COPPE/UFRJ

DADOS AUTONÔMICOS

Marcelino Campos Oliveira Silva

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Jano Moreira de Souza

Rio de Janeiro
Março de 2010

DADOS AUTONÔMICOS

Marcelino Campos Oliveira Silva

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____
Prof. Jano Moreira de Souza, Ph.D.

_____
Prof. Ricardo Oliveira Barros, D.Sc.

_____
Prof. Carlos Kamienski, D.Sc.

_____
Prof. Alexandre de Assis Bento Lima, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
MARÇO DE 2010

Silva, Marcelino Campos Oliveira

Dados Autonômicos / Marcelino Campos Oliveira Silva – Rio de Janeiro: UFRJ/COPPE, 2010.

XI, 127 p.: il.; 29,7 cm.

Orientador: Jano Moreira de Souza

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2010.

Referências Bibliográficas: p. 116-126.

1. Computação autonômica

I. Souza, Jano Moreira de

II. COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação III. Título

Agradecimentos

Em primeiro lugar agradeço a Deus pela oportunidade de ter nascido e por ter me dado capacidade de chegar aonde cheguei.

Gostaria de agradecer aos meus pais Aparecida Campos Meireles Silva e Raimundo Oliveira Silva, por todo o carinho, educação e atenção que me deram durante toda minha vida. Mesmo meu pai não estando vivo para me ver aqui, tenho certeza de que ele está me vendo em algum lugar com orgulho, e a minha mãe devo toda a minha gratidão por ter me dado força mesmo estando distante.

Ao Professor Jano Moreira de Souza pela oportunidade de aprendizado concedida durante o mestrado, por ter se disposto a me orientar e me auxiliar nesta jornada com suas vastas e fantásticas idéias.

Aos professores Ricardo Barros, Carlos Kamienski e Alexandre Assis por, em meio a tantos compromissos, engrandecerem este trabalho ao aceitarem fazer parte desta banca de mestrado e por fazerem tantos comentários pertinentes como sei que farão.

Também devo um agradecimento especial à querida Professora Jonice Oliveira que sempre se dispôs a me ajudar e que teve participação fundamental no inicio deste trabalho e me honra também como membro da banca.

Ao doutorando Wallace Anacleto Pinheiro que se dispôs a me ajudar em minha jornada. Orientou-me em meu caminho com sua sabedoria e experiência e me guiou muito bem. Muito obrigado pela sua amizade, conselhos e auxílio prestado.

Mas, este não é um espaço de agradecimento apenas de pessoas físicas e devo, com isso, agradecer as instituições que de alguma forma me apoiaram. Agradeço ao Instituto Alberto Luis Coimbra de Pesquisa e Pós-Graduação em Engenharia (COPPE) e ao Programa de Engenharia de Sistemas e Computação (PESC) por ter me acolhido como aluno e ter me dado todo o apoio necessário para conduzir meu trabalho. Ao CNPQ que forneceu minha bolsa de fomento e a fundação COPPETEC que me permitiu por em prática muitos dos meus conhecimentos teóricos.

E a tantos outros que de maneira direta ou indireta influenciaram no desenvolvimento deste trabalho.

Obrigado a TODOS!!!

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M. Sc.)

DADOS AUTONÔMICOS

Marcelino Campos Oliveira Silva

Março/2010

Orientador: Jano Moreira de Souza
Programa: Engenharia de Sistemas e Computação

Com o avanço tecnológico, e principalmente com o desenvolvimento da internet nos últimos anos, as pessoas passaram a lidar com o problema de como gerenciar a sobrecarga de dados, resultado de uma imensa quantidade de informações disponíveis atualmente.

Outro fator complicador é a heterogeneidade de ambientes e de usuários em que esses dados devem atuar. Quando um dado se move por entre os diversos sistemas, muitas vezes, deve ser transformado, reprocessado, adaptado, etc. Essas funcionalidades adicionais, presentes nos sistemas, consomem tempo e os tornam cada vez mais complexos. Em vez de se preocupar com as regras do negócio, grande parte do esforço no desenvolvimento dos sistemas é destinado ao tratamento dos dados.

Sendo assim, este trabalho propõe uma estrutura de dados que carregue um pouco da complexidade e da semântica do sistema consigo. Essa complexidade embutida nos dados soluciona alguns problemas que os sistemas têm para manipular os dados facilitando assim o desenvolvimento e manutenção do sistema. Dessa forma, a complexidade adicional que fica embutida nos sistemas, para manipular os dados, é removida e colocada nos próprios dados.

Adicionalmente é apresentado o modelo de dados que permite que a semântica seja incorporada ao dado, além da arquitetura que suporta o Dado Autonômico. Para validação da proposta é feito um estudo de caso na aplicação que foi desenvolvida com Dados Autonômicos deste trabalho.

A aplicação desenvolvida é uma ferramenta para filtragem de notícias de Feeds RSS. É feito um experimento que comprova que utilizando os Dados Autonômicos é possível agregar características autonômicas a um sistema.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

AUTONOMIC DATA

Marcelino Campos Oliveira Silva

March/2010

Advisor: Jano Moreira de Souza
Department: Systems Engineering & Computer Science

With technological progress and especially with the rise of the Internet, people have had to deal with the problem of how to manage data overload. This situation is a result of the huge volume information that is presently available.

Other problem is about heterogeneous nature of environments and applications where this data are used. When the data move through many systems, many times, they have to be transformed, re-processed, adjusted, etc. These additional functionalities, which are embedded in the systems, spend time and make them more complex. In spite of worrying about the business-rule, the bigger part of efforts of software development is destined to data treatment.

In this way, this work proposes a data structure that carries a little bit of systems' complexity and semantic. This strategy transfers the complexity of the systems to the data and allows their re-utilization in different environments, as rules can define how they will behave.

Moreover, the data model that allows semantic being embodied to data is presented, in addition, the architecture that supports Autonomic Data. In order to validate this propose a study case has been done with an application that have Autonomic Data.

The developed application is an aggregator to filtering Feeds RSS news. It's carried out an experiment that proved with Autonomic Data is possible aggregate autonomic characteristics to a system.

Sumário

Índice de Figuras

Índice de Tabelas

Lista de Siglas

API - Application Program Interface

CHOP – regras autonômicas: Auto-Configuração, Auto-Cura, Auto-Otimização e Auto-Proteção

COE - Collaborative Ontology Editor

COPPE – Coordenação dos Programas de Pós-graduação em Engenharia

COTS - Commercial Off-The-Shelf

DBC - Desenvolvimento Baseado em Componentes

IBM - International Business Machines

IME – Instituto Militar de Engenharia

KCE - Knowledge Chains Editor

KIF - Knowledge Interface Format

MDA - Model Driven Architecture

OMG - Object Management Group

OWL - Ontology Web Language

P2P – Peer to Peer

PAL - Protégé Axiom Language

PIM - Platform Independent Model

PMCE - Project Management Collaborative Editor

POO - Programação Orientada a Objetos

PSM - Platform Specific Model

RDF - Resource Description Framework

RSS 1.0 - RDF Site Summary 1.0

RSS 2.0 - Really Simple Syndication

SOA - Service-oriented architecture

SWRL - Semantic Web Rule Language

TI - Tecnologia da Informação

UFRJ – Universidade Federal do Rio de Janeiro

UML – Unified Modeling Language

WWW – Word Wide Web

XML – eXtensible Markup Language

Capítulo 1. Introdução

1.1. Motivação

Estamos vivenciando nos dias de hoje uma era de grandes avanços tecnológicos, estando com isso, ao nosso alcance, facilidades que antes eram inimagináveis. Essas inovações vêm acontecendo constantemente e em um ritmo cada vez mais forte, de modo que o espaço de tempo entre os avanços está cada vez menor.

Essas facilidades proporcionaram a disseminação de dados, de modo que a informação[1] estivesse ao alcance de todos. Na última década a internet foi uma das grandes responsáveis por essa disseminação, pois popularizou o acesso e divulgação de informações. Sendo assim, qualquer pessoa pode postar as mais diversas informações sobre sua vida profissional ou pessoal. Além disso, atualmente essas informações podem ser acessadas dos mais diversos dispositivos. Isto, em contrapartida, também trouxe problemas, como o aumento considerável da quantidade de dados disponíveis. Somam-se a isso, dados oriundos de empresas e instituições na forma on-line de revistas, jornais, artigos, etc.

E ainda, a necessidade de integrar esses vários dispositivos heterogêneos com grandes sistemas corporativos e estender, para além dos limites da companhia, na internet, introduziu novos níveis de complexidade.

Com isso, os sistemas atuais têm sido desenvolvidos de forma muito complexa para suprir os diversos tratamentos de dados que são oriundos de vários sistemas heterogêneos. Adicionalmente, outro problema enfrentado, decorrente disso, é que o usuário e desenvolvedor precisam se preocupar muito com detalhes de implementação devido à complexidade e a inter-relação entre os sistemas.

Um exemplo da complexidade que vem sendo tratada são as diversas especificações dos sistemas que mudam a todo o momento. Com isso os sistemas acabam tendo que manipular uma variedade maior de tipos de dados diferentes. Se antes

[1] *Apesar da sutil diferença entre dado (representação/notação) e informação (significado/denotação), neste trabalho foram usados estes termos sem distinção dos mesmos*

um sistema tratava de dois tipos de dados, no dia seguinte pode ter que tratar quatro, e com isso, o sistema precisa ser modificado para contemplar o tratamento desses novos tipos. Conseqüentemente, os usuários também sofrem com isso, pois acabam tendo que aprender como lidar com o sistema dessa forma diferente.

Então, seria desejável que os dados tivessem mais características autonômicas[2] para tirar um pouco do fardo dos sistemas. Por exemplo, em caso de mudanças de especificação dos sistemas, os dados deveriam encapsular essas novas especificações de forma que o consumidor deste não precisasse saber disto. Além disso, deveria fornecer uma interface de comunicação de modo que os sistemas já soubessem como agir com esses novos dados, e com isso o sistema não precisaria ser modificado facilitando o desenvolvimento.

1.2. Objetivo do Trabalho

O objetivo deste trabalho é apresentado abaixo em dois níveis de abstração, um nível mais abstrato e geral e um nível mais específico que detalha a proposta em partes menores. Além disso, é definido claramente o que não é do objetivo desse trabalho.

Objetivo geral: Definir o conceito de Dado Autonômico e definir um arcabouço[3] que o suporte.

Objetivos específicos:

Os objetivos específicos foram definidos de modo a permitir que o objetivo geral seja alcançado. São eles:

- Definir as características do Dado Autonômico;
- Diferenciar Dados Autonômicos de Agentes e dos Objetos da Orientação a Objetos.
- Criar um modelo de dados que permita que a semântica seja incorporada ao dado, de forma a permitir a criação de regras bem como sua execução;

[2] O conceito de Computação Autonômica será mais bem detalhado no capítulo 4, mas por hora é possível usar o conceito básico que é um programa de computador voltado para o estudo de sistemas complexos com o uso de simulações que é executado com o mínimo de intervenção humana

[3] Adaptou-se a definição de GOVONI (1999) utilizada para *framework* de modo a definir um arcabouço computacional como uma coleção de classes, interfaces e padrões destinada a resolver uma classe de problemas através de uma arquitetura flexível e extensível.

- Criar a arquitetura de suporte ao Dado Autonômico;
- Implementar uma aplicação que utilize o Dado Autonômico;
- Demonstrar que o sistema passa a incorporar propriedades autonômicas com o uso desse dado;
- Demonstrar que parte da complexidade dos sistemas é transferida para o Dado Autonômico facilitando a vida dos programadores e dos usuários finais.

1.3. Justificativa:

Problemas com a complexidade dos sistemas e o desejo de autonomia em relação a estes não é algo novo, muito pelo contrário, desde que o desenvolvimento de sistemas começou a ser feito em larga escala que essas características são almejadas. Na literatura encontramos diversos casos onde foram propostas soluções para esses problemas em separado, mas nenhuma delas reuniu todas as abordagens em um só lugar. Tem-se que a solução proposta pode, ao mesmo tempo, tirar um pouco da complexidade do sistema, facilitando o trabalho do desenvolvedor; aumentar a semântica dos dados e proporcionar soluções autonômicas a diversos problemas enfrentados.

1.4. Delimitação

Propor um novo conceito nunca é fácil, ainda mais se esse novo conceito abrange fatos, perspectivas e métodos já conhecidos. Além disso, novos conceitos sempre estão passíveis de erros e podem acabar evoluindo com o tempo. Pensando nisso, este trabalho se propõe a definir o cerne dessa nova teoria que estará passível a sofrer evoluções e extensões, tão logo novos fatos apareçam.

Este trabalho também propõe cenários reais de aplicação detalhando como seria a implementação e mostrando quais seriam as vantagens obtidas se o conceito fosse aplicado. Adicionalmente, implementa uma solução e a testa em um estudo de casos, mostrando os conceitos sendo aplicados e as vantagens que foram obtidas.

Para verificar as vantagens obtidas na solução implementada foi realizado um experimento, de modo a verificar a qualidade do trabalho feito. Foram utilizadas

técnicas de engenharia de software experimental seguindo as idéias de BARROS *et al* (2002) de modo a obter os melhores resultados possíveis.

Para finalizar essa subseção, é bom deixar claro que este trabalho não tem por objetivo propor um arcabouço geral e definitivo para qualquer tipo de aplicação, pois é entendido que cada área deve ter sua própria implementação de modo a tirar o melhor proveito possível da aplicação. Com isso, pode-se dizer que o presente trabalho objetiva propor um conjunto de recomendações e normas para o desenvolvimento de sistemas de modo a contornar os problemas descritos e aproveitar as vantagens também descritas. Em outras palavras, pode-se dizer que é uma orientação de como o arcabouço deve ser construído.

1.5. Metodologia de Pesquisa

Sendo o presente trabalho de um estudo de natureza predominantemente qualitativa, cujo caráter é exploratório, necessita-se de uma metodologia[4] de pesquisa adequada. Para isso as informações coletadas foram analisadas e discutidas sendo dispostas de acordo com as correntes de idéias apresentadas.

Segundo TRIVIÑOS (1994), a pesquisa de caráter qualitativa deve ser essencialmente descritiva e o objeto de sua atenção devem ser destinados a propósitos que tem utilidade à vida das pessoas. Já segundo RUDIO (2001), na pesquisa descritiva o pesquisador procura conhecer os fatos para então interpretar, descrever e classificar a realidade, sem nela interferir para modificá-la. Para LÜDKE *et al* (1990), este tipo de investigação abrange todo um contexto, cujo ambiente natural é a fonte direta de dados e a preocupação maior deve ser com o processo, o aprendizado e o significado do que com o produto final.

Segundo CRUZ & RIBEIRO (2003) "Uma pesquisa bibliográfica pode visar um levantamento dos trabalhos realizados anteriormente sobre o mesmo tema estudado no momento. Pode identificar e selecionar os métodos e técnicas a serem utilizados, além

[4] Diante das diferentes definições encontradas em diversas fontes sobre metodologia e método, optou-se por utilizar a seguinte definição (HOUAISS, 2001):

- Metodologia - parte de uma ciência que estuda os métodos aos quais ela própria recorre;
- Método - o procedimento, técnica ou meio de se fazer alguma coisa, de acordo com um plano.

de fornecer subsídios para a redação da introdução e revisão da literatura do projeto ou trabalho. Em suma, uma pesquisa bibliográfica leva ao aprendizado sobre uma determinada área". Contudo, uma pesquisa bibliográfica deve ainda ser completada. "Uma pesquisa de laboratório, diferentemente de uma pesquisa bibliográfica ou de campo, permite ao pesquisador manipular suas variáveis, isolá-las ou até mesmo provocar eventos passíveis de controle"(LÜDKE; ANDRÉ, 1990).

Segundo MARCONI & LAKATOS (2007), uma investigação alcança seus objetivos de forma científica quando cumpre ou se propõe a cumprir as seguintes etapas:

- Descobrimento do problema
- Colocação precisa do problema
- Procura de conhecimentos ou instrumentos relevantes ao problema
- Busca de uma solução baseada em princípios encontrados nas mais diversas abordagens pesquisadas
- Invenção de novas idéias e novas perspectivas
- Correção das hipóteses levantadas, teorias, procedimentos ou dados empregados na obtenção da solução incorreta.

Baseando-se em todas essas definições esse trabalho procurou fazer um detalhamento do problema explorando onde ele acontecia e o que abrangia; um levantamento das mais importantes soluções imaginadas destacando as melhores características de cada um. A partir dessas características se buscou formular uma nova solução que atendesse a proposta. Depois de definidos buscou-se validar a proposta e verificar sua viabilidade.

1.6. Organização do Trabalho

O trabalho é organizado da seguinte forma: o capítulo 2 detalha o problema, fazendo-se um levantamento de como ele afeta a vida das pessoas. No capítulo 3 é feita uma revisão literária dos assuntos que foram usados para construir a proposta. No capitulo 4 é definido o conceito de Dado Autonômico bem como detalhado sua arquitetura e suas características. No capítulo 5 é mostrada a implementação proposta

que valida o conceito de Dado Autonômico. No capítulo 6 é apresentada a navegabilidade de ferramenta e é feito um experimento para validar a implementação. Por fim, no capítulo 7 são feitas as conclusões obtidas durante o trabalho e é aberta a discussão para trabalhos futuros.

Capítulo 2.O Problema da Complexidade dos Sistemas para Tratar os Dados

Este capítulo apresenta o problema da complexidade de desenvolvimento e de manutenção dos sistemas. Além disso, mostra algumas soluções que a humanidade tem usado para contornar este problema ao longo desses anos. Devo ressaltar que esse trabalho não é sobre metodologias de desenvolvimento de software, apesar de esta área ter sido explorada. O objetivo de várias técnicas terem sido estudadas foi devido ao caráter exploratório que permitiu conhecer como os vários níveis de complexidade de software são tratados pelas diversas técnicas, de modo a dar uma base de conhecimento maior para a evolução deste trabalho.

2.1. Problema enfrentado pelos sistemas computacionais: complexidade

Em outubro de 2001, a IBM lançou um manifesto observando que o principal obstáculo para o progresso na indústria de Tecnologia da Informação (TI) aparentava ser a crise de complexidade de hardware e software. A companhia citou aplicações e ambientes que possuíam na faixa de 10 milhões de linhas de código e que requeriam profissionais extremamente habilidosos para instalar, configurar, coordenar e mantê-los. (IBM, 2009)

O Manifesto demonstrou que a dificuldade de administrar sistemas computacionais nos dias de hoje vai muito além da administração de ambientes de software individuais. A necessidade de integrar vários ambientes heterogêneos com grandes sistemas corporativos e estender para além dos limites da companhia na internet introduz novos níveis de complexidade. Com isso, a complexidade dos sistemas de computação parece se aproximar dos limites da capacidade humana, pois a tendência de integração e inter-conectividade caminha a passos largos.

Esse passo pode tornar o sonho da computação pervasiva (trilhões de aparelhos conectados a internet) em um pesadelo (IBM, 2009). Segundo KEPHART & CHESS

(2003), inovações nas linguagens de programação têm facilitado a extensão do tamanho e a complexidade dos sistemas que os arquitetos podem projetar, mas utilizar somente essas inovações não irá nos salvar da crise de complexidade.

Mas, essa chamada crise do software não é algo novo. Desde a metade da década de 1980 que BROOKS (1987) já previa uma crise que logo recairia sobre a produção mundial de software. Em seu artigo ele descreve que não existia uma "bala de prata" (em alusão a arma que mata um lobisomem) para solucionar o problema da crescente demanda de software e a complexidade das soluções utilizadas. Já COX (1990), acredita que, apesar da crise ser eminente, essa teria uma solução, mas que seria necessária uma revolução industrial do software.

O manifesto ainda relata que, ao passo que os sistemas se tornam mais interconectados e diversificados, arquitetos de software ficam menos aptos a antecipar e projetar interações entre os componentes, deixando que essas partes sejam tratadas em tempo de execução. Em breve sistemas irão se tornar excessivamente massivos e complexos mesmo para o mais habilidoso integrador de sistemas instalar, configurar ou manter. A seguir é apresentado como algumas linguagens de programação e técnicas de desenvolvimento de software evoluíram para contornar o problema da complexidade.

2.2. Evolução nas linguagens de programação

Para entender o problema enfrentado hoje com a complexidade de desenvolvimento de software é necessário olhar alguns anos atrás e verificar como a complexidade aumentou com o tempo e como as técnicas de programação tiveram que evoluir para tratá-la.

No inicio da década de 50 a atividade de programação de computadores era algo restrito a poucas pessoas e era constituído por uma pequena quantidade de comandos. Os programas eram relativamente simples e executavam tarefas pequenas. Segundo RESENDE & SILVA (2005), os produtos de software eram desenvolvidos utilizando a programação desestruturada e comandos de desvio de fluxo (por exemplo, o comando GOTO) eram usados indiscriminadamente.

Porém o acesso a essa área foi se tornando maior, os programas ganharam mais funções e com isso o código começava a ficar maior e mais difícil de entender. Os comandos de desvio de fluxo que, antes facilitavam o trabalho, começavam a dificultar

o mesmo, pois, tornava muito difícil o entendimento e a reutilização do código-fonte. A demanda dos softwares a serem produzidos exigia uma melhor metodologia para aperfeiçoar a produção dos produtos de software.

Nesse sentido, a programação estruturada foi um grande avanço, pois a funcionalidade era agrupada em funções ou procedimentos, que eram chamados a partir de um programa principal. Quem utilizava a programação estruturada podia reutilizar as funções e os procedimentos, desde que devidamente projetados. Nesta época, a reusabilidade foi explorada, utilizando o conceito de bibliotecas.

Porém, a programação estruturada também foi se tornando ineficiente. À medida que aumentava a complexidade dos produtos de software, surgiu a necessidade de melhores técnicas de programação, objetivando organizar e apoiar o processo de desenvolvimento de produtos de software.

Com o advento da Programação Orientada a Objetos (POO), em meados dos anos 70, com a linguagem de programação Smalltalk, iniciou-se um novo paradigma de desenvolvimento de produtos de software que está presente até os dias atuais. A POO trouxe grandes avanços para o desenvolvimento de produtos de software, permitindo a construção de sistemas mais fáceis de serem projetados, maior reusabilidade de componentes, modularidade, componentização, implementações mais robustas e redução do custo de manutenção (JUNIOR; WINCK, 2006). Muitas aplicações não estão em apenas um único módulo de código contido em um único arquivo. Aplicações são coleções de módulos (classes) que trabalham juntas para fornecer determinadas funções para um conjunto de requisitos bem definidos (GRADECKI; LESIECKI, 2003).

Entretanto, apesar de tais avanços, o aumento da complexidade inseriu novos problemas que a POO é incapaz de resolver. Existem certas propriedades que não são adequadamente encapsuladas em classes. Na concepção de KICZALES *et al.* (1997), nem as técnicas de POO nem as técnicas da programação procedural (estruturada) são suficientes para implementar com clareza importantes decisões do projeto. Entre essas importantes decisões de projeto podemos citar a dificuldade de retirar requisitos não funcionais que ficam entrelaçados com os requisitos do sistema.

Sendo assim, mesmo a POO sendo usada largamente nos dias de hoje ela já mostra sinais de cansaço, e várias outras técnicas vêm surgindo. Existem algumas extensões da POO que tentam solucionar suas limitações visando a uma maior modularidade dos produtos de *software*, tais como Programação Orientada a Aspectos

(*Aspect-Oriented Programming*) (KICZALES *et al*., 1997), Programação Orientada a Sujeito (*Subject-Oriented Programming*) (OSSHER; PERI, 1999) e Programação Adaptativa (*Adaptive Programming*) (LIEBERHERR *et al*., 1994).

2.3. Metodologias de Desenvolvimento que tratam a complexidade independente de linguagem.

Com a integração cada vez maior de sistemas, a administração de complexidade se tornou muito mais complicado do que simplesmente olhar as limitações e vantagens da linguagem de programação utilizada, pois muitos sistemas acabam tendo que ser implementados utilizando várias linguagens de programação, todas de forma integrada.

Com isso, citamos aqui algumas metodologias promissoras que visam construir sistemas integrados, mas diminuindo a complexidade dos mesmos e que são independentes de linguagem de programação. É bom deixar claro que não são discutidos processos de desenvolvimento de software[5], pois estes envolvem algumas questões como produtividade e custo, que não fazem parte do escopo desse trabalho.

2.3.1. Desenvolvimento Baseado em Componentes.

Segundo SPAGNOLI & BECKER (2003), o Desenvolvimento Baseado em Componentes (DBC) se caracteriza como uma abordagem de desenvolvimento estabelecida pela integração planejada de componentes de *software* existentes. Esta abordagem enfatiza o reuso e apresenta vantagens no sentido de possibilitar o aumento da produtividade e qualidade. A técnica de Desenvolvimento Baseado em Componentes visa fornecer um conjunto de procedimentos, ferramentas e notações, possibilitando que, ao longo do processo de desenvolvimento de software, ocorra tanto a produção de novos componentes, quanto a reutilização de componentes existentes.

[5] Segundo Pressman (2004) processo de desenvolvimento de software pode ser definido como um conjunto de atividades com a finalidade de obter um produto de software. É considerado um dos principais mecanismos para se obter um software de qualidade e cumprir corretamente os contratos de desenvolvimento. Entre os mais conhecidos tem-se o modelo em cascata, modelo iterativo e os métodos ágeis.

O DBC facilita a manutenção dos sistemas por possibilitar sua atualização através da integração de novos componentes e/ou substituição dos componentes existentes.

Um componente é definido como uma unidade de software independente que encapsula dentro de si seu projeto e implementação e oferece interfaces bem definidas para o meio externo. Por meio destas interfaces, um componente pode se unir a outros componentes e dar origem aos sistemas baseados em componentes. Um componente apresenta interfaces que separam sua especificação da implementação (D'SOUZA; WILLS, 1998).

Essas interfaces fornecem um encapsulamento que não permite aos usuários conhecer os detalhes de implementação. A especificação de um componente é, normalmente, publicada separadamente de seu código fonte por meio da especificação das interfaces oferecidas por ele. Componentes podem variar desde componentes construídos pelo cliente, ou componentes adquiridos para um domínio específico, até componentes COTS (*Commercial Off-The-Shelf – Software* de prateleira*).* Componentes COTS são componentes genéricos que podem ser prontamente adquiridos no mercado (de prateleira). Existe uma grande motivação para o uso de componentes COTS, mas para que o uso efetivo desses componentes se realize ainda é necessário o desenvolvimento de métodos mais precisos para sua produção e utilização (CARNEY, 1997).

Componentes de software podem ser constituídos por componentes mais simples. Assim o processo de construção de componentes pode ser considerado como recursivo. Nesse contexto, um componente pode ter não apenas código pronto para reutilização, mas também unidades de projeto que têm significado próprio. Por exemplo, no contexto de DBC arcabouços podem ser vistos como componentes flexíveis, em nível de projeto. Para que a reutilização de componentes seja possível é preciso que estes sejam adaptáveis. O projeto de um componente deve ser conduzido de tal forma que, além de tornar a sua execução correta e eficiente, deve ser genérico para que se torne adaptável a vários propósitos e não somente ao propósito ao qual será primeiramente aplicado.

Segundo SPAGNOLI & BECKER (2003), apesar de todas as potenciais vantagens desta abordagem, existem também inúmeras questões em aberto e definições ainda sem consenso que têm dificultado a ampliação do uso dessa metodologia.

Componentes, modelo de componentes e arcabouços de componentes são exemplos de termos básicos que freqüentemente são alvos de confusão.

2.3.2. Padrões de Projeto

Segundo GAMMA *et al* (1995), existiam muitos problemas no desenvolvimento de software que ocorriam repetidamente. Estes problemas acabavam tendo soluções similares, pois sempre tinham raízes em comum. Com o passar do tempo, resolver esses problemas era algo até de certa forma automatizado, pois um programador experiente já percebia logo de início que deveria desenvolver a aplicação de certa forma, para ter o seu código otimizado. Mas, mesmo já sabendo dos possíveis problemas, as mesmas soluções eram refeitas várias e várias vezes, e como se diz na linguagem popular "a roda era sempre reinventada". Pensando nisso, essas soluções foram reunidas, formalizadas e melhor descritas se tornando padrões de projeto.

Sendo assim, os padrões de projeto de software[6] são soluções para problemas recorrentes no desenvolvimento de sistemas de software orientados a objetos. Um padrão de projeto estabelece um nome, define o problema, estabelece a solução, quando deve ser aplicada e suas conseqüências. Um padrão de projeto não é um projeto final que pode ser transformado diretamente em código, mas sim uma descrição ou um modelo de como resolver um problema que pode ser usado em várias situações.

Os padrões de projeto visam facilitar a reutilização de soluções de projeto, isto é, soluções na fase de projeto do software, sem considerar reutilização de código. Também possuem um vocabulário comum de projeto, facilitando a comunicação, documentação e aprendizado dos sistemas de software.

Segundo GAMMA *et al* (1995), entre as vantagens do uso de padrões de projeto temos:

· **Padronização:** um padrão é a documentação de um problema e de sua solução. Cada padrão possui um nome que o identifica. Sua utilização pode tornar o código mais padronizado, facilitar a comunicação entre os membros da equipe, melhorar a documentação, padronizar a nomenclatura das classes, métodos e propriedade.

[6] Muitos autores e desenvolvedores gostam de usar o termo no inglês: *Design Patterns*

· **Abstração:** padrões de projeto ajudam na identificação de abstrações menos óbvias e os objetos que as compõem. Isto é uma chave para se criar um projeto mais flexível. Um exemplo disto são os padrões de criação.

· **Aumento da reutilização de código:** A reutilização de código se baseia na reutilização de soluções já usadas com sucesso aumentando assim a produtividade e qualidade do projeto (MILI, 1995).

Um dos objetivos básicos da orientação a objetos é a reutilização de código. Entretanto, não basta utilizar uma linguagem orientada a objetos para atingir esta meta. É necessário utilizá-la bem. Os padrões de projeto fornecem práticas saudáveis para se atingir este objetivo.

· **Redução de dependência entre objetos:** A redução de dependência é definida como a redução de acoplamento entre os objetos, fazendo com que estes se tornem mais independentes. Isto é muito importante para aumentar os níveis de flexibilidade e reutilização de código.

· **Facilidades em modificações:** A utilização de padrões de projeto facilita modificar e estender as funcionalidades de um aplicativo. Pode-se alterar classes e subsistemas sem afetar os clientes destes, como alterar o comportamento do aplicativo sem sequer precisar recompilar a aplicação, mesmo alterações significativas como qual banco de dados será utilizado. Aliando algumas técnicas à utilização de padrões de projeto isto é possível de forma elegante. O aplicativo fica mais flexível para se adaptar a novos requisitos.

· **Alternativa ao uso de heranças:** A herança consiste em uma classe herdar atributos e métodos de outra classe. Sendo assim, é um recurso muito importante de reutilização de código. Ela possui suas vantagens, mas também possui suas desvantagens. Uma subclasse é definida de forma estática no tempo de compilação. Isto dificulta a alteração do comportamento do aplicativo em tempo de execução. A herança entra em conflito com o princípio de encapsulamento, entre outros problemas. Padrões de projeto oferecem algumas alternativas com objetivo de prover maior flexibilidade.

· **Distribuição de responsabilidades:** A distribuição de responsabilidades consiste na divisão de características do sistema, de modo a deixar funções específicas em determinadas classes, facilitando com isso a coesão do código. O projeto orientado a objetos encoraja a distribuição de características entre objetos. Saber distribuir as responsabilidades entre os objetos é um fator de suma importância quando se deseja maximizar o potencial de reutilização de código.

2.3.3. SOA – Arquitetura Orientada a Serviço

A Arquitetura Orientada a Serviços *(Service-oriented architecture* - SOA) é um estilo de arquitetura de software[7] cujo princípio fundamental preconiza que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços. Pode ser entendida como um paradigma arquitetural que viabiliza a criação de serviços de negócio com baixo acoplamento e interoperáveis entre si, os quais podem ser facilmente compartilhados dentro e fora das corporações. (KRAFZIG *et al*., 2005)

O principio é simples e pode ser entendido como um conjunto de componentes que podem ser acessados e cujas interfaces podem ser divulgadas e pesquisadas. É uma metodologia de projeto e implementação de software baseada em componentes de negócio reutilizáveis, chamados serviços. Esses serviços são feitos de forma a permitir uma interação flexível, com módulos não muito granulares, que podem ser integrados através de interfaces, independentes de plataforma. A Figura 1 ilustra isso:

Figura 1 – Um sistema acessando um serviço

O modelo SOA permite que diferentes aplicações troquem dados facilmente. A Orientação a serviço visa o baixo acoplamento de serviços com sistemas operacionais, linguagens de programação e outras tecnologias que dão suporte às aplicações. A maioria das implementações SOA utilizam serviços Web (SOAP , REST e WSDL). Porém, uma implementação SOA pode se utilizar de qualquer tecnologia padronizada baseada em web.

[7] A arquitetura de software de um sistema consiste dos componentes de software, suas propriedades externas, e seus relacionamentos com outros softwares. O termo também se refere à documentação da arquitetura de software do sistema.

Segundo KRAFZIG *et al* (2005), as seguintes orientações definem as regras básicas para desenvolvimento, manutenção e uso de SOA:

*Reuso, granularidade, modularidade, composição, componentização, portabilidade, e interoperabilidade

*Conformidade com padrões (tanto abertos como específicos de indústrias)

*Identificação e categorização de Serviço, fornecimento e entrega, e monitoramento e entrega.

SOA ainda possui alguns problemas em aberto. Entre os obstáculos do SOA tem-se o gerenciamento de serviços. Ambientes baseados em SOA podem disponibilizar muitos serviços que trocam mensagem para executar as tarefas. Dependendo do projeto, um simples aplicativo pode gerar milhões de mensagens. Gerenciar e fornecer informações de como os serviços interagem pode ser uma tarefa complexa. Isso se torna mais complicado quando esses serviços são entregues por diferentes organizações dentro de uma companhia ou até por diferentes companhias (parceiros, fornecedores) (ROSEN *et al*., 2008).

Outro desafio envolve a falta de testes. Não existem ferramentas sofisticadas que forneçam testabilidade para todos os serviços em uma típica arquitetura. Outro desafio está na parte de segurança. Os modelos de segurança construídos em uma aplicação podem não fornecer o nível de segurança adequado quando os serviços são disponibilizados podendo ser usados por uma gama de aplicações (ROSEN *et al*., 2008).

Mas, apesar dos problemas conhecidos, esse paradigma já é reconhecido como uma das principais maneiras de arquiteturar e organizar as áreas de Tecnologia de Informação (TI) das empresas. Ron Schmelzer, analista sênior da ZapThink, diz que " Pelo menos 60% a 70% do dinheiro que está sendo gasto hoje nas tecnologias tradicionais de integração via *middleware*[8] ou de orientação a objetos migrará para as SOA nos próximos três a quatro anos". (GARNER, 2004).

2.3.4. MDA – Arquitetura Orientada a Modelos

A Arquitetura Orientada a Modelos (*Model Driven Architecture* – MDA) é um padrão criado e mantido pelo OMG (*Object Management Group).* O MDA define uma

[8] *Middleware* ou mediador é uma função que toma a frente de outra em uma chamada, processando os parâmetros antes de passá-los à função original e processando o(s) resultado(s) antes de devolvê-lo(s) à chamada

abordagem para especificação de sistema de TI que separa as especificações das funcionalidades do sistema das especificações da implementação daquela funcionalidade em uma plataforma de tecnologia específica. Para este fim, o MDA define uma arquitetura para modelos que fornece um conjunto de diretrizes para estruturar especificações expressas como modelos (“MDA,” 2009).

A abordagem MDA e os padrões que a apóiam permitem que o mesmo modelo seja usado em plataformas múltiplas por padrões de mapeamento auxiliares adicionais, ou através de pontos de mapeamentos para plataformas específicas, permitindo integrar aplicações diferentes através de seus modelos. Além disso, a interoperabilidade e apoio à evolução dos sistemas é maior quando pensado que o MDA pode proporcionar transformações bilaterais, ou seja, quando é possível tanto a codificação, onde o modelo se transforma em código, como a decodificação onde o código pode se transformar em modelo. Com isso se maximiza a produtividade e qualidade, pois o código gerado já foi testado em outras simulações.

