UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Diogo de Campos

REPRESENTAÇÃO DE DADOS SEMÂNTICOS EM AGENTES BDI

Florianópolis

2014

Diogo de Campos

REPRESENTAÇÃO DE DADOS SEMÂNTICOS EM AGENTES BDI

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Prof. Dr. Ricardo Azambuja Silveira
Coorientador: Prof. Dr. Elder Rizzon Santos

Florianópolis

2014

Diogo de Campos

REPRESENTAÇÃO DE DADOS SEMÂNTICOS EM AGENTES BDI

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 1° de dezembro de 2014.

Prof. Dr. Ronaldo dos Santos Mello
Coordenador do Programa

Prof. Dr. Elder Rizzon Santos
Coorientador
Universidade Federal de Santa Catarina

Banca Examinadora:

Prof. Dr. Ricardo Azambuja Silveira
Orientador
Universidade Federal de Santa Catarina

Profa. Dra. Diana Francisca Adamatti
Universidade Federal do Rio Grande

Profa. Dra. Jerusa Marchi
Universidade Federal de Santa Catarina

Prof. Dr. Mauro Roisenberg
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Bruna Rocha de Oliveira e Edson Carlo de Campos

por me darem motivação para completar este trabalho.

To achieve great things, two things are needed: a plan, and not quite enough time.

Leonard Bernstein

RESUMO

A crescente necessidade de informação e consequente aumento no volume de comunicação tem conduzido à adoção de dados semânticos e resultando numa demanda por ferramentas que manipulam tais dados. Com o avanço de dados semânticos na Web, estamos atingindo um ponto onde ferramentas de software devem se adaptar a este novo formato.

Este trabalho propõe um novo modelo para desenvolvimento de agentes inteligentes da IA baseados em um modelo de lógica BDI, com o objetivo de permitir comunicação livre de ambiguidade e capaz de reutilizar dados semânticos já existentes na Web. O modelo propõe uma alternativa para a representação de dados semânticos no agente, como estes dados podem ser armazenados e utilizados para comunicação com *triplestores* da Web Semântica e também com outros agentes (semânticos e não semânticos).

Com foco na representação destes dados semânticos, são exploradas maneiras de se integrar informação semântica à um agente, seus processos e estados e porque esta integração pode levar a melhores resultados quando acessando informação na Web. Além disso, é demonstrado o ganho que se pode obter ao reutilizar dados já existentes na Web Semântica, e como isto pode facilitar o desenvolvimento de novas aplicações baseadas em agentes.

Finalmente, para avaliar o modelo proposto, é feita uma comparação qualitativa com outros trabalhos na área, levantando as diferenças, motivações e melhorias feitas neste trabalho. Uma implementação deste modelo é apresentada através de um arcabouço criado para demonstrar e validar as intenções deste trabalho. Este arcabouço chamado *PySA* é descrito expondo os principais pontos defendidos na proposta, testando em situações hipotéticas e exemplos reais a comunicação e aprendizado semanticamente rico que são os objetivos do trabalho.

Palavras-chave: Sistemas orientados à Agentes. Modelo de lógica BDI. Web Semântica. RDF. Triplestores.

ABSTRACT

Increasing needs for information and consequent increase in communication volume are leading to a widespread adoption of semantic data and demand for tools that manipulate such data. With the uprising of Semantic Web data, we are reaching a point where software tools must adapt to this new format.

This work proposes a new model for developing intelligent agents based on a BDI reasoning model, with the goal of allowing ambiguity-free communication and capable of reusing semantic data that already exists in the Web. The model proposes an alternative to representing semantic data in agents, and how this data can be stored and utilized to communicate with Semantic Web stores and also other agents (semantic and non-semantic).

Focusing on the representation of this semantic data, this work explores ways to integrate semantic information to an agent, it's processes and states and why this integration can lead to better results when acessing information in the Web. On top of that, this work demonstrates what gain can be obtained from reutilizing data that already exists in the Semantic Web, and how this eases the development of new agent-based applications.

Finally, to evaluate the proposed model, a qualitative comparison is made with similar work in the area, comparing the differences, motivations and improvements made in this project. An implementation of this model is presented through a framework created to demonstrate and validate in practice the intentions of this project. This framework called PySA is described, exposing the main values defended in the proposal, testing in hypothetical situations and real examples the semantically rich communication and learning capabilities that are the main goal of this work.

Keywords: Agent oriented Systems. BDI reasoning model. Semantic Web. RDF. Triplestores.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Este trabalho tem como motivação a crescente necessidade por sistemas automatizados de comunicação semântica e processos de *Web crawling* (programas de computador que navegam de maneira automatizada pela Web, indexando informações)(KOBAYASHI; TAKEDA, 2000). Este tipo de sistema surgiu para resolver o problema de ambiguidade que existe na comunicação tradicional entre sistemas na Web, incapaz de diferenciar conceitos ontológicos sobre um conjunto de dados.

Agentes de software estão sendo propostos como um novo modelo para construção de sistemas computacionais há algum tempo e este tipo de software disponibiliza muitas capacidades como proatividade, reatividade, habilidade social e autonomia. Com a intenção da elevação da Web Semântica como novo rumo de evolução da Internet, nota-se uma falta de apoio para processos que combinam estas duas áreas. Esta integração é vista como benéfica pelo próprio criador da Web Semântica, que tem a intenção de usar agentes inteligentes para populá-la com novos dados (BERNERS-LEE; HENDLER; LASSILA, 2001). Seguindo esta ideia e tendo uma maior sincronia entre agentes inteligentes e dados com informação semântica, será possível facilitar a integração entre diferentes aplicativos automatizados, criando espaço para uma nova gama de ferramentas baseadas em dados comuns e compartilhados, além do conhecimento local introduzido pelo desenvolvedor.

A contribuição principal deste trabalho se faz na forma de um modelo de agentes que integra o modelo de lógica Belief Desire Intention (BDI) (BRATMAN; ISRAEL; POLLACK, 1988)(DICKINSON, 2006)(RAO; GEORGEFF, 1995) com dados semânticos (ANTONIOU; VANHARMELEN, 2004) como uma solução para o problema de ambiguidade na comunicação entre sistemas na Web. O modelo BDI que serve de base para este trabalho cria um paralelo entre a representação de agentes e do conhecimento humano, tentando aplicar a nossa maneira de pensar, ao modelo do agente. Os chamados dados semânticos envolvidos neste trabalho representam um modelo de dados que possui expressividade semântica, capaz de formar relações com outros dados similares (BERNERS-LEE; HENDLER; LASSILA, 2001)(DICKINSON, 2006). Estes tipos de dados são geralmente expressos através de estruturas num formato Sujeito-Predicado-Objeto, e permitem que processos computacionais possuam maior capacidade de interpretação.

