoritmo de classificação para analisar o histórico de compras de um cliente para predizer se ele desejará comprar um determinado tipo de computador. Em outro exemplo, um pesquisador pode empregar algoritmo de classificação para analisar dados obtidos de pacientes sobre um determinado tipo de câncer para predizer dentre os tipos de tratamentos qual o seu paciente deverá receber.

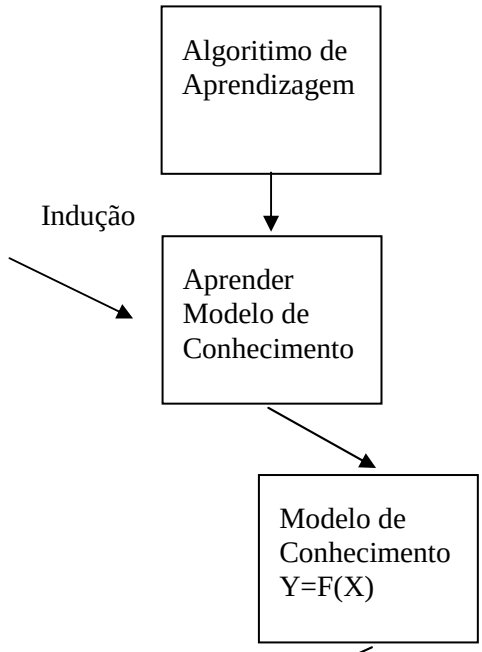
Em cada um destes exemplos, a tarefa de análise consiste na criação de um modelo a partir de um conjunto de dados aprendidos. Tal modelo permite ao sistema ao predizer um rótulo de classe, tais como “computador A” ou “computador B” para o segundo exemplo, ou ainda “tratamento A” ou “tratamento B” ou “tratamento C” para o segundo exemplo por meio de dados dos pacientes. Os dados de entrada da tarefa de classificação são denominados de registros, tuplas ou instâncias. Cada registro pode ser definido por uma dupla (x,y), onde *x* corresponde ao conjunto de atributos preditivos e *y* é um atributo especial, conhecido como atributo-classe (Tan et al., 2009). O conjunto de registros pode ser obtido a partir de uma base de dados, de um histórico de uso, entre outros.

Segundo Han e Kamber (2006) a tarefa de classificação é realizada em duas etapas que são: aprendizagem e predição. A aprendizagem é a etapa onde o algoritmo de aprendizagem constrói o *modelo de conhecimento* por meio da aprendizagem do conjunto de registros de treinamento com seus rótulos de classes conhecidos. O *modelo de conhecimento* obtido pode ser compreendido como uma função, $y=f(X)$, que permite predizer o rótulo, *y,* de uma classe pré-definida para o atributo-classe de um dado *registro de teste*. Uma vez aprendida essa função, a mesma pode ser aplicada posteriormente a vários registros de testes com rótulos de classe desconhecidos de forma a predizer os rótulos, *y,* de classe. Esta etapa é a chamada *predição de classe* ou *etapa de predição*.

A Figura 4.3 apresenta uma visão geral da tarefa de classificação aplicada para o problema de jogar futebol. O objetivo é prever a classe que determina se os jogadores devem jogar futebol. Neste exemplo, o atributo *Jogar* é denominado de atributo-classe, cujos possíveis valores são “sim” ou “não”. Por meio do *modelo de conhecimento* obtido a partir do conjunto de treinamento é possível obter para os registros de testes as classes desconhecidas.

Conjunto de Registros de Treinamento

Tempo	Temperatura	Umidade	Jogar
Sol	30	80	Sim
Sol	26,6	30	Não
Chuva	20	80	Sim
Chuva	16	70	Não
Sol	42	90	Não
Nublado	23	80	Sim
Nublado	9	70	Não



Conjunto de Registros de Testes

Tempo	Temperatura	Umidade	Jogar
Sol	30	80	?
Chuva	20	80	?
Nublado	23	70	?

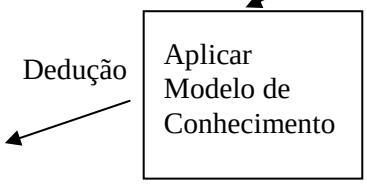


Figura 4.3. Visão Geral da Tarefa de classificação. Adaptado de (Tan et al., 2009)

4.3.2 Introdução a Técnicas de Classificação

Nesta seção será apresentada uma breve introdução de quatro técnicas de classificação de dados: Árvore de Decisão, classificador Bayesiano, Rede Neural Artificial e Máquina de Vetores de Suporte (Tan et al., 2009). Tais técnicas são foram utilizadas em trabalhos como (Silva, 2012; Alves, 2009), e implementadas na abordagem de análise de informações contextuais. De acordo com Tan et al.(2009) uma técnica de classificação é uma abordagem sistemática para criação dos *modelos de conhecimento* ou *modelos de classificação* a partir de um conjunto de registros de entrada. Cada técnica emprega um algoritmo de aprendizagem de máquina específico para criar um *modelo de conhecimento* que possa prevê com maior precisão os rótulos, *y*, de classes dos registros.

4.3.2.1. Indução da Árvore de Decisão

Uma Árvore de Decisão é uma técnica de classificação simples que explora um conjunto de questões e as suas possíveis respostas por meio dos atributos de um registro de teste visando obter o rótulo desconhecido de classe. Conforme ilustrado na Figura 4.4, tais questões e as suas possíveis respostas são organizadas por meio de uma estrutura hierárquica composta por nodos e arestas direcionadas.

A Árvore de Decisão possui nodos terminais ou não terminais. Os nodos terminais ou nodos folha são os que possuem um rótulo de classe, onde cada nodo possui uma aresta que chega e nenhuma que sai. Enquanto, que os nodos não terminais incluem o nodo raiz que é o mais alto na hierarquia da Árvore de Decisão e os demais nodos internos. Cada um desses nodos possui condições de teste sobre os atributos que correspondem às questões e as arestas as possíveis respostas. Desta forma, dado um registro, *X*, de teste para o qual o rótulo, *y*, da classe é desconhecido, os valores dos atributos preditivos do registro são testados contra a árvore de decisão. Na Figura 4.4 um caminho é traçado indicando os resultados das condições de teste sobre os atributos a partir do nodo raiz até um nodo folha, que possui o rótulo, *y*, da classe para este registro de teste.

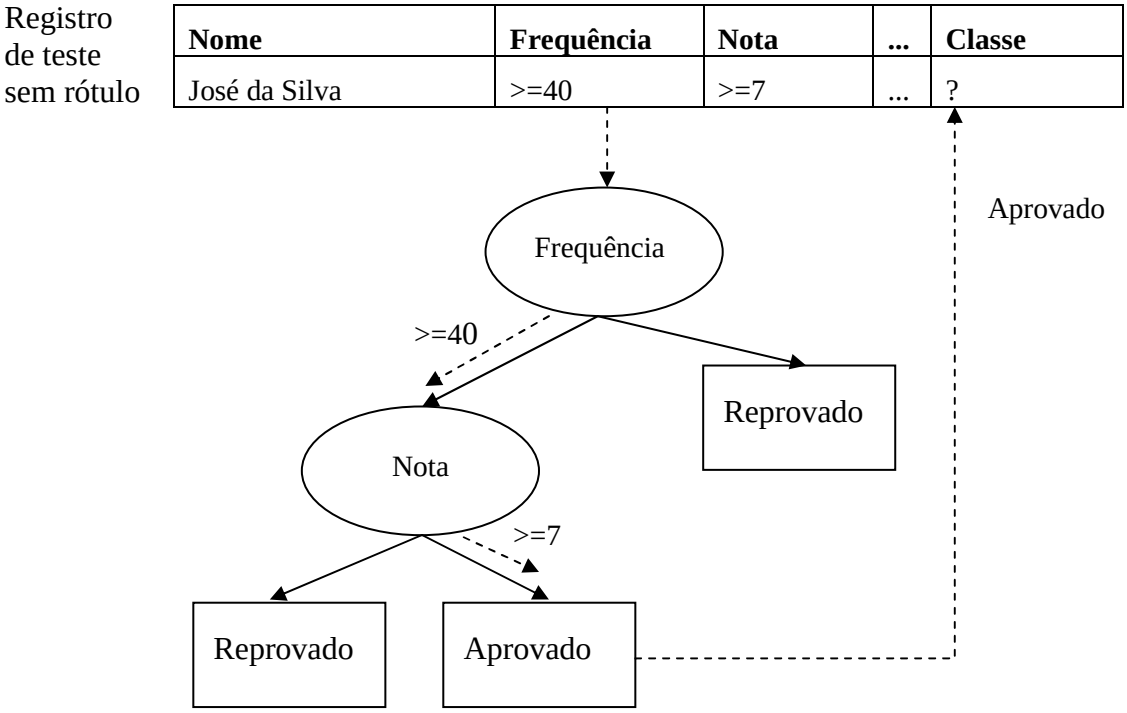


Figura 4.4. Árvore de Decisão classificando um registro de teste sem rótulo. Adaptado de (Tan et al., 2009)

Algoritmos eficientes de Árvore de Decisão têm sido desenvolvidos para tarefa de classificação, e empregados em muitas áreas de aplicação, tais como medicina, análise financeira, astronomia, produção, manufatura, entre outros. O ID3 (*Interactive Dichotomiser*), CART (*Classification and Regression Trees*) e o C4.5 são os algoritmos de árvore de decisão questão entre os mais tradicionais (Han e Kamber, 2006).