Segundo ARLOW & NEUSTADT (2004), MDA divide o trabalho de desenvolvimento em duas áreas principais: o PIM (*Platform Independent Model* – Modelo Independente de Plataforma) e o PSM (*Platform Specific Model* – Modelo Específico de Plataforma). Os modeladores representam uma aplicação em particular, por meio da criação de um PIM. O PIM é o primeiro artefato especificado pela MDA, é a modelagem do sistema nos padrões da UML independente de qualquer tecnologia específica que será utilizada. E através da utilização de ferramentas é possível transformar o PIM num PSM. O PSM é o segundo artefato especificado pela MDA, é o modelo do sistema destinado a um ambiente de execução específico, para uma tecnologia em particular. Deste modo, mais ferramentas podem usar o PSM e gerar código para essa plataforma. Ou seja, o MDA é um modo de separar a arquitetura da aplicação de sua implementação. Assim, através do desacoplamento da arquitetura da aplicação de seu ambiente de execução, o uso de MDA pode resultar em melhores projetos com menor complexidade, pois, entre seus benefícios está o ganho de produtividade através da automatização e reutilização do trabalho, a portabilidade a outras plataformas, a interoperabilidade e o baixo acoplamento. Na Figura 2 temos a como é o ciclo de desenvolvimento utilizando o MDA.

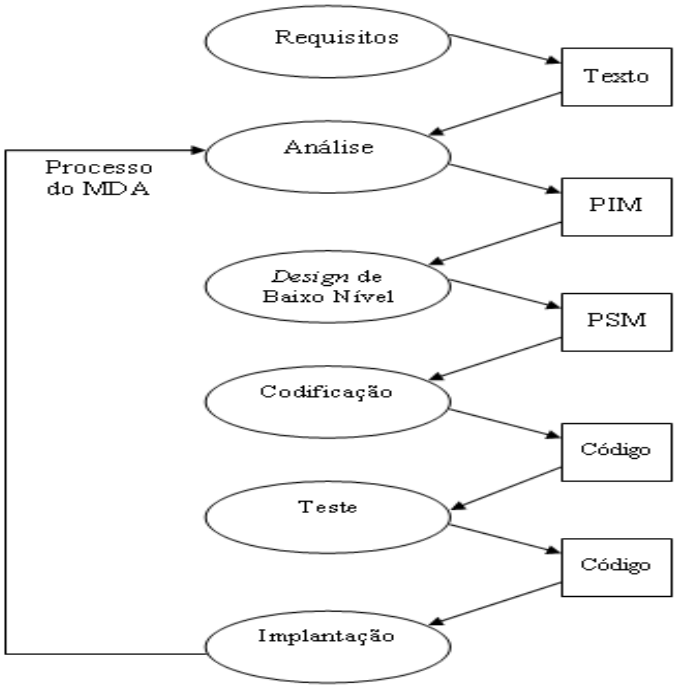

Figura 2 - Ciclo de Desenvolvimento do MDA

Outro fator interessante é que os múltiplos PSM's gerados pelo mesmo PIM podem se relacionar entre si. No MDA, este relacionamento é feito através de pontes. Quando PSM's são destinados a plataformas diferentes, eles não podem se comunicar diretamente. De uma forma ou de outra, é necessário transformar conceitos de uma plataforma em conceitos da outra. Isto é o que a interoperabilidade trata.

Na Figura 3 é mostrada a interoperabilidade do MDA.

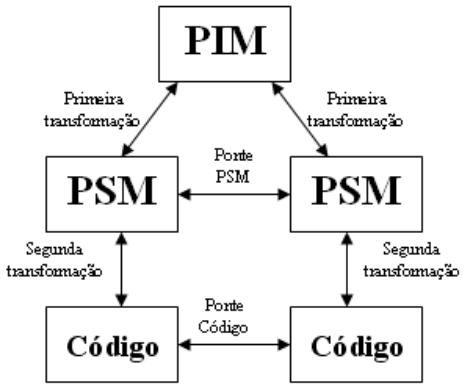

Figura 3 - MDA e sua interoperabilidade

2.4. Lições aprendidas e considerações:

Neste capítulo foi descrito e exemplificado o problema da complexidade de sistemas. Foram apresentadas também as diversas soluções utilizadas para contornar esse problema, permitindo um desenvolvimento de sistemas mais escalável. Porém, mesmo com tantas técnicas e metodologias existentes no desenvolvimento atual, este tem ficado mais complexo, pois os sistemas estão cada vez maiores, e sendo assim, a maioria dos processos de desenvolvimento de software tem tido que considerar os sistemas em unidades cada vez menores que ficam em diversas camadas. Essa complexidade dos sistemas tem dificultado o seu entendimento e administração. A necessidade de integrar vários ambientes heterogêneos com grandes sistemas corporativos vem exigindo projetos de software que forneçam abstrações e mecanismos de composição que suportem claramente os tipos de unidades descritas no projeto. Nesse sentido, tanto o uso de poderosas linguagens de programação quanto o uso de boas metodologias de software já não são mais suficientes, sendo necessárias metodologias de integração de sistemas que suportem a complexidade da integração de vários sistemas heterogêneos. Não há mais lugar para apenas uma abordagem e metodologia, e o presente tem nos mostrado que o futuro englobará o uso de diversas técnicas em conjunto para deixar o desenvolvimento e administração dos sistemas mais simples.

Em relação às técnicas, é possível observar alguns pontos interessantes, Mesmo tendo surgido diversas técnicas, estas têm vários pontos em comum, entre elas estão a modularização e o encapsulamento. Com isso, é possível dividir os problemas em unidades menores e para serem mais facilmente tratados. Porém, esta solução deve ser estudada cuidadosamente, pois a divisão de problemas em muitas camadas dificulta o entendimento do sistema como um todo, assim como dificulta encontrar erros. Outro conceito bastante utilizado por essas metodologias foi o de reuso, que se baseia em usar uma mesma solução em diversos pontos do sistema economizando tempo e facilitando a manutenção do mesmo.

Dentro das metodologias estudadas é fácil ver a aplicação dos padrões de projeto que são usados em larga escala, ao contrário do DBC e do MDA que ainda têm o seu

uso bem restrito devido a desconfianças do mercado e dos poucos trabalhos divulgados de sucesso.

Dentre as quatro, SOA é provavelmente a mais completa e com o futuro mais promissor. Segundo RITTINGHOUSE & RANSOME (2009), o uso de SOA em conjunto com a computação em nuvem[9] tem sido uma solução bastante comentada para a solução de atualização de hardware e software. Dessa forma, não será necessário que as empresas gastem fortunas em atualizações, pois as mesmas serão feitas no servidor que provê o serviço e as empresas pagariam apenas pelo uso do serviço. Baseado nisso, o autor deste trabalho acredita que SOA deve ser a metodologia mais utilizada nos próximos anos.

[9] **Computação em nuvem** ou ***Cloud computing*** é o modelo de desenvolvimento onde as aplicações permanecem em servidores físicos ou virtuais sendo acessadas a partir de uma rede em forma de serviços. Dessa forma, não é necessário dispor de um computador potente para executar aplicações que requerem grande processamento, muito menos seria necessário atualizar esses softwares.

Capítulo 3. Revisão da Literatura

Este capítulo apresenta a conceituação e a revisão bibliográfica realizada em torno dos assuntos que envolvem essa dissertação. Muitas abordagens e tecnologias foram utilizadas neste trabalho, sendo assim, o objetivo desse capítulo é apresentar mais profundamente cada uma delas.

É possível fazer uma divisão lógica dos assuntos estudados em duas partes: a primeira envolvendo a auto-gestão dos sistemas como mais uma alternativa à diminuição da complexidade e a segunda parte envolvendo tipos e semântica de dados.

Na parte de gestão de sistemas é discutida a Computação Autonômica. Depois, na segunda parte, explora-se a semântica de dados, onde é discute-se ontologias, OWL e SWRL.

3.1. A Computação autonômica como solução dos problemas de complexidade

Segundo HARIRI *et al* (2006), uma das opções para lidar com o incrível aumento de complexidade é a Computação Autonômica. Mas, antes de definir o que é esse termo, vamos ver um pouco de sua origem e entender melhor seu propósito.

O termo "Computação Autonômica" surgiu de um termo biológico associado ao "Sistema Nervoso Autonômico" que governa, por exemplo, a batida do coração e a temperatura do corpo, deixando livre a consciência do corpo livre deste fardo de lidar com funções vitais tão complexas. Nosso corpo funciona com uma hierarquia de auto-governância: das simples células até os órgãos e sistema nervoso. Cada nível mantém uma medida de independência, enquanto contribui para o auto-nível da organização que existe no nosso organismo. Com isso, nós nos mantemos despreocupados a maior parte do tempo, pois o sistema nervoso toma conta de si próprio. Porém, nem sempre cada parte de nosso corpo consegue resolver todos os problemas, e então este reporta o problema para que outra parte do organismo o consiga resolver. Muitas vezes entendemos isso na forma de dor, medo, mal estar.

Como exemplo dessa reação, considere o mecanismo que mantém a concentração de glicose no sangue, se a concentração cair abaixo de 0,06%, o tecido estará faminto de sua principal fonte de energia, se a concentração estiver acima de 0,018%, outros indesejáveis efeitos irão ocorrer. Se a concentração de glicose no sangue cair para abaixo de 0,07%, as glândulas renais criam adrenalina, que faz o fígado transformar o glicogênio armazenado em glicose. Isto vai para o sangue e a queda de concentração de glicose no sangue é interrompida. Mais adiante, a queda de glicose também estimula o apetite causando o consumo de comida, que depois da digestão fornece glicose. Por outro lado, se a concentração de glicose no sangue aumenta excessivamente, a secreção de insulina pelo pâncreas é aumentada, fazendo o fígado remover o excesso de glicose do sangue. Além disso, os rins excretam excesso de glicose na urina, os músculos e a própria pele também removem o excesso de glicose do sangue se este ultrapassar 0,18%. Com isso, existem cinco atividades que podem conter danos das flutuações da concentração de glicose no sangue. (ASHBY, 1960)

Assim, baseando em todos esses fatores, não poderia ter sido escolhido melhor nome, uma vez que esse paradigma é inspirado nessas funções naturais e tem por objetivo deixar a complexidade e incertezas do sistema para serem administradas pelo próprio sistema. Isto permite que o mesmo seja capaz de se auto-administrar com o mínimo de intervenção consciente humana, deixando as pessoas mais focadas em seus problemas específicos.

Assim como no sistema nervoso, a Computação Autonômica visa deixar o sistema em equilíbrio com o ambiente. Esse equilíbrio é uma condição necessária para a sobrevivência do sistema. No caso da computação, sobrevivência significa dizer que o sistema deve ter habilidade para proteger a si mesmo, se recuperar de falhas, se reconfigurar caso seja necessário, devido a mudanças no ambiente, e sempre manter suas operações com um mínimo de desempenho (HARIRI *et al.*, 2006).

Para controlar o comportamento do sistema, devem existir mecanismos que possam identificar e alertar sobre mudanças no sistema, disparando ao sinal de alguma mudança expressiva. Com isso, segundo PARASHAR & HARIRI **(**2005**)**, um sistema autonômico requer: (a) canais de sensores que detectem mudanças externas e internas no ambiente e (b) um canal que reaja a efeitos de mudanças no ambiente, mudando o sistema e mantendo o equilíbrio.

As mudanças detectadas pelos sensores têm de ser analisadas para se determinar os estados das variáveis críticas. Este planejamento requer conhecimento prévio do

sistema, de modo a selecionar o comportamento desejado dentro de um extenso número de possíveis comportamentos possíveis. Somente então, o canal reativo executaria a mudança selecionada. Sentir, Analisar, Planejar, Conhecer e Executar são, de fato, as palavras-chave usadas para identificar um sistema autonômico.

KEPHART & CHESS **(**2003**)** dizem que um sistema autonômico é, na verdade, uma coleção de elementos autonômicos que são definidos como sistemas individuais contendo recursos e entregando serviços a humanos ou a outros elementos autonômicos. Seu comportamento interno assim como seus relacionamentos são gerenciados de acordo com políticas estabelecidas por humanos ou outros elementos autonômicos. A Figura 4 ilustra isso.

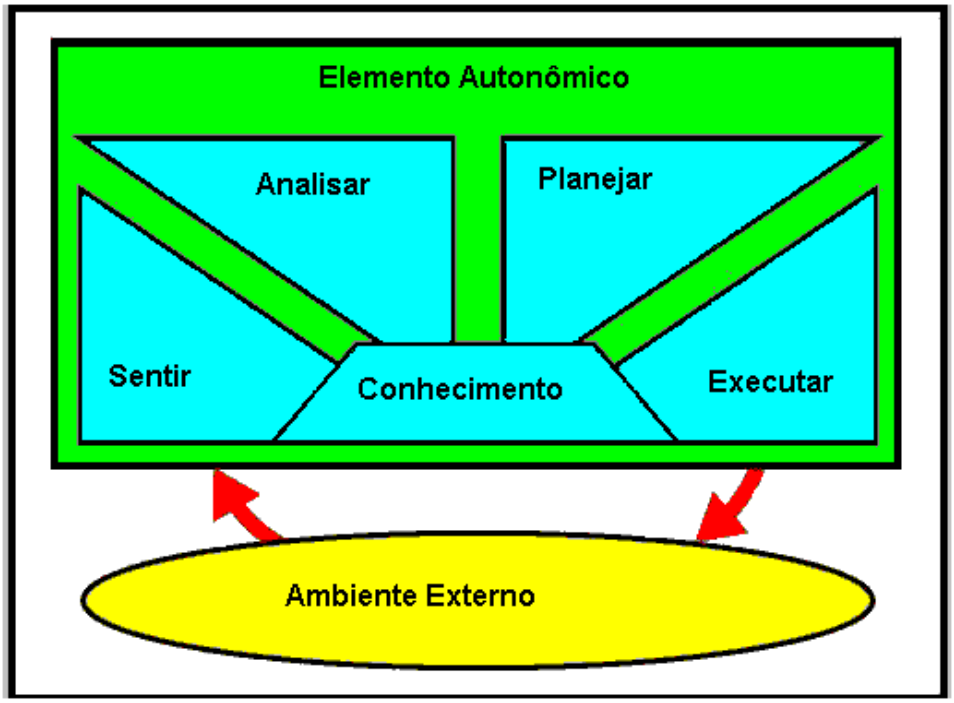

Figura 4 - Elemento Autonômico (adaptado de (Kephart & Chess, 2003))

Na Figura 5, baseado nos conceitos presentes em (IBM, 2006), pode-se verificar como é o relacionamento entre camadas de um elemento autonômico. As setas contínuas indicam os fluxos de dados e eventos entre os processos. A seta tracejada indica a transição do final do processo de volta ao início, o processo EXECUTAR pode influenciar indiretamente o processo MONITORAR. Isto pode ocorrer, quando os dados modificados na execução dispararem eventos que estejam sendo monitorados.

Assim, as camadas responsáveis pela monitoração detectam eventos e coletam dados dos ambientes. As camadas responsáveis pela análise recebem e analisam os

eventos e dados, buscando realizar a detecção de eventos compostos e identificar assinaturas de interesse. As camadas responsáveis pelo planejamento processam as regras armazenadas para realizar o tratamento dos dados recebidos. Finalmente, as camadas responsáveis pela execução, processam no ambiente as ações decididas na fase de planejamento.

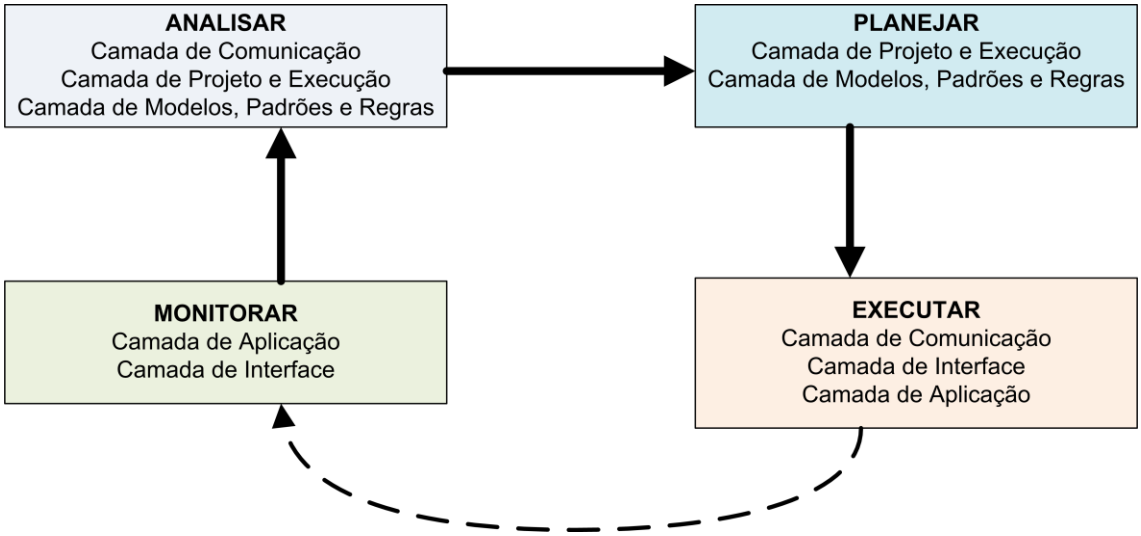

Figura 5 - Relacionamento entre as camadas da arquitetura e as funções de um elemento autonômico (Fonte: (Pinheiro *et al*, 2010a))

3.2. Arquitetura autonômica para manipulação de dados

A Figura 4 e Figura 5, apresentadas anteriormente, demonstram como é um elemento autonômico. Baseado nesses modelos e nos conceitos da Computação Autonômica, o grupo de Computação Autonômica da COPPE/UFRJ propôs a criação de uma arquitetura autonômica de manipulação de dados. Essa arquitetura pode ser baseada na modelagem de comportamento dos diversos componentes derivados do manipulador de dados que sejam disparados pela detecção de eventos. As camadas que compõem a arquitetura são definidas como:

- Camada de Modelos, Padrões e Regras: responsável por armazenar os modelos, padrões e regras, recuperando-os e atualizando-os quando necessário. Ela pode conter os modelos correspondentes aos padrões de eliminação de dados, detecção de eventos compostos e outros modelos e padrões de manipulação de dados;

- Camada de Projeto e Execução: responsável pela criação, manutenção e execução dos modelos e padrões. Ela permite que especialistas modelem os comportamentos dos manipuladores de dados. Ao mesmo tempo, essa camada deve permitir que esses modelos sejam executados através de ferramentas que interpretem e executem diretamente os modelos e padrões ou de implementações executáveis extraídas destes;

- Camada de Comunicação: responsável por fornecer uma interface de entrada e saída de mais alto nível para os modelos. Ela é composta de três componentes: Leitor de Dados, Leitor de Eventos e Gerador de Ações. O Leitor de Dados e o Leitor de Eventos são responsáveis por receber os dados e eventos, respectivamente, advindos das interfaces criadas para as bases de dados, aplicações e agentes. Esses leitores repassam os dados e eventos para a camada de Projeto e Execução. O Gerador de Ações é responsável por repassar as decisões advindas da Camada de Projeto e Execução para a Camada de Interface;

- Camada de Interface: fornece as diferentes formas de comunicação do ambiente externo com os componentes do arcabouço, por meio de interfaces apropriadas;

- Camada de Aplicação: contém as diferentes aplicações e agentes que manipulam dados de forma autonômica.

Ao longo do trabalho será possível verificar que as camadas dessa arquitetura estão presentes no modelo proposto. A Figura 6 ilustra essa arquitetura. É possível ver na figura diversos exemplos do que pode desempenhar o papel de cada camada.

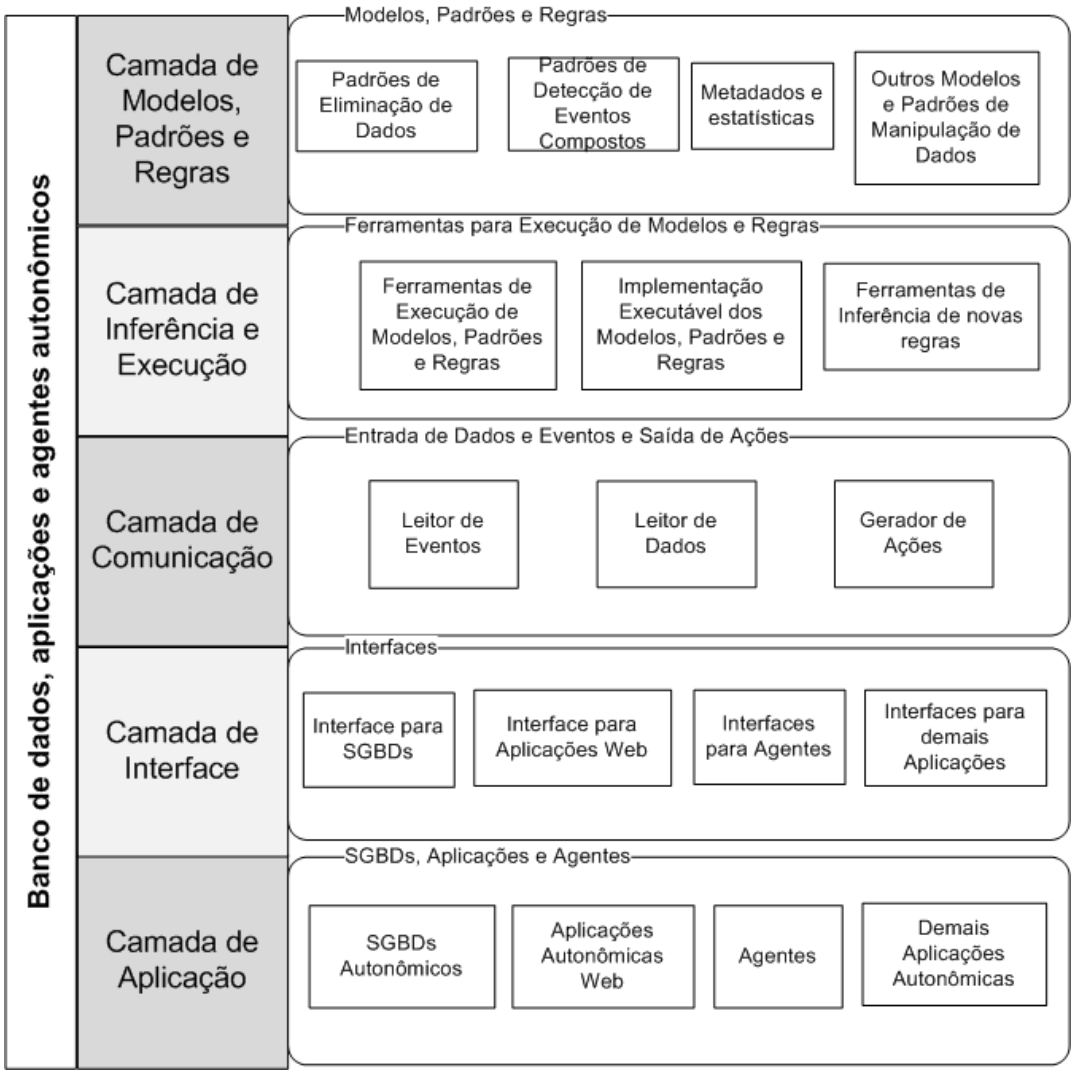

Figura 6 - Arquitetura Autonômica de Manipulação de Dados (fonte: (Pinheiro, 2010b))

A seguir, são mostrados alguns conceitos para identificar as propriedades de um sistema computacional autonômico. Sendo assim, será possível perceber como um sistema não inteligente pode ser convertido em uma aplicação autonômica.

3.3. Propriedades de um sistema autonômico

Antes de definir as propriedades de um sistema autonômico vamos falar mais uma vez sobre o sistema nervoso. Nosso corpo é todo interligado, porém cada parte funciona separadamente com interligações e com uma hierarquia de auto-governança. Dessa forma, um sistema autonômico pode ser definido como uma coleção de componentes autonômicos separados que interagem e se relacionam com os outros em

um alto nível de concordância, sempre pensando no bem maior do sistema, de modo a manter o equilíbrio do organismo. Mas, para os componentes serem autonômicos, estes necessitam possuir algumas características. Segundo PARASHAR & HARIRI (2005), existem 8 elementos-chave, ou características, para um sistema se tornar autonômico. São eles:

1 – Conhecer a si mesmo;

Desde que um sistema possa existir em muitos níveis, um sistema autonômico necessita conhecimento detalhado de seus componentes, seu status atual, capacidade, e todas as conexões como outros sistemas para governar a si próprio. Ele necessita conhecer a extensão de seus recursos, o que pode requisitar, o que pode ceder, o que pode ser compartilhado e o que deve ficar isolado.

Um sistema pode parecer algo simples quando significa uma sala cheia de computadores, ou até mesmo, quando significa uma centena de computadores em uma rede de trabalho de uma companhia.

Mas, se for imaginado que estes podem se interligar com os milhões de computadores da internet, eles passam a se tornar interdependentes. A complexidade ainda aumenta quando se pensa que além de computadores existem acessos via uma crescente quantidade de dispositivos (celulares, TVs, ferramentas inteligentes). Assim, já não se tem mais o termo sistema como algo simples.

Ademais, quando permitimos que todos esses dispositivos podem dividir tempo de processamento, armazenamento e outros recursos, além da possibilidade de utilização por "*leasing*" dos serviços computacionais, chegamos à situação que desafia qualquer definição de um simples sistema.

Mas, é justamente essa percepção de todo o sistema, em alto nível, que a Computação Autonômica precisa. Um sistema não pode monitorar o que não conhece, ou controlar pontos específicos que ainda são desconhecidos.

Para construir essa habilidade em um sistema computacional, políticas claramente definidas e incorporadas em agentes de softwares adaptáveis devem governar a definição do sistema, e de suas interações com as entradas e saídas de dados. Este sistema necessita também da capacidade de unir-se automaticamente com outros

sistemas para formar novos sistemas únicos, mesmo que temporariamente; e se separar, caso necessário.

2 – Deve se configurar e reconfigurar sobre variações de condições imprevistas;

A configuração de um sistema deve ocorrer automaticamente, assim como devem existir ajustes dinâmicos para que essas configurações possam melhor manipular mudanças no ambiente.

Dadas possíveis permutações em um sistema complexo, a configuração pode ser difícil e consumir tempo. Para se ter idéia, alguns servidores sozinhos têm centenas de configurações alternativas. Dessa forma, administradores de sistemas jamais serão capazes de fazer reconfigurações dinâmicas, na mesma velocidade que estas novas configurações apareçam.

Para habilitar a configuração automática, um sistema necessita criar múltiplas imagens críticas do software, assim como faz o sistema operacional (um tipo de clone de *software*), e realocar esses recursos (memória, armazenamento, banda de rede), quando necessário. Se for um sistema distribuído, necessitará alavancar suas múltiplas imagens e fazer um backup para recuperar-se de falhas em partes localizadas de sua rede. Algoritmos adaptativos sendo executado sobre tais sistemas poderiam aprender as melhores configurações para atingir os níveis de desempenho desejado.

3 – Um sistema autonômico nunca deve estar satisfeito com a situação atual, deve sempre procurar otimizações

O sistema irá monitorar suas partes constituintes e o foco do fluxo de trabalho para conseguir objetivos predeterminados, assim como faz um regente à sua orquestra em seus gestos dinâmicos para conseguir uma interpretação musical particular.

Estes esforços o otimizam de uma maneira que ele estará apto a tratar diferentes níveis de complexidade e os freqüentes conflitos de E/S demandados pelo negócio, seus consumidores, fornecedores e empregados. E, desde que as prioridades que regem essas demandas mudem constantemente, apenas a constante auto-otimização irá satisfazê-los.

Auto-otimização é também a chave para habilitar a disponibilidade ubíqua[10] de *e-sourcing*[11], ou para a entrega de serviços computacionais de uma maneira utilizável. E-*sourcing* promete custos previsíveis e acesso simplificado para consumidores de TI. Para fornecedores desses serviços computacionais, entretanto, a promessa de entrega de qualidade de serviço para seus consumidores necessita não apenas da priorização do trabalho e recursos dos sistemas, mas também dos recursos externos suplementares (como armazenamento subcontratado e ciclos extras de processamento). Isto é similar a maneira com que hoje as companhias de energia elétrica compram e vendem a energia que obtiveram em excesso no mercado de eletricidade.

Mas, para ser apto a otimizar a si mesmo, um sistema necessita de mecanismos avançados de *feedback* que permitam monitorar suas métricas e tomar ações apropriadas. Apesar do sistema de *feedback* ser uma técnica antiga, são necessárias novas abordagens para aplicar isso na computação. É necessário responder a questões como: quando um sistema toma decisões, qual é o atraso que pode ser aceito entre duas ações e quais são os seus efeitos? Adicionalmente, como tudo isso afeta toda a estabilidade do sistema?

Inovações na aplicação da teoria de controle para a computação devem ocorrer em conjunto com novas abordagens para toda a arquitetura do sistema. Algoritmos que tomam decisões de controle devem ter acesso a métricas internas. Mais importante ainda, todos os componentes de um sistema autonômico, não importa o quão diversos sejam, devem ser controlados de uma maneira unificada.

4 – Um sistema deve se curar, recuperando-se de falhas críticas.

Um sistema deve ser capaz de descobrir problemas ou problemas em potencial, então encontrar uma maneira alternativa de usar os seus recursos ou reconfigurar-se para manter-se funcionando corretamente. Certos tipos de "cura" têm sido estudados na área da computação já há algum tempo. Por exemplo, a checagem de erros e correções, em 50 anos de tecnologia, tem habilitado a transmissão de dados sobre a internet de maneira consideravelmente confiável, e a redundância de armazenamento de sistemas,

[10] *Computação Ubíqua* é o termo utilizado para referenciar a integração da *computação* móvel e onipresente com o espaço físico. É um tipo de computação distribuída realizada por dispositivos de computação que atuam de forma discreta nos ambientes onde estão implantados

[11] e-sourcing, também conhecido como *reverse auction* é uma plataforma de negócios online que trabalha com o seguinte conceito: a origem da necessidade de compra pode (e deve) partir do consumidor, que tem em seu poder a tomada de decisão quanto à aquisição de um determinado produto ou serviço.

como o RAID, tem permitido a recuperação de dados, mesmo quando uma parte do armazenamento falha.

Mas o crescimento da complexidade de E/S dos sistemas tem tornado as causas das falhas cada vez mais difíceis de serem descobertas. É possível ver isso em nossos computadores pessoais que apresentam falhas difíceis de serem tratadas e que exigem como solução a reinicialização do computador.

Nos sistemas mais complexos, identificar as causas de falha requer análise da causa raiz. Mas, desde que restaurar serviços e minimizar interrupções são as primeiras preocupações, uma abordagem orientada a ações (determinando que ações imediatas devam ser tomadas para deixar um sistema disponível) devem ser tomadas de maneira autonômica.

Inicialmente, a "cura" feita por um sistema autonômico deverá seguir as regras geradas por especialistas humanos. Mas deve-se incorporar mais inteligência aos sistemas computacionais, estes devem descobrir novas regras que os ajudem a ir ao encontro de seus objetivos específicos.

5 – Um sistema deve se proteger.

Um mundo virtual não é menos perigoso que o mundo real. Dessa forma, um sistema autonômico deve se auto-proteger. Ele deve detectar e identificar ameaças, e proteger a si mesmo contra a diversidade de ataques de modo a manter-se íntegro e seguro.

Antes da internet, computadores operavam como ilhas. Então, era fácil protegê-los de ataques proporcionados por vírus ou outros softwares maliciosos. Como a forma de compartilhar dados e programas eram através de disquetes, demorava cerca de semanas ou meses para um vírus se espalhar.

A conectividade das redes mundiais mudou e agora ataques podem vir de qualquer lugar. Vírus se espalham rapidamente (em segundos), pois foram projetados para serem enviados automaticamente para outros usuários. O prejuízo potencial para uma empresa é enorme.

Mais que simplesmente responder a falhas de componentes, ou executar checagens por sintomas, um sistema autonômico necessita ficar em alerta, antecipar ameaças, e tomar as decisões necessárias. Tais respostas necessitam tratar dois tipos de ataque: vírus e intrusão de sistemas por hackers.

Tendo como base o sistema imunológico humano, um sistema digital imune pode detectar códigos maliciosos, enviá-lo automaticamente para um centro de análises, e distribuir a cura para o sistema. Todo o processo acontece sem que nenhum usuário esteja ciente que isso esteja em processo.

Para lidar como ataques maliciosos de hackers, os sistemas de intrusão devem automaticamente detectar e alertar os administradores de sistema de ataques. Atualmente, especialistas de segurança de sistemas devem examinar o problema, analisá-lo e então reparar o sistema. Esse processo deve ser automatizado, pois não haverá especialistas suficientes para lidar com cada incidente.

6 – Um sistema deve conhecer seu ambiente e o contexto ao redor de suas atividades

Isso é quase uma auto-otimização relacionado ao ambiente exterior: um sistema autonômico irá encontrar e gerar regras para melhor interagir com os sistemas vizinhos. Assim, são verificados os recursos disponíveis e subutilizados, e pode-se negociar o uso destes por outros sistemas, facilitando assim a vida de ambos. Essa sensibilidade a contexto inclui melhorias no serviço baseado no conhecimento da transação.

É como uma habilidade do sistema autonômico de manter a confiabilidade, sob uma vasta cobertura, antecipando circunstâncias e combinações de circunstâncias. Porém, mais significantemente, isso irá habilitar o sistema a fornecer informação útil ao invés de informações confusas ou desnecessárias. Por exemplo, mostrar todos os dados de uma sofisticada página *web* seria um exagero e um desperdício, se o usuário estiver conectado via celular e que queira saber apenas o banco mais próximo.

Sistemas autonômicos devem ser capazes de descrever a si mesmo e seus recursos disponíveis para outros sistemas, e esses devem também ser capazes de descobrir automaticamente outros dispositivos no ambiente. Esforços recentes para compartilhar recursos dos supercomputadores, via grade, irão indubitavelmente contribuir com a tecnologia necessária para a habilidade da percepção do sistema. Avanços também serão necessários para fazerem com que os sistemas selecionem ações baseada nas escolhas dos usuários, através de algoritmos que permitam que um sistema determine a melhor resposta em um dado contexto.

7 – Um sistema autonômico não deve existir em um ambiente fechado.

Na natureza, vários tipos de organismos coexistem e dependem um dos outros para sobreviver, nenhum é fechado em seu mundo sem interagir com outros seres. Da mesma forma devem ser os sistemas computacionais. Enquanto independente em sua habilidade de administrar a si mesmo, um sistema autonômico deve funcionar em um mundo heterogêneo e preferencialmente implementar padrões abertos. Segundo (IBM, 2009) um sistema autonômico não deve ser uma solução proprietária de modo a poder interagir melhor com os diversos sistemas.

Entre os diversos motivos que impulsionam isso, está o fato de que um sistema que exista com independência de outros sistemas é algo quase impossível atualmente. Negócios se conectam a fornecedores, clientes e parceiros. Pessoas conectam-se a seus bancos, agentes de viagem e suas lojas. E, além disso, novas tecnologias estão sempre surgindo de modo a integrar cada vez mais os sistemas.

Com isso, para um sistema poder interagir com o mundo, ele necessita de padrões e protocolos abertos, meios padrões de identificação, comunicação e negociação.

8 – Um sistema deve antecipar otimizações de recursos necessários, escondendo a complexidade do usuário.

Este é o último objetivo da Computação Autonômica, organizar os recursos de TI de modo a minimizar a diferença entre os objetivos do negócio e o que foi implementado em TI.