Junto com o avanço na área de agentes, a Web Semântica também tem a ganhar com a presença de agentes semânticos automatizados,

pois ele servem como bons provedores de dados (através de software de *Web crawling*, capaz de buscar, indexar e organizar novas informações) e também porque o uso de ferramentas automáticas fortalece a necessidade de um padrão bem definido de comunicação, e é capaz de detectar problemas difíceis de se ver através de uma análise manual.

O modelo apresentado neste trabalho contribui para preencher a necessidade por sistemas que trabalhem de forma semanticamente consciente na Web. Além disso, alguns autores lidam com a necessidade por reusabilidade no desenvolvimento de agentes (AGGARWAL; SINGH, 2013), mas isto é pouco desenvolvido em trabalhos que integram as áreas de agentes com Web Semântica. Este trabalho resolve parte deste problema pois o modelo traz vantagens em termos de reusabilidade de dados semânticos utilizados pelos agentes, permitindo que novas aplicações sejam desenvolvidas rapidamente.

1.1 PROBLEMÁTICA

O propósito deste trabalho é apresentar uma nova abordagem para lidar com problemas de comunicação e ambiguidade em sistemas de software que interagem com dados na Web. De maneira geral, as perguntas que guiam este trabalho são:

- Como podemos criar sistemas que operam sobre dados na Web e se comunicam sem ambiguidade?

- Como compartilhar e reutilizar informação com conteúdo semântico entre sistemas deste tipo?

Desenvolvendo com base em conceitos atuais de Web Semântica, e de técnicas para desenvolvimento de aplicação orientadas à agentes podemos tornar estas perguntas mais específicas:

- Como representar dados semânticos em agentes BDI?

- Como utilizar esta informação na comunicação entre agentes e outros sistemas?

1.2 OBJETIVOS

1.2.1 Geral

Elaborar um modelo para representação de informação semântica em sistemas de agentes BDI com base nas linguagens e meios de representação de conhecimento utilizados na Web Semântica para proporcionar maior facilidade de uso de agentes na Web.

1.2.2 Específicos

- Mapear os meios de representação usados na Web Semântica.
- Definir um modelo para representação de crenças, desejos e intenções de agentes com informação semântica.
- Implementar o modelo proposto em um arcabouço de desenvolvimento de agentes com conhecimento semântico.
- Testar a implementação e validar o modelo proposto através de um caso de uso prático.
- Validar qualitativamente o modelo proposto em comparação com outros trabalhos relacionados.

Este trabalho está dividido nos seguintes capítulos: esta introdução, um capítulo para revisão da literatura (conceitos comuns e revisão de trabalhos correlatos), apresentação do modelo proposto, apresentação de um arcabouço que implementa este modelo (junto com um caso de uso real implementado neste arcabouço) e finalmente um capítulo para conclusões e trabalhos futuros.

A introdução contém o tema, problemática, objetivos do trabalho e descreve as contribuições feitas nesta dissertação. O capítulo referente à revisão da literatura tem o objetivo de situar o leitor e criar uma visão geral do problema que está sendo tratado, além de apresentar trabalhos já existentes na área e como eles tratam os problemas descritos na problemática.

As contribuições do projeto entram nos capítulos seguintes. Primeiro é apresentada uma proposta de solução para o problema, na forma de um modelo de representação de dados e comunicação entre agentes e entidades externas que contém informação semântica. Os

principais pontos defendidos pela proposta são apresentados e o modelo é descrito no mesmo capítulo.

No capítulo seguinte é apresentado um arcabouço criado para este trabalho com o objetivo de testar o modelo proposto. É feito um paralelo deste arcabouço com o modelo, mostrando onde cada ponto defendido está implementado e como eles interagem, utilizando exemplos teóricos e reais em pseudocódigo. Este capítulo também contém um caso de uso na forma de uma aplicação criada usando o arcabouço desenvolvido. Através deste exemplo, são demonstradas as capacidades introduzidas pelo modelo proposto e ele serve para validar tanto o modelo quanto a implementação do arcabouço.

Usando a definição do modelo, e o arcabouço desenvolvido, o trabalho é avaliado de forma qualitativa, sendo apresentados os principais pontos e vantagens do modelo em comparação com trabalhos similares da área. Esta comparação também é feita através de exemplos de código que comparam como uma mesma aplicação seria implementada usando este modelo, e usando trabalhos similares da área.

Finalmente, o trabalho é sumarizado no capítulo de conclusão, com palavras finais sobre os resultados da dissertação, uma análise geral do modelo proposto, quais pontos positivos ele traz, como se compara com outros trabalhos e qual a contribuição do trabalho. Alguns parágrafos finais apresentam sugestões de trabalhos futuros relacionados à este projeto, e possíveis melhorias ao modelo apresentado.

2 REVISÃO DA LITERATURA

2.1 AGENTES

Um agente pode ser definido como uma entidade computacional que possui um comportamento autônomo, permitindo-lhe decidir suas próprias ações. Esse comportamento autônomo inclui a percepção do ambiente, através de sensores, e suas ações sobre ele, por meio de seus atuadores. (ALVARES; SICHMAN, 1997)

Outro autor define um agente $\mathcal{A}g$ como uma tupla de quatro valores $\mathcal{A}g = \{Sit, Act, Dat, f_{Ag}\}$. Sit é um conjunto de situações onde o agente pode estar (a representação depende da capacidade sensorial do agente), Act é o conjunto de ações que $\mathcal{A}g$ pode utilizar, e Dat é o conjunto de valores representados pelos dados internos de $\mathcal{A}g$. Para determinar a sua próxima ação, $\mathcal{A}g$ utiliza $f_{Ag} : Sit \times Dat \rightarrow Act$ aplicada à situação atual e os valores dos dados internos do agente. (AFSHARCHI et al., 2013)

Independente da definição do que é um agente, pode-se dizer que uma série de características são comuns à todas elas (RUSSELL; NORVIG, 2003):

- Reatividade: o agente tem capacidade de perceber o ambiente e realizar uma ação de acordo com o estímulo recebido;
- Autonomia: o agente pode tomar suas próprias decisões de acordo com os estímulos do ambiente e seu conhecimento prévio;
- Cooperação: capacidade de cooperar com outros agentes para atingir um objetivo em comum;
- Comunicação de conhecimento: capacidade de se comunicar com outros agentes para troca de conhecimento e informações, em alguma linguagem de mais alto nível, e não apenas uma comunicação seguindo algum protocolo;
- Personalidade: demonstração de uma personalidade do agente;
- Adaptabilidade: habilidade de se adaptar a mudanças no ambiente em que atua;
- Capacidade de inferência: capacidade de criar conclusões a partir de informações do ambiente e conhecimento prévio;

- Mobilidade: possibilidade de migrar para uma outra plataforma;
- Continuidade temporal: continuidade do agente com suas características mesmo após longos períodos de tempo.