4.3.2.2. Classificador Bayesiano

Um classificador Bayesiano é fundamentado em um teorema estatístico conhecido como o teorema de Bayes, que pode ser empregado para resolver o problema da previsão (Tan et al., 2009). Desta forma, é uma técnica de classificação que se destaca por empregar o cálculo das probabilidades para classificar um registro de teste com rótulo, *y*, de classe desconhecido.

Suponha que *X* denote um registro formado por um conjunto de atributos, e *Y* o rótulo de uma determinada classe. Se *X* e *Y* possuem um relacionamento não determinístico, então podem ser tratadas como variáveis randômicas, e calculada sua probabilidade condicional *P(Y|X)* também chamada de probabilidade posterior de *Y*, por meio do emprego da Fórmula 1, conhecida como teorema de Bayes:

$$P(Y \mid X) = \frac{P(X \mid Y) \times P(Y)}{P(X)} \quad (1)$$

O teorema Bayes permite obter a probabilidade condicional *P(Y|X)* por meio da combinação da probabilidade condicional da classe *P(X|Y)*, probabilidade anterior *P(Y)*, e a evidência *P(X)*. A probabilidade anterior *P(Y)* é obtida a partir do conjunto de treinamento calculando-se a fração dos registros que pertence a cada classe, já *P(X)* é sempre constante para todas as classes e pode ser ignorada. Com relação à probabilidade condicional de classe *P(X|Y)* podem ser empregadas duas implementações de algoritmos de classificação Bayesianos: o método Bayes simples ou ingênuo e a rede de crenças Bayesiana (Han e Kamber, 2006;Tan et al., 2009).

O processo de aprendizagem consiste em calcular as probabilidades condicionais *P(Y|X)* para cada combinação *X* e *Y* dos registros obtidos do conjunto de treinamento. A partir das probabilidades obtidas, um registro, *X'*, de teste pode ser classificado de acordo com a classe *Y'* que maximiza a probabilidade posterior *P(Y'|X')* (Tan et al., 2009).

Para ilustrar essa técnica, suponha que se deseja prever se uma pessoa que assiste TV deseja assistir um determinado tipo de programa de TV. A Tabela 4.3 apresenta um conjunto de treinamento composto por registros com os seguintes atributos: *Lugar*, *Dia*, *Período*, *Programa de TV* e *Assistir-Programa de TV*. Os programas de TV que devem ser de assistidos pelo usuário são classificados como *Sim*, caso contrário são classificados como *Não*.

Tabela 4.3. Conjunto de treinamento de programas de TV.

Lugar	Dia	Período	Programa de TV	Assistir-Programa de TV
Casa	Domingo	Noite	Filme	Sim
Escritório	Segunda	Manhã	Desenho	Não
Casa	Sábado	Tarde	Desenho	Sim
Carro	Sábado	Noite	Filme	Não
Escritório	Segunda	Manhã	Economia	Sim

Para classificar um registro, *X*, de teste composto pelos seguintes atributos: *X*=(Lugar=Casa, Dia = Domingo, Período = Manhã, Programa de TV= Esportes), deve-se, calcular as probabilidades posteriores *P(Sim|X)* e *P(Não|X)* a partir dos dados dos registros do conjunto de treinamento. Se a probabilidade posterior *P(Sim|X)> P(Não|X)*, então o registro de teste é classificado como *Sim*, senão é classificado como *Não*.

4.3.2.3. Rede Neural Artificial

Redes Neurais Artificiais (RNAs) são algoritmos inspirados nos princípios de funcionamento dos neurônios biológicos para realização de tarefas tais como classificação, previsão de séries temporais, reconhecimento de padrões, entre outras (Tan et al., 2009). O modelo básico de uma rede neural artificial ilustrado na Figura 4.5 é conhecido como *perceptron,* e foi proposto por McCulloch e Pitts (1943).

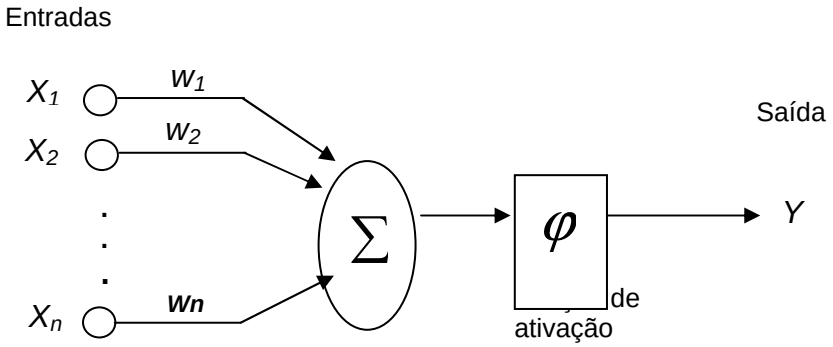


Figura 4.5. Modelo Básico de um perceptron (Han e Kamber, 2006)

Um *perceptron* é composto pelos os seguintes elementos: (i) um conjunto de *n* unidades x_1, x_2, ..., x_n, que representam os atributos de entrada de um registro com os seus pesos multiplicativos correspondentes (w_n); (ii) uma unidade somadora $\sum$ para acumular o produto de cada entrada com o peso multiplicativo; (iii) uma função de ativação φ que determina a saída efetiva *Y* do neurônio.

Para ilustrar o funcionamento de um *perceptron*, suponha que as suas entradas x_1, x_2, ..., x_n recebam valores booleanos (0 ou 1), que os pesos multiplicativos w_1, w_2, ..., w_n possuem valores reais e que tenha um fator *t* de tendência. O *percepron* poderá obter o valor de saída, *Y*, por meio uma função de ativação, que subtraia o valor obtido da unidade somadora $\sum$ por um fator *t* de tendência e então analisa o valor do resultado. Este modelo pode ser expressado matematicamente da seguinte forma:

$$Y = 1, \text{ se } w_1x_1 + w_2x_2 + \ldots w_nx_n - t > 0; \qquad Y = -1, \text{ se } w_1x_1 + w_2x_2 + \ldots w_nx_n - t < 0. \tag{2}$$

Na literatura é possível encontrar propostas de funções de ativação tais como lineares, tangentes hiperbólicas, sigmóides , dentre outras (Han e Kamber, 2006;Tan et al., 2009). Com relação ao processo de aprendizado de um *perceptron,* consiste em ajustar os pesos multiplicativos por meio de algoritmos específicos até que as saídas obtidas sejam consistentes aos rótulos de classes dos registros de treinamento (Tan et al., 2009).

Conforme ilustrado na Figura 4.6 por meio da combinação de *perceptrons* é possível forma uma estrutura mais complexa de rede neural artificial. A arquitetura de uma rede neural pode ser baseada em uma ou múltiplas camadas. Por exemplo, uma rede neural com três camadas (*Perceptron Multicamadas*) é composta por uma camada de entrada, em quem que os neurônios recebem os estímulos (ou sinais); a camada intermediária, onde é realizado o processamento; e a camada de saída, que apresenta o resultado final. Na literatura é possível encontrar diferentes propostas de algoritmos voltados para rede neural artificial como o *backpropagation* (Han e Kamber, 2006).