No corpo humano, quando enfrentamos um perigo potencial ou situação urgente, nosso sistema autonômico se antecipa ao perigo potencial antes que fiquemos sabendo do mesmo. Ou seja, nosso corpo otimiza-se para enfrentar futuras situações. Um exemplo disso são glândulas supra-renais que liberam adrenalina, um hormônio que sobrecarrega a habilidade dos nossos músculos de contrair, de modo a aumentar a quantidade de batidas do coração e a pressão sanguínea. O resultado é que nosso corpo fica preparado para uma determinada ação, como por exemplo, correr, mas nossa consciência permanece sem saber disso. Se decidirmos correr, já estaremos preparados para isso.

Um Sistema Autonômico deve agir da mesma forma, deve antecipar-se a situações de risco e preparar-se para enfrentá-las.

Com isso chegamos ao final das características necessárias para um sistema ser considerado autonômico. De todas as características apresentadas, quatro são as ações que um sistema deve tomar para se auto-administrar, quais sejam: auto-configuração, auto-reparo, auto-otimização e auto-detecção. Essas ações podem ser denominadas regras self.* (do termo original em inglês *self-Configuration*, *self-Healing*, *self-Optimization* e *self-Protection)*, ou regras CHOP (IBM, 2007). Estas regras descrevem atributos que tornam um sistema autonômico auto-gerenciável. Doravante, quando essas quatro regras forem discutidas, será usado o termo regras CHOP.

A IBM (IBM, 2006) ao propor a Computação Autonômica também definiu cinco níveis de maturidade dos sistemas autonômicos, em ordem decrescente da necessidade de intervenção humana. São eles:

- Nível Básico (manual) - múltiplas fontes de dados do sistema são gerenciadas independentemente. O pessoal especializado agrupa os dados coletados das diferentes fontes, tomam as decisões e realizam as ações necessárias;

- Nível Gerenciado - tecnologia de gerenciamento de sistemas para consolidação dos dados. Os especialistas da área, baseados nos dados coletados, tomam as decisões e realizam as ações necessárias;

- Nível Preditivo - sistemas monitoram, correlacionam e recomendam as ações. Pessoal especializado gerencia o desempenho dos sistemas;

- Nível Adaptativo - sistemas monitoram, correlacionam e executam as ações. Pessoal especializado aprova e inicia as ações;

- Nível Totalmente Autonômico – este nível permite a integração dinâmica de componentes, sendo o gerenciamento feito por regras/políticas de negócio. Usuários focam na criação e alteração das regras que descrevem as necessidades do negócio.

Apesar da maioria dos sistemas atuais só alcançarem até o terceiro nível (preditivo), muitas pesquisas propõe novas aplicações em áreas específicas, aumentando a complexidade dos sistemas auto-gerenciáveis em várias aplicações (OLIVEIRA *et al*., 2006)

3.4. Trabalhos relacionados à Computação Autonômica

Tendo em vista que este trabalho lida com Computação Autonômica voltada para dados e modelos de armazenamento, é de suma importância destacar quais são os principais trabalhos da área relacionados a esse assunto. Com isso, as principais pesquisas sobre o assunto são detalhadas a seguir.

Pode-se dizer que o primeiro trabalho a tratar Computação Autonômica com dados foi o de CHANBERLING **(**1998**)** com o DB2. O DB2 UDB é um Sistema de Gerenciamento de Banco de Dados que a IBM lançou em 1983. Historicamente foi concorrente direto da Oracle e do Microsoft SQLServer. (CHAMBERLIN, 1998)

Já em 2002, Segundo LIGHTSTONE *et al* **(**2002**)**, a IBM introduziu elementos autonômicos no DB2, dando a este capacidade de se ajustar a diversas situações. Algumas funções com essas características já existiam, como por exemplo, o otimizador de consultas. Esta é uma função autonômica mais básica, que não necessita de dicas dos usuários e combina regras de reescrita de consultas para transformar as consultas escritas pelos usuários em consultas padronizadas. Outras funções já presentes anteriormente incluem o orientador de configuração e a reorganização automática de índices. A primeira função ajusta *cache* e alocação de memória para que o sistema possua o melhor desempenho possível, e a segunda diminui fragmentação no armazenamento dos dados.

Entre as novas propostas foi incluída a capacidade de avaliação da saúde do sistema, que fornece um monitoramento constante da saúde do sistema que o avalia através de estatísticas e comparação de métricas do sistema contra políticas padrões definidas. Quando algum perigo é detectado, o sistema avisa ao administrador para que este tome a melhor providência.

Outra proposta incluiu a melhoria do otimizador de consultas para que este possa aprender e melhorar sua otimização, tendo como base resultados das otimizações anteriores.

Já no ano de 2003 tivemos o primeiro sistema autonômico que trabalhava com dados. O trabalho de CLARKE *et al* **(**2003**)** descreve uma arquitetura proposta para recuperação de informação em um ambiente distribuído. O foco do trabalho especificamente é sobre uma camada de gerência de persistência que tem como objetivo

a auto-configuração dos *peers*[12] de modo que eles mesmos possam fazer o balanceamento de cargas. Além disso, esse sistema pode detectar a adição de um novo *peer*, ou mesmo a retirada ou a falha de um *peer* e redistribuição da coleção sobre os *peer*s, assim como o processamento de consultas entre eles.

Ainda em 2003 tivemos o trabalho de RHEA *et al* **(**2003**)** o OceanStore. OceanStore é uma central de armazenamento de dados (*data center*), baseado na internet e projetado para ter escalabilidade incremental, compartilhamento seguro e longa durabilidade. A grande questão do projeto é fornecer um sistema com interface de armazenamento expressiva para usuários, enquanto garante alta durabilidade sobre um sistema não confiável (internet) e em constante mudança de base (usuários que entram e saem do sistema).

O projeto é baseado em 2 níveis, sendo que o primeiro, que forma a base do sistema, seria um conjunto de *peers* que se organizam e fazem a gerência e a organização dos dados. Já o segundo nível, o mais funcional e maior, é um conjunto de *peers* que constituem o armazenamento.

A Computação Autonômica entra em dois pontos críticos nesse projeto que são:
1 - Tolerância a falhas: Como os sistemas eventualmente falham, é necessário ter planos de contingência para contornar situações críticas. O sistema reorganiza os *peers*, caso alguns desses falhem, de modo a não comprometer as ações dos usuários.
2 – Percepção de mudanças no sistema: O sistema eventualmente muda, assim como todas as arquiteturas ponto-a-ponto, novos *peer*s entram e outros saem a todo o momento no sistema e o sistema deve se configurar para essas novas situações. A intenção do projeto é que toda essa configuração ocorra de maneira automática e ótima, de modo que a mudança na configuração dos *peer*s não prejudique o usuário.

Em 2003 também tivemos outro projeto de central de armazenamento de dados que é o *Self***-Storage*. Segundo GREGORY & GANGER **(**2003**)**, a idéia da pesquisa é reduzir a carga administrativa enfrentada pelos administradores da central de armazenamento de dados. O artigo foca em *storage bricks*, que são basicamente computadores conectados a internet com médio poder de armazenamento (5 TB). Neste caso, cada computador é um agente inteligente que se auto-organiza e faz balanceamento de cargas, além de acelerar a velocidade de comunicação da rede, codificar e decodificar dados deixando o sistema confiável.

[12] Em um sistema ponto-a-ponto(*peer-to-peer*), *peer* é um computador que compartilhe recursos com outros computadores em um ambiente distribuído.

Em 2004, tivemos mais um artigo sobre armazenamento distribuído. ZHAND *et al* **(**2004**)** propôs um framework para que toda a configuração, em relação à otimização da distribuição, seja feita de forma autonômica. O objetivo do trabalho é fazer com que o framework tenha a melhor relação custo x benefício possível em replicação e distribuição de dados

Em 2005 e 2007 tivemos mais trabalhos relacionados a centrais de armazenamentos, BENNANI & MENASCE **(**2005**)** propõe um método estatístico com modelo de filas para melhor balanceamento de cargas enquanto que WANG. *et al* **(**2007**)** propõe uma arquitetura para virtualizar terceirização de centrais de armazenamento, para que um administrador não precise saber que existam diversas centrais.

Para finalizar, em 2009 tivemos o trabalho de NGUYEN *et al* **(**2009**)** para distribuição de dados em grade. Este gerencia o problema de diferentes políticas de acesso quando se tem computadores de vários locais, geograficamente distantes

Como consideração final a ser feita sobre esses trabalhos é possível citar algumas falhas dessas propostas, ou mesmo citar algumas características que seriam desejáveis, mas que estas não possuem. Uma característica a ser destacada nos trabalhos de GREGORY & GANGER **(**2003**)** e ZHANG *et al* **(**2004**)** é que, em ambos os casos, os dados não são necessariamente autonômicos, os sistemas é que são autonômicos e têm como objetivo fazer a organização dos dados. Dessa forma, se os dados forem transportados para outro sistema, não terão as propriedades autonômicas, pois os dados serão organizados somente naquele sistema específico.

Outra desvantagem é que a implementação desses sistemas fica muito dependente dos tipos de dados, se os dados forem mudados a implementação terá que ser mudada.

Outro ponto ainda a ressaltar é que os dados não levam parte da semântica do sistema com eles. Regras de negócio são voláteis e mudam constantemente de acordo com a necessidade do negócio, dessa forma, um sistema, ou uma parte do sistema, pode ficar desatualizado com relação a essas mudanças. Ou ainda, se a semântica dos dados mudarem, e se o sistema não for alterado com essas mudanças, o sistema irá interpretá-los de maneira errônea. Se os dados carregassem a semântica do sistema com eles, tudo isso poderia ser evitado sem a necessidade de alterar o sistema e este iria possuir realmente Dados Autonômicos. Um exemplo de como isso pode ser feito pode ser visualizado na aplicação proposta desse trabalho.

3.5. Ontologias

Ontologia é um tema que tem sido estudado em diversas áreas como: Filosofia, Linguagem e Cognição, Ciência da Informação e Ciência da Computação. Diferentes definições podem ser encontradas para a ontologia, dependendo da área em que esse conceito esteja sendo utilizado. Mesmo dentro de uma mesma área podem ser encontradas diferentes definições. OSCAR (2003) ressalta que muitas definições têm sido criadas para ontologias e que tais definições têm mudado e evoluído ao longo do tempo. Algumas definições são apresentadas a seguir:

Na área de Sistemas de Informação: é definida como um conjunto de conceitos e termos que podem ser usados para descrever alguma área do conhecimento ou construir uma representação para o conhecimento (SWATOUT, 1999).

Na área de Inteligência Artificial: GUARINO (1997) define ontologia como uma caracterização axiomática do significado do vocabulário lógico. Já para SOWA (2000), a ontologia define os tipos de coisas que existem ou podem existir em um mesmo domínio.

Na área de Banco de Dados: conhecimento genérico que pode ser reusado em aplicações de tipos diferentes (SPYNS *et al*., 2002).

Berners-Lee, considerado o pai da Web, prefere uma definição mais técnica: "ontologia é um documento ou arquivo que formalmente define os relacionamentos entre termos". Nota-se, a partir desta definição, a preocupação em relacionar ontologias com o problema prático de uso de ontologias na Web (BERNERS-LEE *et al.* 2001).

OSCAR *et al* (2003) comenta que a comunidade de ontologia distingue dois tipos principais de ontologias: leves e pesadas. As ontologias leves incluem conceitos, taxonomia de conceitos, relações entre conceitos e as propriedades que os descrevem. Neste sentido, os tesauros podem ser considerados ontologias leves. Já as ontologias pesadas acrescentam axiomas e restrições às ontologias leves.

NOY (2001) esclarece que uma ontologia juntamente com um conjunto de instâncias de conceitos constitui uma base de conhecimento. No entanto, afirma que uma linha tênue separa estes conceitos, delimitando os pontos onde uma ontologia acaba e onde uma base de conhecimento começa. É identificado o processo de construção de uma ontologia como sendo ortogonal ao processo de construção do conhecimento. Segundo esse autor, ontologias constituem o elo que liga os subprocessos de conhecimento, permitindo que seus itens sejam ligados, combinados e

usados. As ontologias fornecem a infra-estrutura que, de forma coerente, une diferentes subprocessos que guiam o desenvolvimento e uso de aplicações de gerenciamento do conhecimento. No trabalho de MAEDCHE (2002) encontra-se um conjunto de propriedades que objetivam caracterizar as diferenças entre base de conhecimento e ontologia. Estas idéias podem ser visualizadas na Tabela 1.

Tabela 1 - Diferenças entre ontologia e base de conhecimento

	Ontologia	Base de Conhecimento
Possui Conjunto de Declarações Lógicas	Sim	Sim
Tipo de Teoria	Teoria Geral	Teoria Circunstancial
Declarações são predominantemente	Intencionais	Extensões de idéias
Construção da Teoria	No momento da construção e manutenção.	Baseada em mudança contínua.

BÉZIVIN (1998) enfatiza que as principais propriedades de uma ontologia são o compartilhamento e filtragem. O compartilhamento se baseia em um acordo sobre o entendimento comum de um conceito, ou seja, o uso de uma ontologia comum entre dois ou mais agentes diferentes. A filtragem é vista sob o ponto de vista da abstração. Geralmente as pessoas consideram modelos da realidade. Estes modelos, por definição, expressam somente uma parte da realidade. A ontologia define o que poderia ser extraído dessa realidade (características mais relevantes para aquele domínio do problema) de modo a se construir um modelo para aquele sistema.

Este autor comenta ainda que uma ontologia pode conter informações de diferentes naturezas e, geralmente, possui três tipos de informação:

- **Terminológica**: conjunto de conceitos e relações entre esses conceitos;
- **Axiomática**: regras de inferência aplicadas aos conceitos e relações;
- **Pragmática**: informações adicionais sobre os conceitos, como por exemplo, a forma de imprimir diferentes conceitos e relações de uma ontologia.

GUARINO (1997) classifica as ontologias em duas dimensões: nível de detalhe e nível de dependência de uma tarefa particular ou ponto de vista. A primeira

classificação mostra o nível de profundidade na especificação de um vocabulário. Podem existir ontologias com termos genéricos, expressando o consenso de uma maioria de pessoas que usam este vocabulário e ontologias que detalham tais termos, através de termos específicos. No segundo nível existem quatro tipos distintos: ontologia de alto-nível, ontologia de domínio, ontologia de tarefa e ontologia de aplicação. Isto pode ser visto na Figura 7.

Pode-se entender uma ontologia de alto-nível como uma descrição de conceitos genéricos, tais como espaço, tempo, objeto, ação, etc., que são conceitos independentes do domínio. Uma ontologia de domínio e uma ontologia de tarefa procuram descrever uma conceitualização para um domínio genérico (automóveis, medicina, conferências, por exemplo) ou uma tarefa genérica (diagnósticos, vendas, leitura de artigos, por exemplo), especializando conceitos da ontologia de nível superior. Uma ontologia de aplicação especifica conceitos da ontologia de domínio e da ontologia de tarefa para certa atividade dentro desses domínios. Define regras a serem seguidas por conceitos do domínio quando certa tarefa é realizada (avaliação de um artigo para uma conferência, por exemplo).

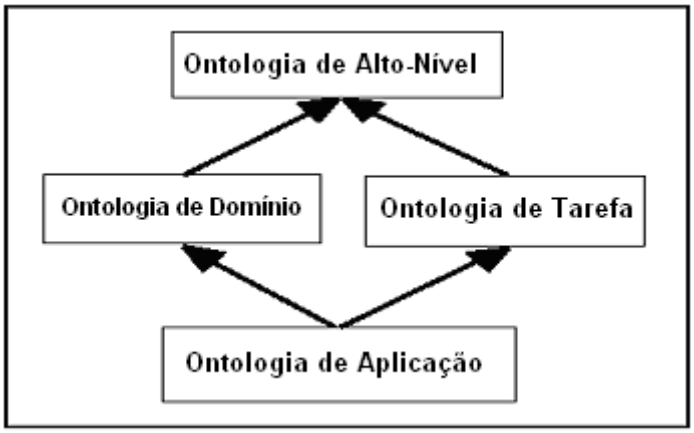

Figura 7 - Classificação de ontologias (fonte: (GUARINO, 1997)).

Outro ponto importante diz respeito às características desejáveis em uma ontologia. HWANG (1999) comenta que é impossível construir uma ontologia suficientemente rica para todos os fins e domínios, e cita cinco características desejáveis em uma ontologia:

- **Aberta e Dinâmica**: para se ajustar às mudanças e novos desenvolvimentos em um domínio, uma ontologia deve ser aberta e dinâmica, tanto em termos de seus algoritmos quanto da sua estrutura. Os sistemas deveriam ser capazes de "criar" conceitos com o mínimo de ajuda humana;
- **Escalável e Interoperável**: deve ser facilmente escalável, considerando um domínio amplo e adaptável a novos requisitos. Deve também ser possível integrar várias ontologias em uma nova ontologia quando o tratamento de diferentes vocabulários conceituais é requerido;
- **Fácil Manutenção**: deve ter uma estrutura simples, limpa e modular para ser de fácil entendimento pelas pessoas, o que facilita a sua manutenção;
- **Semanticamente Consistente**: o domínio a ser abordado deve guiar a escolha dos termos escolhidos;
- **Independente do Contexto**: não deve conter termos muito específicos para não tornar complexa a associação com as fontes de dados e futuras integrações com outras ontologias.

USCHOLD (1996) destaca três categorias principais de uso para ontologias: **comunicação** (entre pessoas e organizações), **interoperabilidade** (entre sistemas) e a **construção de sistemas** (especificação, confiabilidade e reuso de componentes). Ele também classifica o grau de formalidade usado na definição dos termos. Estes podem ir de altamente informal (linguagem natural), passando pela semi-informal (linguagem natural restrita e estruturada de forma a reduzir ambigüidades) e semiformal (linguagem artificial definida formalmente) até chegar ao rigorosamente formal (termos definidos meticulosamente através de semântica formal, teoremas e provas de propriedades, tais como validade e completeza).

3.5.1. Tecnologias para representação e uso de Ontologias

Existem diversas tecnologias disponíveis pela W3C para representar uma ontologia. Atualmente, pode-se destacar:

- O *Extensible Markup Language* (XML, 2009), que provê a sintaxe para documentos estruturados, mas sem qualquer imposição de restrições semânticas ao significado dos documentos;
- O *Extensible Markup Language Schema* ("XML Schema," 2001) que restringe a estrutura de um documento XML;
- O *Resource Description Framework* (RDF, 1999) que é um modelo de dados para descrever recursos e as relações entre eles, também baseado na sintaxe XML;
- O *Resource Description Framework Schema* (RDF SCHEMA, 1999), um vocabulário para permitir a descrição de propriedades e classes, embora com alguma expressividade limitada.
- O *Ontology Web Language* (OWL), originado a partir da linguagem DAML-OIL, que acrescenta vocabulário e definições mais formais para a descrição de ontologias, tais como relações entre classes, cardinalidade, igualdade, tipos de propriedades complexos, etc. A OWL é proposta pelo W3C como um padrão de linguagem ontológica e atualmente apresenta o *status* de recomendação (OWL, 2003). Por ser uma das tecnologias usadas no trabalho esta é mais bem detalhada a seguir:

3.5.2. OWL

O OWL (*Ontology Web Language*) é uma linguagem de marcação semântica para publicação e compartilhamento de ontologias da internet. É a linguagem recomendada atualmente pelo W3C (http://www.w3.org/News/2003#item203). (OWL, 2003)

Ela é derivada da linguagem DAML-OIL e construída sobre o RDF. Sua sintaxe tem poucas diferenças em relação à sintaxe DAML-OIL, podendo se destacar a remoção da restrição do número de qualificadores e a habilidade de declarar diretamente que propriedades podem ser simétricas.

Assim como a DAML-OIL, a sintaxe da linguagem OWL consiste de objetos do tipo: *headers*, *class elements*, *property elements* e *instances*. A OWL é subdividida em três linguagens: OWL Lite, OWL DL, e OWL Full. A OWL Lite é um subconjunto da OWL DL, que é subconjunto da OWL Full, e qualquer OWL Lite é uma ontologia OWL DL. Qualquer ontologia OWL DL é uma ontologia OWL Full.(OWL, 2003)

A OWL Lite é específica para necessidades básicas dos usuários, com restrições simples. Cardinalidade é suportada apenas com valores 0 ou 1. A OWL Lite é a mais simples a ser implementada, e pode ser uma boa alternativa para migração de tesauros e taxonomias.

A OWL DL e OWL Full têm o mesmo vocabulário (um superconjunto da OWL Lite), mas diferem nas restrições à linguagem. A OWL DL impõe a separação de tipos (uma classe não pode ser uma propriedade ou uma instância, e uma propriedade não pode ser uma classe ou uma instância). Portanto, restrições não podem ser aplicadas aos elementos da própria linguagem. Já na OWL Full, essas flexibilidades são permitidas. Outra diferença básica é que na OWL DL tem-se a distinção entre tipos de dados e objetos.

As propriedades são ou *ObjectProperties* ou *DatatypeProperties*. A intenção da definição da OWL DL é deixá-la o mais próximo de uma lógica de descrição para o aproveitamento de todas as ferramentas e implementação relacionadas a esse campo de pesquisa.

3.5.3. Protégé

O Protégé foi utilizado durante este trabalho para a criação da ontologia. É um projeto desenvolvido na Universidade de Standford, Estados Unidos, pelo Departamento de Informática Médica da Escola de Medicina (SMI - *Stanford Medical Informatics*). Tem por objetivo permitir a interoperabilidade com outros sistemas de representação do conhecimento, além de ser extensível.

É um ambiente de edição de base de conhecimento compatível com OKBC (*Open Knowledge-Base Connectivity protocol*), protocolo que facilita a interoperabilidade por prover uma API (*Application Program Interface*) que serve como uma interface de consulta e construção comum para sistemas baseados em *frames*.

O Protégé usa a linguagem PAL (Protégé *Axiom Language*), baseada em lógica de primeira ordem, que é uma variante da linguagem KIF (*Knowledge Interface Format*). A linguagem KIF foi desenvolvida em 1990 como uma sintaxe padrão para lógica de primeira ordem (“PROTÉGÉ. User's Guide,” 2000).

Uma ontologia Protégé consiste de:

- Classes: são conceitos no domínio de discurso, que constituem uma hierarquia taxonômica;
- *Slots*: descrevem propriedades de classes e instâncias;
- *Facets*: descrevem propriedades de *slots*, sendo uma forma de especificar restrições (*constraints*) nos valores de *slots*;
- Axiomas: especificam regras adicionais.

Apesar de possuir sua linguagem própria, o Protégé ainda permite construir ontologias usando outras linguagens como a OWL. Além disso, permite a importação e exportação de representações em formato texto, RDF *Schema* e bases de dados através do JDBC, usando a arquitetura de metaclasse (extensão do OKBC) que permite o uso de modelos de conhecimento diferentes do existente no Protégé. Metaclasse (classe cujas instâncias são também classes) é um modelo (*template*) usado para definir novas classes em uma ontologia. Além disso, permite a especificação de herança múltipla e classes abstratas.

Uma característica importante do Protégé é em relação a possibilidade de utilização de suas funcionalidades em um programa Java através de sua API. Importando algumas bibliotecas específicas do Protégé é possível manipular um arquivo OWL com os mesmos recursos presentes no Protégé.

3.5.4. Inferências

As tentativas de construir grandes ontologias têm sido dificultadas pela falta de definições claras das ontologias. Entre os problemas temos inconsistências e contradições que usualmente ocorrem, além da falta de mecanismos para identificar relações que são conseqüências lógicas de outras. OWL tem recursos para tentar melhorar esse aspecto através de Lógicas de Descrição (*Description Logic*).

Segundo BAADER *et al* (2005), Lógica de Descrição é uma família de formalismos baseados em conceitos para representação de conhecimento. O nome Lógica de Descrição refere-se, por um lado, a semântica baseada em lógica de primeira ordem, e por outro lado, ao conceito de descrições para descrever um domínio. Esta descreve um domínio em termos de: conceitos (classes, tipos), que são conjuntos de objetos com características em comum; papeis (propriedades, relações) e indivíduos (objetos, instâncias) que são os elementos individualizáveis do universo de discurso. A

sub-linguagem de OWL que foi feita para suportar a lógica de descrições é o OWL-DL, mas como OWL-FULL é a versão mais abrangente, então esta também suporta.

As inferências que podem ser construídas com o OWL-DL são constituídas de um conjunto de axiomas para declaração de fatos sobre conceitos, propriedades e indivíduos. Porém, as regras de inferência são restritas a apenas composições entre hierarquias de classes e propriedades, não provendo construtores de composição. É Impossível capturar relacionamentos entre composição de propriedades. Segundo IAN HORROCKS (2005), embora a OWL seja uma linguagem rica em conjunto de construtores de classes, esta fornece uma linguagem fraca para relacionar propriedades.

Para exemplificar imagine que tenhamos uma propriedade temPai, que identifica qual individuo é pai de outro, e que queremos definir uma nova propriedade temAvô, a partir dessa propriedade. Considerando três indivíduos A,B e C; se A temPai B e B temPai C, a partir dessas informações podemos inferir que A temAvó C. A Figura 8 ilustra esse exemplo.

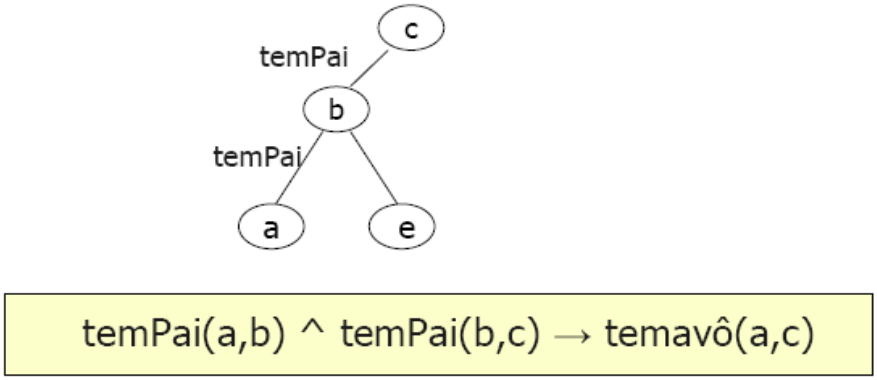

Figura 8 – Representação de uma inferência possível em OWL (fonte:(MOURA, 2006))

Porém, se quisermos ser um pouco mais ousados e definir outras propriedades familiares esbarraremos na limitação de capturar relacionamentos entre composição de propriedades. Imagine que tenhamos agora a propriedade temIrmão e temPai, e que, baseado na composição dessas duas propriedades, quiséssemos construir uma outra propriedade temTio. Considerando três indivíduos A,B e C sendo que A temPai B e B temIrmão C, fazendo uma analogia ao exemplo da Figura 8, poderíamos inferir que A temTio C. Porém, essa inferência não será possível, pois A e C estão em hierarquias diferentes. A Figura 9 ilustra a inferência que gostaríamos de fazer.

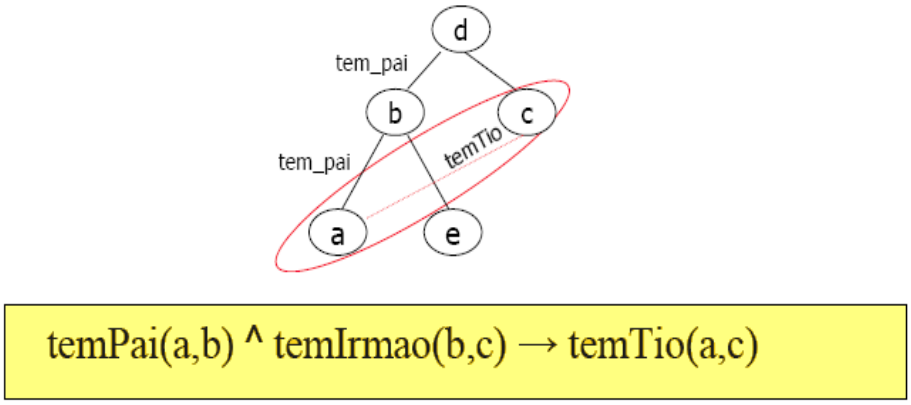

Figura 9 - Representação de uma inferência que não é possível em OWL (fonte: (MOURA, 2006))

Uma maneira possível de superar esses problemas de restrição da OWL seriam linguagens de regras. Segundo IAN HORROCKS (2005) a união de lógica de descrição com linguagens de regra, combinando os melhores recursos de cada um é um alternativa bastante razoável, principalmente para problemas cruciais de decidibilidade. Uma linguagem que combina esses dois paradigmas é a SWRL.

3.5.5. SWRL

O SWRL (*Semantic Web Rule Language*) é uma linguagem de regras baseada na combinação das sub-linguagens do OWL que são o OWL DL e o OWL lite. Ela possui também uma sublinguagem de marcação de regras Unária e Binária do Datalog[13]. Essa linguagem estende o conjunto de axiomas do OWL incluindo clausulas horn[14]. Sendo assim, isto a torna apta para combinar regras horn com a base de dados OWL.

As regras são da forma de implicação entre antecedente (corpo) e conseqüente (cabeça). O significado pode ser lido como: sempre que as condições no antecedente acontecerem, então as condições do conseqüente devem acontecer. Tanto o antecedente quanto o conseqüente consistem de zero ou mais átomos. Um antecedente vazio será sempre verdade, dessa forma o conseqüente será sempre atendido; Já um conseqüente

[13] Datalog é uma linguagem de consulta não procedural baseada na linguagem de programação lógica Prolog.
[14] Regra Horn é uma implicação de um antecedente (um conjunto de formulas atômicas) para um consequente (uma única formula atômica). Todas as variáveis no conseqüente devem ocorrer em pelo menos um átomo do antecedente, e todos são considerados universalmente quantificados.

vazio é tratado sempre como falso. Múltiplos átomos são tratados sempre como conjunções. É importante notar que regras com múltiplas conseqüentes podem ser transformado em várias regras sendo cada uma com um conseqüente único. Por exemplo:

a ^ b → p ^ q então cada átomo pode ser tratado separadamente aplicando-se a distributividade:

(~(a ^ b) V (p ^ q))

(~(a ^ b V p)) ^ (~(a ^ b) V q)

Os átomos das regras podem ser da seguinte forma C(x) (descrição OWL DL), P(x,y) que são propriedade entre valores , sameAs(x,y) e DifferentFrom(x,y) que são propriedades da OWL ou simplesmente x que pode representar variáveis ou valores de OWL.

- Sintaxe:

Sobre a sintaxe, se quiséssemos representar a regra descrita na Figura 9, esta ficaria da seguinte forma:

`temPai(?x,?y) ∧ temIrmao(?y,?z)` ⇒ `temTio(?x,?z)`

Sendo que x,y e z são variáveis da classe Pessoa. Não é necessário nesse caso explicitar isso, pois as propriedades utilizadas são aplicáveis apenas a classe pessoa. Porém, caso houvesse outra classe contendo essas propriedades, seria necessário explicitar. Tendo em vista esse caso o nosso exemplo ficaria então da seguinte forma:

`Pessoa(?x)∧Pessoa(?y)∧ Pessoa(?z)∧ temPai(?x,?y)` ∧ temIrmao(?y,?z) ⇒ temTio(?x,?z)

- Built-Ins

Além dessas opções de aplicação, ainda temos os *built-ins* que são funções que permitem futuras extensões a linguagem. O sistema de *built-in* também pode ajudar na interoperabilidade do SWRL com outros formalismos fornecendo uma infra-estrutura extensível e modular para linguagens de web semântica, serviços web e aplicações web. Entre os diversos built-ins oferecidos podemos citar:

swrlb:equal(?x,?y): retorna true se x for igual a y

swrlb: notEqual(?x,?y): retorna true se x for diferente de y

swrlb:lessThan(?x,?y):: retornar true se x e y forem do mesmo tipo e se x tiver um valor inferior ao de y.

Para ter acesso completo a lista completa dos built-ins basta visitar o página http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

- Limitações

Embora a SWRL seja uma linguagem rica em recursos, dando muito mais recursos a uma ontologia, também possui suas limitações. Algumas dessas limitações são citadas abaixo no FAQ[15] sobre SWRL ("SWRLLanguage FAQ," 2009).

Não tem Nenhuma característica não-monotônica. WRL apenas suporta inferências monotônicas. Então, não é possível usá-lo para modificar informações em uma ontologia. Se as regras SWRL permitissem modificações, a não-monotônicidade iria acontecer. Por essa razão não é possível retirar ou remover informações de uma ontologia usando SWRL.

Por exemplo, assuma que exista uma regra que indica que um motorista tem mais que 25 anos e é segurável:

```
Motorista(?d)∧temIdade(?d, ?idade)∧swrlb:greaterThan(?idade, 25)-> seguravel(?d, true)
```

Esta regra irá adicionar o valor *true* para a propriedade segurável para todos os motoristas que satisfizerem o antecedente. Esta não muda o valor existente para a propriedade. Se, por exemplo, um motorista tem um valor prévio de falso para essa propriedade, um disparo bem sucedido dessa regra para os motoristas irá resultar na propriedade tendo dois valores.

Inexistência de disjunções. Apenas conjunções de átomos são suportados. Dessa forma, a seguinte regra não é possível:

```
A(?x) V B(?x) -> C(?x)
```

Porém, na maioria dos casos essa restrição é fácil de contornar. No caso do exemplo, as seguintes regras irão produzir o mesmo efeito.

```
A(?x) -> C(?x)

B(?x) -> C(?x)
```

[15] *FAQ* é um acrônimo da expressão inglesa *Frequently Asked Questions*, que significa Perguntas Freqüentes. È basicamente uma compilação de perguntas freqüentes acerca de determinado tema

Inexistência de Negação por falha. Uma conseqüência da monotonicidade do SWRL é que negação por falha[16] não é suportada. A seguinte regra, por exemplo, não é possível:

```
Pessoa(?p) ∧ ¬ temCarro(?p, ?c) -> PessoaSemCarro(?p)
```

Mas, é possível contornar essa situação, no exemplo acima outra forma de escrevê-la seria assim:

```
Pessoa(?p) ^ (temCarro = 0)(?p) -> PessoaSemCarro (?p)
```

Contudo, a negação clássica é possível no SWRL com o uso da propriedade owl:complementOf nas regras. Por exemplo, é possível reescrever a seguinte regra que define se um indivíduo não é membro de uma classe de pessoa, então ele poderia ser classificado como membro da classe não humana.

```
(not Pessoa)(?x) -> NãoHumano(?x)
```

[16] **Negação por falha:** interpretação da negação da lógica, na qual a negação de uma fórmula é Verdade se a fórmula não puder ser provada Verdade.
Ex: não a(b). Se a(b) for Falso então não a(b) é Verdadeiro

Capítulo 4. Dados Autonômicos

Este capítulo tem por finalidade apresentar os conceitos associados aos Dados Autonômicos, principal tema dessa dissertação de mestrado. Adicionalmente, serão apresentadas suas propriedades e várias aplicações que podem se beneficiar dessas idéias.

4.1. Definição

Assim como os sistemas computacionais têm sua complexidade em constante crescimento, os dados também têm seguido essa tendência e, hoje em dia, temos uma explosão da quantidade de dados. Porém, os dados não estão apenas crescendo em quantidade, mas também em complexidade. Dados cada vez mais complexos e interligados têm surgido e isso dificulta enormemente o trabalho de gerentes de TI que não conseguem lidar com tanta informação.

Segundo HAN & KAMBER (2006), estamos inundados por esses dados, eles são cientifícos, médicos, financeiros, entre outros, e é a partir deles que conduzimos nossas vidas. As pessoas não têm condições de analisar todos essa informação. É importante considerar que a atenção humana é um recurso muito precioso. Com isso, sempre estão sendo buscadas técnicas para analisar, classificar, resumir, indentificar tendências e indentificar anomalias dos dados automaticamente.