Existem algumas classificações mais específicas sobre agentes (agentes reativos, agentes cognitivos, etc (BITTENCOURT, 2001)), e este trabalho é baseado em agentes que seguem o modelo BDI (*Belief-Desire-Intention* (RAO; GEORGEFF, 1995), descrito na próxima seção.

2.2 MODELO BDI

O modelo BDI é inspirado no conceito de estados mentais com origem em estudos do raciocínio prático humano na teoria do filósofo Michael Bratman (BRATMAN, 1987)(BRATMAN, 1999). A representação de uma teoria do raciocínio prático humano pela IA é uma tentativa de se criar entidades ainda mais inteligentes e autônomas. Além de uma representação de um modelo interno do ambiente, agentes seguindo a arquitetura BDI também possuem a capacidade de tomar decisões lógicas através de seu raciocínio. Um agente BDI utiliza seu conhecimento, representado por seu modelo interno, como parte do seu processo de raciocínio que acaba por alterar seu estado, também referido como estado mental (WOOLDRIDGE, 2009). Esse estado é composto pelos três conceitos propostos pela teoria de Bratman, crenças, desejos e intenções, assim como dois novos conceitos utilizados em sistemas que seguem esse modelo: objetivos e planos. Esses cinco componentes definem o chamado estado mental do agente BDI e podem ser caracterizadas como (BRENNER; ZARNEKOW; WITTIG, 2012):

- Crenças (Beliefs): representam o conhecimento do agente sobre o mundo, e não necessariamente são informações corretas, podendo ser incompletas em muitos casos.
- Desejos (Desires): estão relacionados a estados do mundo que o agente gostaria de provocar. Podem existir desejos conflitantes entre si, ou com poucas chances de se tornar realidade, e influenciam na tomada de decisão sobre a próxima ação a tomar.
- Intenções (Intentions): podem ser definidas como objetivos que o agente se comprometeu a realizar. Usualmente, o agente não possui recursos suficientes para tentar atingir todos seus objetivos, e assim, intenções são um subconjunto dos objetivos que o

agente escolhe para efetivamente tentar alcançar, de acordo com os recursos disponíveis.

- Objetivos (Goals): formam um subconjunto dos desejos onde o agente tem possibilidade de atuar. Por esse motivo, diferentemente dos desejos, não devem existir objetivos contraditórios, devendo ser consistentes.

- Planos (Plans): são uma combinação das intenções. Cada intenção constitui um subplano de um plano geral de um agente, e o conjunto de todos os planos reflete as intenções do agente.

Aplicando a definição de (AFSHARCHI et al., 2013) ao modelo BDI, sobre um agente $\mathcal{A}g = (Sit, Act, Dat, f_{Ag})$ pode-se dizer que Dat é um outro conjunto $Dat = (Blf, Dsr, Int)$ que representa as bases de dados do agente (Crenças, Desejos, e Intenções respectivamente), Act é a base de planos e f_{Ag} é a função de deliberação lógica do Agente onde $f_{Ag} : Sit \times (Blf, Dsr, Int) \rightarrow Act$.

2.3 WEB SEMÂNTICA

A Web Semântica é um movimento guiado pela World Wide Web Consortium (W3C) que promove um padrão de formatos de dados semânticos na Internet (W3C, 2014c). O objetivo da inclusão de dados semânticos é de converter a atualmente desestruturada Web de dados em uma forma mais organizada, conectando informações similares.

O principal propósito da Web Semântica é guiar a evolução da Web atual (BERNERS-LEE; HENDLER; LASSILA, 2001), permitindo que usuários encontrem e compartilhem informação mais facilmente. Humanos são capazes de utilizar a Internet para várias tarefas, porém as páginas atuais foram feitas somente para o uso humano, e é muito difícil de se criar máquinas capazes de cumprir estes mesmo objetivos. A Web Semântica é uma visão onde esta informação possa também ser interpretada por máquinas, para que estas sejam capazes de realizar tarefas comuns como busca e combinação de informações na Web.

Aqui também estão situados *Web Services* Semânticos (*Semantic Web Service* ou SWS), que assim como *Web services* tradicionais, funcionam através de um sistema cliente-servidor, mas fazem parte da Web Semântica, pois os dados transmitidos possuem anotações e ontologias que servem para facilitar a comunicação e remover ambiguidade mesmo em comunicações entre máquinas (GESSLER et al., 2009).

A Web Semântica pode ser considerada um trabalho em progresso com o objetivo de melhorar o entedimento que as máquinas tem da informação na Web. Processos devem ser capazes de entender os dados que encontram, para isso, é necessário o uso de bases de conhecimento que relacionem o conhecimento puramente léxico com suas possibilidades semânticas (BERNERS-LEE; HENDLER; LASSILA, 2001).

2.3.1 Ontologias

Ontologias na ciência da computação representam um conjunto de conceitos dentro de um domínio, e relações entre pares de conceitos. Elas são utilizadas para modelar domínios e facilitar a dedução sobre conceitos semânticos. Alguns exemplos de tipos de ontologia são as Ontologias de Linguagem Natural, que relacionam conceitos diferentes e as Ontologias de Domínio, que agrupam conhecimento sobre um domínio (ANTONIOU; HARMELEN, 2008).

Extensamente utilizadas na Web Semântica, estas ontologias representam um dos maiores problemas para serem resolvidos na área, principalmente quando se trata do mapeamento e conexão entre diferentes ontologias. A criação destas conexões pode ser feita manualmente, mas existem várias técnicas construídas para auxiliar na automação da construção de conexões, extraindo ou deduzindo dados e relações a partir de informação já existente na Web (como métodos linguísticos, lógicos e estatísticos) (BERNERS-LEE; HENDLER; LASSILA, 2001) e (ANTONIOU; HARMELEN, 2008).

2.3.2 Linked Data

Linked Data ou dados linkados é o nome dado a um método para publicações de dados estruturados conectados uns aos outros. Segundo o criador do termo, Tim Berners-Lee, alguns poucos conceitos definem as regras que regem a organização de dados linkados (W3C, 2014a):

- Devem utilizar *URIs* (Uniform Resource Identifier) para denominar e referenciar recursos.
- Devem utilizar *URIs* HTTP para que estes recursos possam ser referenciados e pesquisados por pessoas e agentes.
- Devem providenciar informação útil sobre o objeto sendo referenciando, favorecendo padrões como *"SPARQL Protocol and RDF*

Query Language" (SPARQL)[1] e *Resource Description Framework* (RDF).