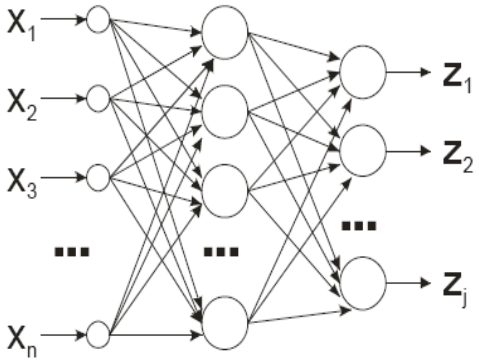


Figura 4.6. Rede Neural Artificial Perceptron Multicamadas

4.3.2.4. Máquina de Vetores de Suporte

Máquina de Vetores de Suporte (SVM, do inglês, *Support Vector Machine*) é uma técnica de classificação que é embasada na Teoria da Aprendizagem Estatística (TAE), desenvolvida por Vapnik (1995). Esta técnica tem recebido cada vez mais atenção por parte de pesquisadores por apresentar resultados promissores em muitas aplicações como reconhecimento de padrão e classificação de textos (Tan et al., 2009). Uma Máquina de Vetores de Suporte tem como base o emprego de condições matemáticas oriundas da TAE que permitem classificar um conjunto de dados em duas classes diferentes. A fronteira entre as duas classes de dados é denominada de limite de decisão ou fronteira de decisão, que pode ser linear ou não linear, e determina o tipo de Máquina de Vetores de Suporte.

Uma classificação de dados por meio de uma Máquina de Vetores de Suporte do tipo linear consiste na aprendizagem de uma função ou classificador baseado em um hiperplano ou uma representação espacial para dividir linearmente um conjunto de dados em duas classes conforme ilustrado na Figura 4.7 (a). O classificador é obtido por meio de um conjunto de treinamento durante a etapa de aprendizado. Na etapa de predição, dado um conjunto de dados de testes em que os dados podem assumir os valores (+,-) o classificador separa estes dados de modo que um dos lados do hiperplano contenha somente dados da classe (+), e outro lado dados da classe (-).

A classificação de um conjunto de dados com dados não separáveis linearmente, como ilustra a Figura 4.7 (b) deve ser realizada por meio de uma Máquina de Vetores de Suporte do tipo não linear. Para isso, a Máquina de Vetores de Suporte é construída a partir da transformação do conjunto de dados inicial em um novo conjunto de dados linearmente separáveis por meio de uma série de princípios do teorema de Cover da separabilidade de padrões (Han e Kamber, 2006). Com relação aos algoritmos de Máquina de Vetores de Suporte, o algoritmo de Otimização Mínima Sequencial (SMO, do inglês, *Sequential Minimal Optimization*) é um dos que estão entre os mais tradicionais (Platt, 1998).

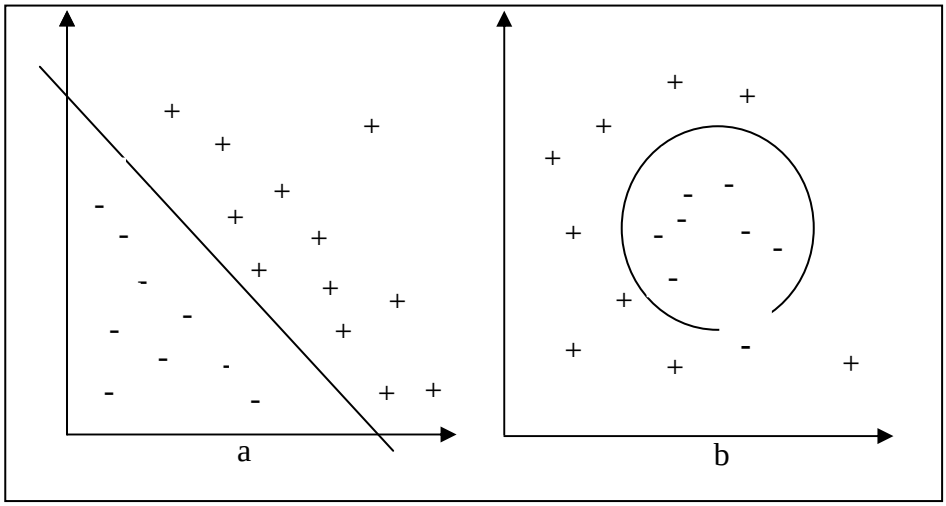


Figura 4.7. (a) Classificação de dados com limite de decisão linear; (b) Classificação de dados com limite de decisão não linear

4.4. Abordagem Geral para a Análise de Informações Contextuais

A abordagem de análise apresentada nesta seção, tem como base o emprego da tarefa de classificação por meio de etapa de aprendizagem supervisionada para criação de *modelos de conhecimento*, posteriormente utilizados na etapa de predição de rótulos de classes. Desta forma, são utilizadas as técnicas de classificação da área de aprendizagem de máquina, e o conceito de *registro contextual*, que será apresentado na subseção a seguir.

4.4.1. Registro Contextual

Um *registro contextual* é definido, neste trabalho, como sendo o registro a ser utilizado nas tarefas de aprendizagem supervisionada e predição, sendo formado por uma combinação de atributos preditivos e de classe, ou seja, o atributo alvo com ou sem o rótulo de classes pré-definidas que correspondem as dimensões semânticas 5Ws+1H.

Cada classe pré-definida pode se referir a um determinado tipo de conteúdo, uma atividade do usuário, um tipo de necessidade, um nível de satisfação de um serviço, uma avaliação de um produto etc. Por exemplo, em um sistema de recomendação de conteúdo para TV Digital, uma classe pode corresponder a um gênero de programa de TV que será recomendado para o usuário. Um rótulo de classe pré-definida, pode ser atribuído ao atributo-classe pelo usuário ou inferida para o mesmo dependendo da etapa de classificação que esteja sendo realizada com o *registro contextual*. Assim, um *registro contextual* pode conter ou não um rótulo de classe no atributo-classe.

No que se refere à obtenção do *registro contextual* pode ser empregada a técnica realimentação de relevância (do ingles, *feedback relevance*) que tem como base a interação explícita e implícita do usuário com o sistema computacional (Blanco-Fernandez, 2007). Toda vez que o usuário interage com o sistema, como assistir ou avaliar um programa de TV, acessar informações sobre um produto, ou solicitar recomendações de conteúdo, um *registro contextual* poderá ser gerado.

A Tabela 4.4 apresenta alguns atributos preditivos e de classe que podem ser combinados para compor um *registro contextual*, destacando a dimensão semântica contextual correspondente e possíveis valores dos mesmos. Torna-se importante destacar que de acordo com os requisitos do sistema computacional, tais atributos podem ser complementados ou até mesmo substituídos por outros de maior expressividade. Por questões de privacidade, não aparece um atributo preditivo de identificação do usuário (ID do usuário) referente à dimensão *Who* no *registro contextual*, pois esta informação é opcional no processo de predição.

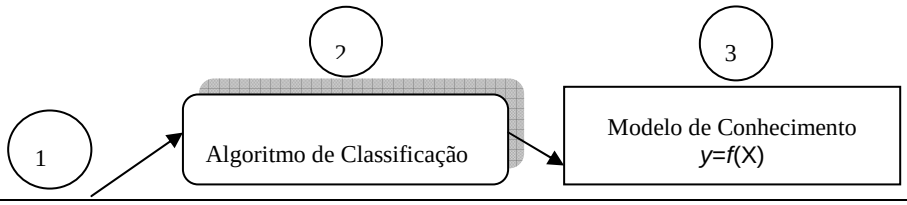
Tabela 4.4. Exemplos de atributos preditivos e atributos-classe que podem ser combinados para compor um registro contextual.

Dimensão Semântica	Atributos	Valores candidatos
Where	**Preditivos**	Casa, escola, shopping, escritório, ônibus, restaurante
	Lugar	
	Zona	Norte, sul, leste, oeste, centro
	Estado	AM, SP, RJ, RR, MG, RS
When	Dia	Segunda, terça, quarta, quinta, sexta, sábado, domingo
	Período	Manhã, tarde, noite, madrugada
	Tipo do dia	Fim de semana, dia de semana, feriado
How	Dispositivo	Tablet, smartphone, TV, set-top box, notebook, PC
	Tipo de dispositivo	Fixo, móvel, portátil
	Aplicação	Web, móvel, desktop
What	**Atributos-classe**	Policial, musical, ação, romântico, comédia, ficção
	Gênero	
	Necessidade	Hedônica, utilitária
	Conteúdo	Esporte, notícias, entretenimento, musica, educação, compras

4.4.2. Aprendizagem Supervisionada baseada em Contexto

Esta é a etapa onde o algoritmo de classificação constrói o *modelo de conhecimento* por meio da análise do histórico de contexto composto por *registros contextuais* dos usuários com o atributo-classe instanciado, ou seja *registros contextuais* pré-classificados. No processo da classificação os *registros contextuais* dos usuários podem ser referenciados como tuplas, instâncias, exemplos, ou casos de testes. Uma tupla, *X*, pode ser representada por meio de vetor de atributos n-dimensional, $X=(x_1, x_2, x_3,..., x_n)$. A Figura 4.7 ilustra o processo de aprendizagem supervisionada baseada em contexto.