Junto com a grande quantidade e complexidade dos dados veio a complexidade dos sistemas, que, como foi descrito no capítulo 3, tem se tornado um dos grandes desafios nos tempos atuais. Várias técnicas de programação surgiram para administrar essa complexidade, objetivando organizar e apoiar o processo de desenvolvimento de produtos de *software*. Com isso, no âmbito da Ciência da Computação, essas técnicas de programação vêm evoluindo, desde construções de baixo nível – como linguagens de máquina – até abordagens de alto nível – como a Programação Orientada a Objetos (POO) (ELRAD *et al*., 2001).

Um dos problemas enfrentados, que não é resolvido pelas técnicas atuais de desenvolvimento de software, é que o dado não tem características pró-ativas, ele

sempre depende da manipulação de outros sistemas. Essas características geralmente sao implementadas no ambiente, o que impede a livre movimentação dos dados por ambientes heterogêneos. Além disso, implementando essas características no ambiente, o deixamos mais complexo, dificultando sua manutenção e seu reuso.

Tendo em mente a situação enfrentada pelas pessoas, este trabalho propõe um conceito de estrutura de dado que objetiva minimizar os efeitos desse problema. O conceito relativo aos Dados Autonômicos segue a filosofia da Computação Autonômica onde os elementos se auto gerenciam. Mas, além disso, os Dados Autonômicos também seguem a filosofia da POO, não visando concorrer com esta e sim complementá-la.

A idéia dos Dados Autonômicos foi apresentada pela primeira vez em SILVA *et al* (2008) e pode ser caracterizada por utilizar dados com estrutura mais complexa, carregando um pouco de semântica consigo. Este conceito soluciona alguns problemas de complexidade dos dados facilitando a manipulação dos dados em geral. Dessa forma, a complexidade adicional, que normalmente é embutida nos sistemas, é transferida para os próprios dados. De certa forma, podemos dizer que estamos fatorando a complexidade comum existente no sistema e colocando num lugar que todos os sistemas teriam acesso, neste caso: nos dados. A Figura 10 ilustra essa idéia da fatoração da complexidade.

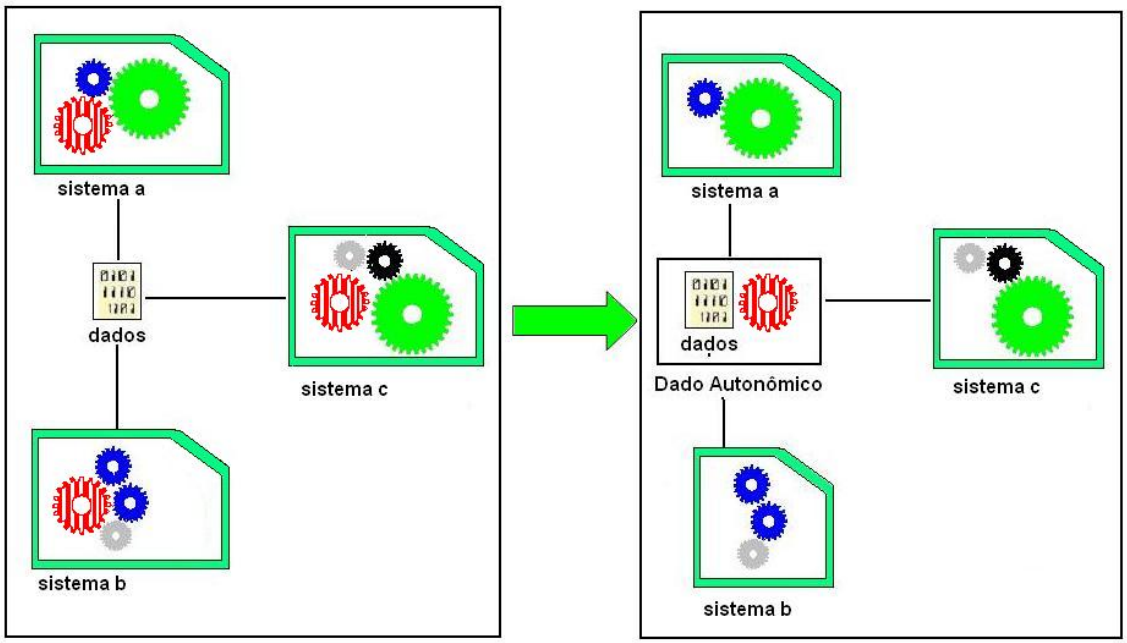

Figura 10 – Sistema com dados normais e o sistema com Dados Autonômicos

Na figura podemos verificar a existência de vários sistemas trocando dados entre si. Na representação as funcionalidades do sistema são representadas por engrenagens, ou seja, cada uma dessas engrenagens representa um pouco da complexidade do sistema. É possível notar que a engrenagem listrada é comum a todos os sistemas da figura. A partir deste cenário, no segundo quadro, é mostrado como a complexidade pode ser acoplada aos dados, de modo a termos um Dado Autonômico. Dessa forma, os sistemas continuam tendo acesso a essas funcionalidades sem ter que implementá-las.

Essa complexidade, que fica no Dado Autonômico, está na forma de regras dinâmicas e informações adicionais que podem ser representadas por meta-informações. A Figura 11 ilustra a visão arquitetural do Dado Autonômico.

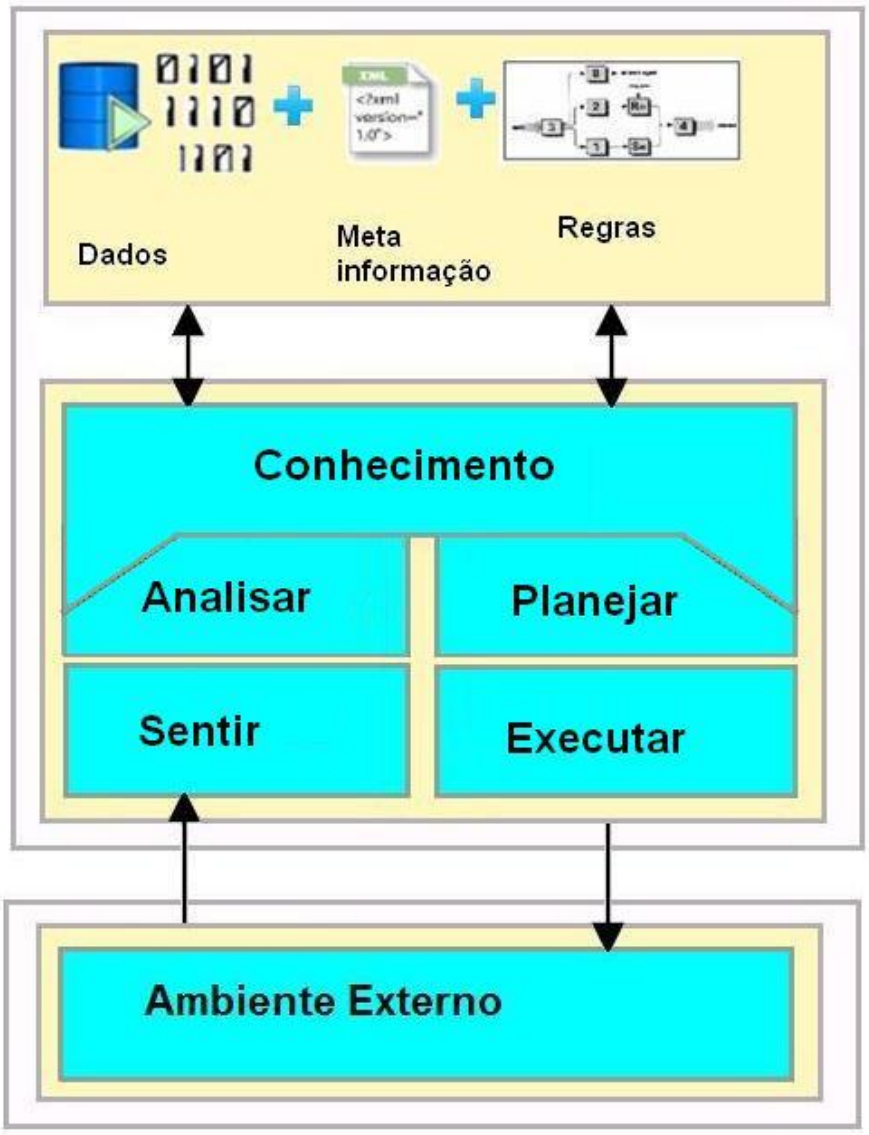

Figura 11 - Modelo Estrutural do Dado Autonômico

Entre as meta-informações, destacam-se: sua localização, classificação, estatísticas e conceitos. Como exemplo, temos que um dado sobre uma atividade pode conter informações que vem da França, ser classificado como atividade de finalização de projeto, e ter a estatística de 20 acessos. A partir desses dados, inúmeras ações poderão ser tomadas.

Dentre as regras embutidas nos dados autonômicos podem existir regras de negócios ou de eventos.

As regras de negócio definem em que contexto os dados podem ser aplicados, e alguns tipos de resultados esperados. Como exemplo, pode-se imaginar que se determinado dado descreve uma atividade, a regra de negócio irá especificar que essa atividade pode durar de uma a quatro horas, que suporta gerentes com mais de 4 anos de experiência e que valores fora desse intervalo não devem ser aceitos a não ser que a atividade seja do tipo de finalização de projeto.

Já as regras de evento definem ações a serem tomadas, caso ocorra determinado(s) evento(s). Essas regras por sua vez não são as únicas a serem verificadas, e se juntarão às regras já estabelecidas no sistema onde o Dado Autonômico está inserido, compondo assim as regras gerais do sistema. Como exemplo, tem-se que um dado possui a seguinte regra: se o número de acessos ao dado for superior a 100 por dia, é feita a replicação. Já o sistema pode conter a seguinte regra: se o dado não for acessado nenhuma vez por dia então ele é apagado. Dessa forma, da junção das duas regras temos que se o dado não for acessado, será apagado e se for acessado mais de cem vezes por dia, será replicado. Assim, é facil perceber que um mesmo Dado Autonômico pode ser executado de formas diferentes em sistemas diferentes, pois cada sistema tem suas próprias regras.

Todas as ações e verificações são feitas por um elemento autonômico, composto pelo raciocinador, que processa as regras existentes, e executor, que possui os sensores e percebe as mudanças no ambiente. Estruturalmente, este elemento autonômico não está junto ao dado, está junto do sistema que lê o dado, mas semanticamente ambos estão juntos.

É bom ressaltar que podem ocorrer conflitos ao se combinar as regras do Dados Autonômicos e as regras do sistemas. Por isso, a regra deve conter também prioridade de execução, para saber qual irá sobrepor sobre qual, resolvendo assim os conflitos.

Outro ponto a se ressaltar é o caracter dinâmico das meta-informações e das regras. Dessa forma, o sistema poderá se aperfeiçoar, atualizando sempre as

classificações, estatísticas, conceitos e mesmo as regras. O ideal inclusive é que ocorra isso, e que as atualizações sejam feitas pelo próprio sistema autonômico depois de identificar melhorias que possam ser feitas.

4.1.1. Composição: Características dos Dados Autonômicos e seus 10 mandamentos

Como se trata de uma definição nova, todas as propriedades e características devem ser bem definidas para não haver ambiguidades nem confusões quanto ao uso.

Segundo PINHEIRO & SILVA *et al* (2009b) o Dado Autonômico deve ser uma entidade que encapsula o dado e provê uma interface de comunicação para o acesso ao mesmo. O Dado Autonômico deve abstrair informações, mostrando apenas dados que seja relevantes ou importantes para o sistema. O sistema não necessita saber todos os detalhes dos dados, precisa saber apenas o que for relevante para ele. .

O Dado Autonômico deve ser polifórmico, ou seja, deve se adaptar ao ambiente de forma a contornar situações imprevistas. Para isso, o Dado Autonômico deve possuir pelo menos uma das características das regras CHOP, quais sejam: auto-configuração, auto-proteção, auto-cura e auto-otimização.

Baseado nas características necessárias para um sistema ser considerado autonômico (IBM, 2009), redefiniu-se essas propriedades de modo a adaptá-las para o conceito do Dado Autonômico. Essas características deverm guiar o comportamento do Dado Autonômico nos diversos ambientes em que se encontra. Dessa forma, definimos os 10 mandamentos que orientam o comportamento dos dados autonômicos. São eles:

1 – Deveis ter conhecimento sobre si mesmo:

O Dado Autonômico deve carregar consigo um pouco de sua semântica de forma que seja possível saber um pouco mais sobre ele. As informações adicionais que o Dado Autonômico carrega consigo podem ser diversas, entre eles podemos citar: estatísticas, meta informações e regras. O objetivo disso é que os Dados Autonômicos não sejam uma simples informação, mas que tenham uma semântica e um significado, sendo assim mais próximo de ser um conhecimento.

2 – Deveis conhecer teu ambiente e o contexto ao redor de tuas atividades

A grande diferença dos Dados Autonômicos para um dado normal é a possibilidade de diferentes execuções de acordo com o sistema em que esteja. Um dado normal sempre será executado da mesma forma, porém, um Dado Autonômico tem informações a mais, e essas informações podem indicar como os dados serão executados. Dentre essas informações adicionais incluem-se as informações do ambiente e das atividades que estão sendo executadas.

3 – Deveis não se restringir a um único ambiente fechado. Deveis ter a capacidade de se mover para outros ambientes.

O Dado Autonômico não deve ficar restrito a apenas em um sistema, deve ser possível a sua migração para outros sistemas. Uma das principais características deste dado é levar a sua semântica (dependente de contexto) consigo, dessa forma para cada sistema diferente aonde ele vá, este terá uma interpretação diferente.

Para o acesso das informações do Dado Autonômico, este deverá fornecer uma interface de comunicação.

4 – Deveis fornecer a informação adequada ao ambiente em que se encontra, escondendo a sua complexidade;

O Dado Autonômico deve abstrair a complexidade dos dados e fornecer apenas as informações importantes e relevantes. E, caso a complexidade dos dados aumente, ou seja alterada, o sistema deve ficar alheio a isso, recebendo as informações da mesma forma.

5 – Deveis cooperar uns com os outros visando aperfeiçoar a si mesmo e ao conjunto;

Os Dados Autonômicos devem procurar trocar informações com os demais dados para otimizar o sistema. Essas informações podem ser as mais diversas, como meta-informações, estatísticas de acesso, regras que tiveram sucesso com determinado usuário etc.

6 – Deveis encapsular informações e forneceras uma interface de comunicação para que outro sistema possa acessar teus dados;

O Dado Autonômico deve encapsular informações e não deixar o sistema acessá-los diretamente. O Dado Autonômico tem que fornecer uma interface de comunicação por onde o sistema possa obter as informações necessárias. Essa interface pode ser polimórfica, de modo que em diferentes sistemas ela possa ser usada diferentemente. Por exemplo, considere que exista uma interface que forneça ao sistema os dados de forma ordenada, em um sistema pode ser usado o *quick sort*[17], enquanto que outro sistema pode usar o *merge sort*, dependendo para isso das características e das regras de negócio do sistema.

7 – Deveis se configurar e reconfigurar sob variações de condições imprevistas;

O Dado Autonômico deve ter a percepção de mudanças em si, deve analisar essas mudanças e então tomar decisões de modo que melhore a sua condição. Por exemplo, digamos que para um determinado sistema o Dado Autonômico tem estatísticas de acesso ao mesmo. Com isso, o Dado Autonômico tem a percepção quando está sendo muito ou pouco acessado. Dessa forma, com essa informação é possível prever se um determinado horário tem mais acessos que outros. Então, o Dado Autonômico, possuindo essa informação, poderá sugerir ao sistema que ocorra uma reconfiguração para que não aconteça nenhum problema relativo a sobrecarga por acessos.

8 – Deveis nunca estar satisfeito com a situação atual do ambiente, deveis sempre procurar otimizações;

O Dado Autonômico carrega informações do sistema consigo. Assim, ele tem consciência de como o sistema está. Deve então tomar ações para melhorar a si mesmo e com isso melhorar o seu ambiente. Deve haver uma melhoria contínua para que a semântica do mesmo reflita da melhor forma possível a realidade.

[17] *Quick sort* e *merge sort* são dois tipos de algoritmos de ordenamento de dados, cada um tem características que os melhor qualifiquem para a situação que serão usados.

9 – Deveis se curar.

O Dado Autonômico deve zelar pela integridade dos seus dados, não permitindo a inserção de anomalias, dados errôneos ou mesmo incompletos. Se possível, deve corrigir dados que foram inseridos incorretamente e eliminar dados redundantes.

10 – Deveis ter informações para proteger a si mesmo.

O Dado Autonômico deve possuir informações do ambiente que são atualizadas constantemente. Através dessas informações é possivel prever situações de risco para o dado e para o sistema. Por exemplo, se o dado tiver estatísticas de acesso, este pode prever quando será muito acessado e prever assim uma replicação.

Depois de apresentadas todas as características e propriedades podemos resumir a definição de Dados Autonômicos como uma estrutura de dados pró-ativa que contém informações adicionais descrevendo suas características e semântica (dentro dos diferentes contextos em que atua). O Dado Autonômico deve encapsular informações fornecendo uma interface para acesso a informação, deve abstrair dados quando necessário e deve ser polimórfico, se adaptando a situações diversas, tomar decisões para a própria otimização e ter pelo menos uma das características autonômicas das regras CHOP.

Considerando os 10 mandamentos do Dado Autonômico podemos verificar que o Dado Autonômico possui algumas camadas de abstração. Essas camadas podem ser divididas em três, sendo a mais interna a de dados, a intermediária é a de controle, onde são feitas os processamentos autonômicos e a mais externa é a camada de comunicação que oferece a interface de comunicação com o ambiente. Na camada de dados se encontram a informação em si, meta-informações dos dados e regras. Na camada de controle é feita a transformação de dados e o processamento. Na camada de comunicação é fornecida ao ambiente a maneira de ambos de comunicarem, ficando assim as interfaces de comunicação e as várias formas de poliformismo. A Figura 12 ilustra a arquitetura do Dado Autonômico em camadas.

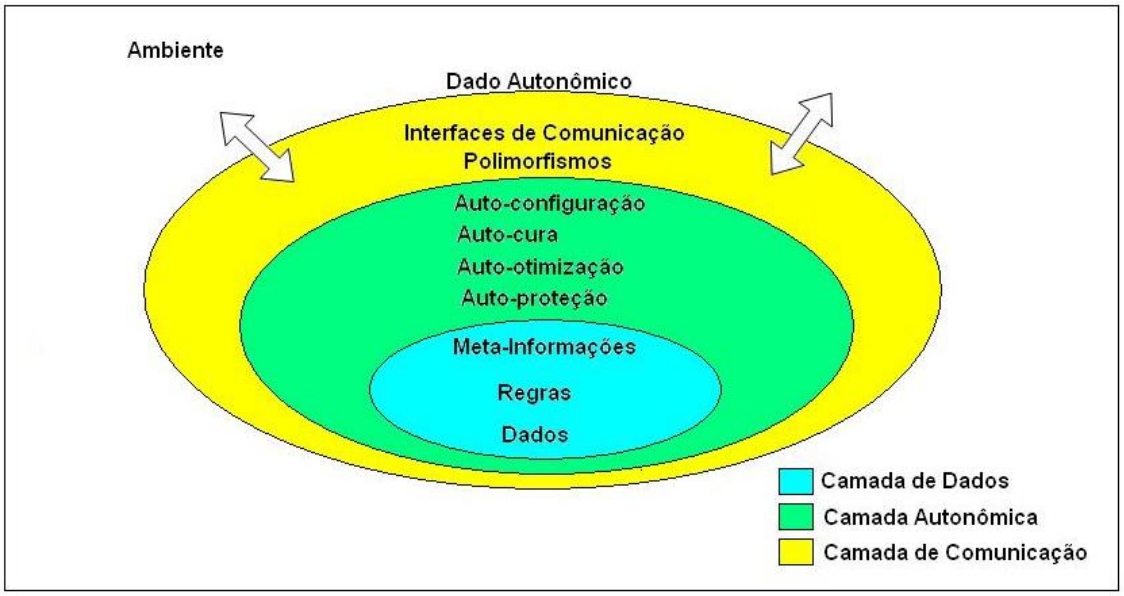

Figura 12 – Estrutura em camadas do Dado Autonômico

4.1.2. Dado Autonômico como agente e objeto

Segundo TECUCI (1998), um agente é basicamente um sistema de software que interage e percebe seu ambiente, interpreta essas percepções, faz inferências, resolve problemas, determina ações e age sobre o ambiente de modo que realize tarefas ou objetivos para o qual foi designado. Já RUSSEL & NORVIG (2004) tem uma definição mais geral onde um agente é uma entidade que é capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores. Além disso, a inteligência seria uma característica apenas desejável para se ter um agente autônomo, mas certas aplicações não precisam de autonomia, nem precisam de algum grau de inteligência.

Com isso, é fácil perceber que muitas das características apresentadas dos Dados Autonômicos correspondem também a características dos agentes. Então, pode se inferir, baseado nessas definições, que o Dado Autonômico é uma especialização de agente. Sendo assim, olhando por esse ângulo de visão, temos que o Dado Autonômico é um agente composto por dados que percebe seu ambiente por meio de atuadores e pode se modificar para melhor se adaptar ao seu ambiente.

Além do campo dos agentes, podemos notar as semelhanças com o campo da Orientação a Objetos. Segundo SCHACH (2005), um objeto é uma entidade de software ativa criada a partir de um modelo de dados em programação a objetos, ou em outras palavras, uma instância de uma classe. Um objeto é constituído basicamente de

atributos, métodos e um identificador. Os atributos são variáveis de qualquer tipo, sendo que essas podem ser outros objetos e o estado (valores) dessas variáveis é mantido enquanto o objeto estiver em atividade. Já os métodos são procedimentos e funções que executam alguma determinada ação. Tanto os atributos quanto os métodos têm uma visibilidade que define quem pode acessá-los. Dessa forma, alguns atributos das classes não são acessíveis para outros objetos, podendo ser acessado e/ou modificados somente através dos métodos.

Além disso, os objetos ainda possuem características de encapsulamento de dados, herança e polimorfismo. O encapsulamento é a capacidade de agrupar informações de modo modularizado, escondendo detalhes que não necessitam ser conhecidos. Herança é a capacidade de uma classe herdar métodos e atributos de outra classe e, finalizando, poliformismo é a capacidade de um método de classes filhas terem comportamentos diferentes da classe pai (SCHACH, 2005).

É possível notar novamente que temos uma definição similar aos Dados Autonômicos, onde várias características são comuns. Isso não é por acaso, pois várias das características dos Dados Autonômicos foram pensadas baseando-se na Orientação a Objetos.

Dessa forma, com o objetivo de verificar as diferenças e as semelhanças entre essas três áreas, as características do Dado Autonômico são discutidas abaixo. As características analisadas são:

1 – Deveis ter conhecimento sobre si mesmo:

Um agente normalmente não precisa conhecer a si mesmo, pois sua preocupação é atuar sobre o ambiente através de atuadores. Caso o agente faça parte do ambiente, de modo que o estado atual do ambiente modifique o agente, então ele necessita conhecer a si mesmo, mas isso ocorre somente em casos específicos. Em relação ao objeto, podemos dizer que ele conhece a si mesmo, pois necessita saber a visibilidade de seus métodos e variáveis, seu atual estado e o seu polimorfismo para saber qual o método correto a invocar.

2 – Deveis conhecer teu ambiente e o contexto ao redor de tuas atividades:

Um agente conhece seu ambiente através de seus atuadores, mas nem sempre conhece todo o ambiente, tendo a consciência da parte específica que lhe cabe. O objeto normalmente não conhece o seu ambiente, mas pode saber referenciar outros objetos. Dessa forma, um objeto conhece o seu ambiente apenas se este for também for um objeto e o primeiro tiver a referência deste. Normalmente, a comunicação entre objetos é fruto da troca de mensagens entre objetos interconectados.

3 – Deveis ter a capacidade de se mover para outros ambientes:

Os agentes normalmente não saem de seus ambientes. Apesar disso, existe uma especialização de agente chamada agentes móveis que pode se mover para outros ambientes mantendo seu estado (BRAUN; ROSSAK, 2005).

Os objetos, por sua vez, podem se mover de um ambiente para outro facilmente, usando tecnologias como RMI e CORBA, mantendo o seu estado. (PLÁŠIL; STAL, 1998)

4 – Deveis fornecer a informação adequada ao ambiente em que se encontra, escondendo a sua complexidade:

Normalmente os agentes atuam sobre o ambiente, mas o ambiente não tem acesso ao agente de forma que este não fornece nenhuma informação àquele. Por sua vez, os objetos possuem a características de encapsulamento e de visibilidade, dessa forma o objeto fornece apenas as informações necessárias e permitidas ao requerente escondendo as demais.

5 – Deveis cooperar uns com os outros visando aperfeiçoar a si mesmo e ao conjunto:

Um agente normalmente não precisa cooperar com outros agentes. Mas, existe uma especialização de agente chamada agentes colaborativos que se comunicam entre si para melhor interagir sobre o ambiente (ALLEN *et al*., 2002).

Os objetos seguem raciocínio similar e podem ou não colaborar uns com os outros, dependendo do fim a que se destinam.

6 – Deveis encapsular informações e forneceras uma interface de comunicação para que outro sistema possa acessar teus dados:

Um agente geralmente não fornece uma interface de comunicação para sejam acessados seus dados. Os objetos por sua vez oferecem essa interface através do uso de métodos.

7 – Deveis configurar e reconfigurar sobre diferentes condições e ambientes:

Normalmente os agentes não se reconfiguram, mas recentemente têm surgido muitos trabalhos em que os agentes tenham essa habilidade. No trabalho de WU *et al* (2001) os agentes possuem mecanismos para se reconfigurar e continuar atuando mesmo com determinadas falhas na rede. Já no trabalho de BLAKE (2001), o termo agentes auto-configuráveis é usado para designar a função do agente coordenador baseado em regras, que neste trabalho, atua como mediador num ambiente de componentes distribuídos. Neste trabalho o agente configura os aspectos de comunicação, administração de dados, e execução de políticas.

Os objetos normalmente não se auto-configuram, exceto se existir um método criado especificamente para esse fim

8 – Deveis nunca estar satisfeito com a situação atual do ambiente, deveis sempre procurar otimizações:

Um agente é feito com o propósito de perceber algo no ambiente e atuar sobre ele. Algumas vezes, os agentes podem reter o conhecimento que obtiveram do ambiente para ter uma atuação melhor, são esses os agentes adaptativos (RESCONI; JAIN, 2004). Sendo assim, é possível classificar essa classe de agentes como portadora de auto-otimização, pois melhora sua atuação de acordo com a percepção de seus atuadores. Além disso, ainda encontramos o trabalho de MANCINI *et al* (2006) que propõe uma auto-otimização para agentes em seu trabalho, baseado nas características da Computação Autonômica.

Com relação aos objetos, estes geralmente não se auto-otimizam.

9 – Deveis se curar:

Existem diversos casos onde são usados agentes para uma auto-cura do ambiente, como no trabalho de PARK *et al* (2005) em que um sistema de agentes faz o auto-diagnostico e provê também uma solução para a cura. Apesar disso, não há relatos de agentes que se curem. Um dos fatos para isso é que, na quase totalidade dos casos, os agentes não são alvos de ataques. As falhas que atrapalham os agentes normalmente acontecem no ambiente.

Os objetos geralmente não contêm a propriedade de auto-cura.

10 – Deveis ter informações para proteger a si mesmo:

Assim como no caso da auto-cura, são diversos os casos onde são usados agentes para a auto-proteção do ambiente. Temos como exemplo disso o trabalho de STERRIT & HINCHEY (2005), onde é proposto um sistema de auto-proteção com agentes autonômicos para proteção dos sistemas de exploração espaciais. Apesar disso, não são encontrados casos onde os agentes buscam a auto-proteção.

Os objetos normalmente não possuem a propriedade de auto-proteção.

4.2. Aplicações sugeridas de Dados Autonômicos

Durante o estudo do conceito de Dados Autonômicos, várias aplicações foram estudadas para verificar quais poderiam se beneficiar do uso de Dados Autonômicos. Algumas envolveram estudos elaborados e bem aprofundados, mas que não foram implementados. Duas aplicações em específico foram bem estudadas, uma é o XML Ativo e o outro é COPPEER. A seguir é descrita como ficariam essas duas aplicações com o uso de Dados Autonômicos.

4.2.1. Uso de Dados Autonômicos como XML Ativo

A tecnologia de serviços Web se tornou uma alternativa importante para a interoperabilidade de aplicações e sistemas de integração de informações. Uma

tecnologia que explora esse potencial de uso de serviços Web é o XML Ativo (AXML). Essa classe de documentos XML, surgida recentemente, possui conteúdo dinâmico, pois combina dados XML com chamadas de serviços Web. Uma grande vantagem dos documentos AXML é a possibilidade de manter atualizadas informações voláteis, como a temperatura atual de uma cidade ou um saldo bancário. (ABITEBOUL *et al*., 2002)

Neste contexto, o XML Ativo seria um Dado Autonômico onde meta-informações e regras seriam incluídas para otimizar o sistema em diversos aspectos. Outro aspecto a se aproveitar seria o uso de chamadas a serviços Web, onde os provedores de serviço web forneceriam também meta-informações e regras.

O sistema de XML Ativo é executado basicamente sobre duas camadas, a parte da aplicação e a parte dos serviços web. Considerando isso, foram pensados os seguintes cenários utilizando os Dados Autonômicos e as regras CHOP;

1 - Auto-Otimização:

Otimização de consultas baseado no tipo da aplicação: Podem ser feitos vários tipos de pesquisa em uma mesma aplicação, alguns mais refinados do que outros. Assim, uma otimização a ser feita seria, para um mesmo serviço, deixar várias consultas preparadas, na qual seria acessada a mais apropriada, de acordo com o nível de refinamento da pesquisa e de acordo com os campos requeridos. Dessa forma, com buscas mais específicas, teríamos uma otimização nas buscas.

Materialização específica para cada tipo de aplicação: Os dados podem prover informações sobre eles mesmos. Dessa forma, o processo de ordenamento de materialização dos campos seria facilitado.

Proximidade dos serviços web: Em aplicações onde ocorra grande número de requisições de serviços web, a velocidade de resposta dos servidores de serviços pode ser crucial. Com isso, achar os pares mais próximos da aplicação ou os menos ocupados pode representar um grande ganho para aplicação. Assim, o Dado Autonômico teria regras e meta-informações, baseados em estatísticas, para encontrar os melhores servidores de serviços Web.

2 - Auto-Cura:

Descarte de marcações inseguras: Algumas invocações a serviços web podem ser inseguras e representar risco para a aplicação, ou ainda podem trazer recursos que não serão visualizadas na tela (por exemplos, alguns navegadores não suportam certos recursos). Assim, seria interessante que aplicação não permitisse que tais marcações fossem materializadas e muito menos que esses serviços fossem invocados. Os Dados Autonômicos teriam regras e meta-informações para identificar quais serviços seriam prejudiciais à aplicação.

3 - Auto-Proteção:

Identificação de recursão infinita: Quando é feita uma solicitação de serviço web para um servidor A, esse servidor pode não dispor daquele serviço e com isso pode redirecionar o pedido para um servidor B que tenha esse serviço. Porém esse servidor B pode não dispor do serviço também e pode redirecionar para um servidor C. Dessa forma, esses redirecionamentos podem acabar levando a uma recursão infinita e a aplicação pode ficar esperando eternamente pela resposta. Para evitar esse problema haveria meta-informações e regras no XML Ativo que iriam indicar alguns possíveis sinais de recursão infinita, e quais seriam os melhores servidores a serem procurados.

4.2.2. Framework COPPEER

Nessa subseção será apresentada a arquitetura do COPPEER e alguns projetos que se beneficiam dele para depois mostrar as propostas. O COPPEER é um projeto de pesquisa em progresso no laboratório de Banco de Dados da COPPE/UFRJ. O objetivo deste projeto é a implementação de um arcabouço para desenvolvimento e execução de aplicativos ponto-a-ponto colaborativos (MIRANDA *et al*., 2006).

Figura 13 - Arquitetura lógica da plataforma COP*PEER(fonte:* (MIRANDA *et al.*, 2006)*)*

Este arcabouço possui uma arquitetura lógica de vários níveis, ilustrada na Figura 13, destinada a facilitar o tratamento da complexidade combinada de sistemas. O nível mais baixo é ocupado pela camada de agência que executa computações ponto-a-ponto genéricas para atender as necessidades das outras camadas. Dentro dessa camada, o sistema é visto como um conjunto de agências colaborativas. Uma agência é a entidade interna de um nó da rede na qual os agentes do COPPEER são executados. Um agente COPPEER é um artefato de software que troca informações com outros agentes e move-se de uma agência para outra para executar computações distribuídas (MIRANDA *et al*., 2006).

Sobre a camada de agência, apóia-se a camada de integração, que abrange os problemas de locação de aplicações, gerenciamento de sessões e integração remota de sistemas ponto-a-ponto. Essa camada é responsável por fornecer locações virtuais de alta disponibilidade para aplicações colaborativas (MIRANDA *et al*., 2006).

4.2.2.1. Projetos que usam o COPPEER

Esta subseção tem por intenção falar dos programas alvos, que usam o COPPEER que iriam se beneficiar com o uso de Dados Autonômicos. Os programas PMCE, COE e KCE são todos ponto-a-ponto e são executados sobre a plataforma COPPEER. A seguir é mostrada uma visão geral desses projetos, e descrita a estrutura de armazenamento de cada um, pois esta é uma informação primordial visto que este projeto é justamente sobre dados.

O PMCE (*Project Management Collaborative Editor*) é um editor ponto-a-ponto para colaboração e compartilhamento de conhecimento para a gerência de projetos, que possibilita a disseminação do conhecimento gerado e das experiências adquiridas durante a realização das atividades do projeto por seus responsáveis, através de ferramentas de cunho colaborativo. (DE REZENDE *et al*., 2008)

Em relação aos dados (fato que interessa diretamente a este projeto) cada *peer* tem sua própria base de dados e, quando acontece alguma atualização, o banco de dados de todos os *peers* é atualizado.

O COE (*Collaborative Ontology Editor*) é um editor colaborativo de ontologias em um ambiente ponto-a-ponto com a utilização da plataforma COPPEER, com suporte à criação, edição, compartilhamento e reuso de ontologias na linguagem OWL. O COE faz a busca de ontologias, através de uma palavra-chave, usando a rede ponto-a-ponto e a combinação das ontologias retornadas pela busca com a ontologia editada (VILELA, 2007),

Por fim, temos o KCE (*Knowledge Chains Editor*) que é um editor de aprendizagem colaborativa projetado para permitir a troca de cadeias de conhecimento. Os dados utilizados neste editor são arquivos OWL, assim como o COE. Na verdade, as cadeias de conhecimento, principal dado dessa aplicação, são todas ontologias em OWL (SOUZA *et al*., 2006).

4.2.2.2. Detalhamento da Proposta

A proposta é apresentada sobre a perspectiva das regras CHOP

1 - Auto-Configuração dos dados:

a) Replicação de Bases de Dados: Todos os projetos apresentados contêm base de dados distribuídos pelos *peers*. Alguns *peers* contêm dados que são mais acessados que outros, e com isso ficam sobrecarregados. Uma solução para esse problema é a replicação. Porém, esses acessos podem ser variados, pois um grupo de usuários tem necessidades diferentes de outros, o que torna as buscas também diferentes. Os acessos também podem ser diferentes por horários (em certos horários há mais fluxo que outros). Essa situação torna difícil o planejamento da replicação.