- Devem incluir conexões para outros objetos relacionados (através de suas *URIs*) quando publicados na Web.

O conceito bastante simples é solidificado nos padrões definidos pela W3C, analisados nas seções seguintes. A utilização de padrões e um modelo comum permite que qualquer aplicação possa atuar sobre um conjunto global de dados e informações semânticas, que é o propósito da Linked Data.

2.3.3 Formato RDF

Resource Description Framework (RDF) é um mecanismo utilizado para descrever objetos informacionais, e seus relacionamentos, ao contrário do padrão HTML que apenas liga documentos sem se preocupar com a natureza das conexões. O padrão RDF cumpre os passos definidos na seção sobre Linked Data através da maneira como descreve dados. O padrão permite que recursos sejam identificados através de suas *URIs* e conectados, formando um conjunto de dados estruturados. Aplicações que conhecem a *URI* de um artefato, podem a partir dele acessar todo este conjunto relacional. (W3C, 2014b) A representação de dados no formato RDF é feita através de triplas onde cada tripla é formada por um sujeito, um predicado e um objeto, do tipo "A se relaciona com B". O sujeito representa o recurso em si, através de uma *URI*, o objeto alvo pode ser um recurso RDF, ou apenas um valor literal. O predicado representa o tipo de relação entre os dois, isto é feito através de uma *URI* que referencia um de vários tipos de predicados disponíveis para o domínio.

Ao analisar várias destas triplas interconectadas, é possível imaginar um grafo de conexões de informação. O modelo de dados RDF possui algumas implementações padronizadas pela W3C, como os formatos RDF/XML e RDFa (Resource Description Framework in Attributes) (W3C, 2014b) e neles existem alguns tipos básicos de links RDF com conceitos externos:

- Links de vocabulário; que lidam com situações onde um termo semântico possui várias representações léxicas diferentes.

[1] SPARQL é uma linguagem utilizada para fazer buscas sobre bases de dados RDF (HEBELER et al., 2011).

- Links de identidade; que associam objetos do mundo real com uma ou várias *URIs*, através de *URI aliases*, que utilizam um simbolismo especial para indicar que múltiplas *URIs* referenciam o mesmo objeto.

- Links de relação; que fazem a conexão entre diferentes objetos, criando o citado grafo de dados semânticos.

2.3.4 DBPedia

Uma das principais razões para o avanço atual da Web Semântica, é a existência de conjuntos de triplas RDF disponíveis gratuitamente na Internet, através de projetos como a DBpedia, que extrai informações da Wikipedia e as torna disponíveis na Web utilizando tecnologias da Web Semântica e de dados linkados.

A DBPedia armazenam seus dados no formato RDF em *triplestores* (bases de dados para triplas) (RUSHER, 2003) e permite que aplicações independentes se comuniquem com uma grande base comum de dados, tornando o esforço da determinação de todas as relações semânticas muito menor. Segundo (LEHMANN et al., 2014), a DBPedia em inglês atualmente possui mais de 400 milhões de fatos que descrevem 3.7 milhões de "coisas" (*things*) conectadas através de um base de ontologias que contém 320 classes e 1.650 propriedades.

Segundo Tim Berners-Lee, a DBPedia é dos projetos mais famosos do esforço da disseminação de dados linkados (BERNERS-LEE, 2008).

2.4 TRABALHOS CORRELATOS

O problema de integração entre agentes e a Web Semântica é um problema conhecido, e alguns trabalhos existentes tratam deste problema de maneiras diferentes. Nesta seção serão apresentados alguns trabalhos e suas soluções serão avaliadas como possíveis alternativas aos problemas que motivaram este trabalho.

(NAKAYAMA; HARA; NISHIO, 2005) é um trabalho em forma bastante inicial onde os autores sugerem uma integração entre agentes inteligentes e *Web Services* Semânticos. É proposto o desenvolvimento de um objeto que encapsula a comunicação de dados de aplicações de comunicação (email, agendas, etc) e converte estes dados para o formato RDF. Os autores terminam o trabalho apenas com esta proposta,

mas não há menção sobre a capacidade destes objetos se comunicarem com outras bases de dados RDF já existentes.

Existem vários outros trabalhos que utilizam ontologias da Web Semântica como solução para o problema de comunicação ambígua entre agentes (LI; WU; YANG, 2005)(LIU; CHEN, 2012)(AFSHARCHI et al., 2013)(DICKINSON; WOOLDRIDGE, 2003)(KARDAS; DEMIREZEN; CHALLENGER, 2010)(KLAPISCAK; BORDINI, 2009)(MASCARDI et al., 2011)(HALAC; EKINCI; DIKENELLI, 2011) e um resumo deles é apresentado a seguir.

No artigo (LI; WU; YANG, 2005) os autores apresentam um novo protótipo de arcabouço para desenvolvimento de agentes que lida com o problema de compartilhamento de ontologias entres os agentes. O artigo propõe um modelo onde um agente pode servir um papel de integração de ontologias, chamado *OntologyAgent*. Utilizando este agente, outros agentes podem compartilhar e sincronizar suas ontologias particulares com as de outros agentes.

Apesar de claramente lidar com o problema de ambiguidade na comunicação entre agentes, este artigo não cita como o modelo proposto lida com a comunicação com bases de dados semânticas externas, e não reutiliza dados existentes na Web (apenas consegue reutilizar dados presentes em outros agentes).

Em (LIU; CHEN, 2012), os autores utilizam agentes BDI com conhecimento ontológico para se comunicar com SWSs. O trabalho foca na dificuldade de sincronizar a ontologia do agente com a ontologia usada pelos vários serviços existentes. De uma certa forma, o trabalho também trata do problema de comunicação ambígua, ao permitir que o agente se comunique com serviços semânticos, cujos dados utilizam técnicas da Web Semântica para evitar este problema. O trabalho também vai na direção de reaproveitar dados previamente existentes, desde que estes estejam disponíveis em algum SWS conhecido do agente.

Apesar disso, os autores não especificam como os dados que são obtidos destes serviços são armazenados na base de crenças do agente, o que indica que eles perdem esta conexão com a "base de dados" remota, agentes que venham a obter estas crenças no futuro não possuem mais conexão com o dado original.