O histórico de contexto (1) é obtido por meio da técnica de realimentação de relevância, e posteriormente analisado pelo algoritmo de classificação (2) para criação do *modelo de conhecimento* também conhecido como modelo de classificação (Tan et al., 2009). Vale ressaltar que o *modelo de conhecimento* (3) é construído de acordo com a técnica de classificação empregada, e que pode ser compreendido como uma função, $y=f(X)$, que permite predizer o rótulo, *y*, de classe do atributo-classe de um dado *registro, X, contextual* de um usuário.



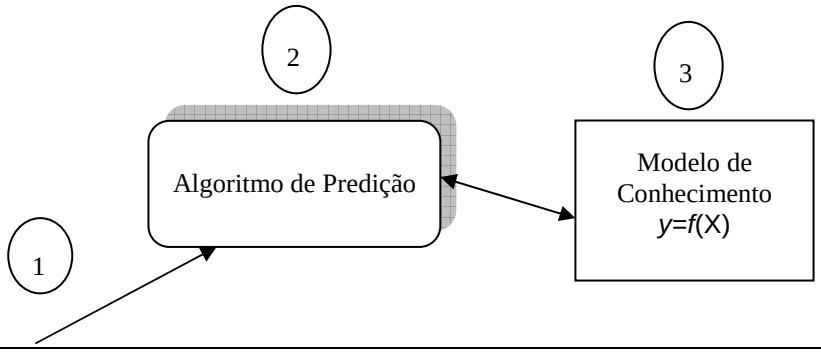
Registros Contextuais (Histórico de Contexto)				
Atributos Preditivos				**Atributo-classe**
Lugar (*Where*)	Dia (*When*)	Período (*When*)	Dispositivo (*How*)	Conteúdo (*What*)
Casa	Domingo	Noite	TV	Entretenimento
Escritório	Segunda	Manhã	Notebook	Economia
Casa	Sábado	Tarde	TV	Esporte
Carro	Sábado	Noite	Navegador GPS	Música
Shopping	Domingo	Tarde	Smartphone	Compras

Figura 4.7. Etapa de Aprendizagem Supervisionada. Adaptado de (Han e Kamber, 2006)

4.4.3. Predição de Classes Contextuais

Conforme ilustrado na Figura 4.8 esta é a segunda etapa da abordagem, onde dado um *registro, X, contextual* com um atributo-classe, não instanciado (1), ou seja, um registro contextual sem o rótulo, *y*, de classe, deseja-se predizer um rótulo, *y*, de classe para o atributo-classe deste registro. Vale lembrar que um atributo-classe pode ser um tipo de conteúdo, um gênero, uma necessidade, dentre outros. Para que a predição ocorra, é necessário que o algoritmo de predição (2) utilize o *modelo de conhecimento y=f*(X) (3) obtido na etapa anterior, que permite predizer o rótulo, *y*, de classe do atributo-classe de um dado *registro, X, contextual de teste.*

O *registro, X, contextual de teste* é obtido automaticamente partir de uma ação explícita ou implícita do usuário. Por exemplo, ao solicitar uma recomendação de conteúdo, ou ao acessar uma lista de produtos. Assim, a partir do rótulo, *y*, de classe que fora obtido se torna possível a sua utilização posterior na fase de disseminação de contexto como parâmetro de entrada para serviços como personalização e adaptação de conteúdo, filtragem de informação, entre outros.



Registro Contextual de Teste				
Atributos Preditivos				**Atributo-classe**
Lugar (*Where*)	Dia (*When*)	Período (*When*)	Dispositivo (*How*)	Conteúdo (*What*)
Casa	Domingo	Noite	TV	Classe y?

Figura 4.8. Etapa de Predição. Adaptado de (Han e Kamber, 2006)

4.5. Implementação da Abordagem para Análise de Informações Contextuais

Nesta seção propõe-se apresentar como implementar a abordagem de análise de informações contextuais sobre um histórico de contexto ou base de dados. O objetivo é criar um *modelo de conhecimento* que classifique o conteúdo de acordo com o contexto do usuário. Por meio do rótulo de classe inferido será possível posteriormente obter uma lista de conteúdos correspondentes a partir de técnicas de filtragem de informação como a técnica de filtragem baseada em conteúdo (Adomavicius e Tuzhilin, 2005), implementada em sistemas de recomendação.

Para a implementação das técnicas de classificação (Árvore de Decisão, classificador Bayesiano, Rede Neural Artificial e Máquina de Vetores de Suporte) exploradas pela abordagem apresentada pode ser empregado o modo programado da ferramenta Weka desenvolvida na Universidade de Waikato na Nova Zelândia (Hall et al., 2009). Desta forma, deve-se utilizar a API Weka que contempla uma série de técnicas de aprendizagem de máquina implementadas em Java.

4.5.1. O Ambiente de Desenvolvimento

Para programar aplicações baseadas em Weka, deve-se ter instalado a plataforma de programação Java (J2SE, *Java 2 Standard Edition*). Assim, é necessário instalar o *J2SE Development Kit* (J2SE, 2012), e um ambiente de desenvolvimento integrado (do inglês, IDE *Integrated Development Environment*) como o Eclipse (Eclipse, 2012) usado na programação das aplicações.

Além disso, para que o Weka seja utilizado nas aplicações, deve-se adicionar a API Weka no projeto da aplicação, por meio da *tab libraries* da caixa de dialogo *Java Settings*. Deve ser usado o botão *Add External JARs* para adicionar o arquivo *weka.jar*. Todos os exemplos apresentados neste trabalho foram implementados por meio de tais ferramentas.

4.5.2. Criação da Base de Dados para Tarefa de Classificação

A Weka utiliza um arquivo no formato ARFF (*Attribute-Relation File Format*) que contém um conjunto de registros que são utilizados na tarefa de classificação pelas técnicas de aprendizagem de máquina implementadas na API Weka. No caso desta abordagem o conjunto de registros ou base de dados corresponde ao histórico de contexto (Figura 4.8) composto por *registros contextuais* com atributos preditivos e atributos-classe instanciados, ou seja, com rótulos de classes.

Tais registros são tratados pela API do Weka como instâncias. Na Figura 4.9 é apresentado um trecho de um arquivo ARFF. Vale ressaltar que API Weka também fornece recursos que permitem que as instâncias sejam obtidas a partir de um sistema de gerenciamento de banco de dados. Para isso, algumas configurações devem ser feitas no arquivo DatabaseUtils.props para definir a conexão com o banco de dados. Informações detalhadas sobre o acesso por meio de banco de dados podem ser encontradas em (Weka, 2012).