Com isso, uma solução encontrada é deixar o próprio sistema fazer dinamicamente essa replicação. Os Dados Autonômicos guardam estatísticas de acesso

que tornam possível antecipar o problema e então fazer toda a configuração, deixando tudo transparente para o usuário. As estatísticas a serem usadas contabilizam o número de acessos médios por dia e por horário. Assim, o agente irá ler todos os dados e identificar em qual dia e horário geralmente ocorrem situações críticas (por exemplo, às segundas na parte da tarde há um grande volume de acesso) e, dessa forma, preparar a replicação dos dados para computadores ociosos (que estão ligadas sem objetivo justamente esperando por dados de replicação), desafogando a máquina mais acessada. Após o horário, a replicação é desativada e as máquinas ficarão novamente ociosas esperando por outros dados.

b) Configuração de envio de mensagens: Assim como existem *peer*s que são mais acessados (retornam mais respostas), existem outros que costumam não ter respostas para determinados grupos de usuário. Apesar destes *peer*s não serem "úteis", eles sempre recebem mensagens de requisição, e essas mensagens gastam processamento e banda de rede. Para resolver este problema, as mesmas estatísticas de acesso, que são usadas para identificar *peers* que precisam de replicação, podem ser usadas para identificar *peer*s que não irão retornar resultados. Com isso, esses *peer*s são descartados da lista de *peer*s que irão receber mensagem de requisição, economizando processamento e banda da rede.

As estatísticas a serem usadas fazem uma relação dos resultados retornados por *peer* e por busca semântica. Cada *peer* terá essa relação, sendo assim, é possível saber de antemão pra onde mandar as mensagens de requisição.

2 - Auto-Cura dos dados:

Validação de arquivos: Os projetos estudados neste trabalho fazem uso de ontologia e utilizam a linguagem OWL para tal (OWL, 2003). O arquivo do OWL é padronizado e com isso marcações colocadas erroneamente podem invalidar este arquivo. Tendo esse problema em mente, o projeto de dados autonômicos irá verificar os dados que estão para serem inseridos de modo a não permitir a invalidação dos demais dados. Dessa forma, é garantida a consistência dos arquivos, pelo menos na parte de entrada de dados.

Mas, ainda há outro problema, quando o arquivo já estiver inválido. Em muitos casos o que invalida o arquivo são pequenos erros (uma marcação que não foi fechada), que até são fáceis de serem corrigidos, mas difíceis para um usuário leigo o fazer. Além

disso, mesmo para um usuário mais avançado no assunto, encontrar erros é sempre trabalhoso. Então, para resolver este problema, o agente irá verificar a integridade dos arquivos e, caso seja constatado um pequeno erro, este vai tentar validá-lo.

3 - Auto-Otimização

Entre as otimizações algumas já foram mencionadas anteriormente, como a auto-replicação e a configuração de envios de mensagem que otimiza o processamento dos *peers* e a utilização da rede.

Agrupamento de dados similares: Além das citadas anteriormente, outra otimização é a cerca do agrupamento de dados similares. No caso dos arquivos OWL, dados semelhantes devem ficar próximos, na mesma ontologia, pois deixa o arquivo com uma maior semântica. Para fazer isso serão usadas estatísticas de resultados de dados semânticos (o mesmo usado para filtro de mensagem, detalhada na auto-configuração). De posse desses dados, é possível saber quais *peers* têm um maior aparato de dados sobre determinado assunto. Valendo-se disso, é possível enviar determinado conjunto de dados para um lugar no qual estes ficarão mais bem agrupados por semelhança, facilitando buscas.

4 - Auto-proteção

Duas proteções já foram citadas anteriormente, que é a proteção de sobrecarga de acessos (antecipar a replicação) e a validação de arquivos OWL inválidos.

Verificação da Integridade dos arquivos: Outra auto-proteção proposta é a verificação da integridade dos arquivos. Como os arquivos são transportados em um ambiente distribuído, estes podem ser interceptados e modificados, de modo que se comprometa a segurança dos dados e a execução de todo o sistema, caso seja inserido dados cujo propósito é prejudicar os sistemas. Dessa forma, um arquivo md5, que conterá o resumo do arquivo, irá acompanhar os arquivos enviados para verificar se realmente o arquivo veio com sua integridade intacta.

Capítulo 5. RSS Autonômico

Esse capítulo tem como objetivo detalhar a aplicação desenvolvida que valida a proposta realizada. Porém, se limita a detalhar a arquitetura e as funcionalidades da aplicação, bem como a teoria que embasou e encorajou o uso de algumas funcionalidades. A parte relacionada à usabilidade, navegabilidade e interface gráfica da ferramenta é detalhada somente no capítulo 6.

5.1. Definição

Assim como foi relatado na introdução, estamos vivendo na era da informação e por isso estamos ficando a cada dia mais sobrecarregados com o imenso volume de informação. No mundo das notícias isso não é diferente, a grande quantidade de veículos de comunicação e a facilidade de acesso a esses nos dão uma quantidade quase infinita de possibilidades de acesso a notícias em todo mundo.

Entre as facilidades oferecidas estão os *Feeds RSS. Feeds* são listas de atualização de conteúdo de um determinado site, escritos com especificações baseadas em XML. Atualmente, existem três principais especificações para a criação de arquivos *feed*: RSS 1.0 (*RDF Site Summary 1.0*), RSS 2.0 (*Really Simple Syndication* – Distribuição muito simples *2.0*) e Atom. As informações acerca das notícias estão divididas em marcações XML apresentando assim informações de cada notícia e do canal que a provê. É necessária uma marcação do tipo <*channel*> que representa o canal, e várias marcações do tipo <*item*> onde cada uma dessas representa a notícia em si (WINER, 2003).

A Tabela 2 lista todas as marcações presentes no RSS 2.0. Nesta tabela, pode-se verificar as sub-marcações que ficam dentro da marcação <*channel*>, ou seja, que descrevem o canal e na Tabela 3 pode-se verificar as sub-marcações de <*item*> que descrevem a notícia.

Tabela 2 – Sub-marcações que representam o canal

Marcação	Descrição	Exemplo	Obrigatório
title	Nome do canal	GoUpstate.com News Headlines	sim
link	A URL do *website* correspondente ao canal.	http://www.goupstate.com/	sim
description	Frase que descreve canal.	The latest news from GoUpstate.com, a Spartanburg Herald-Journal Web site.	não
language	A linguagem do canal	en-us	não
copyright	Direitos de cópia relativa ao canal.	Copyright 2002, Spartanburg Herald-Journal	não
managingEditor	Pessoa responsável pelo conteúdo editorial do canal.	geo@herald.com (George Matesky)	não
webMaster	Pessoa responsável por assuntos técnicos do canal	betty@herald.com (Betty Guernsey)	não
pubdate	Data de publicação da notícia	<pubDate>Sun, 19 May 2002 15:21:36 GMT</pubDate>	não
lastBuildDate	Última vez que o conteúdo do canal foi alterado	Sat, 07 Sep 2002 09:42:31 GMT	não
category	Categoria em que se enquadra a notícia	<category>Sports</category>	não
generator	Nome do programa usado para gerar o canal	MightyInHouse Content System v2.3	não
docs	Documentação para o formato usado no arquivo RSS.	http://www.rssboard.org/rss-specification	não
cloud	Especifica um serviço web que suporta a interface de rssCloud	<cloud domain="rpc.sys.com" port="80" path="/RPC2" registerProcedure="pingMe" protocol="soap"/>	não
ttl	Número de Minutos que indica a frequência de atualização	<ttl>60</ttl>	não
image	Um arquivo de imagem	<image> <url>http://www.w3c.	não

		__com/logo.gif</url> <title>W3c.com</title> </image>	
rating	Avaliação do canal, se, por exemplo, a notícia pode ser vista apenas por maiores de 17 anos.	<rating>(PICS-1.1 "http://www.classify.org/safesurf/" l r (SS~~000 1))</rating>	não
textInput	É uma entrada de texto e contém informações para onde enviar os dados.	<textinput> <description>Search Google</description> <link>http://www.google.br/search?</link> </textinput>	não
skipHours	Horas que não serão publicadas novas notícias no canal.	<skipHours> <hour>0</hour> <hour>1</hour> <hour>23</hour> </skipHours>	não
skipDays	Dias que não serão publicadas novas notícias no canal	<skipDays> <day>Saturday</day> <day>Sunday</day> </skipDays>	não

Tabela 3 - Sub-marcações que representa cada notícia

Marcação	Descrição	Exemplo	Obrigatório
Title	Título da notícia	Festival de Veneza está Aberto	Sim
Link	URL da notícia	http://nytimes.com/2004/ 12/07FEST.html	Sim
description	Sinopse da notícia	<description>O Festival de Filmes de Veneza, um dos mais importantes festivais de cinema do mundo, está acontecendo essa semana.</description>	Sim
Author	Autor da notícia	<author> lawyer@boyer.net (Lawyer Boyer) </author>	não
category	Categoria em que se enquadra a notícia	<category>Sports</category>	não

comments	URL para uma página que comentários da notícia	<comments>http://ekzemplo.com/ entry/4403/comments</comments>	não
enclosure	Arquivo de mídia que uma notícia pode conter	<enclosure url="http://www.scripting.com/ mp3s/weatherReportSuite.mp3" length="12216320" type="audio" />	não
Guid	Valor único que identifica um Item	<guid>http://some.server.com/ weblogItem3207</guid>	não
pubdate	Data de publicação da notícia	<pubDate>Sun, 19 May 2002 15:21:36 GMT</pubDate>	não
Source	Link do canal	<source url="http://www.tomalak.org/links2.xml" >Tomalak's Realm</source>	não

Esses arquivos XML encontram-se disponíveis através de uma URL. Sendo assim, os usuários incluem o *link* desses arquivos *feed* em seu programa leitor de *feed* (agregador) e recebem as informações, sem visitar o site, sobre as atualizações que ocorreram (WINER, 2003). Os *feeds* são usados para que um usuário da internet possa acompanhar os novos artigos e demais conteúdos de um *site* ou *blog* sem a necessidade de visitar o site em si. Sempre que um novo conteúdo for publicado em determinado site, o "assinante" do *feed* poderá lê-lo imediatamente. Entre os principais agregadores de *feeds* da atualidade podemos citar o Mozilla Firefox, Internet Explorer 8, FeedGhost, Juice, Akregator. Além disso, ainda existem algumas ferramentas online como o Google Reader[18] e o Yahoo Pipes[19] que ainda permitem a busca de notícias através de palavras chaves.

Apesar dos arquivos RSS terem um grande potencial e disporem de grande popularidade, os agregadores oferecem opções muito limitadas. A exemplo do cliente de *feeds* do Firefox, os leitores de *feeds* atuais armazenam as notícias e as ordenam por data, não oferecendo recursos adicionais (filtragem considerando o perfil do usuário, palavras-chave, regras, etc). Em virtude dessas deficiências, o usuário muitas vezes recebe um grande volume de notícias, sendo que várias apresentam pouca ou nenhuma relevância para o mesmo.

[18] http://www.google.com/reader
[19] http://pipes.yahoo.com/pipes/

Então, Para resolver o problema de sobrecarga de informação, este trabalho propõe um sistema que utiliza Dados Autonômicos que encapsulam os arquivos RSS com regras. Estas regras filtram as notícias recebidas baseando-se em vários critérios, fornecendo, assim, notícias mais relevantes ao usuário. A arquitetura dessa proposta é mostrada a seguir.

5.2. Arquitetura do RSS autonômico

Resumidamente o sistema é baseado em um servidor que constrói o Dado Autonômico e um cliente que o recebe e o processa. A seguir detalhamos a cada uma das partes do sistema.

5.2.1. RSS+

O RSS+ é uma especialização do Dado Autonômico feito para se adequar ao sistema RSS Autonômico. O RSS+ encapsula notícias RSS juntamente com regras. Para o desenvolvimento do RSS Autonômico, utilizamos a especificação dos *Feeds* RSS 2.0 (WINER, 2003).

Essa escolha foi feita para diminuir a complexidade da aplicação desenvolvida e porque esta é a especificação de arquivos *feed* mais recente e a mais utilizada atualmente, sendo, por isso mesmo, a que apresenta maior quantidade de notícias disponíveis.

Como estrutura de dados, para o armazenamento das notícias, foi escolhida a OWL pela possibilidade da representação das informações como uma ontologia, podendo assim dar maior semântica aos dados. Existem diversos tipos de arquivos que suportam a representação de uma ontologia, mas a OWL foi escolhida por ser a linguagem padrão mantida pela W3C, sendo assim, melhor revisado e tendo uma maior aceitação pelos diversos softwares que lidam com ontologia. Para manipulação de dados OWL foi usada a API do Protégé que permite a manipulação de ontologias na linguagem JAVA, podendo ser obtido no site http://Protégé.stanford.edu/. Para representação das regras foi escolhida a linguagem SWRL pelos mesmos motivos que foi escolhida a linguagem OWL, ou seja, por ser a linguagem de regras padrão da W3C podendo ser usada com a OWL.

Além disso, um Dado Autonômico ainda necessita de um raciocinador, que interpreta regras e dispara decisões. Para esse cargo foi escolhido o JESS. Segundo FRIEDMAN (2003), JESS é um processador de regras e ambiente de execução de *scripts*[20] feito totalmente em Java por Ernesto Friedman-Hill. Entre as vantagens do JESS destaca-se que é um dos poucos raciocinadores que suporta o processamento de regras SWRL. Além disso, é uma aplicação pequena, gasta pouca memória e tem um rápido processamento. Diante disso, o JESS foi escolhido para ser o raciocinador do RSS+.

Para facilitar o desenvolvimento do RSS+, foi feito um diagrama de classes que organiza a divisão das informações contidas no RSS e esclarece como as regras se relacionam com esses dados. As classes e atributos foram todas feitas baseando na especificação F*eeds* RSS 2.0. A conversão das informações de RSS 2.0 para OWL obedece a essa modelagem. A Figura 14 ilustra o diagrama.

[20] conjunto de instruções executado sem a intervenção do usuário

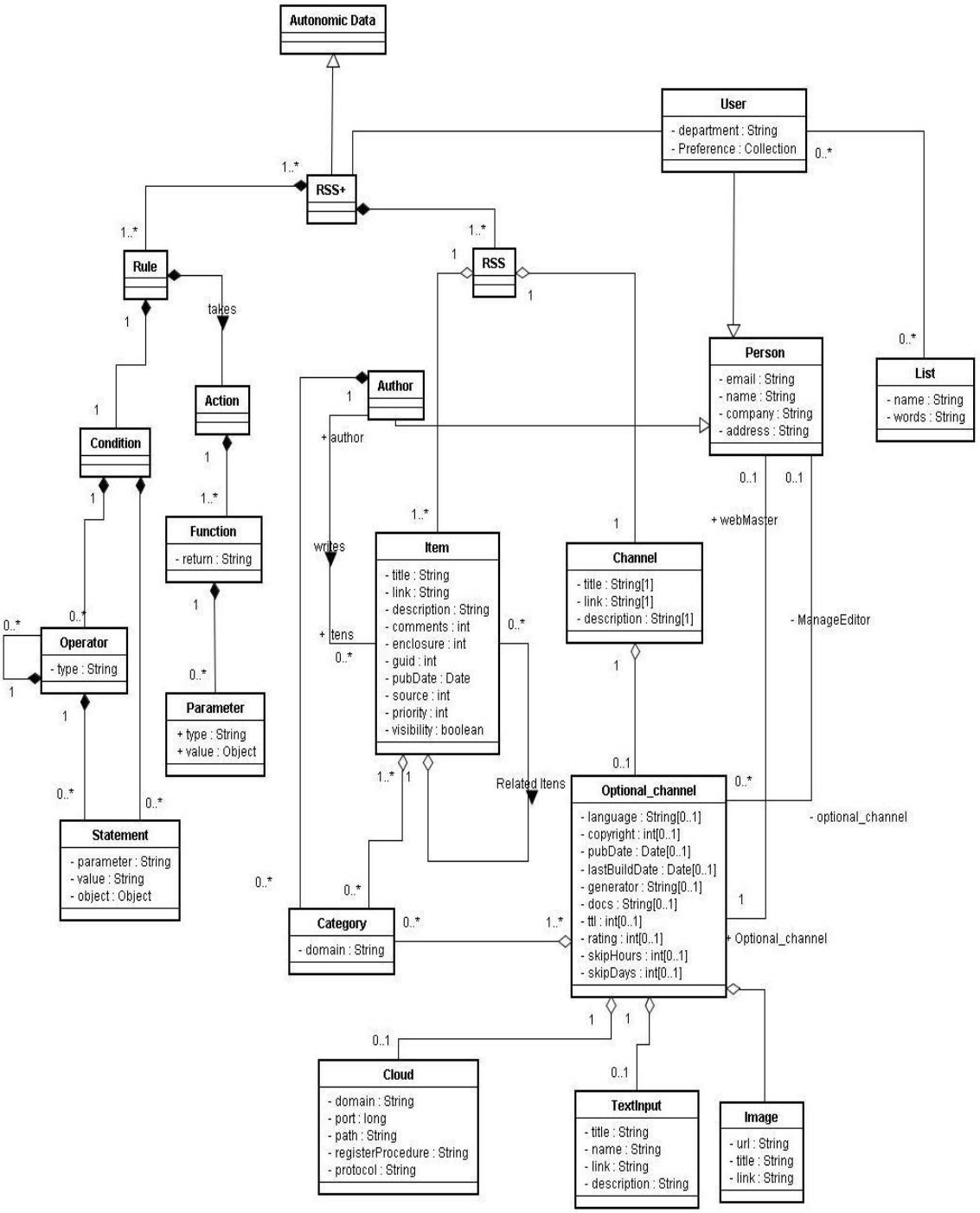

Figura 14 – Diagrama de classes do conteúdo do RSS+

Na figura é possível verificar que o RSS+ é uma especialização de Dado Autonômico, sendo basicamente composto por vários RSS e várias regras. A herança dos Dados Autonômicos obriga a relação entre RSS+ e *Rules* ser não nula, ou seja, é obrigatória a presença de pelo menos uma regra. A classe RSS representa um arquivo RSS. Ele é constituído por 1 canal e por vários *Itens*, sendo que *Itens* são as notícias em si. Algumas das relações presentes nesse diagrama não existem nos arquivos RSS como,

por exemplo, a relação entre *Category* (categoria) e *Author*(autor) e o relacionamento entre os vários *Itens*. Essas relações foram criadas para dar maior semântica aos dados e são obtidas por inferência. Por exemplo, se um autor escreve uma notícia da categoria esportes, então, essa categoria é atribuída automaticamente ao autor.

Sobre a parte do diagrama que descreve as regras (parte mais a esquerda da figura), destaca-se que foi implementada para ser a mais geral possível, podendo interpretar as mais diversas regras. Porém, como é utilizado na aplicação proposta o SWRL como linguagem de regras, o diagrama acaba sofrendo com as limitações do mesmo. Por exemplo, a classe *Operator* poderia suportar os operadores de junção, disjunção e negação, mas a SWRL apenas aceita junção de termos. Deste modo, essa generalidade foi perdida neste caso. Mas, esse modelo não serve apenas para a aplicação construída, e dessa forma, se for usado em outra aplicação, toda a potencialidade de descrição de regras pode ser utilizada.

5.2.2. Servidor RSS+

No sistema é necessária a existência de um servidor que construa a instância do RSS+ a ser consumido. Este papel fica com o servidor RSS+. O servidor é responsável por fazer o download dos arquivos XML dos *Feed* RSS. Esses arquivos XML são convertidos para OWL, obedecendo à modelagem da Figura 14, e são incluídas regras SWRL, criadas pelo servidor, além do OWL *Schema* (que descreve a estrutura dos arquivos). As regras construídas pelo servidor são mandatárias em relação às regras que o usuário incluir. Um exemplo de regra que pode ser incluído pelo servidor seria: se for imaginado que essa ferramenta vá ser usada por uma companhia, de modo a permitir que os funcionários recebam notícias, esta pode desejar que seus funcionários não recebam notícias da concorrente. Porém, a equipe de marketing deve receber notícias relacionadas ao concorrente, para elaborar as estratégias de captação de clientes. Sendo assim, a regra seria: se o usuário não for do departamento de marketing não receberá notícias do concorrente.

Depois de ser criado no servidor, o RSS+ chega ao cliente. Neste por sua vez são incluídas as regras criadas pelo usuário. Dessa forma, os conjuntos de regras semânticas vindos do servidor em conjunto com as regras do usuário irão determinar as notícias a

serem exibidas para o usuário. Com isso, um mesmo arquivo enviado para clientes diferentes pode derivar em resultados diferentes.

5.2.3. Cliente RSS+

Como dito anteriormente, para acessar o RSS+ é usada a parte cliente do RSS Autonômico que fornece uma interface de comunicação com o usuário, onde são apresentadas as notícias do interesse do leitor. Para utilizar a ferramenta, o usuário deverá criar uma conta, fornecer os seus dados pessoais, assinar os *Feeds* que desejar, selecionar os amigos entre os usuários existentes e preencher algumas de suas preferências em relação a notícias que resultarão nas regras para filtragem de notícias. Maiores detalhes de como funciona a ferramenta são dados no capítulo 6.

A Figura 15 ilustra todo o sistema. É possível ver o servidor RSS+ realizando o download do arquivo XML de um fornecedor de RSS, fazendo a conversão para RSS+ e então o repassando para os diversos clientes. Também é possível ver o aspecto colaborativo onde os vários clientes fazem troca de informações.

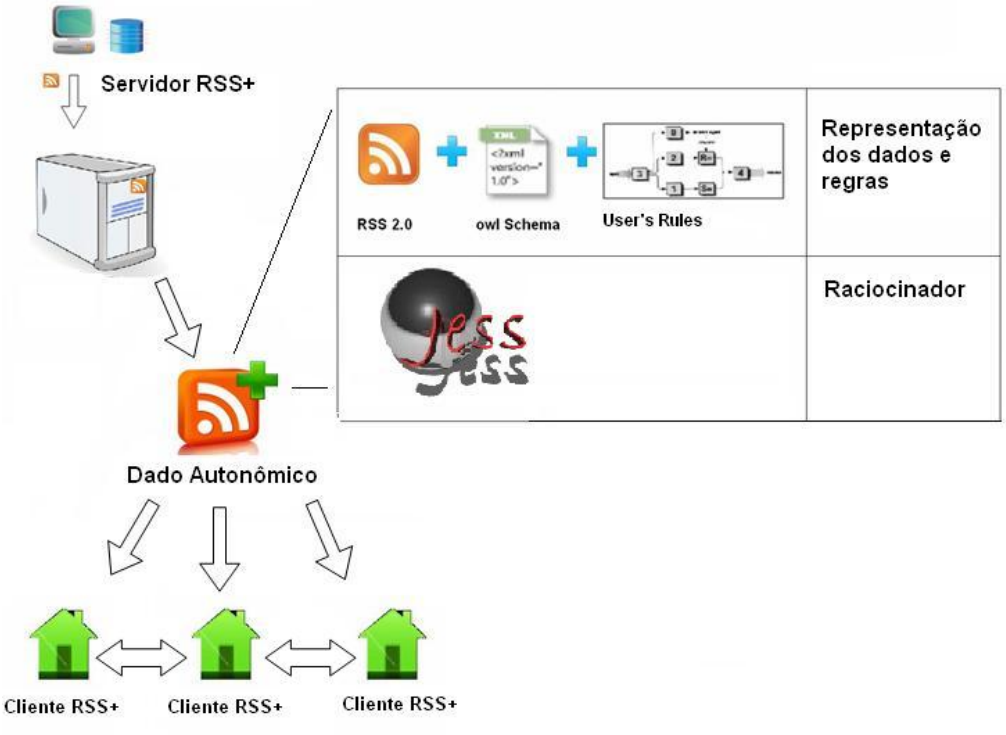

Figura 15 – Arquitetura do Sistema

5.3. Problemas relacionados ao RSS e soluções propostas

Como relatado anteriormente, o formato RSS tem um grande potencial, mas esse potencial esbarra em alguns problemas que serão discutidos a seguir. Para verificar como é o uso dos arquivos RSS e verificar como eles são distribuídos foram coletados arquivos RSS de diversas categorias (cultura, economia, esportes, política, meio ambiente, entretenimento, saúde e tecnologia) de grandes veículos de comunicação. Os veículos escolhidos foram Herald Tribune (http://www.heraldtribune.com/), New York Times (http://www.nytimes.com/), The Guardian (http://www.guardian.co.uk), Washington Post (http://www.washingtonpost.com/) , The Sun (http://www.thesun.co.uk) e Financial Times (http://www.ft.com/). Estes veículos foram escolhidos devido à relevância internacional destes, além de terem em acervo notícias com o idioma inglês que é primordial devido à existência de várias ferramentas para o tratamento de textos somente no idioma inglês.

Entre os pontos interessantes descobertos nessa pesquisa pode-se destacar a falta de padrão, onde não existe um consenso para divulgar informações. Toda essa falta de padrões dificulta a identificação e a análise de informações quando se recebe notícias dos mais diversos veículos de comunicação. Todos os problemas identificados foram divididos nos seguintes tópicos.

5.3.1. Marcações Dublin Core

Apesar da existência de padrões e recomendações de marcação para os documentos RSS, foram encontradas algumas marcações fora desse padrão. Isso ocorre devido a possibilidade de extensão de marcações que os documentos RSS permitem. Entre as marcações encontradas a mais comum foram as do tipo Dublin Core.

O Dublin Core (DC) é um padrão de metadados, composto por 15 elementos, planejado para facilitar a descrição de recursos eletrônicos. O padrão Dublin Core (DC) é mantido pela *The Dublin Core Metadata Initiative* que é uma organização aberta engajada com o desenvolvimento de padrões de metadados online inter-operáveis que suporta uma grande quantidade de modelos de negócio e outros propósitos.

SOUZA *et al*. (2000) destaca que muitos veículos utilizam esse padrão devido a algumas vantagens, tais como: a simplicidade na descrição dos recursos, o entendimento semântico universal (dos elementos), o escopo internacional e a extensibilidade (o que permite adaptações às necessidades adicionais de descrição). O DC pode ser inserido em uma página HTML e utiliza a linguagem XML. Adota a sintaxe do RDF, possui um conjunto de 15 elementos básicos, que podem ser implementados livremente para atender as necessidades de cada usuário e, ainda é um formato padrão adotado para efetuar a interoperabilidade entre outros formatos. (DUBLIN CORE METADATA INITIATIVE, 2004). Na Tabela 4 podemos verificar todos os elementos e sua descrição.

Tabela 4 - Elementos metadados Dublin Core adaptado de (ALVES; SOUZA, 2007)

Title(Título)	Nome dado aos recursos
Creator(Criador)	Entidade originalmente responsável pela criação do conteúdo do recurso
Subject(Assunto)	Tema do conteúdo do recurso. Pode ser expresso em palavras-chaves e/ou categoria. Recomenda-se o uso de vocabulários controlados
Description(Descrição)	Relato do conteúdo do recurso. Exemplos: sumário, resumo e texto livre
Publisher(Publicador)	Entidade responsável por tornar o recurso disponível
Contributor(Colaborador)	Entidade responsável pela contribuição intelectual ao conteúdo do recurso
Date(Data)	Data associada a um evento ou ciclo de vida do recurso
Type(Tipo)	Natureza ou gênero do conteúdo do recurso. Exemplos: texto, imagem, som, dados, software
Format(Formato)	Manifestação física ou digital do recurso. Exemplos: html, pdf, ppt, gif
Identifier(Identificador)	Referência não-ambígua (localizador) para o recurso dentro de dado contexto
Source(Fonte)	Referência a um recurso do qual o presente é derivado
Language(Idioma)	Língua do conteúdo intelectual do recurso
Relation(Relação)	Referência para um recurso relacionado
Coverage(Cobertura)	Extensão ou escopo do conteúdo do recurso; pode ser temporal e espacial
Rights(Direitos)	Informação sobre os direitos assegurados dentro e sobre o recurso

Nos arquivos RSS as marcações Dublin Core aparecem com os nomes dos metadados citados acima, precedidos de "dc". Assim, se consideramos o campo "*creator",* por exemplo, ter-se-ia o seguinte:

<dc:creator> Jonh Connors </dc:creator>

Mas, apesar das vantagens do uso do padrão Dublin Core, o seu uso não foi modelado no sistema RSS autonômico, pois não faz parte da semântica do mesmo. Dessa forma, para resolver esse problema o sistema faz um mapeamento das marcações Dublin Core para a correspondente do formato RSS 2.0. Os mapeamentos podem ser visto na Tabela 5:

Tabela 5 – Mapeamento entre os valores das marcações Dublin core e os do RSS 2.0

Marcação em Dublin core	Correspondente em RSS 2.0
<dc:title>	<title>
<dc:creator>	<author>
<dc:subject>	<category>
<dc:description>	<description>
<dc:Publisher>	< managingEditor>
<dc:contributor>	Não existe uma marcação equivalente, mas essa informação foi colocada juntamente com a <author>
<dc:date>	<pubDate>
<dc:type>	Não existe uma marcação equivalente, mas essa informação é colocada junto com a <description>
<dc:format>	<docs>
<dc:identifier>	<guid>
<dc:source>	<source>
<dc:language>	<language>
<dc:relation>	Não existe marcação equivalente no RSS 2.0, mas essa informação é colocada junto com a marcação <item>.

<dc:coverage>	Não existe uma marcação equivalente, mas essa informação é colocada junto com o <enclosure>
<dc:rights>	<copyright>

5.3.2. Marcação <author>

Os valores contidos nos arquivos RSS relativos aos autores não seguem um padrão, sendo que os veículos de comunicação preenchem esses valores seguindo critérios próprios. Dessa forma, um mesmo autor pode escrever várias notícias, mas ter seu nome divulgado de formas diferentes, dificultando assim a identificação do mesmo. Por exemplo, um mesmo autor pode aparecer referenciado pelo nome:

<author> by John Connors </author>

ou ainda pelo e-mail:

<author> Johnconnors@ht.com</author>

Além disso, em alguns casos, são escritas outras informações que podem atrapalhar a identificação de um mesmo autor, como a escrita do email em conjunto com o nome:

<author> done by John Connors (email:John@ht.com)</author>

Esse conjunto de fatores dificulta a identificação dos autores. Dessa forma, para contornar esse obstáculo, o Dado Autonômico possui regras que realizam uma auto-configuração, tratando o valor obtido para encontrar o autor correto. O tratamento é baseado em regras de expressões regulares e no acesso a uma base de dados de autores. Através da análise de expressões regulares, é possível extrair as informações e verificar, por exemplo, se existe um e-mail nesse campo e então compará-las com a base de dados existente para recuperar o autor correto.

Adicionalmente, pode existir mais de um autor com o mesmo nome. Por conta disso, todos os autores com esse nome são retornados. Existe ainda a possibilidade de um autor abreviar seu nome do meio, então, se existir a coincidência de 2/3 dos nomes entre a regra do usuário e o nome do autor, este é retornado. Como exemplo, se o

usuário buscar por “NEIL LEWIS” outras ocorrências como “NEIL A. LEWIS” serão retornadas. Isso pode implicar na ocorrência de dados irrelevantes (um autor com nome parecido), mas garante que um possível dado relevante não seja descartado por estar escrito de forma diferente.

5.3.3. Marcação <category>

A parte de categorias é ainda mais complicada que a de autores. Foi verificado que cada veículo de comunicação define o seu nome para a categoria, e, assim, duas notícias relativas a uma mesma categoria podem ter a descrição de suas características (valores na marcação *category*) totalmente diferentes. No RSS do New York Times, por exemplo, as categorias são bem específicas. Considere duas notícias sobre negócios, uma pode ter a categoria “*trade and market*” e a outra “*stoke exchange*”, dificultando o agrupamento de notícias semelhantes por categoria.

Para resolver esse problema foi necessário uma auto-classificação de dados, para não depender da classificação sem padronização das mídias. A auto-classificação é um assunto antigo e teve BORKO (1963) como primeiro trabalho importante. Porém, neste trabalho, os algoritmos eram ainda ineficientes chegando a pouco mais de 45% de acerto. A auto-classificação evoluiu desde então, começaram a ser usadas técnicas de inteligência artificial, bases de conhecimento e ontologias. Como exemplo disso, tem-se os trabalhos de LIN (1995) e ANTONIE & ZAIANE (2002). Sendo assim, atualmente, para determinados contextos, a auto-classificação já atinge quase 100%.

Baseado nessas técnicas foi escolhido utilizar uma técnica de associação de termos conhecida como “vizinho mais próximo” presente em ANTONIE & ZAIANE (2002) para resolver este problema. A idéia neste caso é, através das categorias fornecidas, generalizar as notícias para domínios específicos. No exemplo citado das categorias “*trade and market*” e “*stoke exchange*”, ambos seriam generalizados para a categoria “economia” através da relação entre as palavras. Inicialmente, para este trabalho, foram pré-definidas as seguintes categorias gerais: economia, esportes, entretenimento, política, meio ambiente, saúde, tecnologia e notícias gerais. Para que seja feita essa generalização, verifica-se a relação entre as palavras, consultando uma ontologia de termos, extraído do Wordnet.

O Wordnet é um amplo banco de dados léxico do idioma inglês, desenvolvido sobre a direção de George A. Miller. Substantivos, verbos, adjetivos e advérbios são agrupados em conjuntos de sinônimos cognitivos que são chamados de synsets. Cada um destes expressa um conceito diferente e são interligados pelo significado de sua semântica e por sua relação léxica. Através dessas relações, é possível produzir combinações de dicionários e tesauros, e fornecer suporte para análise de textos automática e aplicações de inteligência artificial. Wordnet é gratuito e pode ser usado livremente (WORDNET, 2009).

A utilização do Wordnet ocorre da seguinte forma: um termo é pesquisado na base de dados do Wordnet, então é feita a navegação nas relações semânticas do termo até encontrar um dos termos gerais (já pré-definidos na ontologia), citados acima, e o termo encontrado é retornado. Na Figura 16 temos o exemplo da palavra "White House", na navegação entre as relações semânticas consegue-se chegar ao termo política, sendo assim a palavra "White House" é enquadrado na categoria de política.

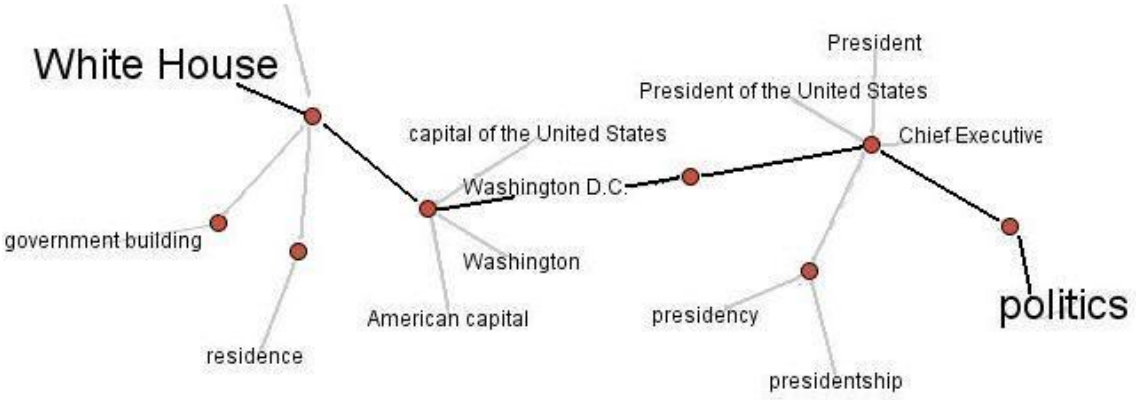

Figura 16 – Relação semântica entre as palavras no wordnet

A técnica proposta para esse caso não é tão poderosa nem tão sofisticadas quanto às observadas na literatura, porém, contorna o problema específico relatado anteriormente.

Poderíamos imaginar que a generalização de notícias pode não ser tão útil caso o usuário deseje receber notícias de uma categoria específica. Por exemplo, se a pessoa deseja notícias sobre esportes, então ela pode assinar um *feed* de esportes. Porém, isso não é tão simples, pois, hoje em dia as notícias estão bem entrelaçadas com os diversos canais e conseguimos encontrar notícias de categorias diferentes, mesmo em *feeds* bem específicos. É possível encontrar notícias que estejam relacionadas à saúde em um canal

de esportes, meio ambiente em um canal entretenimento ou política em um canal de negócios. Considerando isso, uma pessoa pode encontrar uma notícia de esportes dentro de um *feed* específico de negócios (a estratégia usada por um clube para o marketing de venda de ingressos). Assim, através dessa técnica, busca-se identificar as várias categorias gerais em que uma notícia se enquadra independente do canal que esta se encontra.