Os autores de (AFSHARCHI et al., 2013) focam seu trabalho no desenvolvimento de um método que permita que agentes aprendam ontologias de outros agentes. Segundo os autores, apesar de já existirem vários trabalhos que tratam sobre como utilizar ontologias em agentes, existe pouco esforço na direção de permitir que agentes com ontologias

heterogêneas consigam se comunicar entre si. Isto é feito através de um método onde um agente inserido em um contexto com vários outros agentes pode perguntar à estes agentes se a sua interpretação de uma ontologia está correta (através de um série de perguntas que pode ter um resultado positivo ou negativo). Após questionar um número suficiente de agentes, o agente aluno (*LearningAgent*) associa esta nova ontologia à uma ontologia conhecida.

Esta capacidade de aprender novas ontologias isenta os agentes da necessidade por uma base ontológica comum, o que é uma grande vantagem, mas também sofre da desvantagem inerente à não-centralização de dados, que é a duplicação de informação.

O artigo (DICKINSON; WOOLDRIDGE, 2003) descreve uma arquitetura flexível para o desenvolvimento de agentes BDI em aplicações de Web Semântica, chamado *"Nuin"*. São apresentadas as principais características de projeto da plataforma, e é descrito como ela pode ser usada para prover grande utilidade aos projetistas de agentes.

A motivação do trabalho é bastante parecida com a desta dissertação, segundo os autores, agentes de software estão sendo propostos como um novo modelo para construção de sistemas computacionais faz algum tempo, e este tipo de software promete muitas capacidades inovadoras, como proatividade, reatividade, habilidade social e autonomia. Com a intenção da elevação da Web Semântica como novo rumo de evolução da Internet, nota-se a falta de aplicações de agentes semânticos, que é o foco do trabalho.

Para solucionar este problema, os autores apresentam *Nuin*, que, apesar de focar na ferramenta, possui em seu projeto várias decisões teóricas interessantes para o modelo proposto. O arcabouço tem seu projeto baseado na linguagem AgentSpeak(L) (RAO, 1996), onde um agente AgentSpeak(L) consiste em um conjunto de crenças construídas a partir de um alfabeto de termos de primeira ordem e um conjunto de planos. Neste artigo, a conexão entre dados semânticos e o conhecimento dos agente é feito através de chamados "Knowledge Sources" (KS), que devem atender a um conjunto de interfaces que permitam acesso a meta-dados semânticos, na forma RDF. A própria configuração do agente é especificada através de um modelo RDF (a arquitetura *Nuin* pode ser vista na figura 1).

No momento da escrita do artigo, o desenvolvimento da ferramenta ainda estava em progresso, então os autores baseiam boa parte do modelo em construções pré-existentes, como a utilização de uma versão modificada de interpretadores clássicos (DICKINSON; WOOLDRIDGE, 2003) e modelos de comunicação FIPA.

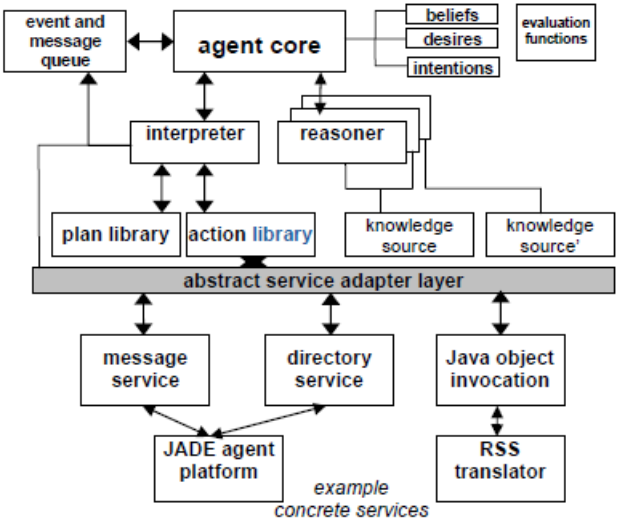

Figura 1 – Arquitetura Nuin
(DICKINSON; WOOLDRIDGE, 2003)

Os autores de (KARDAS; DEMIREZEN; CHALLENGER, 2010) utilizam *Nuin* como uma base para desenvolver uma nova linguagem de domínio (*Domain Specific Modeling Language* ou DSML) para modelagem de agentes. Esta linguagem tem como objetivo deixar mais explícito o papel de ontologias na criação de modelos de agentes, e incorpora conceitos como Agentes da Web Semântica e SMSs diretamente no metamodelo apresentado.

O modelo é bastante genérico, e apesar dos autores introduzirem um estudo de caso prático que segue este modelo, não fica claro como estes novos elementos do modelo permitem que um agente consiga se comunicar com outros agentes e serviços utilizando ontologias compartilhadas. Todas as referências a dados semânticos ficam nos agentes, ou seja, não há conexão direta dos dados com bases externas (e por consequência não há reutilização de dados comuns entre agentes).

Em (KLAPISCAK; BORDINI, 2009), os autores fazem outra proposta para a solução deste problema. Este artigo descreve a ferramenta JASDL (JasonAgentSpeak-DescriptionLogic), uma extensão da plataforma de agentes Jason (BORDINI; WOOLDRIDGE; HUBNER, 2007) que utiliza OWL-API para prover capacidades semânticas aos agentes.

Os autores apresentam a estrutura básica da ferramenta, bastante similar ao Jason, e explicam a proposta de utilizar "pontes" nos conversores Jason para que o motor de lógica do agente consiga acessar a OWL-API e converter os literais presentes nas bases de dados clássicas Jason (ver figura 2).

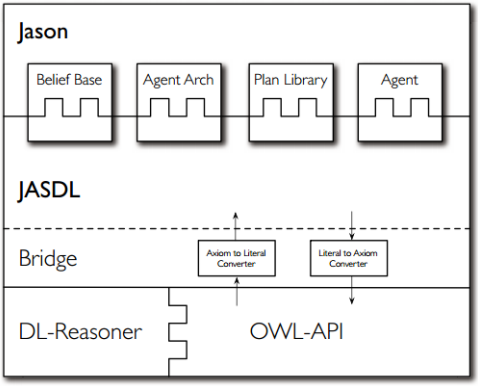

Figura 2 – Arquitetura JASDL
(KLAPISCAK; BORDINI, 2009)

Este trabalho propõe um modelo para inserção de ontologias em agentes Jason, mas não reutiliza dados existentes na Web Semântica, apenas permite a comunicação entre agentes que conheçam uma mesma ontologia.