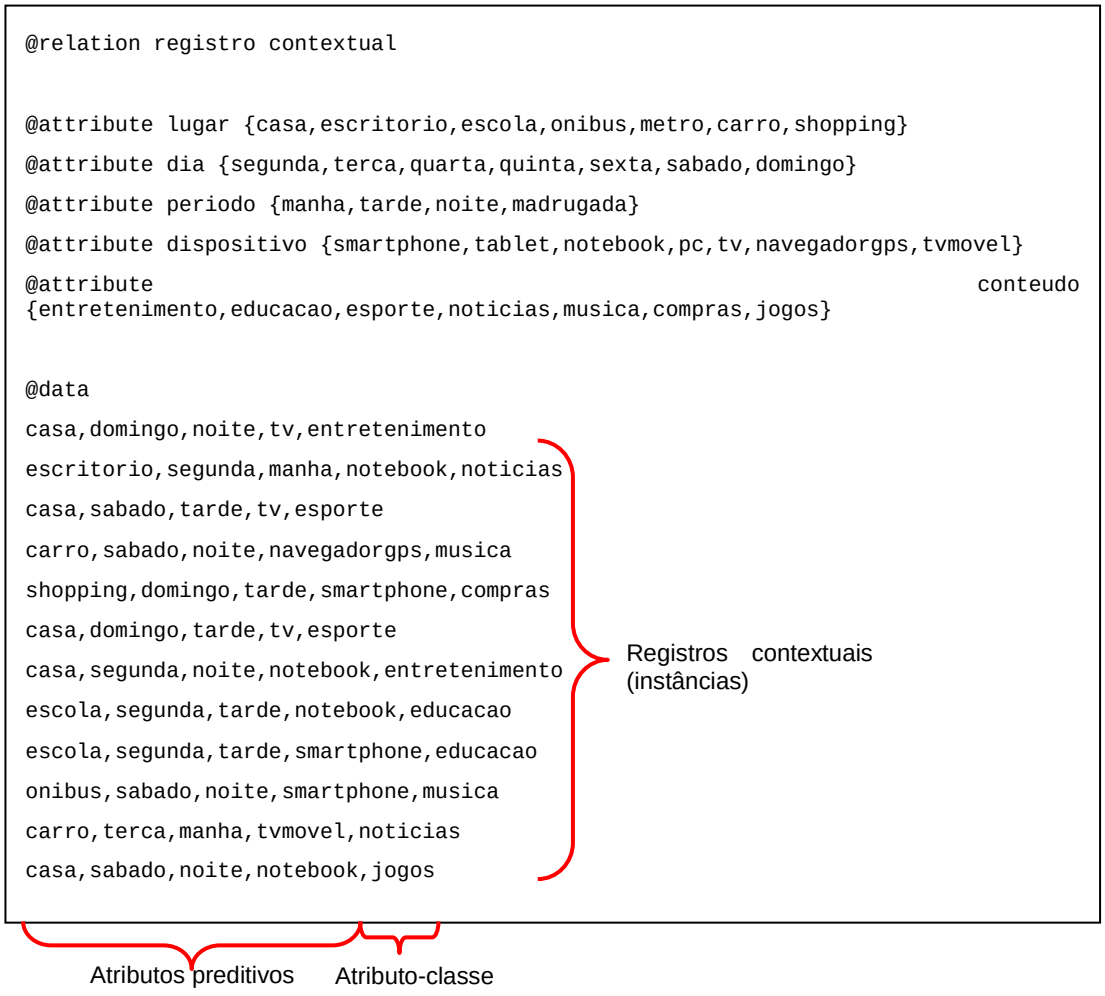


Figura 4.9. Arquivo ARFF com alguns registros contextuais

O arquivo no formato ARFF é um arquivo de texto dividido em três partes:

- Relação (*@relation*), se refere ao termo utilizado para identificar a relação de registros;
- Atributos (*@Attribute*), correspondem aos atributos preditivos que compõe um registro. Para cada atributo é definido seu nome e os seus possíveis valores. Geralmente, o último atributo é o atributo-classe com seus rótulos de classes pré-definidas;
- Dados *(@data)*, cada linha corresponde a uma instância, ou seja um *registro contextual*, com os valores separados por virgula e seguindo a ordem dos atributos.

A seguir serão descritas as implementações das principais etapas da abordagem de análise de informações contextuais por meio de exemplos práticos das técnicas de classificação: Árvore de Decisão, classificador Bayesiano e Rede Neural e SVM.

4.5.3. Obter Instâncias de Registros Contextuais

Esta tarefa consiste em recuperar e converter os registros do arquivo ARFF para um formato mais apropriado visando uso posterior nas etapas de aprendizagem supervisionada e predição. Com isso, os registros do arquivo ARFF são carregados e convertidos como instâncias (ou vetores de atributos), e o atributo-classe também é definido. O trecho de código a seguir apresentado na Figura 4.10 ilustra a implementação de Java que utiliza classes da API Weka.

```
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

(...)

public Instances getInstanciasRegistrosContextuais(){
                Instances instancias = null;
                try
                {
                        DataSource  fonteDeDados = new DataSource(ARFF);
                        instancias = fonteDeDados.getDataSet();

instancias.setClassIndex(instancias.numAttributes() - 1);
                        System.out.println(instancias);
                }
                catch(Exception e)
                {
                        System.out.println("O arquivo arff "+ARFF+" nao
existe...");
                        e.printStackTrace();
                }
```

Figura 4.10. Exemplo de código para obter instâncias de registros do ARFF

A API Weka fornece, por meio do pacote *weka.core.converters*, um conjunto de classes para pré-processamento de registros do arquivo ARFF. No exemplo, *fonteDeDados* é uma variável de instância da classe *weka.core.converters.ConverterUtils.DataSource*. Por meio de *fonteDeDados* é possível chamar o método *getDataSet()* para obter do arquivo ARFF um conjunto de registros tratados como instâncias da classe *weka.core.Instances*. A localização do arquivo ARFF

é passado como argumento para o contrutor *DataSource()*. Por fim, é definido o atributo-classe das instâncias por meio do método *setClassIdex()* que recebe como argumento o índice que identifica qual será o atributo-classe usado na aprendizagem e predição. Por padrão no arquivo ARFF o atributo-clase é o ultimo atributo, o que explica o motivo para o índice está definido como *numAttributes-1*.

4.5.4. Criação da Instância do Registro Contextual de Teste

A API Weka trata o registro de teste como um vetor de atributos preditivos sem o atributo-classe instanciado, ou seja uma instância sem o rótulo, *y*, de classe. Assim, torna-se necessária a criação de uma instância correspondente ao *registro contextual de teste* que será utilizado posteriormente na etapa de predição. Para isso, deve-se utilizar o pacote *weka.core*, pois fornece a classe *weka.core.Instance* responsável pela codificação de uma instância. O trecho de código a seguir apresentado na Figura 4.11 ilustra a criação de uma instância a partir de atributos preditivos e atributo-classe de um *registro contextual* de teste.

No exemplo, a classe *weka.core.Instance* é instanciada por meio do construtor *Instance()* que recebe como argumento a quantidade de atributos que a instância deve ter, incluindo o atributo-classe da classe. No exemplo, um objeto do tipo *weka.core.Instance* é referenciado pela variável *instancia*. Por meio de *instancia* é possível chamar o método *setDataSet()* para definir o conjunto de instâncias (obtido na etapa de pré-processamento.) que a *instancia* pertence. Em seguida, por meio do método *setValue()* é possível definir cada atributo preditivo que pertence ao registro de teste. Com isso, uma instância será criada posteriormente utilizada na tarefa de predição conforme será descrito a seguir.

```
import weka.core.Instance;

(...)

public Instance getInstanciaDeTeste(String lugar, String dia, String
periodo, String dispositivo,
                    Instances instancias){
                Instance instancia = new Instance(5);
                instancia.setDataset(instancias);
                instancia.setValue(0, lugar);
                instancia.setValue(1, dia);
                instancia.setValue(2, periodo);
                instancia.setValue(3, dispositivo);
                System.out.println("A instancia de TESTE e -->
"+instancia);
            return instancia;
       }
```

Figura 4.11. Criação da Instância de um Registro Contextual de Teste

4.5.5. Aprendizagem Supervisionada baseada em Árvore de Decisão

O trecho de código a seguir apresentado na Figura 4.12 ilustra a implementação da etapa de aprendizagem por meio da técnica Árvore de Decisão. Para isso, deve-se utilizar o pacote *weka.classifiers.trees* que fornece classes que implementam diversos algoritmos de Árvore de Decisão.

```
import weka.classifiers.trees.J48;

(...)

public void arvoreDeDecisao_Aprendizagem(Instances
instancias){
                              try
                {
                        j48 = new J48();

j48.buildClassifier(instancias);
                }
                catch(Exception e)
                {
                         e.printStackTrace();
                }
        }
```

Figura 4.12. Implementação da Árvore de Decisão

No exemplo, é utilizado o algoritmo C4.5 (Witten e Franck, 2005; Tan et al., 2009) por meio da implementação J48 disponível na API Weka. Assim, *j48* é uma instância da classe *weka.classifiers.trees.j48*. Por meio do objeto *j48* é possível chamar o método *buildClassifier()* para construir um *modelo de conhecimento* (baseado em Árvore de Decisão) usado na classificação a partir do conjunto de instâncias recebidas como argumento.