5.4. Listas Brancas e Negras

A técnica de listas brancas e negras consiste respectivamente em uma lista que pode permitir ou bloquear. Essas técnicas são muito utilizadas no âmbito da Ciência da Computação, principalmente na internet, para bloquear sites indesejados. A seguir é discutido sobre o uso dessa técnica em alguns campos como os spams.

5.4.1. Spams

Junto com a facilidade de troca de informações surgiu uma prática danosa chamada de Spam. O Spam consiste do envio de mensagens em massa objetivando principalmente a disseminação de propagandas de produtos e serviços, disseminação de vírus, correntes de boatos e mensagens ofensivas ou golpes. Está pratica é bem conhecida dos usuários de internet principalmente na ferramenta de email. Ao longo dos anos o Spam foi se expandindo e passou a ser disseminado em outras plataformas como jogos online, em mensagens de celular e blogs. Isso fez com que usuário tenha cada vez mais cuidado com as informações que recebe.

Segundo (K SOFT, 2009), com a popularidade das notícias via Feeds RSS a prática de Spam também passou a ser exercida sobre essa área. Segundo esse autor, as práticas de spam sobre Feeds RSS podem acontecer sob 3 aspectos:

A primeira é a de inclusão de palavras-chave para um tópico especifico. Isso não faz diferença para humanos, mas traz diferenças na busca de notícias pelos robôs de busca para redirecionar para um site web. Essa técnica é antiga e é utilizada também com relação a paginas, mas esta é barrada pela maioria dos sites de busca.

A segunda técnica envolve a inclusão de links nos Feeds RSS: Esses artigos geralmente contêm muito pouco ou nenhum conteúdo. A principal atração é o título. Ao

clicar no título do *feed*, o usuário é direcionado a um blog contendo centenas de outros blogs e Feeds RSS, cada um redirecionando para mais links. O objetivo desse tipo de spam RSS é enganar o usuário para clicar em propagandas e redirecioná-lo para sites de um produto.

A terceira envolve a criação de um RSS falso. É feita a duplicação inteira de um *Feed RSS*, com todo o conteúdo, mas são retirados os anúncios das empresas e são colocados outros. Geralmente não prejudica os usuários com relação ao conteúdo.

Uma forma de combater esses problemas é através da técnica de listas negras. As listas negras são listas contendo mensagens que foram classificadas como spam. Sendo assim, quando uma mensagem é recebida, suas informações são comparadas com a da lista, caso seja encontrado, a mensagem é classificada como spam e então é descartada.

A aplicação deste trabalho utiliza essa técnica para barrar mensagens enquadradas como spam.

5.4.2. Listas para Conteúdos indesejados

As notícias que o usuário recebe podem possuir um conteúdo não desejado. Por exemplo, alguns usuários não gostam que existam palavras pornográficas na notícia, e seria do desejo destes que não recebessem essas notícias. Contudo, existem ainda outras notícias que o usuário não gostaria de receber como notícias relacionadas à violência, política ou religião. Sendo assim, o usuário poderia aplicar um filtro para não receber essas notícias. Contudo, a solução não é tão simples, pois uma notícia com essas característica pode ser difícil de identificar se não houver uma palavra bem explícita no título ou na categoria. O usuário pode tentar barrar textos que contenham algumas palavras relacionadas a esse assunto, porém, o número de palavras que caracterizam um assunto é grande, fazendo com que esse filtro consiga barrar somente algumas notícias se tornando ineficaz.

Uma solução para esse problema é o uso de listas negras de palavras por assunto. É usada, por exemplo, uma lista negra sobre violência que vai conter um conjunto de palavras relacionadas a esse assunto. Sendo assim, o texto da notícia é comparado à lista, se o número de ocorrências for significativo, então o texto é enquadrado dentro da categoria violência e pode ser barrado se esse for o desejo do usuário.

As listas foram montadas conforme o site thesaurus.com que contém relação entre as diversas palavras. A aplicação do RSS Autonômico permite ao usuário escolher entre usar as listas negras de violência, pornografia, política e religião.

5.5. Funções implementadas

Além de sugerir possíveis soluções para problemas identificados na análise do arquivo RSS, este trabalho propõe a utilização de algumas funções que podem ajudar na filtragem das notícias. Essas funções foram baseadas nos trabalhos de PINHEIRO *et al* (2008) e PINHEIRO *et al* (2009a)

Para que ocorra o uso das funções, estas devem ser descritas em OWL na formas de regras, sendo que as especificações de execução das funções ficam necessariamente na parte antecedente da regra, mantendo o Dado Autonômico encapsulado. Assim, mesmo que um sistema venha a ter uma nova função, uma regra contendo essa função deve ser criada no Dado Autonômico para a sua utilização. Nesse caso, um Dado Autonômico pode ter uma regra que usa uma função não presente no sistema (afinal, o Dado Autonômico pode ir de um sistema para outro levando regras consigo, e nem todas as funcionalidades de um sistema existem em outros). Dessa maneira, essa regra não será executada a menos que se implemente essa função.

Como o Dado Autonômico suporta a inserção de novas regras, estas podem ser inseridas, bastando para isso estarem no formato reconhecido pelo Dado Autonômico.

As funções implementadas neste trabalho têm como objetivo comparar palavras chaves fornecidas pelo usuário, com informações das notícias, e inferir se as notícias são relevantes para o usuário. A seguir são descritas essas funções.

5.5.1. Seleção de Dados Permitidos

Em muitos casos os usuários podem escolher selecionar notícias por palavras chaves. Na maioria das vezes, eles preferem visualizar notícias que contenham algum termo específico, como o time favorito ou uma marca de roupa. Dessa forma, a função de seleção de dados permitidos foi implementada com esse objetivo.

Mas a função não é apenas uma busca por termos exatos, pois isso torna a busca muito limitada, restringindo-se a encontrar apenas a palavra exata. Por exemplo, se o

usuário digitar "*corredor*", seria apropriado se encontrasse textos contendo as palavras "*correr*" ou "*corrida*", mas a busca por palavras exatas não encontraria esses textos. Assim, para contornar esse problema é usado o recurso de *Stemming.* Segundo AHMED & MITHUN (2004), *Stemming* é a técnica de reduzir uma determinada palavra para o seu radical, ou seja, são retirados os sufixos e prefixos de modo que esta se reduza a sua forma raiz. Assim, no caso citado anteriormente a palavra corredor é reduzida para "corre". No texto da notícia essa técnica também é aplicada de forma que "correr" e "corrida" também são reduzidos para "corre" e assim a palavra chave é encontrada. Para esta pesquisa, utiliza-se o algoritmo Snowball Stemmer para a realização do *Stemming* e, apesar do exemplo com palavras em português, consideram-se apenas textos de notícias em inglês.

Depois de realizar o *Stemming* no texto é usado o coeficiente de Jaccard para verificar se o conjunto de palavras fornecido na entrada é considerado similar ao texto analisado. Segundo BAEZA-YATES & RIBEIRO-NETO (2007), o coeficiente de Jaccard é uma medida de similaridade entre dois conjuntos e é definido como o tamanho da interseção dividido pelo tamanho da união desses conjuntos. A equação (1) demonstra o coeficiente de similaridade de Jaccard, sendo A e B os dois documentos comparados:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \qquad (1)$$

Usando o coeficiente de Jaccard é necessário ainda fornecer o nível de similaridade desejado, com objetivo de comparar este valor com o valor obtido pelo coeficiente. Se o valor obtido pela função for igual ou maior que o nível de similaridade fornecido, diz-se que os textos são similares. Para efeitos de exemplificação, vamos imaginar que exista um documento A que será composto de uma palavra, representando uma busca de um usuário, e que exista um documento B que será o texto buscado, composto por 100 palavras. Se o nível de similaridade escolhido for de 2%, é necessário que existam pelo menos duas ocorrências da palavra do documento A no documento B para que ambos sejam considerados similares.

Resumindo, a função busca por similaridade ao texto, compara o conteúdo de um ou mais atributos da notícia com o conteúdo de um parâmetro. Assim, toda a notícia que tiver(em) o(s) atributo(s) considerado(s) similar(es) ao parâmetro é retornada.

5.5.2. Negação de Dados Proibidos

É a função inversa da descrita anteriormente. Em alguns casos os usuários não querem receber notícias que contenham determinados termos, entre eles podemos citar o time rival ou uma marca que não agrada. Além disso, a escolha de certos termos pode evitar a ocorrência de spams ou pornografias. Esta função funciona com o mesmo principio da anterior comparando o conteúdo de um ou mais atributos das notícias com um parâmetro. Mas, nesse caso, toda notícia que tiver o(s) atributo(s) considerado(s) similar(es) ao parâmetro não é retornada, sendo então descartada.

5.5.3. Descarte de Dados Similares

Um dos maiores problemas de se assinar vários *Feeds RSS* é a ocorrência de notícias similares ou repetidas. Notícias de maior relevância são noticiadas por vários veículos de comunicação, e sendo assim, se o usuário assina *feeds* de canais diferentes sobre um mesmo assunto, com certeza irá receber notícias repetidas. Evitar isso é o propósito dessa função. Para fazer isso é utilizado novamente o coeficiente de Jaccard, e o usuário fornece qual o nível de similaridade desejado. Os textos das notícias são comparados um a um, se essa comparação obtiver um valor maior ou igual ao da similaridade escolhida, então a notícia mais antiga é descartada. Isso é feito até que todas as notícias tenham sido comparadas.

Resumindo, os dados das notícias são comparados entre si e, se existirem similaridades de acordo com o valor escolhido, as notícias mais antigas são descartadas e somente a mais recente é retornada.

5.6. Auto Otimização de Regras

Com a utilização da ferramenta é possível obter notícias que atendam às regras inseridas. Porém, às vezes as regras inseridas pelo usuário podem não corresponder ao desejo deste. Entre as razões podemos citar a severa restrição da regra que não encontra nenhum resultado, regras mal-formuladas, falta de notícias daquele interesse, ou mesmo porque podem existir notícias do interesse do usuário sem que ele saiba da existência das mesmas. Sendo assim, é proposta a auto-otimização das regras onde a ferramenta altera as regras do usuário com o objetivo de obter resultados mais relevantes para este. A auto-otimização se baseia no número esperado de notícias do usuário e acontece em duas vertentes, remoção de regras se o resultado for muito restrito e adição de regras caso o resultado seja muito amplo.

Primeiramente, é verificado se com as regras fornecidas é possível encontrar a quantidade de notícias esperadas. E, caso o resultado seja muito restrito, o sistema irá remover as regras de menor prioridade (prioridade esta que o usuário fornece ao inserir a regra), até que seja obtido o número de resultados esperados.

Caso o resultado seja muito amplo, o sistema irá buscar regras dos amigos do usuário (amigos estes, escolhidos pelo próprio usuário entre os outros usuários do sistema) para restringir o resultado. A busca por regras do usuário segue algumas normas. A primeira norma é que são buscadas as regras de usuários que tenham um perfil similar. Para calcular a similaridade entre os perfis são comparados os dados de canais preferidos, categorias de notícias preferidas, palavras chaves, companhia e departamento que trabalha. Para cada um desses quatro quesitos é dado o peso de um quarto. Todos esses campos devem ser fornecidos pelo usuário.

Na segunda norma, observa-se outro campo fornecido pelo usuário, que são as palavras proibidas. Essas palavras proibidas não são usadas para o cálculo de similaridade, elas são usadas em segunda instância e servem para que não sejam adicionadas regras que estejam fora do interesse do usuário. Um exemplo da importância dessa norma é que se um usuário A tem um perfil de receber notícias de futebol, este poderá receber regras de um usuário B com este perfil. Porém, o usuário A pode torcer pelo time Flamengo e não gostar de receber notícias do Vasco, que pode ser justamente o time pelo qual o usuário B torce. Dessa forma, se A colocar ‘Vasco’ dentre as palavras proibidas não correrá o risco de receber notícias relativas a ‘Vasco’.

A terceira norma é a compatibilidade da regra a ser adicionada. Existem regras que se forem unificadas não irão produzir nenhum resultado. Como exemplo pode-se citar regras de datas: se o usuário tiver uma regra para obter notícias de 3 a 7 de agosto, não pode ser adicionada uma regra para obter notícias de 8 a 9 de agosto, pois a conjunção dessas regras não iria trazer nenhum resultado.

Por fim, a quarta norma verifica o nível de seletividade da regra, para que seja adicionada uma regra em que a conjunção resultante consiga obter o número de resultados mais próximo possível do esperado pelo usuário. O fator de seletividade é obtido através do histórico de consultas e fica no próprio Dado Autonômico.

Resumindo, são obtidas as regras dos usuários amigos com perfil semelhantes, destas regras são excluídas as que possuem palavras proibidas e as regras incompatíveis, e por fim elas são ordenadas pelo nível de seletividade de cada uma.

5.7. Características Autonômicas do Sistema

Depois de descrever a arquitetura do sistema e todas as funcionalidades implementadas, é chegada a hora de reapresentar estes tópicos sob o ponto de vista dos 10 mandamentos do Dado Autonômico. A aplicação foi construída seguindo esses mandamentos de modo a prover um sistema com todas as vantagens descritas no capítulo anterior. Segue abaixo as especificações do sistema olhando pelo ponto de vista dos 10 mandamentos do Dado Autonômico.

1 – Deveis ter conhecimento sobre si mesmo:

O RSS Autonômico carrega consigo um pouco da semântica do seu ambiente de origem em forma de regras e meta-informações. Com isso, este tem conhecimento sobre seu conteúdo para saber a que veio e como validar as informações que obtiver no sistema em que está. Dessa forma, este sabe as especificações dos dados e saberá se um determinado dado inserido condiz com sua semântica.

2 – Deveis Conhecer teu ambiente e o contexto ao redor de tuas atividades:

Ao chegar a um novo sistema, o RSS Autonômico imediatamente verifica as regras existentes do usuário e já as incorpora. Além disso, este tem acesso às regras dos

demais usuários, assim ele pode verificar quais são as melhores regras, e incorporá-las se isso for benéfico. Sendo assim, o RSS+ tem acesso as regras que foram inseridas através do servidor e as regras dos demais usuários do sistema.

3 – Capacidade de se movimentar para outros ambientes, quando necessário:

Como o RSS Autonômico fornece interfaces para acesso a seus dados, através da API do Protégé, ele pode se movimentar livremente por outros sistemas, desde que eles implementem as interfaces de comunicação exigidas.

4 – Deveis fornecer a informação adequada ao ambiente em que se encontra, escondendo a sua complexidade:

O RSS Autonômico esconde toda a complexidade com relação aos seus dados. Os sistemas não necessitam conhecer detalhes do arquivo OWL, pois, através da interface de comunicação, este provê todas as informações que são necessárias para o sistema.

5 – Deveis cooperar uns com os outros visando aperfeiçoar a si mesmo e ao conjunto:

O RSS Autonômico provê interfaces para troca de regras entre os diversos RSS+ existentes. Dessa forma, este pode se auto-otimizar conseguindo as melhores regras podendo ajudar a otimizar todo o grupo, fornecendo regras para que os demais possam conseguir aperfeiçoar-se também.

6 – Deveis encapsular informações e forneceras uma interface de comunicação para que outro sistema possa acessar teus dados:

O RSS Autonômico é constituído de um arquivo XML descrevendo uma ontologia, regras SWRL e chamadas de função. Para o desenvolvedor do sistema, esses detalhes não necessitam ser conhecidos, pois o RSS Autonômico fornece uma interface de comunicação através da API do Protégé. Dessa forma, os detalhes de construção do dado ficam encapsulados e o desenvolvedor do sistema não precisa preocupar com os detalhes da construção dos dados para recuperar ou inserir elementos.

7 – Deveis Configurar e Reconfigurar sob diferentes condições e ambientes:

O RSS Autonômico se auto-configura em relação aos valores relativos a autores de notícias e categorias. O RSS Autonômico encontra as categorias relativas navegando através da relação entre os termos do Wordnet. Além disso, ele auto-configura a conversão de outras marcações, como a Dublin core, para as marcações do RSS 2.0.

8 – Deveis nunca estar satisfeito com a situação atual do ambiente, deveis sempre procurar otimizações:

O RSS Autonômico procura otimizações em relação à filtragem de notícias. Caso o número de notícias resultantes seja muito alto, a ferramenta irá procurar automaticamente outras regras através dos amigos de forma a dar uma melhor filtragem para o usuário. Além disso, pode ocorrer de o conjunto de regras deixarem a filtragem muito restrita, resultando em poucas notícias retornadas. Dessa forma, o RSS Autonômico irá retirar algumas das regras do usuário para que a pesquisa retorne mais resultados.

9 – Deveis se curar:

Não existe nenhuma função de auto-cura no RSS Autonômico.

10 – Deveis Proteger a si mesmo:

A proteção do RSS Autonômico se da através das listas negras. A lista negra relativa a bloqueio de spams já é incluída automaticamente. Além disso, existem as listas negras de conteúdo, para bloquear certos conteúdos. A auto-proteção utiliza esse recurso. O sistema identifica tendências sobre assuntos que estão incomodando os amigos de um usuário A. A identificação dessa tendência acontece da seguinte forma: se mais de 20% dos amigos do usuário A escolherem uma determinada lista negra, o usuário A receberá essa regra automaticamente protegendo-o desses assuntos.

Capítulo 6. Experimento

Este capítulo apresenta experimentos que objetivam avaliar a viabilidade de uso dos Dados Autonômico através da análise de resultados obtidos pela ferramenta RSS Autonômico. Os principais objetivos foram o de verificar se a ferramenta realmente obteve propriedades autonômicas com o uso de Dados Autonômicos e se é eficaz na busca de notícias. Além disso, é apresentada a interface da ferramenta e sua navegabilidade.

A avaliação é baseada numa análise comparativa dos resultados obtidos com o uso da ferramenta e das opiniões de usuários sobre a mesma.

6.1. Interface da Ferramenta e Navegabilidade

No capítulo anterior foi apresentada a arquitetura da ferramenta RSS Autonômico. E, além da arquitetura, foram relatadas todas as suas funcionalidades e quais foram as soluções adotadas para os diversos problemas que surgiram. Nessa subseção é apresentada a interface da ferramenta com o usuário.

Para utilizar a ferramenta, primeiramente, o usuário deverá realizar um acesso (*login*). Caso não possua uma conta, pode criar uma clicando no botão *"new Account"* e fornecendo alguns dados. A Figura 17 mostra a tela de *login* e a Figura 18 mostra a tela de cadastro de usuário.

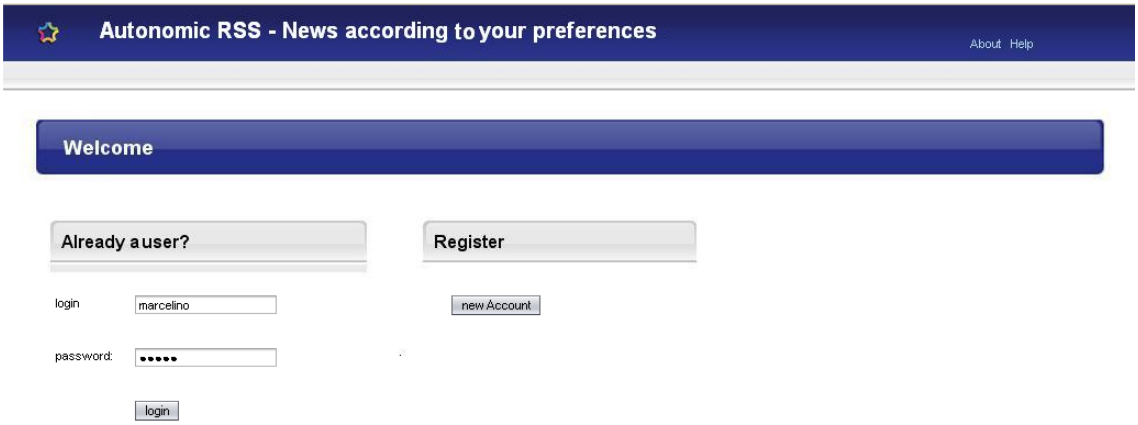

Figura 17 – Ferramenta RSS Autonômico – Tela de login.

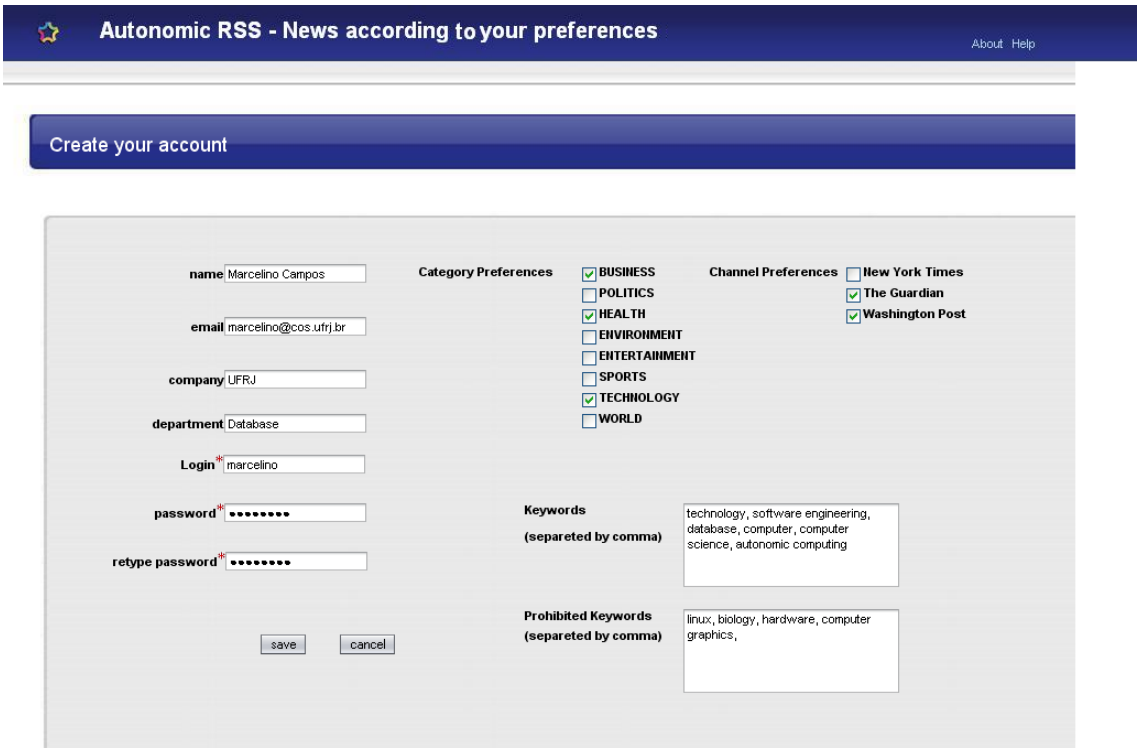

Figura 18 – Tela de Cadastro de Usuário

Logo após, o usuário é redirecionado para tela principal onde este pode editar suas informações pessoais, tendo acesso às diversas funcionalidades do sistema. A primeira ação a ser adotada pelo usuário é a escolha de quais canais RSS ele deseja

receber notícias. Assim, ele terá acesso à funcionalidade “Visualizar Notícias”. Assim, caso ainda não tenha definido nenhum canal RSS, o sistema irá mostrar uma mensagem de erro explicando que o usuário deve selecionar pelo menos um canal RSS. Na tela principal o usuário terá acesso às configurações gerais, visualização de amigos, tela de manipulação de canais RSS e visualização das notícias selecionadas. A Figura 19 ilustra a tela principal. Já na tela de manipulação de canais RSS é possível inserir e remover canais (através de seus respectivos links Web) de preferência do usuário. A Figura 20 ilustra a tela de manipulação de canais RSS.

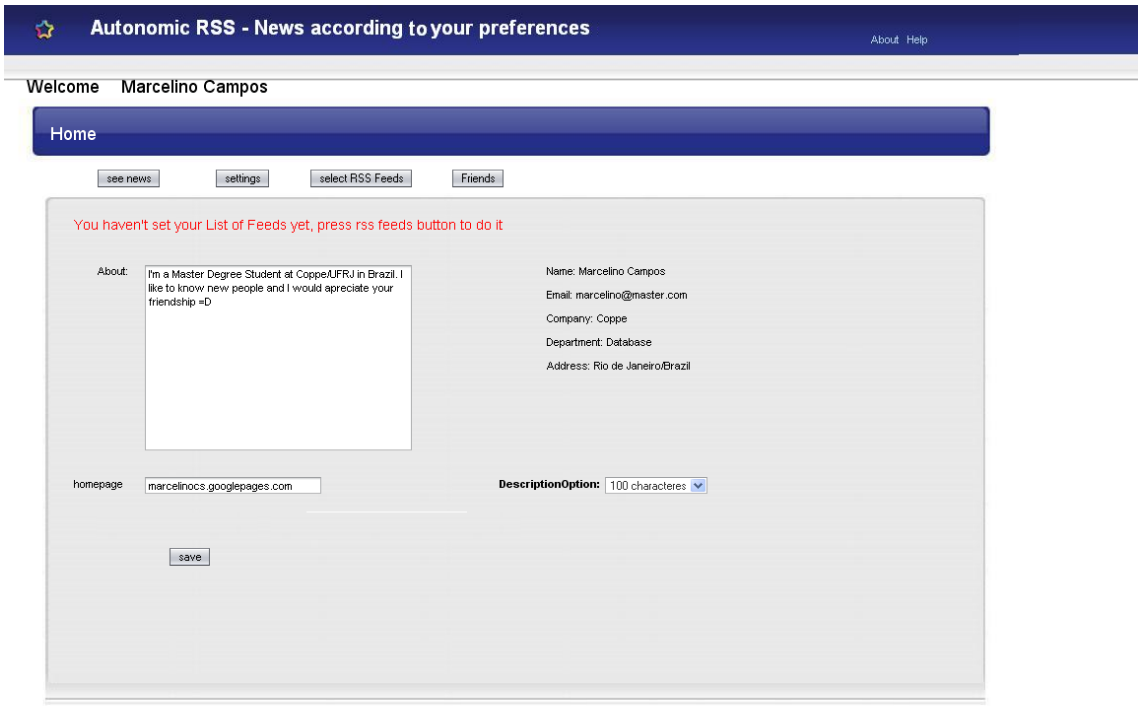

Figura 19 – Tela principal

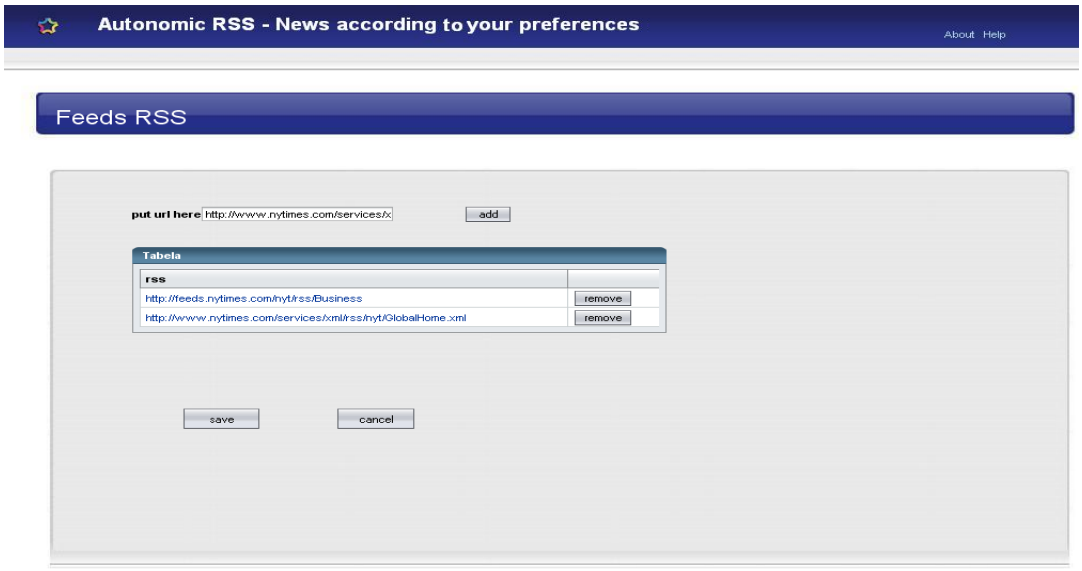

Figura 20 – Tela de manipulação de canais RSS

Para configurações gerais, relativas à edição de regras, escolha de amigos, e listas negras e brancas, o usuário pode ir ao botão "*settings*" na tela principal. Assim, após clicar no botão, o usuário é redirecionado à página de configurações gerais, que é dividida em quatro abas. A primeira aba tem opções gerais de filtragem, onde o usuário pode escolher as palavras que ele quer ou não encontrar no texto, opção de descarte de notícias similares, escolha do intervalo da data de publicação da notícia, escolha da categoria, entre outros. A Figura 21 ilustra a primeira aba da página de configurações gerais.

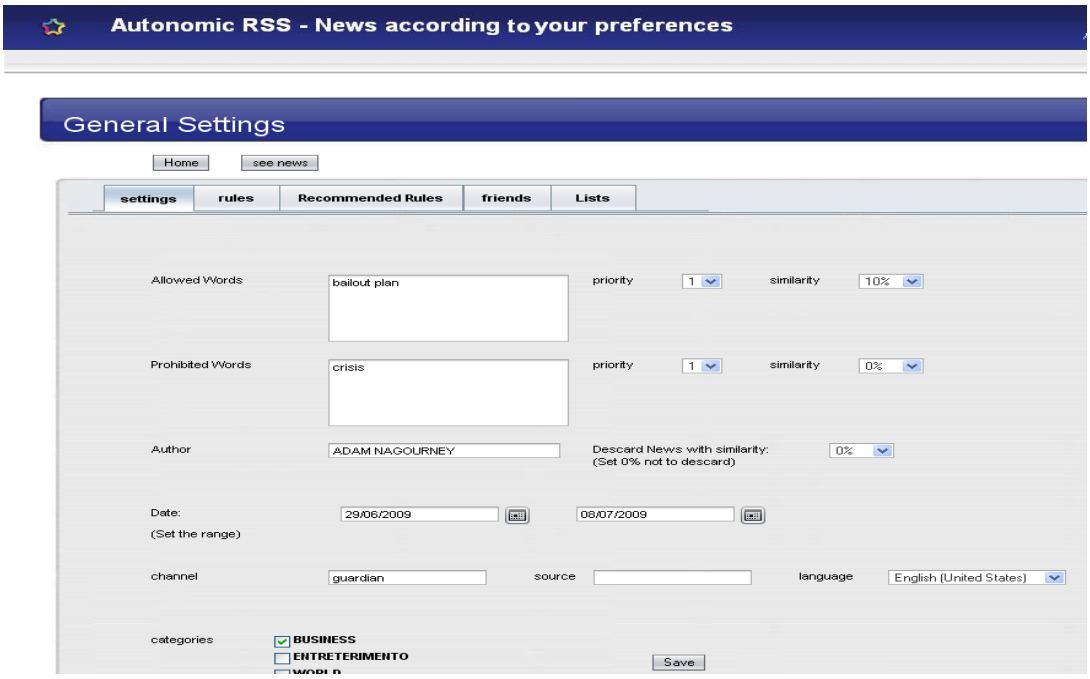

Figura 21 – Configurações Gerais Aba 1 – Opções de Filtragem

Após preencher os campos com os valores de sua escolha, o usuário deve clicar no botão de salvar, convertendo estas preferências em regras SWRL. Para a visualização de todas as regras do usuário, ele deve ir à segunda aba, onde além de visualizar é possível incluir e remover regras. A Figura 22 ilustra a aba de regras. Nesta aba, é possível incluir regras SWRL que não estejam disponíveis na primeira aba de opções de filtragem. Sendo assim, a primeira aba, de opções de filtragem, pode ser considerada uma opção mais amigável ao usuário para inclusão de regras, enquanto que a aba de regras pode ser considerada a opção para usuários avançados.

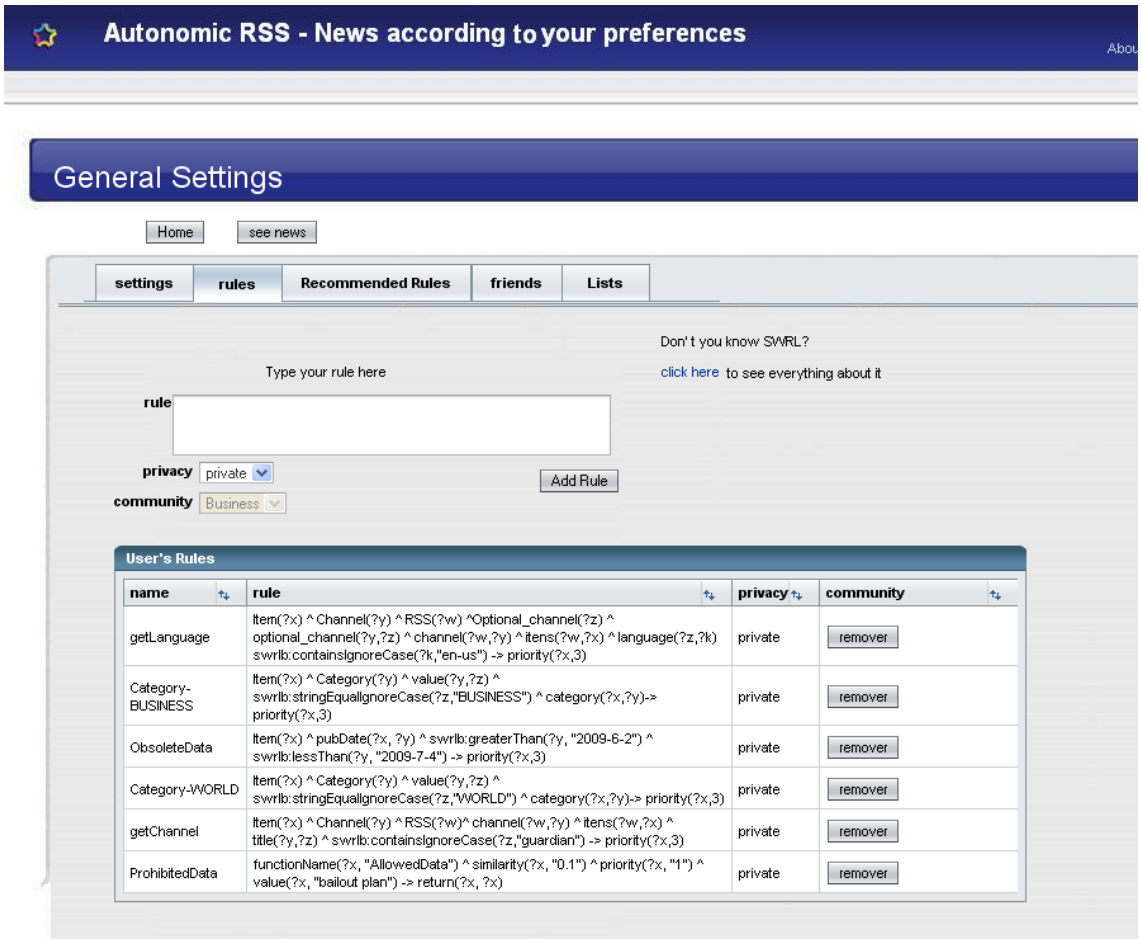

Figura 22 - Configurações Gerais Aba 2 – Opções Avançadas de Regras

A terceira aba é responsável por mostrar ao usuário regras recomendadas pela ferramenta. Essa tela mostra as regras agrupadas por usuário, sendo que estas são ordenadas em relação à compatibilidade do perfil do usuário. Em outras palavras, são mostradas todas as regras do amigo com perfil mais compatível, depois todas as regras do segundo perfil mais compatível e assim por diante. A Figura 23 ilustra a tela de recomendação de regras.