Em (MASCARDI et al., 2011) os autores descrevem um modelo de agentes *CooL-AgentSpeak* baseado na linguagem *AgentSpeak-DL*, que adiciona capacidades para troca de planos através de serviços ontológicos. A motivação dos autores é que é possível que um agente tenha um objetivo a ser cumprido, mas não conheça nenhum plano que atinja este objetivo. Enquanto isso, outro agente conhece um plano que resolve este problema, mas como os agentes usam ontologias diferentes o primeiro agente não consegue aprender este plano. O modelo descrito pelos autores permite que um agente identifique ontologias similares em outros agentes e reconheça que o outro agente possui um plano que é útil para cumprir seus objetivos atuais.

De maneira similar, (HALAC; EKINCI; DIKENELLI, 2011) também apresenta um modelo baseado em *AgentSpeak-DL*, mas seu modelo é capaz de compartilhar objetivos ao invés de planos. Apesar de serem problemas válidos, os dois trabalhos se restringem ao compartilhamento

de planos ou objetivos, e não tratam da comunicação em geral entre agentes e serviços externos. Além disso, não existem referências às bases comuns de dados, os agentes nos dois casos só são capazes de compartilhar planos ou objetivos (mesmo usando ontologias heterogêneas).

Considerando todos os trabalhos apresentados, é possível ver que existe bastante desenvolvimento na área de comunicação sem ambiguidade através de ontologias, mas nenhum deles trata do problema de duplicação de dados semânticos espalhados pelos agentes e bases de dados na Web. Este trabalho tem o objetivo de cobrir esta deficiência e integrar comunicação sem ambiguidade com reutilização de dados semânticos já existentes, como será visto no próximo capítulo.

3 MODELO PROPOSTO

Para resolver o problema de ambiguidade e duplicação de dados em sistemas na Web, este trabalho propõe uma abordagem que integra a área de desenvolvimento orientado à agentes BDI com a área da Web Semântica. A proposta vem na forma de um novo modelo para desenvolvimento de agentes, que permita aos agentes comunicarem-se claramente com dados na Web, sistemas externos e até outros agentes de software.

No modelo aqui proposto, são seguidos os seguintes critérios cujas motivações serão apresentadas nas seções seguintes:

- Em um agente, dados semânticos são associados somente à base de crenças e não afetam as outras bases de conhecimento.
- A maior parte da informação semântica contida nas crenças do agente é obtida de bases de dados RDF (ao invés de ser duplicada para cada nova aplicação desenvolvida).
- A conexão entre as crenças do agente e triplas em uma bases de dados RDF é feita usando *URIs* que identificam as crenças do agente e as relacionam com recursos na base remota.

Com base nas definições apresentadas no capítulo de Revisão da Literatura, podemos definir este modelo como uma interação entre dois conjuntos $\mathcal{T}s$ e $\mathcal{A}g$:

$$\mathcal{T}s = \{RDF \mid RDF = (Sbj, Prd, Obj)\}$$

Este primeiro conjunto representa uma *triplestore* que contém dados no formato RDF, onde cada dado RDF é um tripla (Sbj, Prd, Obj) no formato Sujeito-Predicato-Objeto (W3C, 2014b).

O outro conjunto representa o agente em si:

$$\mathcal{A}g = \left\{ Sit, Act, Dat, f_{\mathcal{A}g}, {f'}_{\mathcal{A}g} \middle| \begin{array}{l} Dat = (Blf, Dsr, Int) \\ {f'}_{Ag} : Blf \rightarrow \mathcal{T}s \end{array} \right\}$$

Este agente $\mathcal{A}g$ é similar ao modelo base de (AFSHARCHI et al., 2013) onde Dat continua a representar as bases de dados do Agente

($Dat = (Blf, Dsr, Int)$ contém crenças, desejos e intenções), porém adiciona uma função $f'_{\mathcal{A}g}$ que mapeia sua base de crenças à um recurso RDF dentro da *triplestore*.

Com este mapeamento, é possível associar cada crença de um agente a um dado RDF compartilhado via *triplestore* remota. Isto permite que qualquer agente ou serviço que utilize a mesma ontologia desta *triplestore* entenda exatamente o que significa cada crença de um agente seguindo este modelo. Além disso, ao criar uma conexão direta entre as crenças e dados remotos, é fácil expandir a base de crenças do agente lendo mais dados da *triplestore*. Qualquer dado disponível nesta base remota pode ser utilizada pelo agente sem ter que ser inserida manualmente na base de crenças, isto permite que novas aplicações sejam desenvolvidas mais rapidamente, reaproveitando todos os dados já existentes.

A seguir, é apresentada a motivação por trás de cada critério utilizado no desenvolvimento deste modelo.

3.1 MOTIVAÇÕES DAS CARACTERÍSTICAS DO MODELO

3.1.1 Dados semânticos em crenças

A ideia de colocar informação somente nas crenças (ao invés de permear todo o agente) segue o princípio de "*Separation of Concerns*" (DIJKSTRA, 1982).

"*Separation of Concerns*" ou separação de conceitos é uma estratégia que envolve dividir o software de maneira que ele se torne modular, e é bastante comum no desenvolvimento orientado à objetos(LAPLANTE, 2007). No modelo proposto por este trabalho, o conhecimento semântico deve ficar isolado nas crenças do agente, assim o resto do agente funciona como um agente comum (sem conhecimento semântico). Isso permite que o modelo proposto sofra o mínimo de mudanças possíveis sobre a sua base (modelo BDI), sem ter que revolucionar a maneira de se escrever este tipo de agente.

O modelo também propõe que a base de dados de crenças do agente se torne o lugar para manter estes dados, pois é aqui que ficam os literais que representam dados conhecidos pelo agente, e a relação entre eles. Esta estratégia tem um paralelo muito grande com a Web Semântica, onde dados dentro de uma mesma ontologia se relacionam através de uma tripla (similar às crenças do agente). Desta maneira, as crenças do agente podem estar associadas a uma informação semântica,

mas o modelo também deve permitir que elas não tenham associação nenhuma, desta maneira o desenvolvimento se mantém flexível para suportar uma mistura de dados semânticos e não-semânticos.

Como as crenças estão envolvidas em várias partes da execução de um agente, a introdução de novas informações pode afetar alguns processos de decisão como a seleção de objetivos e execução de planos. De maneira geral, o motor de deliberação não deve ser afetado, pois ele lida com a base de objetivos e de planos em um nível mais alto. Os objetivos e planos em si costumam ter contato com as crenças, e estes podem fazer uso dos dados semânticos quando necessário (por exemplo, se existir ambiguidade entre crenças que possam ser diferenciadas através de ontologias).