4.5.6. Predição baseada em Árvore de Decisão

Para implementação da tarefa de predição baseada em Árvore de Decisão deve-se utilizar além do pacote *weka.classifiers.trees*, também o pacote *weka.core* , pois fornece a classe *weka.core.Instance* responsável pela codificação de uma instância (*registro contextual*), ou seja, um vetor de atributos, e a classe *weka.core.Attribute* responsável pela codificação de um atributo. O trecho de código a seguir apresentado na Figura 4.13 ilustra a implementação da tarefa de predição por meio da técnica Árvore de Decisão.

```
import weka.core.Instance;
import weka.core.Attribute;
import weka.classifiers.trees.J48;

(...)

public String arvoreDeDecisao_Predicao(Instance instancia){

              String classeInferida = null;
              try {
                    int indice =
Double.valueOf(j48.classifyInstance(instancia)).intValue();
                    Attribute atributoClasse = instancia.classAttribute();
                    classeInferida = atributoClasse.value(indice);

              } catch (Exception e) {
                    e.printStackTrace();
              }
              return classeInferida;
       }
```

Figura 4.13. Implementação da Predição baseada em Árvore de Decisão

No exemplo, por meio do objeto *j48* é possível chamar o método *classifyInstace()* usado para classificar uma instância com rótulo, *y*, de classe desconhecido, ou seja, obter uma classe para um *registro contextual de teste*. Para isso, o método *classifyInstace()* recebe como argumento a instância que deve ser classificada, e retorna um índice que posteriormente será utilizado para obter o rótulo, *y*, de classe inferido. O objeto *atributoClasse*, cujo o tipo é *weka.core.Attribute* se refere ao atributo-classe obtido via o método *classAttribute()* da instância de teste. Por meio do método *value()* do objeto *atributoClasse* é possível obter o rótulo, *y*, de classe inferido.

4.5.7. Aprendizagem Supervisionada baseada em um Classificador Bayesiano

O trecho de código a seguir apresentado na Figura 4.14 ilustra a implementação da tarefa de aprendizagem por meio de um classificador Bayesiano. Para isso, deve-se utilizar o pacote *weka.classifiers.bayes* que fornece classes que implementam diversos algoritmos de classificadores Bayesianos.

No exemplo, é utilizada uma implementação do algoritmo Naive Bayes (Han e Kamber, 2006;Tan et al., 2009). Assim, *naiveBayes* é uma instância da classe *weka.classifiers.bayes.NaiveBayes*. Por meio de *naiveBayes* é possível chamar o método *buildClassifier()* para construir um *modelo de conhecimento* (baseado no teorema de Bayes) usado na classificação a partir do conjunto de instâncias recebidas como argumento.

```
import weka.classifiers.bayes.NaiveBayes;

(...)

public void classificadorBayesiano_Aprendizagem(Instances
instancias){
                                try
              {
                      naiveByes = new NaiveBayes();
                      naiveByes.buildClassifier(instancias);
              }
              catch(Exception e)
              {
                     e.printStackTrace();
              }
       }
```

Figura 4.14. Implementação do Classificador Bayesiano

4.5.8. Predição baseada em um Classificador Bayesiano

Para implementação da tarefa de predição baseada em Naive Bayes deve-se utilizar além do pacote *weka.classifiers.bayes*, também o pacote *weka.core*, pois fornece a classe *weka.core.Instance* responsável pela codificação de uma instância (*registro contextual*), ou seja, um vetor de atributos, e a classe *weka.core.Attribute* responsável pela codificação de um atributo. O trecho de código a seguir apresentado na Figura 4.15 ilustra a implementação da tarefa de predição por meio do algoritmo Naive Bayes.

```
import weka.core.Instance;
import weka.core.Attribute;
import weka.classifiers.bayes.NaiveBayes;

(...)

public String classificadorBayesiano_Predicao(Instance instancia){

String classeInferida = null;
      try {
      int indice =
Double.valueOf(naiveBayes.classifyInstance(instancia)).intValue();
      Attribute atributoClasse = instancia.classAttribute();
      classeInferida = atributoClasse.value(indice);
      } catch (Exception e) {
                  e.printStackTrace();
      }
                        return classeInferida;
      }
```

Figura 4.15. Implementação da Predição baseada em Naive Bayes

No exemplo, por meio do objeto *naiveBayes* é possível chamar o método *classifyInstace()*usado para classificar uma instância com rótulo de classe desconhecido, ou seja, obter um rótulo, *y*, de classe para um *registro contextual de teste*. Para isso, o método *classifyInstace()* recebe como argumento a instância que deve ser classificada, e retorna um índice que posteriormente será utilizado para obter o rótulo, *y*, de classe inferido. O objeto *atributoClasse*, cujo o tipo é *weka.core.Attribute* se refere ao atributo-classe obtido via o método *classAttribute()* da instância de teste. Por meio do método *value()* do objeto *atributoClasse* é possível obter o rótulo, *y*, de classe inferido.

4.5.9. Aprendizagem Supervisionada baseada em Rede Neural Artificial

O trecho de código a seguir apresentado na Figura 4.16 ilustra a implementação da tarefa de aprendizagem por meio de uma Rede Neural Artificial. Para isso, deve-se utilizar o pacote *weka.classifiers.functions* que fornece uma miscelânea de classes que implementam diversas técnicas de aprendizagem, como Multlayer Perceptron (Han e Kamber, 2006;Tan et al., 2009)

```
import weka.classifiers.functions.MultilayerPerceptron;

(...)

public void redeNeural_Aprendizagem(Instances instancias){

   try {
                     multilayerPerceptron = new
MultilayerPerceptron();

multilayerPerceptron.buildClassifier(instancias);
                }
                catch(Exception e)
                {
                           e.printStackTrace();
                }
```

Figura 4.16. Implementação da Rede Neural Artificial

No exemplo, é utilizada a implementação do algoritmo Multlayer Perceptron da API Weka. Assim, *multlayer Perceptron* é uma instância da classe *weka.classifiers.functions.MultlayerPerceptron*. Por meio de *multlayerPerceptron* é possível chamar o método *buildClassifier()* para construir um *modelo de conhecimento* (baseado em Rede Neural Artificial Multlayer Perceptron) usado na classificação a partir do conjunto de instâncias recebidas como argumento.

4.5.10. Predição baseada em Rede Neural Artificial

Para implementação da tarefa de predição baseada em rede neural artificial deve-se utilizar além do pacote *weka.classifiers.functions*, também o pacote *weka.core*, pois fornece a classe *weka.core.Instance* responsável pela codificação de uma instância (*registro contextual*), ou seja, um vetor de atributos, e a classe *weka.core.Attribute* responsável pela codificação de um atributo. O trecho de código a seguir apresentado na Figura 4.17 ilustra a implementação da tarefa de predição por meio da técnica rede neural artificial.

No exemplo, por meio do objeto *multilayerPerceptron* é possível chamar o método *classifyInstace()*usado para classificar uma instância com rótulo de classe desconhecido, ou seja, obter um rótulo, *y*, de classe para um *registro contextual de teste*. Para isso, o método *classifyInstace()* recebe como argumento a instância que deve ser classificada, e retorna um índice que posteriormente será utilizado para obter o rótulo, *y*, de classe inferido. O objeto *atributoClasse*, cujo o tipo é *weka.core.Attribute* se refere ao atributo-classe obtido via o método *classAttribute()* da instância de teste. Por meio do método *value()* do objeto *atributoClasse* é possível obter o rótulo, *y*, de classe inferido.

```
import weka.core.Instance;
import weka.core.Attribute;
import weka.classifiers.functions.MultilayerPerceptron;

(...)

public String redeNeural_Predicao(Instance instancia){

		String classeInferida = null;
		try {
			int indice =
Double.valueOf(multilayerPerceptron.classifyInstance(instancia)).intValue();
			Attribute atributoClasse = instancia.classAttribute();
			classeInferida = atributoClasse.value(indice);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return classeInferida;
	}
```

Figura 4.17. Implementação da Predição baseada em Rede Neural Artificial

4.5.11. Aprendizagem Supervisionada baseada em Máquina de Vetores de Suporte

O trecho de código a seguir apresentado na Figura 4.18 ilustra a implementação da tarefa de aprendizagem por meio de uma Máquina de Vetores de Suporte. Para isso, deve-se utilizar o pacote *weka.classifiers.functions* que fornece uma miscelânea de

classes que implementam diversos algoritmos de aprendizagem, como o algoritmo de Otimização Mínima Sequencial (SMO) (Platt, 1998).

```
import weka.classifiers.functions.SMO;

(...)

public void svm_Aprendizagem(Instances instancias) {

            try {
                  smo = new SMO();
                  smo.buildClassifier(instancias);
            } catch (Exception e) {
                                    e.printStackTrace();
            }

}
```

Figura 4.18. Implementação de uma Máquina de Vetores de Suporte

No exemplo, é utilizada a implementação do algoritmo SMO da API Weka. Assim, *smo* é uma instância da classe *weka.classifiers.functions.SMO*. Por meio de *smo* é possível chamar o método *buildClassifier()* para construir um *modelo de conhecimento* (baseado em Máquina de Vetores de Suporte) usado na classificação a partir do conjunto de instâncias recebidas como argumento.