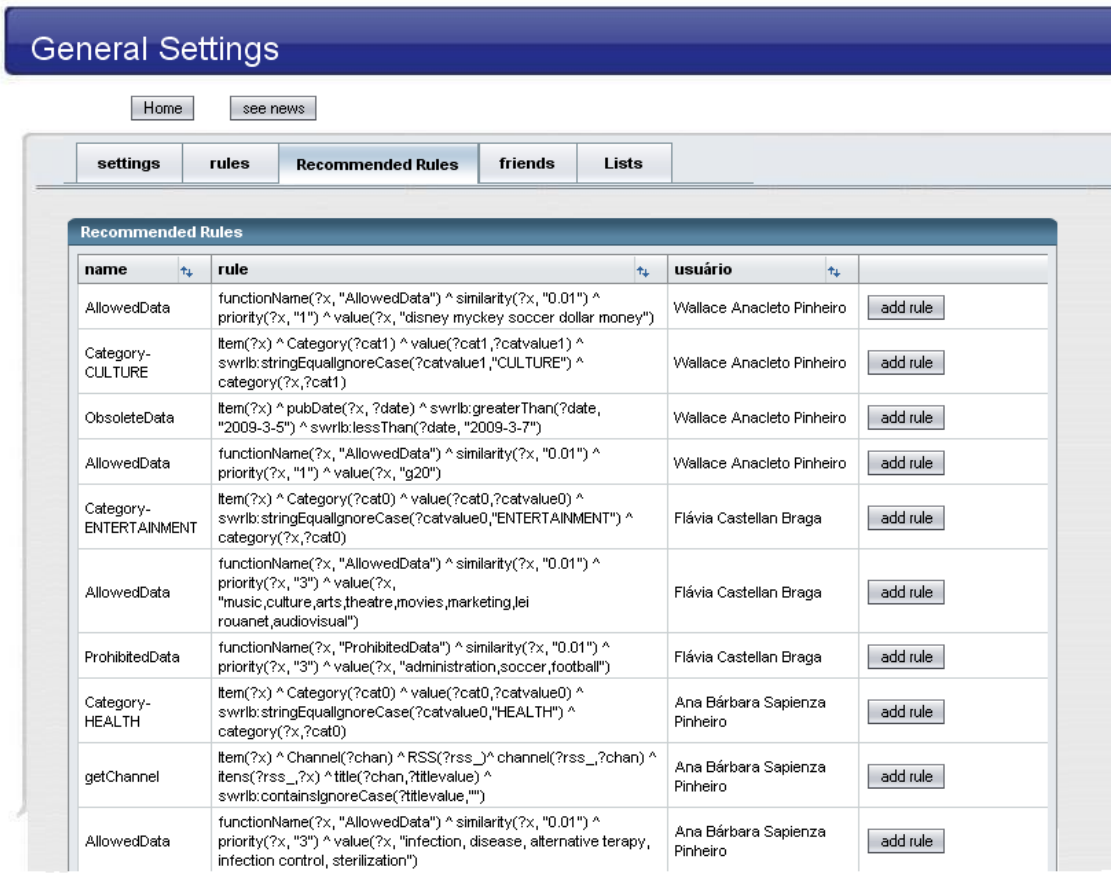

Figura 23 - Configurações Gerais Aba 3 – Recomendação de Regras

A quarta aba é a responsável pela escolha de amigos. Nessa aba são listados todos os usuários do sistema, sendo assim, o usuário pode escolher quem será seu amigo. Os usuários selecionados como amigos poderão visualizar o perfil do usuário que os escolheu. Se um usuário A estiver na lista de amigos de um usuário B, A poderá visualizar o perfil de B, podendo adicionar as regras que B deixar disponível, melhorando as suas opções de filtragem. A Figura 24 ilustra a aba de escolha de amigos.

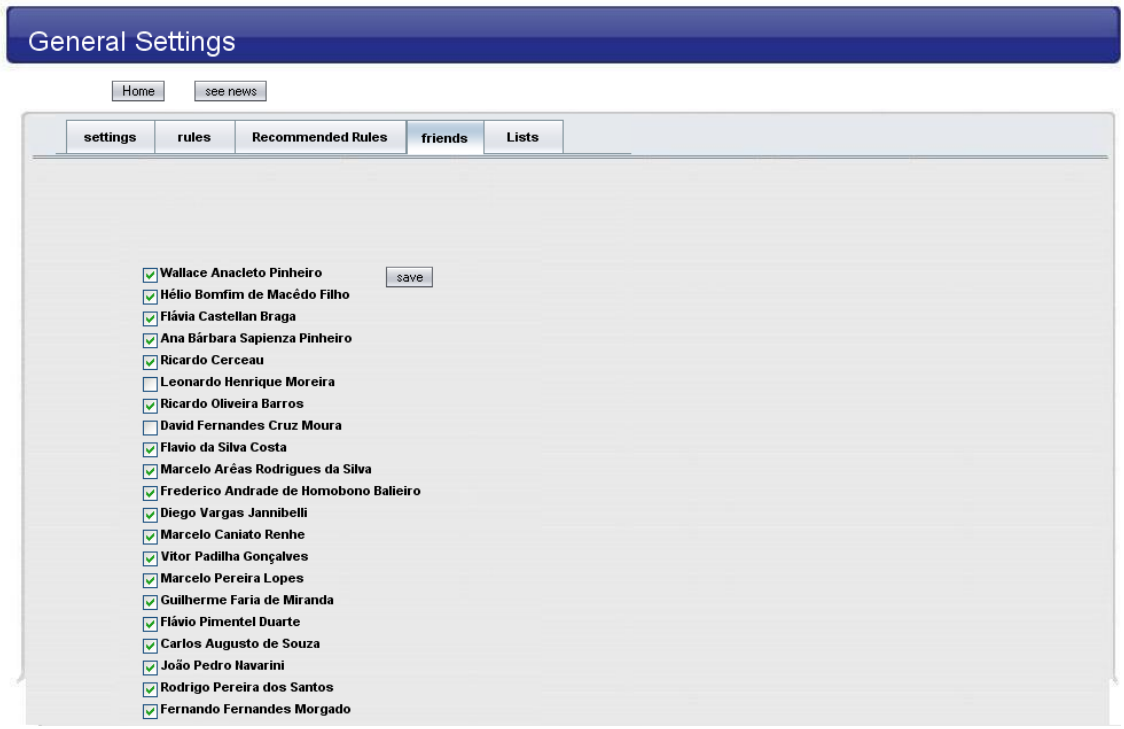

Figura 24 - Configurações Gerais Aba 4 – Opção de Escolha de amigos

A quinta e última aba da tela de configurações gerais é responsável pela escolha das listas negras e brancas de conteúdo. Através da funcionalidade dessa aba, será possível filtrar notícias que contenham palavras da lista dos assuntos disponibilizados. Vários assuntos podem ser escolhidos. Para fins do experimento apresentado neste trabalho, são considerados os seguintes assuntos: violência, política, religião e pornografia. A lista negra de RSS com spam já fica incluída automaticamente, descartando spams automaticamente. A Figura 25 ilustra a tela de listas.

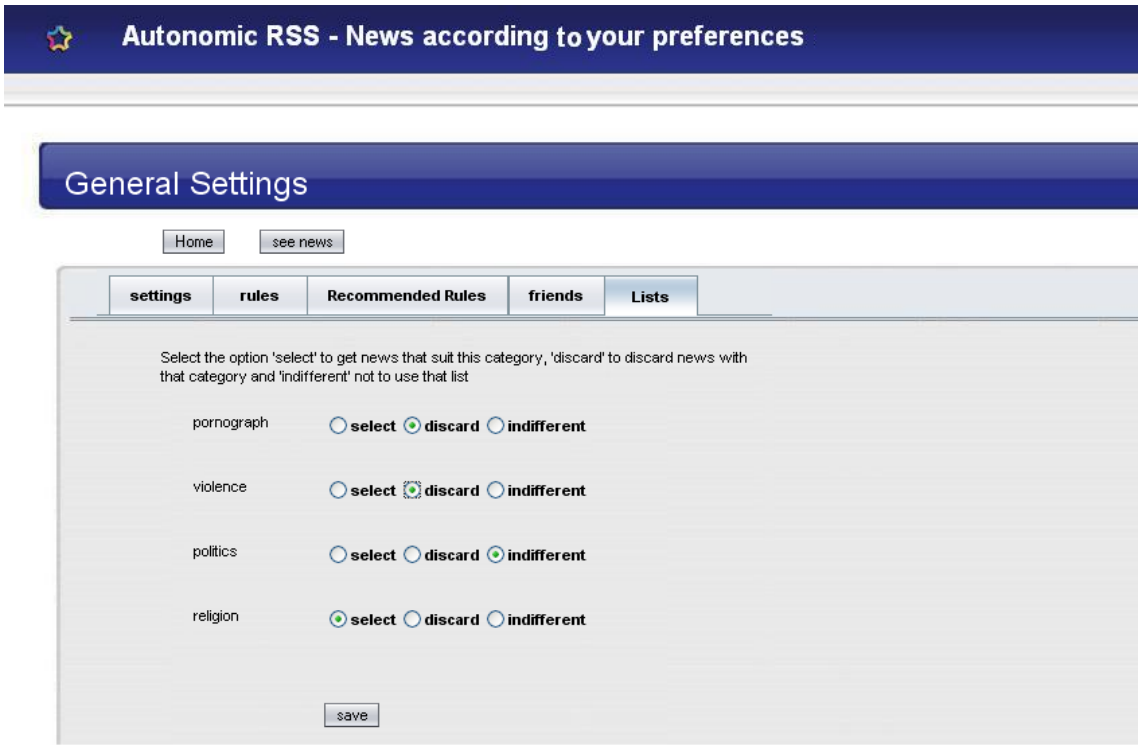

Figura 25 - Configurações Gerais Aba 5 – Opções de Listas Negras e Brancas

Assim, depois de acertar todas as configurações, o usuário pode escolher clicar no botão "*see news*" para ver as notícias com as regras aplicadas ou clicar no botão "*home*" para voltar para a tela principal. Voltando à tela principal, o usuário poderá ver outros usuários que o adicionaram na tela de amigos, através do botão "*friends*". Ao clicar no botão, o usuário é redirecionado à tela de amigos que é ilustrada na Figura 26.

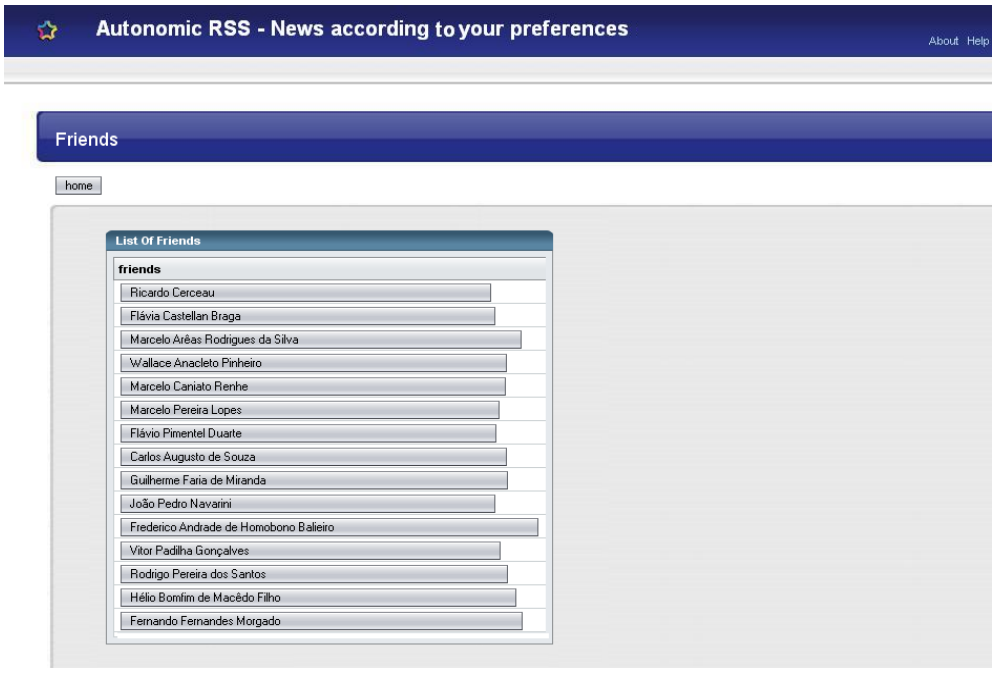

Figura 26 – Tela de amigos

Clicando em um dos amigos listados, o usuário poderá ver o perfil deste amigo. Na tela de “perfil de amigo”, o usuário pode ver informações do amigo e quais são suas regras públicas. Em cada regra pública existe um botão ao lado para adição da regra. Ao clicar no botão, a regra é adicionada para a lista de regras do usuário. A Figura 27 ilustra a tela do perfil do amigo.

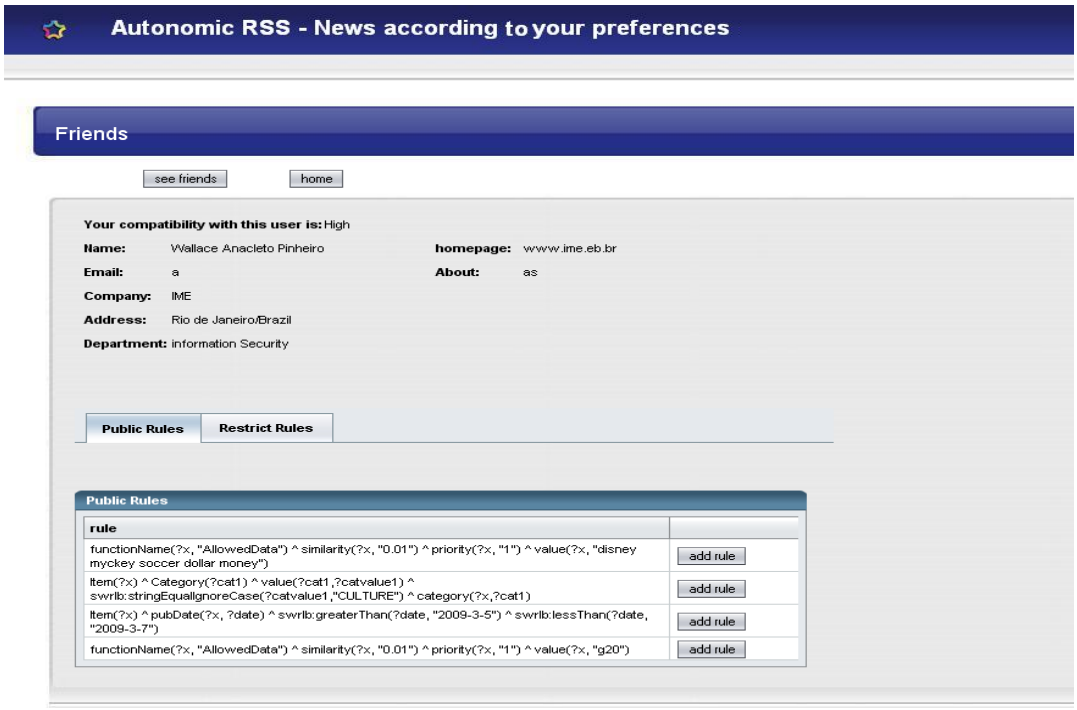

Figura 27 – Tela do perfil do amigo

Por fim, temos a funcionalidade que permite visualizar as notícias resultantes dos filtros aplicados. Para isso, o usuário deve clicar no botão “*see news”* da tela principal. Ao clicar neste botão, todas as regras SWRL e as funções serão executadas sobre o arquivo OWL, que contém as notícias, e o resultado é apresentado ao usuário. A Figura 28 ilustra a tela do resultado de filtragem de notícias. Se for do interesse do usuário, ele pode ver o registro de operações (detalhando todas as regras usadas e importadas) e pode baixar o arquivo RSS correspondente ao resultado.

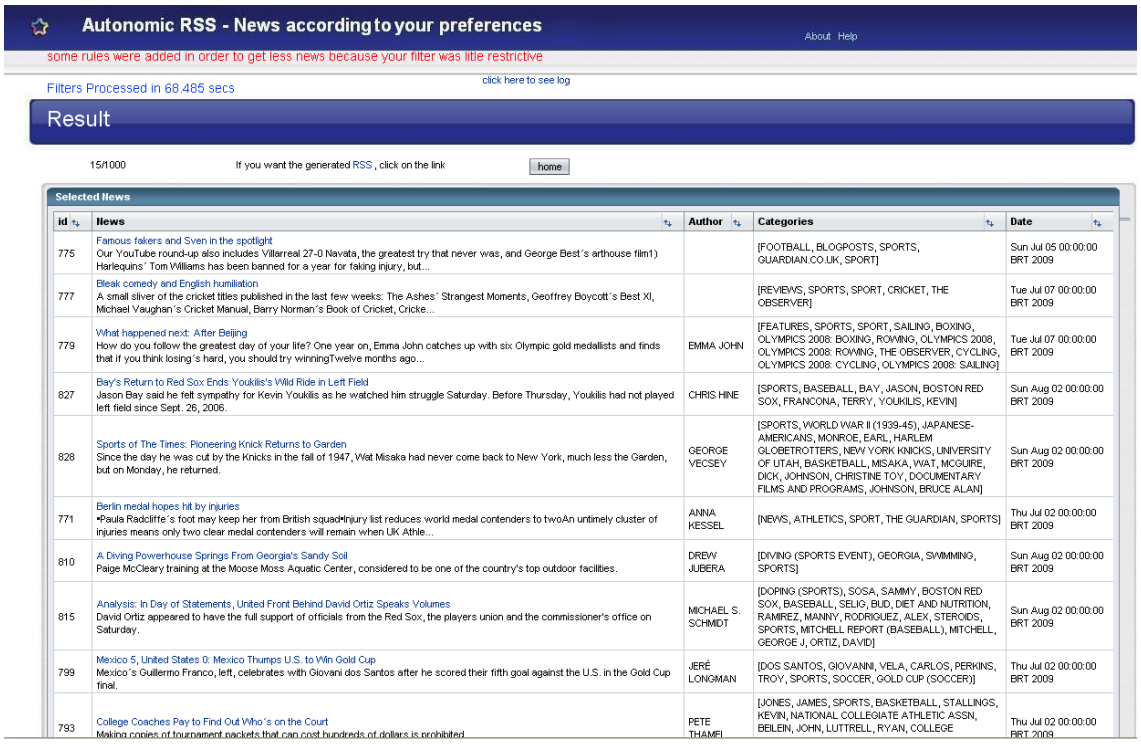

Figura 28 – Tela do Resultado da Filtragem

6.2. Visão Geral do Experimento

Com o objetivo de validar as técnicas apresentadas anteriormente e verificar a viabilidade dos Dados Autonômicos na aplicação proposta, foi realizado um experimento envolvendo um grupo de participantes de diversos lugares, sendo a maioria alunos do IME e da UFRJ. Além desses, participaram profissionais do mercado de empresas como a Chemtech, Totvs e Politec. Com os grupos formados por alunos do Instituto Militar de Engenharia (IME) e Universidade Federal do Rio de Janeiro (UFRJ), o experimento foi presencial, e com os demais foi via internet. No total foram avaliados 30 participantes. Eles receberam formulários onde cadastraram suas informações e preferências. Nos experimentos presenciais realizados, os participantes, após preencherem os formulários, tiveram a oportunidade de utilizar a ferramenta e testar suas funcionalidades. Os demais somente preencheram os formulários. O experimento utilizou uma carga de 1000 notícias provenientes de diferentes veículos de comunicação (Herald Tribune, New York Times, The Guardian, Washington Post, The Sun e Financial Times) coletadas entre os meses de fevereiro a agosto de 2009.

O planejamento e execução dos experimentos foram feitos seguindo os conceitos e técnicas apresentadas em BARROS *et al*. (2002). Segundo este trabalho a realização de um estudo experimental geralmente é dividida em cinco fases: a **definição**, o **planejamento**, a **execução**, a **análise** e o **empacotamento do estudo**. A definição de cada uma dessas fases e os procedimentos são discutidos nas próximas seções.

Os experimentos foram realizados no mês de outubro de 2009. Inicialmente, foram definidas as etapas envolvidas no estudo e como estas etapas foram realizadas. Em seguida, foram apresentadas as observações obtidas durante o estudo.

6.2.1. Definição

Essa fase consiste em resumir os objetivos do estudo, seu foco de qualidade e os objetos que serão analisados. Para os experimentos realizados, têm-se:

- Objeto de Estudo: A ferramenta RSS Autonômico, aplicação que se utiliza de Dados Autonômicos e técnicas autonômicas para o filtro de notícias RSS.

- Objetivo: São dois os objetivos principais, primeiro identificar a eficiência em buscas da ferramenta desenvolvida, e o segundo é verificar se através da utilização de Dados Autonômicos a aplicação ganhou características autonômicas. Este objetivo é divido em três etapas, sendo que cada uma dessas deve ser cumprida para que o objetivo seja alcançado. As etapas desse objetivo são:

    a) Verificar se o sistema se auto-otimiza em relação à junção de regras de modo a obter a quantidade de notícias que o usuário deseja.

    b) Verificar se o sistema se auto-protege, não permitindo spam entre as notícias e nem notícias que estejam na lista negra do usuário. São obtidas as configurações dos amigos, as regras são adicionadas se a porcentagem de amigos que escolheram uma determinada regra atingir certo valor que é configurável. De inicio a porcentagem é 20%, podendo ser alterada.

    c) Verificar se o sistema realizou corretamente a auto-configuração de classificação de categorias.

- Foco de Qualidade: O foco da qualidade também é dividido em duas vertentes, uma para cada um dos objetivos citados acima. A primeira vertente consiste

em avaliar se os resultados da ferramenta são relevantes, para verificar se a utilização da mesma é viável. Essa avaliação se baseia em comparar as notícias retornadas pela ferramenta com as classificadas pelos participantes como relevantes. A segunda vertente irá verificar se as técnicas utilizadas tiveram o efeito desejado, ou seja, se o sistema realmente adquiriu propriedades autonômicas. Como o segundo objetivo foi dividido em três partes, o respectivo foco de qualidade também o é, avaliando assim se cada um dos objetivos parciais será cumprido. Sendo assim, os focos de qualidade específicos para cada etapa do objetivo são:

a) A ferramenta deve obter notícias filtradas e a quantidade destas deve estar de acordo com o indicado pelo usuário no formulário. Caso ocorra um número diferente de notícias daquele desejado pelo usuário, a ferramenta deverá adicionar ou remover regras. Atingir a quantidade desejada pelo usuário é o parâmetro que indicará a qualidade deste item.

b) O sistema não deve permitir nenhuma notícia que contenha spam e nenhuma que esteja enquadrada nas listas negras marcadas pelo usuário. O descarte de notícias que não sejam do interesse do usuário, ou seja, que não estejam de acordo com as regras, indicará a qualidade deste item.

c) A classificação dada à categoria da notícia deve ir ao encontro do interesse do usuário. A coincidência dos valores preenchidos no formulário com os valores obtidos pela ferramenta indicará a qualidade deste item.

Uma observação a ser feita é que este estudo não está diretamente interessado no tempo gasto para a utilização das técnicas propostas e isso não será levado em consideração.

- Perspectiva: o estudo será desenvolvido sob a ótica do pesquisador. É verificada a viabilidade de utilização dos Dados Autonômicos.

- Contexto: demonstração das propriedades autonômicas da ferramenta RSS Autonômico.

6.2.2. Planejamento

O planejamento engloba a descrição do perfil dos participantes, dos instrumentos, do processo de execução e uma avaliação crítica dos problemas que podem ser encontrados ao longo desta execução. Dessa forma, tem-se:

- Contexto global: utilização de Dados Autonômicos para incorporar conceitos autonômicos a uma ferramenta sem aumentar sua complexidade.

- Contexto local: avaliação das técnicas autonômicas presentes no Dado Autonômico.

- Participantes: alunos da UFRJ e do IME e profissionais do mercado de Tecnologia de Informação e Engenharia.

- Treinamento: Não foi necessário treinamento, pois o uso direto da ferramenta não foi obrigatório.

- Instrumentos: Todos os participantes, independentes se foram presenciais ou não, receberam questionários contendo perguntas para caracterização da sua formação e experiência. Eles os preencheram com informações sobre suas preferências e questões específicas sobre as atividades propostas.

- Critérios: o foco de qualidade do estudo exige critérios que avaliem os ganhos proporcionados pela utilização dos Dados Autonômicos a fim de fornecer propriedades autonômicas à aplicação. Os ganhos obtidos pela utilização das técnicas são avaliados quantitativamente valendo-se das respostas dos participantes. Esta análise tem o objetivo de avaliar a ferramenta, seus pontos fontes, fracos e possibilidades de melhorias futuras.

- Variáveis independentes: os dados pessoais, experiência com uso de *feeds* e os tipos de agregadores de *feeds* utilizados são informações independentes coletadas durante o estudo.

- Variáveis dependentes: todas as demais variáveis (por exemplo, notícias e regras de filtragem) são dependentes.

- Capacidade aleatória: pode ser exercida na seleção dos participantes do estudo e na distribuição dos objetos de análise entre eles. O objeto da análise é o mesmo para todos os participantes. A turma de participantes foi composta de alunos da UFRJ,

alunos do IME e profissionais do mercado. Dessa forma, considerando a população do experimento todo aluno da UFRJ ou IME tem a mesma chance de ter participado desse experimento. Os demais profissionais do mercado foram escolhidos por disponibilidade de participar do experimento. Isto indica uma amostra aleatória da população descrita no item 'Participantes'.

- Classificação em bloco: Não se verificou a necessidade de separação de blocos uma vez que a diferença de escolaridade da população era mínima (graduandos e profissionais formados) e que as diferenças de conhecimento de cada um não apresentam qualquer obstáculo no preenchimento dos formulários.

- Balanceamento: não houve necessidade de balanceamento, pois não foi observada nenhuma característica que recomendasse isso.

- Mecanismo de análise: as variáveis dependentes serão apresentadas utilizando-se as escalas próprias de cada variável. Além disto, os resultados serão discutidos utilizando-se por base os resultados obtidos pela ferramenta comparados às preferências obtidas a partir da opinião dos participantes em cada atividade.

- Validade interna do estudo: é definida como a capacidade de um novo estudo repetir o comportamento do estudo atual com os mesmos participantes e objetos com que ele foi realizado. A validade interna do estudo é dependente do número de participantes executando o estudo. Como são avaliados 30 participantes, isso garante um bom nível de validação interna. Outro ponto que pode influenciar o resultado do estudo é a troca de informações entre os participantes que já realizaram o estudo e os que não o realizaram. Para evitar este problema, tem-se como requisito que os participantes não troquem informações a respeito dos questionários.

- Validade externa do estudo: mede sua capacidade de refletir o mesmo comportamento em outros grupos de participantes e profissionais do meio acadêmico, ou seja, em outros grupos além daquele em que o estudo foi aplicado. A validade interna do estudo é considerada suficiente, visto que o experimento contou com uma gama bem variada de pessoas de diferentes locais.

- Validade de construção do estudo: refere-se à relação entre os instrumentos e participantes do estudo e a teoria que está sendo provada por este. É importante enaltecer que a aplicação foi feita sobre uma área amplamente conhecida, que não exigiria conhecimento específico prévio, ou seja, a área de notícias Web. Isto, de certa

forma, neutraliza o efeito da experiência dos participantes e evita que experiências anteriores criem uma interpretação incorreta do procedimento a ser adotado.

- Validade de conclusão do estudo: mede a relação entre os tratamentos e os resultados, determinando a capacidade do estudo em gerar alguma conclusão. O estudo utilizará somente medidas objetivas e quantitativas como foi descrito anteriormente.

6.2.3. Execução

A execução permite a realização do estudo experimental pelos participantes, utilizando os instrumentos e os processos definidos no planejamento. Desse modo, as seguintes etapas foram realizadas:

- Seleção dos Participantes: os participantes foram selecionados, conforme descrito no item "Participantes" da seção anterior. Foram selecionados 12 alunos da UFRJ, 10 alunos do IME e 8 profissionais do mercado totalizando 30 participantes.

- Instrumentação: foram distribuídos questionários, conforme descrito no item 'Instrumentos' da seção anterior.

- Procedimentos de Participação: Após concordarem em participar do experimento, cada participante recebeu dois formulários e responderam objetivamente as questões.

- Execução: Como os participantes não executaram a ferramenta diretamente a única etapa da execução foi o preenchimento dos formulários.

A próxima etapa consiste da análise dos resultados, realizando o preparo dos resultados gerados pelo experimento e discutindo as inferências feitas sobre este. Além disso, serão mostradas quais foram as questões do formulário.

6.3. Avaliação dos Resultados

Antes das análises principais será demonstrada uma análise preliminar para descobrir qual o conhecimento dos usuários acerca dos *feeds* de notícias. Foram distribuídas aos participantes as seguintes perguntas, relacionadas aos Feeds RSS, sendo que as perguntas de 3 a 5 só deveriam ser respondidas se a pergunta 2 fosse afirmativa:

1. Forneça o seu nome completo:

2. Qual a sua experiência com RSS?

3. Como você escolhe o uso dos feeds?

4. Dê a sua nota para o nível de similaridade (o quão parecido) às notícias que recebe. Considere os últimos quinze dias e que as notas variam de nota 0 (para nenhuma notícia repetida ou parecida) a nota 10 (para mais de dez notícias parecidas).

5. Quais os agregadores (clientes) de feeds que você utiliza? Ex: Mozilla Firefox, Internet Explorer 7, FeedGhost, Juice, Akregator, Vienna, etc. Assinale também como vc classifica esses agregadores (clientes) de feeds: as notas variam de nota 0 (para pobres, que não fornecem opções) a nota 10 (para os completos, que fornecem muitas opções).

O resultado da questão 2 é mostrado na Figura 29 e mostra que a maioria dos usuários conhecia Feeds RSS.

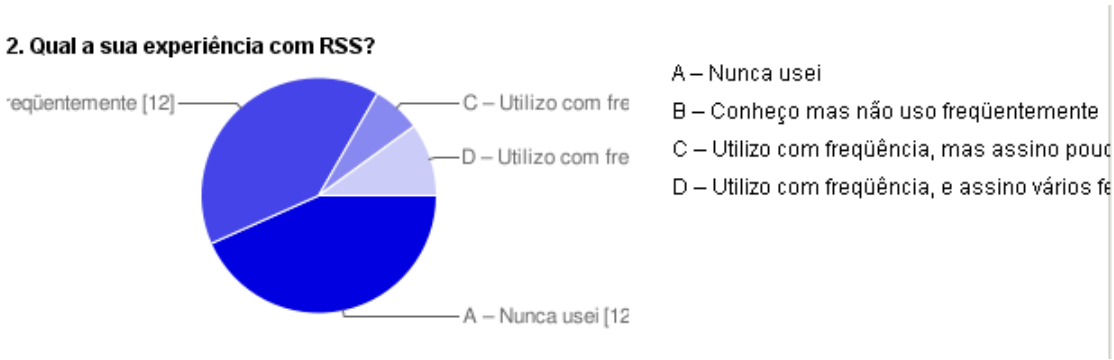

Figura 29 – Experiência dos usuários com Feeds RSS

Dentre os usuários que conheciam Feeds RSS, a maioria assina notícias de sites gerais e apenas uma pequena quantidade assina de vários sites. O resultado é ilustrado na Figura 30.

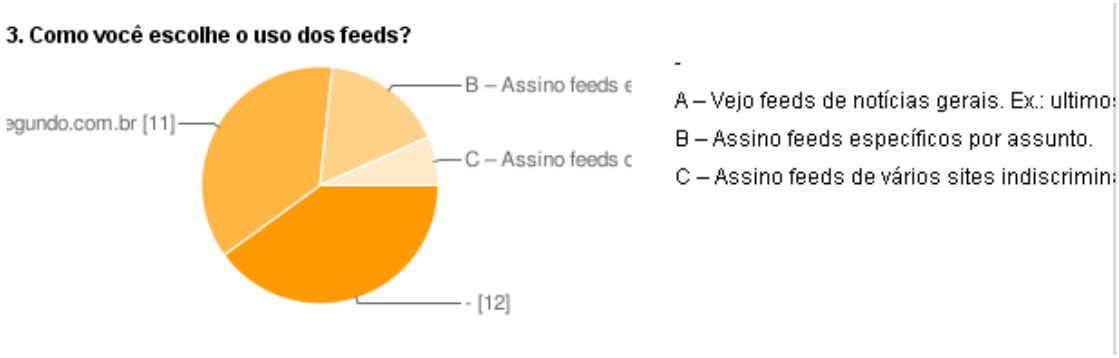

Figura 30 – Como o usuário escolhe o uso de feeds

Com relação à quantidade de notícias similares que cada usuário recebia, uma grande parte escolheu 7 e 8 notícias similares por semana, o que mostra que os usuários recebem relativamente muitas notícias similares e que ferramentas para reduzir essa quantidade de notícias similares realmente era necessária. O gráfico com a quantidade escolhida por cada um é ilustrado na Figura 31.

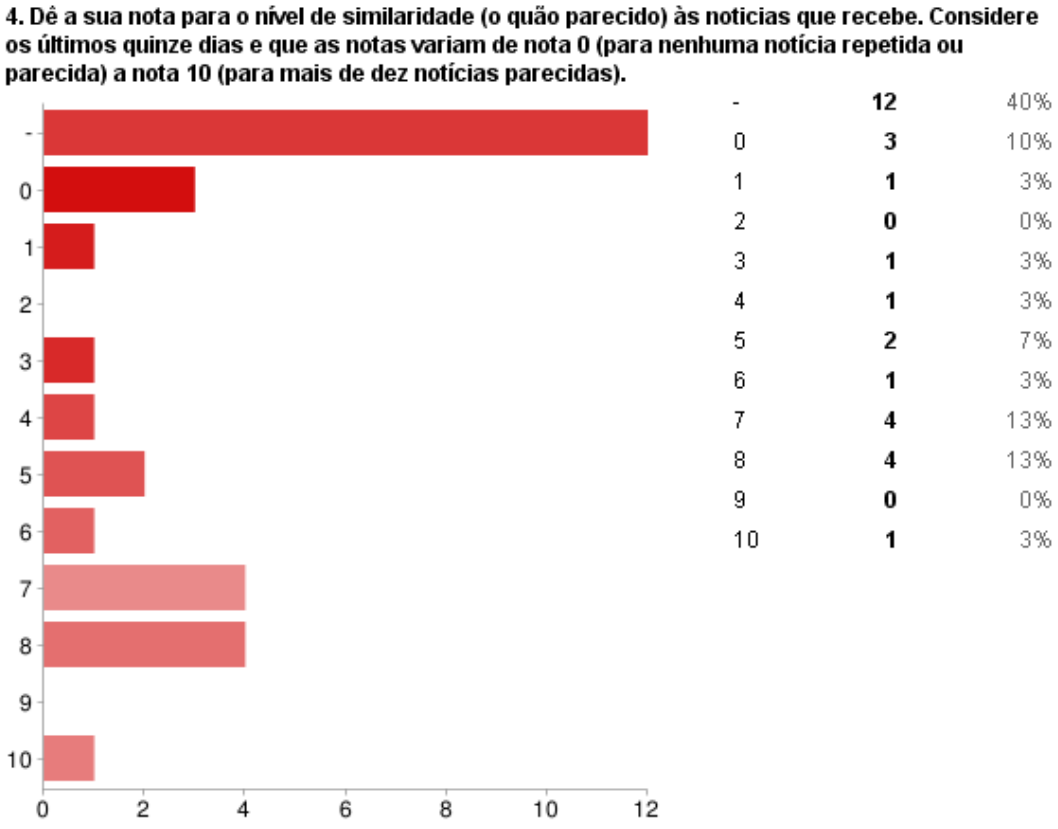

Figura 31 – Número de notícias similares que o usuário recebe em uma semana

Agora serão demonstradas as análises principais. A avaliação dos resultados foi dividida em quatro partes, analisando separadamente cada um dos focos do experimento.