3.1.2 Aproveitando dados de triplestores

Outro ponto importante desta proposta é reaproveitar a enorme quantidade de dados com anotações semânticas já existentes na Internet.

Para ilustrar a importância deste ponto, é possível ver no seguinte exemplo alguns dados tirados da *DBPedia*, usando uma busca no formato *SPARQL* para se obter uma lista de presidentes nascidos no Brasil (LEHMANN et al., 2014):

```
SELECT DISTINCT ?p WHERE {
?p a dbpedia-owl:President .
?p dbpedia-owl:birthPlace <http://dbpedia.org/resource/Brazil> .
}

...

"http://dbpedia.org/resource/Carlos_Luz"
"http://dbpedia.org/resource/Hermes_da_Fonseca"
"http://dbpedia.org/resource/Carlos_Arthur_Nuzman"
"http://dbpedia.org/resource/Manuel_Ferraz_de_Campos_Sales"
"http://dbpedia.org/resource/Artur_Bernardes"
"http://dbpedia.org/resource/Eurico_Gaspar_Dutra"
"http://dbpedia.org/resource/Fernando_Collor_de_Mello"
"http://dbpedia.org/resource/Floriano_Peixoto"
"http://dbpedia.org/resource/Francisco_de_Paula_Rodrigues_Alves"
"http://dbpedia.org/resource/Prudente_de_Morais"
(... mais resultados)
```

Cada um destes resultados está conectado a muitas outras in-

formações, por exemplo o período de mandato, data de nascimento, partido político, etc.

Veja alguns dados para um dos presidentes:

```
foaf:name
Carlos Coimbra da Luz

dbpedia-owl:party
dbpedia:Brazilian_Social_Democracy_Party

dbpedia-owl:activeYearsEndDate
1955-11-11 (xsd:date)

dbpedia-owl:activeYearsStartDate
1955-11-09 (xsd:date)

dbpprop:birthDate
1894-08-04 (xsd:date)

dbpprop:birthPlace
Três Corações, Minas Gerais, Brazil

dbpprop:deathDate
1961-02-09 (xsd:date)

dbpprop:deathPlace
Rio de Janeiro, Rio de Janeiro, Brazil
```

O modelo visa aproveitar toda esta informação que já está disponível, pois isso deve reduzir bastante a quantidade de dados novos que são inseridos em uma nova aplicação. Como o desenvolvimento de aplicações com agentes BDI implica a descrição das três principais bases de dados do agente (base de crenças, base de objetivos e base de planos), é possível acelerar o desenvolvimento destas aplicações ao permitir que os desenvolvedores obtenham boa parte das suas crenças de uma base pré-existente.

Este ponto diferencia este trabalhos de outros apresentados na seção de trabalhos correlatos, pois nenhum deles apresentava a ideia de utilizar bases de triplas da Web Semântica como fonte direta de dados para os agentes.

3.1.3 Crenças locais e dados semânticos remotos

Finalmente, neste modelo determina-se que as crenças de um agente devem estar conectadas à objetos remotos (instâncias de dados) nestas *triplestores*. Isto permite criar uma associação direta dos nossos dados locais, com uma referência externa. Através desta conexão podemos obter mais informações sobre cada crença do agente e por consequência qualquer conexão de dados feitas nas triplas remotas se estende para a nossa aplicação (outro exemplo de reutilização).

Para criar esta conexão, propõe-se que as crenças literais dos agentes estejam associadas à uma *URI* que representa o mesmo valor do dado em uma *triplestore* (caso ele exista). Tendo esta referência explícita, se torna trivial acessar os dados remotos (já que temos o *link* à mão) e também ganha-se um identificador único e padronizado para as crenças do agente.

Como estas *URIs* são únicas e bem definidas, a comunicação entre agentes de diferentes aplicações também fica mais fácil. É possível ter literais com nomes e utilidades diferentes em cada aplicação, mas se eles estiverem associados à mesma *URI*, dois agentes conseguem se comunicar e entender as crenças disponíveis uns nos outros.

A figura 3 apresenta a conexão de um agente à uma *triplestore* através das *URIs* embutidas nas suas crenças:

3.2 AVALIAÇÃO

O modelo descrito resolve o problema de ambiguidade na comunicação, e também é capaz de reaproveitar dados previamente existentes. Ao contrário de outros trabalhos na área, esta integração consegue de maneira simples resolver estes dois problemas, e é especialmente útil para o desenvolvimento de novas aplicações que contenham dados semânticos em bases remotas.

Ao utilizar uma mesma base de dados e ontologias, o modelo fica livre de problemas comuns em sistemas de ontologias heterogêneas (como sincronização de crenças, planos e objetivos entre agentes), mas por outro lado, dificulta a comunicação com agentes que não utilizem esta mesma base de dados. Esta deficiência é compensada pelo fato que a comunicação se torna mais simples, e todos os agentes podem compartilhar uma quantidade de dados enormes sem muito esforço. Ao compartilhar a fonte de dados, o significado de cada dado é automaticamente compartilhado entre os agentes (seguem a mesma ontologia).

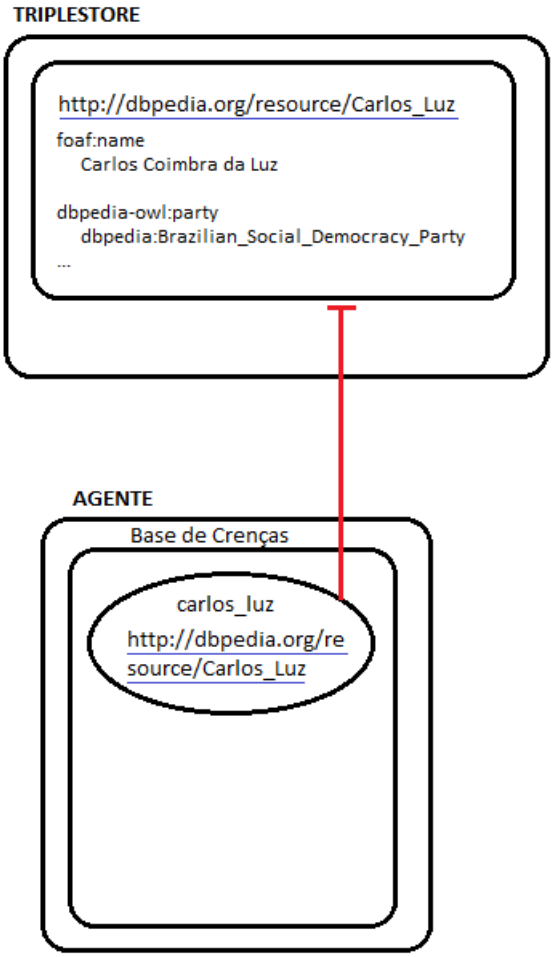

Figura 3 – Conexão entre crenças e *triplestores*

No modelo é possível identificar as seguintes características:

- **Interatividade**: Agentes seguindo este modelo são altamente interativos. Ao compartilhar uma ontologia comum, permite-se que todos os agentes operem livremente sobre as crenças de outros agentes, criando uma rede de conhecimento facilmente compreendida por todos os agentes envolvidos.