4.5.12. Predição baseada em Máquina de Vetores de Suporte

Para implementação da tarefa de predição baseada em Máquina de Vetores de Suporte deve-se utilizar além do pacote *weka.classifiers.functions*, também o pacote *weka.core*, pois fornece a classe *weka.core.Instance* responsável pela codificação de uma instância (*registro contextual*), e a classe *weka.core.Attribute* responsável pela codificação de um atributo. O trecho de código a seguir apresentado na Figura 4.19 ilustra a implementação da tarefa de predição por meio do algoritmo SMO da API Weka.

No exemplo, por meio do objeto *smo* é possível chamar o método *classifyInstace()*usado para classificar uma instância com rótulo de classe desconhecido, ou seja, obter um rótulo, *y*, de classe para um *registro contextual de teste*. Para isso, o método *classifyInstace()* recebe como argumento a instância que deve ser classificada, e retorna um índice que posteriormente será utilizado para obter o rótulo, *y*, de classe inferido. O objeto *atributoClasse*, cujo o tipo é *weka.core.Attribute* se refere ao atributo-classe obtido via o método *classAttribute()* da instância de teste. Por meio do método *value()* do objeto *atributoClasse* é possível obter o rótulo, *y*, de classe inferido.

```
import weka.core.Instance;
import weka.core.Attribute;
import weka.classifiers.functions.SMO;

(...)

public String svm_Predicao(Instance instancia){

            String classeInferida = null;
            try {
                  int indice =
Double.valueOf(smo.classifyInstance(instancia)).intValue();
                  Attribute atributoClasse =
instancia.classAttribute();
                  classeInferida = atributoClasse.value(indice);
            } catch (Exception e) {
                  e.printStackTrace();
            }
            return classeInferida;
      }
```

Figura 4.19. Implementação da Predição baseada em Máquina de Vetores de Suporte

4.5.13. Disseminação do Rótulo de Classe Inferido

A disseminação pode ser implementada de acordo com o tipo do serviço oferecido pelo sistema sensível ao contexto. Deste modo, o rótulo de classe inferido pode ser empregado, por exemplo, como parâmetro para serviços de adaptação e personalização de conteúdo, para modificar o comportamento de um sistema ou prover recomendação de conteúdo por meio de técnicas de adaptação ou filtragem de informação da área de Recuperação de Informação como a técnica de filtragem baseada em conteúdo (Adomavicius e Tuzhilin, 2005), implementada em sistemas de recomendação.

Em cenários como Web, TV Digital Interativa, e Computação Ubíqua, a exploração do contexto pode enriquecer a experiência de interação do usuário, principalmente quando empregado em aplicações interativas para prover conteúdos e/ou informações de forma personalizada (Silva, 2012; Alves, 2009, Golularte, 2003). A Figura 4.20 apresenta a implementação do método *main* da classe *AnalisadorDeContexto* responsável gerenciamento das etapas de aprendizagem e predição.

```
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.trees.J48;
import weka.classifiers.functions.MultilayerPerceptron;
import weka.core.Attribute;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

public class AnalisadorDeContexto {

(...)

public static void main(String[] args) {

AnalisadorDeContexto analisadorDeContexto = new
AnalisadorDeContexto();

            Instances instancias =
analisadorDeContexto.getInstanciasRegistrosContextuais();


      analisadorDeContexto.setAprendizagemSupervisionada(NAIVEBAYE
S, instancias);
            Instance instancia =
analisadorDeContexto.getInstanciaDeTeste("casa","domingo","noite",
"tv", instancias);
            System.out.println("Classe inferida --->
"+analisadorDeContexto.getPredicao(NAIVEBAYES,instancia));
                        }
}
```

Figura 4.20. Implementação do método main da classe AnalisadorDeContexto

No exemplo, o método *main* executa sistematicamente os métodos responsáveis pela implementação das etapas de aprendizagem e predição. O método *getInstanciasRegistrosContextuais()* recupera e trata como instâncias os registros armazenados no arquivo ARFF. Em seguida, o método *setAprendizagemSupervisionada()* gera o modelo de conhecimento a partir das instâncias e da técnica de aprendizagem passadas como parâmetros.

Finalmente, a etapa de predição é executada por meio do método *getPredicao()* que retorna o rótulo de classe inferido. Para isso, o método *getPredicao()* recebe como parâmetros a instância correspondente ao *registro contextual de teste* e a técnica que será utilizada na tarefa de predição. A instância do *registro contextual de teste* é obtida por meio do método *getInstanciaDeTeste ()* que recebe como parâmetros as informações contextuais correspondentes aos atributos preditivos. Por fim, o analisador de contexto de posse do rótulo de classe obtido pode passá-lo como parâmetro para um método que implementa, por exemplo, uma filtragem de conteúdo.

4.6. Considerações Finais

A Computação Sensível ao Contexto é uma área de pesquisa que vem despertando cada vez mais a atenção e interesse de pesquisadores. O objetivo desta área é desenvolver sistemas computacionais que sejam mais adaptáveis, flexíveis, e fáceis de usar de modo

a diminuir a quantidade de intervenções por parte dos usuários na solicitação de serviços ou na buscar de informação.

Para isso, tais sistemas devem ser capazes de automaticamente identificar o usuário e o seu contexto corrente para oferecer informações personalizadas ou serviços adaptados às necessidades dos usuários. Pesquisadores estudam diversas técnicas, abordagens e métodos para implementar os principais requisitos de software de sistemas sensíveis ao contexto. No entanto, ainda não há um consenso de quais são as melhores ferramentas para suporte ao desenvolvimento deste tipo de sistema.

Desta forma, este capítulo apresentou uma visão geral sobre as áreas Computação Sensível ao Contexto e Aprendizagem de Máquina. Foi destacado que o desenvolvimento de sistemas sensíveis ao contexto exige a superação de vários desafios técnicos como aquisição automática, representação, privacidade e segurança, e especialmente a análise ou processamento de informações contextuais.

Neste sentido, o trabalho discutiu a proposta de uma abordagem para a tarefa de análise de informações contextuais através da tarefa de classificação implementada por meio de técnicas de classificação que utilizam aprendizagem máquina. Com isso, foram apresentados os conceitos introdutórios sobre as principais técnicas de classificação experimentadas na abordagem proposta neste trabalho.

Para o desenvolvedor foram apresentados exemplos práticos de como empregar as técnicas de classificação para aprendizagem de máquina e predição de modo programático através do uso da API Weka. Torna-se importante ressaltar que o assunto discutido neste trabalho não está esgotado, e que outras técnicas para aprendizagem de máquina como Raciocínio Baseado em Casos, ou ainda Classificador de Vizinho mais Próximo (Tan et al., 2009) podem ser experimentadas por meio da abordagem proposta para análise de informações contextuais.

O principal objetivo deste texto foi apresentar uma visão geral sobre Computação Sensível ao Contexto e os conceitos introdutórios sobre Aprendizagem de Maquina que serviram de base para concepção da abordagem de análise de informações contextuais discutida. Para o desenvolvedor de sistemas sensíveis ao contexto este texto pode auxiliar na aprendizagem de novas técnicas que viabilizem o desenvolvimento de sistemas cada vez mais atrativos. Na Computação Sensível ao Contexto, muitos assuntos estão em aberto. Com isso, surge à demanda por realização de novos estudos, e oportunidades de pesquisas para alunos e professores.

4.7. Referências

Abowd, G. D.; Mynatt, E. D. (2000) “Charting past, present, and future research. In: Ubiquitous Computing”, ACM Transactions on Computer-Human Interaction (TOCHI), v. 7, n.1, p. 29-58.