6.3.1. Análise 1: verificação da Auto-Otimização de regras

Essa atividade teve por intenção verificar se a ferramenta conseguia auto-otimizar as regras de modo a conseguir o número desejado de notícias. No formulário foi pedido aos participantes que preenchessem os seguintes itens:

a) Supondo uma consulta em uma ferramenta que retorna notícias, em relação ao número de notícias retornadas, qual seria a faixa aceitável? (por exemplo: 5-25)

b) Qual o número específico de notícias, dentro da faixa dada anteriormente, você gostaria de ler?

c) Quais palavras-chave você gostaria de encontrar nas notícias retornadas?

d) Quais palavras você NÃO gostaria de encontrar nas notícias retornadas?

e) Notícias Similares: Se quiser que sejam descartadas notícias similares forneça um nível de similaridade para o descarte das mesmas, usando um valor de 1 a 100%

f) Escolha qual a sua categoria preferida?

g) Você NÃO gostaria de receber notícias de quais domínios?

h) Dentre os outros participantes do experimento, indique alguns que tenham um perfil parecido com o seu e que queira que seja associado como amigo:

i) Você acredita que a importação de regras de amigos que tem perfis parecidos pode ser benéfica para a filtragem de notícias? Comente a respeito:

A partir dessas respostas foi construído o perfil do usuário, com regras, listas negras e amigos. Só depois que todos os perfis foram cadastrados e completados (com todas as regras do usuário, dos amigos e listas negras) é que o processamento das notícias foi feito. Depois de processar todos os usuários, foi obtido o resultado apresentado na Tabela 6:

Tabela 6 – Vezes que a Ferramenta acertou o número de notícias

Acerto das notícias			
	acertou o número de notícias	não acertou o número de notícias	Total
quantidade	19	11	30
porcentagem	63,33333333	36,66667	100

O resultado pode ainda ser mais bem visualizado no gráfico da Figura 32. É possível ver que na maioria das vezes foi obtida a quantidade desejada pelo participante.

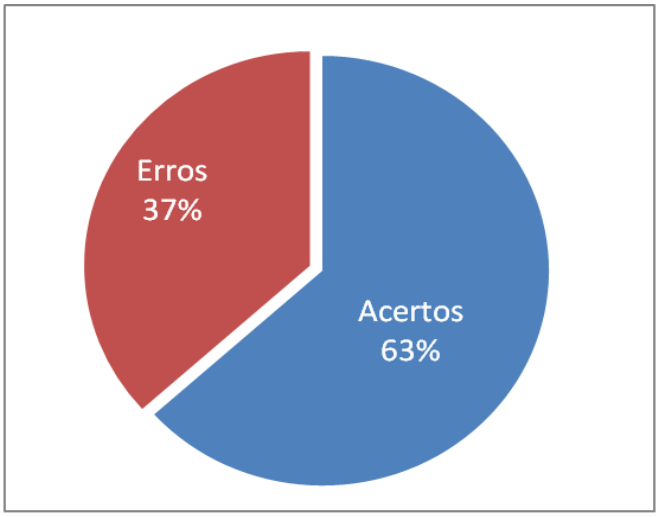

Figura 32 – Número de acertos da Ferramenta em relação à quantidade de notícias

6.3.2. Análise 2: verificação da viabilidade da ferramenta

As demais análises fazem uso do segundo formulário que os usuários responderam. Para fazer essas atividades foram selecionadas 100 notícias das 1000 existentes no repositório. Essas 100 notícias foram divididas em 10 formulários diferentes. Como tivemos 30 participantes, garantimos que cada notícia tenha sido lida por pelo menos 3 pessoas diferentes, garantindo uma maior confiança dos dados. Nesse formulário, foi questionado em qual categoria o usuário classifica determinada notícia e se esta é relevante, considerando as seguintes regras:

Palavras Proibidas: *exercise, acid, japan, barcelona*;

Palavras Permitidas: *acupuncture, health, mexico, liverpool, manchester, nadal, g20,*

phelps, dylan, inflation, software, microsoft;

Assunto proibido: política

Na ferramenta, usando essas regras de filtragem, 12 resultados foram retornados. Para definir se uma notícia foi considerada relevante ou não, o seguinte procedimento foi adotado: se nenhum ou um participante marcou uma notícia como relevante, então a maioria não a achou relevante e, dessa forma, ela é considerada irrelevante; se dois ou três participantes marcaram a notícia como relevante, então a notícia é considerada relevante. Das 100 notícias analisadas, o total considerado relevante, segundo os usuários, foi igual a 36.

Sendo assim, os resultados da ferramenta foram comparados com os classificados como relevante pelos usuários. O gráfico de precisão x cobertura pode ser visto na Figura 33

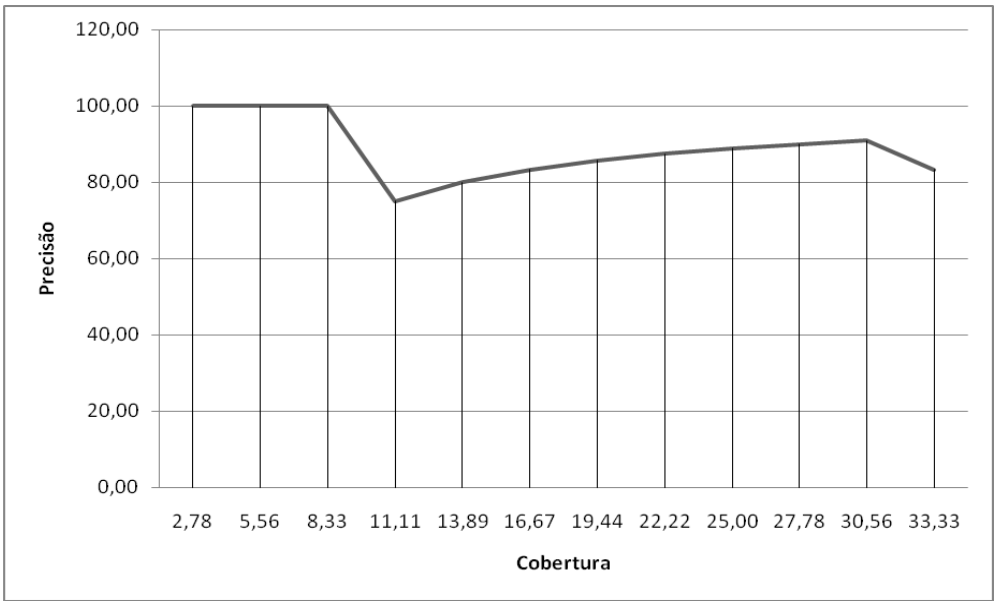

Figura 33 – Precisão X Cobertura das notícias retornadas pela ferramenta

É possível ver pelo gráfico que a precisão obtida foi alta, variando entre 100 % e 75%, atingindo uma media de 88%. Isso mostra que as notícias retornadas pela ferramenta são relevantes para o usuário. Porém, outro fato a ser notado é que a cobertura atingiu apenas 33,33%, ou seja, foi retornado somente um terço de todas as notícias consideradas relevantes para o usuário. Isso pode ser explicado pelo fato de que os participantes terem feito muito mais associações do que a ferramenta. Por exemplo, foi dado o termo ‘phelps’ que é um nadador, e, baseando-se nesse termo, vários usuários

relacionaram notícias de natação como relevante, e a ferramenta não faz essa inferência. Sendo assim, a quantidade de notícias consideradas como relevantes para os usuários é mais ampla. Por outro lado, se for pensado que a ferramenta tem o propósito de diminuir a sobrecarga de informação é melhor que venham menos resultados. E, entre uma notícia que tem relação com o termo e outra que tenha o termo, a segunda opção é obviamente mais relevante.

6.3.3. Análise 3: verificação da Auto-Proteção

Essa atividade teve por intenção verificar se, com os Dados Autonômicos, o sistema conseguiu algum nível de auto-proteção. Para verificar isso, foram utilizados o formulário 1, questão *i* e o formulário 2. A análise se deu em dois pontos: Primeiramente, era necessário verificar se o usuário concorda com a importação de listas dos outros usuários. Depois, verificar se as listas são efetivas, ou seja, se realmente descartam notícias irrelevantes para o usuário.

Para checar o primeiro ponto verificaram-se as respostas da questão *i* do formulário 1. O resultado da opinião é ilustrado na Figura 34:

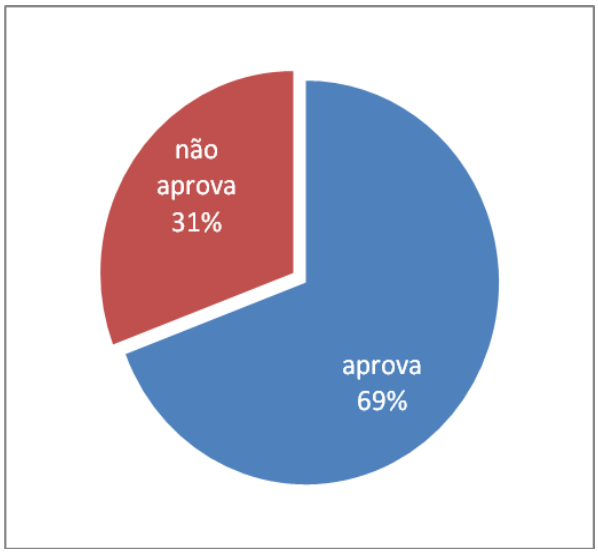

Figura 34 – Aprovação dos usuários sobre a importação de regras

Isso demonstra que a maioria concorda com a idéia de importação de regras, satisfazendo o requisito de aceitação do primeiro ponto. Para verificar o segundo ponto foram obtidas todas as notícias que foram descartadas pela lista negra. O total de notícias foi igual a 6 (lembrando que a lista negra é último operador a ser aplicado, ou seja, essas 6 notícias foram pré-selecionadas pelas outras regras e só foram descartadas

na aplicação do operador lista negra). Diante dessas 6 notícias, estas foram comparadas com as classificadas pelos participantes como relevante para verificar se realmente as notícias descartadas eram irrelevante à pesquisa. O resultado pode ser conferido no gráfico da Figura 35

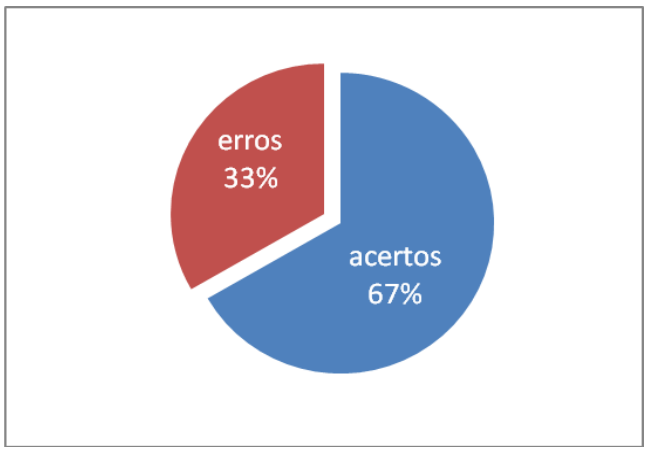

Figura 35 – Taxa de acertos no descarte das notícias pela lista negra

Sendo assim, a eficiência do descarte de notícias por parte da lista negra se mostrou razoavelmente eficiente. Portanto, como se conseguiu comprovar que os dois pontos citados acima ocorrem, pode se afirmar que a auto-proteção ocorre.

Outro ponto a se respaldar é que na questão g do formulário 1, com relação à aplicação das listas, 100% dos usuários responderam a questão com alguma lista, ou seja, todos escolherem usar pelo menos uma lista negra. Isso mostra que as listas são um recurso importante para os usuários.

6.3.4. Análise 4: verificação da Auto-Configuração

Para verificação da auto-configuração, foi utilizado o mesmo formulário da análise 3. Como explicado anteriormente, cada usuário atribuiu uma categoria para cada notícia. Foram coletados esses dados e comparados com as classificações obtidas pela ferramenta. Para essa análise, não foi feita a média em relação às respostas dos usuários. Dessa forma, para que fosse feita a comparação, a quantidade existente de notícias de cada categoria definida pela ferramenta foi multiplicada por 3. Como exemplo, considere que na categoria esportes existem 20 notícias, dentre as 100, que foram

classificadas pela ferramenta na categoria esportes. Como existem 3 respostas de participante para cada notícia, então temos 60 análises de notícias de esportes para comparar. O resultado pode ser visto na Tabela 7.

Tabela 7 – Comparação da classificação dos usuários com a ferramenta

Categoria	n° de notícias analisadas	Acertos	Porcentagem
Economia	54	39	72,2%
Esportes	60	54	90,0%
Entretenimento	30	23	76,7%
Meio Ambiente	30	19	63,3%
Política	30	23	76,7%
Tecnologia	30	20	66,7%
Saúde	60	53	88,3%
Gerais	6	4	66,7%
Total	300	235	78,3%

É possível ver pela tabela que em algumas categorias os resultados coincidiram muito bem com a ferramenta, como, por exemplo, esportes e saúde. Por outro lado, outras categorias não obtiveram uma coincidência tão alta como no caso de economia e meio ambiente. Contudo, por que algumas categorias têm resultados melhores do que outras? Existem algumas hipóteses, uma delas é relacionada aos vários assuntos paralelos que envolvem uma notícia. Por exemplo, uma notícia pode estar relacionada a esportes ou economia, quando, por exemplo, se fala dos problemas financeiros de um clube de futebol. Sendo assim, a ferramenta pode ter classificado a notícia como esporte e o usuário como economia e os dois estarem certos. Porém, apesar destes fatores, o resultado pode ser considerado satisfatório, já que observando a média geral de acertos foi de 78.3%.

Sendo assim, chegamos ao final das 4 análises e é possível verificar que os dois pontos principais do experimento foram atingidos, ou seja, a ferramenta se mostrou eficaz na busca de notícias e adquiriu propriedades autonômicas com o uso de Dados Autonômicos.

Capítulo 7. Conclusões e Trabalhos Futuros

Os avanços tecnológicos trouxeram uma grande facilidade e acesso à informação, sendo que isto também resultou numa explosão da quantidade da informação, fazendo com que cada vez mais os sistemas tenham que possuir mais complexidade para tratar esses dados e informações.

A fim de resolver esse problema, várias técnicas, estratégias e metodologias foram estudadas, como o SOA e o MDA. Após esses estudos, foram selecionadas as melhores características de ambas para que fosse integrado em uma solução única. Sendo assim, essa dissertação propôs uma nova abordagem de programação onde se é usada uma estrutura de dados pró-ativa chamada de Dados Autonômicos. Dessa forma é possível resolver os problemas de complexidade, inserindo-a no próprio Dado Autonômico e permitindo o reuso.

Os Dados Autonômicos não têm por intenção substituir as técnicas de desenvolvimentos atuais, mas sim complementá-las. Foram demonstradas suas características e benefícios, além de apresentada a arquitetura que suporta o Dado Autonômico.

Neste trabalho, foi feita uma comparação em relação à POO e a programação utilizando agentes. Foi possível verificar, através de várias comparações, que os Dados Autonômicos completam em vários quesitos essas duas técnicas.

Com objetivo de validar o estudo feito, uma ferramenta foi produzida implementando os Dados Autonômicos. O cenário escolhido foi o de Feeds RSS. A ferramenta visou melhorar a filtragem de notícias, dando ao usuário a opção de usar regras para fazer a filtragem, além de poder usar regras de outros usuários do sistema. Uma grande dificuldade encontrada foi a grande falta de padronização na disponibilização de informações nos arquivos RSS.

Foi feito um experimento com essa ferramenta que objetivou verificar se, através dos Dados Autonômicos, essa ferramenta ganhou propriedades Autonômicas. O experimento demonstrou que sim, mostrando bons resultados em relação às

características autonômicas. Existem técnicas que conseguem resultados melhores para algumas técnicas autonômicas testadas. Porém não era do objetivo desse trabalho, conseguir os melhores resultados para cada uma das técnicas empregadas, mas sim fazer com que a ferramenta tivesse um conjunto de técnicas autonômicas de modo a provar que o sistema ganhava propriedades autonômicas.

Além disso, outros exemplos de aplicação foram citados, mostrando que é possível incluir os Dados Autonômicos em diversas aplicações para que estas também possam gozar de características autonômicas.

Como trabalhos futuros têm-se o aprimoramento da arquitetura dos Dados Autonômicos para que seja mais bem explorada e a proposição de novas aplicações que possam usar Dados Autonômicos. Além, claro, de fazer comparações entre aplicações construídas com e sem Dados Autonômicos para verificar como é o desempenho de cada uma.

Em relação à ferramenta produzida, tem-se como objetivo aprimorar as funções de Auto-Otimização e Auto-Proteção para gerarem melhores resultados. Para isso, é necessário um estudo mais aprofundado de técnicas específicas para esses campos. Além disso, propõe-se usar melhores técnicas para recomendação de regras através do *feed back* do usuário, técnica que não foi explorada neste trabalho.

Além disso, é desejável que sejam feitos mais experimentos com a aplicação, para testar os diversos operadores em separado, verificando a eficiência dos mesmos. Outros experimentos ainda são desejáveis, como comparar a aplicação com outras que têm sido desenvolvidas recentemente, e que também tem o objetivo de descartar noticias autonomicamente.

Por fim, propõe-se, como um trabalho futuro mais ambicioso, integrar todas as técnicas que estão sendo pesquisadas atualmente no laboratório de Banco de Dados da COPPE/UFRJ e fazer um único portal para filtragem e leitura de notícias.

Capítulo 8. Referências Bibliográficas

ABITEBOUL, S., 2002. "Active XML: A Data-Centric Perspective on Web Services" ***Journees Bases de Donnees Avancees (BDA)***, Evry, France

AHMED, S.; MITHUN, F., 2004. Word stemming to enhance spam filtering. *IN PROCEEDINGS OF CONFERENCE ON EMAIL AND ANTI-SPAM (CEAS)*, Mountain View, USA. Disponível em:<http://citeseerx.ist.psu.edu/viewdoc /summary?doi=10.1.1.59.9479>.

ALLEN, J. *et al*., 2002. "A problem solving model for collaborative agents". In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, Bologna, Italy. pp.774-781. Disponível em:<http://portal.acm.org/citation.cfm?id=544923>

ALVES, M. D. D. R.; SOUZA., 2007."Estudo de correspondência de elementos metadados: DUBLIN CORE e MARC 21". ***Revista Digital de Biblioteconomia e Ciência da Informação***, v. Volume 4, N° 2, pp.20-38.

ANTONIE, M.; ZAIANE, O., 2002. "Text document categorization by term association". In: *Data Mining, 2002. ICDM 2002. Proceedings. 2002 IEEE International Conference on*. pp.19-26.

ARLOW, P. J.; NEUSTADT, I., 2004. ***Enterprise Patterns and MDA***. Addison-Wesley, ISBN: 032111230X, 9780321112309, 2004.

ASHBY, P. W. R. 1960 ***Design for a Brain***. 2° ed. Michigan University 268 pgs.

BAADER, F. *et al*. 2005 "Mechanizing Mathematical Reasoning",

Cap. Description Logics as Ontology Languages for the Semantic Web. pp. 228-248.

BAEZA-YATES, P. R.; RIBEIRO-NETO, B., 2007. ***Modern Information Retrieval***. 2nd. ed, Canada, Pearson Education, ISBN: 9780321416919, .

BARROS, A. *et al*., 2002."Um Estudo Experimental sobre a Utilização de Modelagem e Simulação no Apoio à Gerência de Projetos de Software". In: *XIX Simpósio Brasileiro de Engenharia de Software proceedings*, Uberlândia, MG.

BENNANI, M. N.; MENASCE, D. A, 2005. "Resource Allocation for Autonomic Data Centers Using Analytic Performance Models". Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.7181>.

BERNERS-LEE, T; HENDLER, J.; LASSILA, O. 2001. "The Semantic Web". Scientific American. vol. 284, nº. 5, p. 34-43

BÉZIVIN, J., 1998. "Who's Afraid of Ontologies?". Disponível em:<http://www.metamodel.com/oopsla98-cdif-workshop/bezivin1/>.

BLAKE, M. 2001."Rule-driven coordination agents: a self-configurable agent architecture for distributed control". In: *Proceedings 5th International Symposium on Autonomous Decentralized Systems*, pp.271-277 Dallas, TX, USA.

BORKO, H.; BERNICK, M., 1963. "Automatic Document Classification" ***Journal of ACM***, v. 10, n. 2, pp 151-162.

BRAUN, P.; ROSSAK, W., 2005 ***Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*** Morgan Kaufmann, ISBN: 1558608176, 9781558608177

BROOKS, F. 1987. "No Silver Bullet: Essence and Accidents of Software Engineering" ***Computer***, v. 20, n. 4, pp 10-19.

CARNEY, D. 1997. Assembling Large Systems from COTS Components: Opportunities, Cautions, and Complexities. pp.14. Disponível em:<http://www .sei.cmu.edu/activities/cbs/monographs/assembling.systems.pdf>. Acesso em: Fevereiro 26, 2009.

CHAMBERLIN, P. D. D. 1998. ***A Complete Guide to DB2 Universal Database***. Morgan Kaufmann, ISBN: 1558604820, 9781558604827.

CLARKE, C. L. A. *et al*. 2003"A reliable storage management layer for distributed information retrieval systems". In: *Proceedings of the twelfth international conference on Information and knowledge management,*. ACM. pp.207-215. New Orleans, LA, USA

COX, B. J., 1990. "Planning the Software Industrial Revolution" ***IEEE Software***, v. 7, n. 6, pp 25-33.

CRUZ, P. C.; RIBEIRO, U. 2003. ***Metodologia científica: teoria e prática***. 1st. ed, Rio de Janeiro, Axcel Books do Brasil, ISBN: 8573231866, 9788573231861.

D'SOUZA, P. D. F.; WILLS, A. C. 1998. ***Objects, Components, and Frameworks with UML***. Addison-Wesley, ISBN: 0201310120, 9780201310122.

DUBLIN CORE METADATA INITIATIVE. 2009. Dublin Core Metadata Element Set, Version 1.1: Reference Description. 2004. Disponível em:<http://dublincore.org/documents/dces/>. Acesso em: Abril 4, 2009.

ELRAD, T. *et al*. 2001. "Discussing aspects of AOP" ***Commun. ACM***, v. 44, n. 10, pp 33-38.

FRIEDMAN, E. 2009. ***Jess in action: rule-based systems in java***. Manning Publications Co., 2003. Disponível em:<http://portal.acm.org/citation. cfm?id=1407161>. Acesso em: Março 1, 2009.

GAMMA, E. *et al*. 1995. ***Design Patterns: Elements of Reusable Object-Oriented***

Software. Addison-Wesley, ISBN: 0201633612, 9780201633610.

GARNER, R. SOA=MBA - IT Channel., 2004. Disponível em: <http://www.crn.com /it-channel/26806302>.

GRADECKI, P. J.; LESIECKI, N., 2003 ***Mastering AspectJ***. Wiley, ISBN: 0471431044, 9780471431046.

GREGORY, R.; GANGER, J., 2003. *Self-* Storage: Brick-based storage with automated administration*. Technical report. Carnegie Mellon University. Pittsburgh:

GUARINO, N., 1997. "Information Extraction A Multidisciplinary Approach to an Emerging Information Technology". **Cap. Semantic matching: Formal ontological distinctions for information organization, extraction, and integration**. pp. 139-170.

HAN, P. J.; KAMBER, M., 2006. ***Data Mining***. Morgan Kaufmann, ISBN: 1558609016, 9781558609013.

HARIRI, S. *et al*., 2006 "The Autonomic Computing Paradigm" ***Cluster Computing***, v. 9, n. 1, pp 5 - 1.

HOUAISS, A., 2001. Cap. Dicionário Houaiss da Língua Portuguesa.

HWANG, C. H. 1999. "Incompletely and Imprecisely Speaking : Using Dynamic Ontologies for Representing and Retrieving Information "Sweden. Disponível em:<http://www.cs.aue.auc.dk/ ~legind/ IRspring2004/ StudSem_EMNER/semArtikler/hwang.pdf>. Acesso em: Fevereiro 14, 2009.

IAN HORROCKS.2005. "OWL Rules, OK?" Washington, D.C., USA

IBM. An architectural blueprint for autonomic computing. 2006. Disponível em:<http://www-01.ibm.com/software/tivoli/autonomic/pdfs/

AC_Blueprint_White_Paper_4th.pdf>. Acesso em: Abril 21, 2009.

IBM., 2009. "Autonomic Computing Manifesto". Disponível em:<http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf>. Acesso em: Fevereiro 14, 2009.

JUNIOR, P. V. G.; WINCK, D. V., 2006. ***AspectJ - Programação Orientada a Aspectos com Java***. Novatec, ISBN: 857522087X, 978857522087.

K SOFT., 2009 "Your RSS Feed Might Look Like Spam. *RSS Especification"*, Disponível em:<http://www.rss-specifications.com/spam-rss.htm>. Acesso em: Julho 18, 2009.

KEPHART, J.; CHESS, D., 2003 "AC Vision of Autonomic Computing". Disponível em:<http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf>. Acesso em: Fevereiro 14, 2009.

KICZALES, G. *et al*., 1997 "Aspect-Oriented Programming" Springer-Verlag, pp.220-242. Finland.

KRAFZIG, P. D. *et al*., 2005 ***Enterprise SOA: Service-Oriented architecture Best Practices***. Prentice Hall, ISBN: 0131465759, 9780131465756.

LIEBERHERR, K. J. *et al*. 1994 "Adaptive object-oriented programming using graph-based customization" ***Commun. ACM***, v. 37, n. 5, pp 94-101.

LIGHTSTONE, S. S. *et al*., 2002 "Toward autonomic computing with DB2 universal database" ***SIGMOD Records.***, v. 31, n. 3, pp 55-61.

LIN, C. 1995. "Knowledge-based Automatic Topic Identification." Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.59>. Acesso em: Novembro 2, 2009.

LÜDKE, M.; ANDRÉ, M. E. A., 1990. ***Pesquisa em Educação: Abordagem***

Qualitativa. 1 ed. São Paulo. EPU.

MAEDCHE, A. OI-Modeler User´s Guide., 2002. Disponível em: <http://kaon.semanticweb.org/docus/Manual_KAON-OI-Modeler_ November_2002.pdf>. Acesso em: Fevereiro 14, 2009.

MANCINI, E. P. *et al*., 2006. "Mobile Agents Self-optimization with MAWeS.". In: *PARA*, Springer, pp.1158-1167. Disponível em:<http://dblp.uni-trier.de/db/conf/para/para2006.html#ManciniRVV06>. .

MARCONI, P. M. D. A.; LAKATOS, E. M., 2007. ***Metodologia científica***. 5th. ed, Atlas, ISBN: 8522447624, 9788522447626.

MDA. *OMG Model Driven Architecture*, 2009. Disponível em: <http://www.omg.org/mda/>. Acesso em: Fevereiro 27, 2009.

MILI, H., 1995 "Reusing software: issues and research directions" ***Software Engineering, IEEE Transactions on***, v. 21, n. 6, pp 528-562.

MIRANDA, M. *et al*., 2006 "Building Tools for Emergent Design with COPPEER" 10th International Conference on Computer Supported Cooperative Work in Design . pp. 550-555. Nanjing, China.

MOURA, A. M. C., 2006. "Linguagens de Regras p/ a Web", Notas de aula do curso de Banco de Dados. IME, 2006.

NGUYEN, T. *et al*.2009 "Autonomic data management system in grid environment" ***Journal of Algorithms & Computational Technology***, v. 3, pp 155-177.

NOY, N. F.; MCGUINNESS, D. L. 2001 "Ontology Development 101: A Guide to Creating Your First Ontology". Disponível em:<http://eprints.kfupm.edu.sa/55795/>. Acesso em: Fevereiro 14, 2009.

OLIVEIRA, J. *et al*. 2006 "Symptom Analysis of a Web Server Log" LAACS-Latin

American Autonomic Computing Symposium, Campo Grande, MS.

OSCAR *et al*. 2003. "Methodologies, tools and languages for building ontologies. Where is their meeting point?" ***Data & Knowledge Engineering***, v. 46, n. 1, pp 41-64.

OSSHER, H.; TARR, P. 1999 "Multi-dimentional separation of concerns in Hyperspace". In: Aspect-Oriented Programming Workshop – 13th European Conference on Object-Oriented Programmming (ECOOP'99). Lisboa, Portugal.

OWL. OWL Web Ontology Language Overview. 2003. Disponível em:<http://www.w3.org/TR/2003/PR-owl-features-20031215/>. Acesso em: Fevereiro 14, 2009.

PARASHAR, M.; HARIRI, S. 2005 "Unconventional Programming Paradigms",. **Cap. Autonomic Computing: An Overview.** pp. 257-269.

PARK, J. *et al*., 2005 "Proactive Self-Healing System based on Multi-Agent Technologies". In: *Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications*, IEEE Computer Society, pp.256-263.

PINHEIRO, W., SILVA, Marcelino. RODRIGUES, Thiago. SOUZA, Jano, XEXEO, Geraldo. 2010a. " Discarding Similar Data with Autonomic Data Killing Framework based on High-Level Petri Net Rules: an RSS Implementation" In: 6th International Conference on Autonomous & Autonomic Computing. ICAS (10), Cancun, Mexico.

PINHEIRO, Wallace., 2010b, Arcabouço Autonômico de Padrões Para Eliminação de Dados. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

PINHEIRO, Wallace. SILVA, Marcelino. RODRIGUES, Thiago. SILVA, Marcelo, SILVA, Marcio, SOUZA, Jano, XEXEO, Geraldo. 2009a."Autonomic RSS: Discarding Irrelevant News" In: *5th* International Conference on Autonomous &

Autonomic Computing. *ICAS (09),* Valencia, Spain.

PINHEIRO, W. SILVA, M. BARROS. R. SOUZA, J, XEXEO, G. 2009b. "Autonomic Collaborative RSS: an Implementation of Autonomic Data using Data Killing Patterns". In: *13th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2009)*, Santiago, Chile.

PINHEIRO, W. A. SOUZA, J. XEXEO, G. 2008. "Autonomic Patterns: Modelling Data Killing Patterns Using High-Level Petri Nets". In: International Conference on Autonomic and Autonomous Systems. IEEE Computer Society pp.198-203. Los Alamitos, CA, USA

PLÁŠIL, F.; STAL, M. 1998. "An architectural view of distributed objects and components in CORBA, Java RMI and COM/DCOM" ***Software - Concepts & Tools***, v. 19, n. 1, pp 14-28,.

PRESSMAN, R. S. 2004. ***Software engineering***. 6th. ed, McGraw-Hill, ISBN: 007301933X, 9780073019338.

PROTÉGÉ. User's Guide. 2000. Disponível em:<http://protege.stanford.edu/publications/UserGuideA4.pdf>. Acesso em: Fevereiro 14, 2009.

RDF. Resource Description Framework (RDF) Model and Syntax Specification. 1999. Disponível em:<http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>. Acesso em: Fevereiro 14, 2009.

RDF SCHEMA. 2009 Resource Description Framework (RDF) Schema Specification. 1999. Disponível em:<http://www.w3.org/TR/1998/WD-rdf-schema/>. Acesso em: Fevereiro 14, 2009.

RESCONI, G.; JAIN, L. C. 2004. ***Intelligent agents***. 1 ed. Springer, ISBN: 3540220038, 9783540220039.

RESENDE, P. A. M. P. D.; SILVA, C. C. D., 2005. ***Programação orientada a***

aspectos em Java desenvolvimento de software. brasport, ISBN: 8574522120, 9788574522128.

DE REZENDE, J. *et al*. 2008 "Sharing the knowledge generated during a Water Resources project". In: *12th International Conference on Computer Supported Cooperative Work in Design. CSCWD 2008.* 2008. pp.300-305.

RHEA, S. *et al*. 2003 "Pond: the oceanstore prototype" ***Proceeding of Conference on File and Storage Technologies,*** *San Francisco, USA.* pp 1--14.

RITTINGHOUSE, John. RANSOME, James. 2009. **Cloud Computing**. CRC Press, 320 pp. ISBN: 9781439806807.

ROSEN, M. *et al*. 2008. ***Applied Soa: Service-Oriented Architecture And Design Strategies***. Wiley India Pvt. Ltd., ISBN: 8126517662, 9788126517664.

RUDIO, F. V. 2001 .***Introdução ao Projeto de Pesquisa Científica***. 29th. ed, editora Vozes.

RUSSELL, P. S. J. *et al*. 2004 ***Artificial Intelligence***. 2nd ed. Prentice Hall, ISBN: 9780137903955.

SCHACH, P. S. R. 2005 ***Object-oriented and Classical Software Engineering***. McGraw-Hill, ISBN: 0072865512, 9780072865516.

SILVA, Marcelino. OLIVEIRA, Jonice. SOUZA, Jano 2008 "Autonomic Data: Use of Meta Information to Evaluate, Classify and Reallocate Data in a Distributed Environment" LAACS-Latin American Autonomic Computing Symposium, pp.73-76 Gramado - RS, Brasil.

SOUZA, J. *et al*. 2006 "Knowledge Chains Learning Certification Using Intelligent Agents to Create Personalized Dynamic Tests" pp.175-182 San Sebastian, Lisboa, Portugal..

SOUZA, M. I. F. *et al*.2000 "Metadados para a descrição de recursos de informação eletrônica:utilização do padrão Dublin Core". ***Ciência da Informação***, v. 29, pp.93-102.

SOWA, P. J. F., 2000. ***Knowledge Representation***. ISBN: 0534949657, 9780534949655

SPAGNOLI, L.; BECKER, K., 2003. *Um estudo sobre o Desenvolvimento Baseado em Componentes*. pp.48. PUC-RS. Porto Alegre RS:

SPYNS, P. *et al*., 2002. "Data modelling versus ontology engineering" ***SIGMOD Rec.***, v. 31, n. 4, pp 12-17,.

STERRITT, R.; HINCHEY, M. 2005. "Engineering Ultimate Self-Protection in Autonomic Agents for Space Exploration Missions". In: *Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, IEEE Computer Society. pp.506-511. Greenbelt, MD, USA

SWRLLanguage FAQ. 2009. Disponível em:<http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ>. Acesso em: Fevereiro 15, 2009.

SWATOUT, W., 1999 "Ontologies". ***IEEE Inteligent Systems & their applications***, v. 14 n. 1, jan/fev 1999.

TECUCI, P. G.; DYBALA, T., 1998. ***Building Intelligent Agents***. 1st ed. Morgan Kaufmann Publishers Inc., ISBN: 0126851255, 9780126851250.

TRIVIÑOS, P. A. N. S., 1994 ***Introdução à pesquisa em ciências sociais***. editora Atlas, 4° ed. ISBN: 8522402736, 9788522402731.

USCHOLD, P. M. *et al*., 1996 ***Ontologies: Principles, Methods and Applications***. The Knowledge Engineering Review, v.11, n.2, pp 93-155.

VILELA, C., 2007. "*COE Um Editor Colaborativo de Ontologias*". Projeto Final de

Curso (Ciência da Computação) Universidade Federal do Rio de Janeiro.

WANG, X. *et al*., 2007. "Appliance-Based Autonomic Provisioning Framework for Virtualized Outsourcing Data Center". In: *Autonomic Computing, International Conference on*, Los Alamitos, CA, USA. IEEE Computer Society, pp.29.

WINER, D. RSS 2.0 Specification. 2003. Disponível em:<http://cyber.law.harvard.edu/rss/rss.html>. Acesso em: Março 1, 2009.

WORDNET., 2009. Disponível em:<http://wordnet.princeton.edu/>. Acesso em: Julho 25, 2009.

WU, C. *et al*. 2001 "Configurable Mobile Agent and Its Fault-Tolerance Mechanism". In: *Computer Networks and Mobile Computing, International Conference on*, IEEE Computer Society, pp.380. Los Alamitos, CA, USA

XML. Extensible Markup Language (XML). 2009. Disponível em:<http://www.w3.org/XML/>. Acesso em: Fevereiro 14, 2009.

XML Schema. 2001. Disponível em:<http://www.w3.org/XML/Schema#dev>. Acesso em: Fevereiro 14, 2009.

ZHANG, Z. *et al*. 2004. "RepStore: A Self-Managing and Self-Tuning Storage Backend with Smart Bricks" ***Proceedings of International Conference on Autonomic Computing***, pp 122-129, New York, USA.