Alves, L. G. P.; Silva, F. S.; Bressan, G. (2009) “CollaboraTVware: A Context-Aware Insfrastructure with Support for Collaborative Participation in Interactive Digital Environment”, In: International Journal of Advanced Media and Communication (IJAMC), v. 3, n.4.

Adomavicius, G.; Tuzhilin, A. (2005) “Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions”, In: IEEE Transactions on Knowledge and Data Engineering, v.17, n.6, p.734-749.

Blanco-Fernandez, Y. (2007) “Propuesta Metodológica para el Razonamento Semántico en Sistemas de Recomendación Personalizada y Automática. Aplicación Al Caso de Contenidos Audiovisuales”, Tese (Doutorado) - Departamento de Enxeneria Telemática E.T.S.E. de Telecomunicación, Universidade de Vigo.

Chen, G.; Kotz, D. (2002) “ SOLAR: A pervasive-Computing Infrastructure for Context-Aware Mobile Applications”. Technical Report TR2002-421, Dartmouth College.

Chen, G.; Kotz, D. (2004) “Na Intelligent Broker Architecture for Pervasive Context-Aware Systems”, Ph. D. Thesis, University of Maryland, USA.

Dey, A.K. and Abowd, G.D. (2000) “Towards a better understanding of context and context-awareness”, In the Workshop on the What, Who, Where, When and How of Context-Awareness, affiliated with the 2000 ACM Conference on Human Factors in Computer Systems (CHI 2000).

Dey, A. K., Abowd, G. D. (2001) “A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications”, In: Human-Computer Interaction, v. 16, n. 2-4, p.97-166.

Dey, A. K. (2001) “ Understanding an Using Context”, In: ACM Personal and Ubiquitous
Computing Journal, v.5, n.1, p.4-7.

Dockhorn Costa, P. D. (2003) “Towards a Services Plataform for Context-Aware Applications”, Master Thesis, University of Twente, The Netherlands.

Eclipse (2012). The Eclipse Foundation open source community. Disponível em: <http://www.eclipse.org/downloads/>. Acesso em: 19 agos 2012

Forstadius, J.; Lassila, O.; Seppanen, T. (2005) “RDF-Based Model for Context-Aware Reasonig in Rich Service Environment.” In: Proceedings of the 3rd International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), p.15-19, 2005.

Goularte, R. (2003) “Personalização e adaptação de conteúdo baseadas em contexto para TV Interativa**”,** Tese (Doutorado) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-23092004-153330/pt-br.php>

Gu, T.; Pung, H. K.; Zhang, D. Q. (2005) “A Service-Oriented Middleware for Building Context-Aware Services” In: Journal of Network and Computer Applications, v.28, n.1, p.1-18.

Han, J.; Kamber, M. Data Mining: Concepts and Techniques. Morgan & Kaufmann, San Francisco, 2006.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reautmann, P., Witten, I. (2009) “The WEKA data mining software: an update”, ACM SIGKDD Explorations Newsletter, v. 11, n.1, p. 10-18, 2009.

Henricksen, K.; Indulska, J.; Mcfadden, T.; Balasubramaniam, S. (2005), “Middleware for distributed context-aware systems”, In: *Lecture Notes in Computer Science*, 3760:846–863.

J2SE (2012). *J2SE Development Kit.* Disponível em: <http://www.oracle.com/technetwork/java/javase/index-jsp-135232.html>. Acesso em: 19 agos 2012

McCulloch, W. S.; Pitts, W. (1943) "A Logical Calculus of the Ideas Immanent", in Nervous Activity**.** Bulletin of Mathematical Biophysics**,** n.5, p.115-133, 1943.

Neles, K.; Malvezzi, W.; Bressan, G. (2012) "Dealing with uncertainties in the monitoring of patients through sensors networks", Health Care Exchanges (PAHCE), 2012 Pan American, p. 50-57, doi:10.1109/PAHCE.2012.6233439.

Papazoglou, M. P. Service-Oriented Computing: Concepts, Characteristics and Directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE'03), p.3-12, 2003.

Pessoa, R. M. (2006) " Infraware: Um Middleware de Suporte a Aplicações Sensíveis ao Contexto", Master Thesis, Universidade Federal do Espírito Santo (UFES).

Platt J. Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schoelkopf and C. Burges and A. Smola, editors, Advances in Kernel Methods - Support Vector Learning, 1998.

Raatikainen, K.; Christensen, H. B. T.; Nakajime, T. (2002), " pplication requirements for middleware for mobile and pervasive system**s**", In: *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):16-24.

Ranganathan, A.; Cambell, R.H. (2003) "A Middleware for Context-Aware Agents in Ubiquitous Computing Environments", ACM/IFIP/USENIX Int'l Middleware Conf., LNCS 2672, Springer-Verlag.

Samaras, G.; Kshemkalyani, A.D.; Citron, A. (1996) "Context management and its applications to distributed transactions", Proceedings of the 16th International Conference on Distributed Computing Systems.

Santos, L. O. B da S. (2004) " Semantic Services Support for Context-Aware Platforms" Dissertação de Mestrado, Programa de Pós Graduação em Informática, Universidade Federal do Espírito Santo.

Silva, F. S.; Alves, L. G. P.; Bressan, G. (2012) " PersonalTVware: An Insfrastructure to Support the Context-Aware Recommendation for Personalized Digital TV", In: International Journal of Computer Theory and Engineering (IJCTE), v. 4, n.2.

Silva, F. S. (2011) "PersonalTVware: uma infraestrutura de suporte a sistemas de recomendação sensíveis ao contexto para TV Digital Personalizada", Tese (Doutorado em Sistemas Digitais) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2011. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3141/tde-31052011-171129/pt-br.php>.

Strang, T.; Linnhoff-Popien, C. (2004), "A context modeling survey", In: First International Workshop on Advanced Context Modelling, Reasoning And Management, Nottingham, England.

Tan, P-N., Steinbach, M., Kumar, V. Introdução ao Data Mining, Ciência Moderna,Rio de Janeiro, 2009.

Truong, K. N.; Abowd, G. D.; Brotherton, J. A. (2001) “Who, What, When, Where, How: Design Issues of Capture &Access Applications.”, In: Proceedings of Ubicomp 2001: Ubiquitous Computing. p. 209-224.

Vapnik, V. N., The nature of Statistical learning theory. Springer-Verlag, New York, 1995.

Vieira,V.; Souza, D.; Salgado, A. C.; Tedesco, P. (2006), “Uso e Representação de Contexto em Sistemas Computacionais”, Mini-curso apresentado no Simpósio de Fatores Humanos em Sistemas Computacionais (IHC 2006), Natal, Brasil.

Vieira, V.; Tadesco, P.; Salgado, A. C. (2009) “Modelos e Processos para o Desenvolvimento de Sistemas Sensíveis ao Contexto”, Mini-curso apresentado no XXIX Congresso da Sociedade Brasileira de Computação (CSBC 2009), Bento Gonçalves-RS, Brasil.

W3C. Web Services Activity. World Wide Web Consortium (W3C) Recommendation. Disponível em:<http://www.w3.org/2002/ws>. Acesso em: 18 agos. 2012.

W3C. Simple Object Acess Protocol (SOAP) 1.2. World Wide Web Consortium (W3C) Recommendation. Disponível em:<http://www.w3.org/TR/soap12-part0/>. Acesso em: 18 agos. 2012

Weka (2012), Use Weka in your Java Code, Disponível em:<http://weka.wikispaces.com/Use+Weka+in+your+Java+code>. Acesso em: 19 agos 2012

Witten, I. H.; Franck, E. Data Mining: Practical Machine Learning Tools and Techniques”, Elsevier, 2005.

Zimmer, T. (2004) “Towards a Better Understanding of Context Attributes”. Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, p.23, March 14-17.

Zimmermann, A.; Specht, M.; Lorenz, A.(2005) “Personalization and Context Management”, User Modeling and User – adapted Interaction. Volume 15, Numbers 3-4 (2005), 275-302, DOI: 10.1007/s11257-005-1092-2

PROMOÇÃO

SBC

ORGANIZAÇÃO

IME-USP PoliUSP

PATROCÍNIO

cgi.br nic.br CAPES CNPq 60 ANOS FAPESP SAP

USP PRCEU LARC IME-USP PRÓ REITORIA Pós-Graduação USP EMC² Google

WebMedia em cooperação com

acm In-Cooperation SIGMM sigweb

ISBN: 978-85-7669-265-2