- **Interoperabilidade**: Qualquer implementação deste modelo permite comunicação entre outros agentes, desde que todas as implementações estejam conectadas à mesma *triplestore* como fonte de dados e ontologias.

- **Escalabilidade**: Aplicações que seguem este modelo são altamente escaláveis, pois é possível reutilizar uma enorme quantidade de dados, o que acelera muito o desenvolvimento de novos agentes e serviços.

- **Reusabilidade**: O modelo permite alto grau de reusabilidade a nível de dados. Utilizando fontes já existentes através de *triplestores* remotas, é possível reutilizar uma grade quantidade de dados facilmente durante o desenvolvimento de um agente.

- **Confiança**: Ao associar as crenças dos agentes à uma base de dados remotas, criamos um nível de confiança bastante alto nestes dados (tão confiáveis quando a base de dados de onde se originaram). Como todo agente possui acesso a estes dados, é fácil verificar a origem das afirmações feitas por outros agentes.

No próximo capítulo é apresentada uma implementação deste modelo, na forma de um arcabouço para desenvolvimento de agentes.

4 PYTHON SEMANTIC AGENTS

Este capítulo descreve o desenvolvimento de um arcabouço criado para implementar o modelo proposto no capítulo anterior. PySA (*Python Semantic Agents*) é uma implementação *Python* de um arcabouço de agentes BDI, escrito de forma imperativa (ao contrário da forma declarativa mais tradicional). Uma visão imperativa desse tipo de arcabouço permite interagir com os agentes de uma maneira mais familiar, e apesar de ter seus defeitos (arcabouços declarativos requerem menor especificidade) demonstrou-se capaz o suficiente para servir a este propósito demonstrativo.

Decidiu-se criar uma ferramenta nova para permitir maior liberdade durante a implementação das características do modelo proposto, sem restrições devido às capacidades de expansão de ferramentas existentes (um problema encontrado por (KLAPISCAK; BORDINI, 2009)). É importante enfatizar que esta ferramenta não está sendo proposta como uma implementação completa, nem tanto um substituto para outros arcabouços existentes e já consolidados. PySA foi criado para provar que é possível criar uma ferramenta seguindo o modelo proposto neste trabalho, e esta ferramenta ajuda a resolver o problema de ambiguidade na comunicação e duplicação de dados existentes na Web Semântica.

Algumas características de projeto e um caso de uso prático são apresentados nas seções seguintes, para demonstrar PySA e validar o modelo. Uma versão completa da documentação de classes e métodos vem como anexo desta dissertação.

4.0.1 Arquitetura

Como mencionado no modelo proposto, esta solução envolve conectar as crenças dos agentes à *URIs* RDF. A arquitetura de PySA claramente separa este conhecimento das outras bases de dados do agente, mantendo as conexões somente na base de crenças (figura 4).

Na figura 4 podemos ver três componentes básicos do agente: uma base de Crenças, uma base de Objetivos e uma base de Planos.

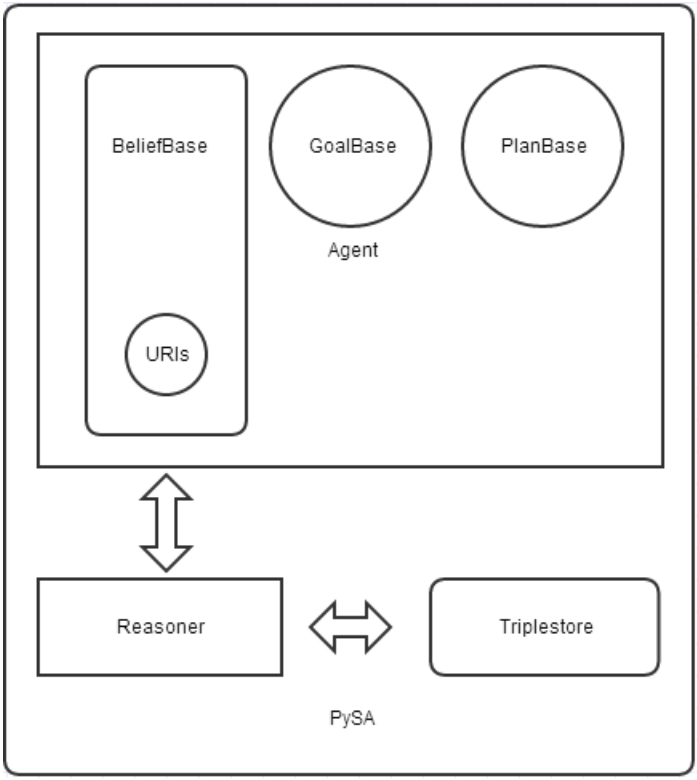

Figura 4 – Interação PySA-Triplestore

4.0.1.1 Base de Crenças

Agentes PySA guardam informação sobre o ambiente onde estão inseridos nesta estrutura de dados. Neste arcabouço estas crenças também estão acompanhadas de uma *URI* que as conecta a uma base de dados comum na Internet. Isto se aplica tanto a dados literais (por exemplo, o agente acredita que um objeto *existe*) quanto a relações entre os dados (por exemplo, o agente acredita que um objeto *possui uma relação* com outro). No caso de dados literais, a *URI* que acompanha um dado aponta para um recurso na DBPedia que representa o mesmo dado, e no caso de relações, temos uma crença que representa a relação entre dois literais (de novo com uma *URI* que aponta para um recurso) feita através de uma relação que também acompanha uma *URI* (que

aponta para uma ontologia na DBPedia).

Esta informação combinada com o nosso motor de deliberação é o que nos permite trabalhar com dados semânticos na nossa aplicação, e nos relacionar com fontes exteriores.

Abaixo temos um exemplo de um literal livre (ainda não foi associado a um agente), mas tudo o que é guardado na base de crenças é obtido a partir disto:

```
james_dean = URI(
'http://live.dbpedia.org/page/
James_Dean'
)
```

Localmente possuímos apenas uma identificação única para este literal, que é a própria *URI*, mas existem muitos mais dados que podem ser obtidos externamente. A maneira que os dados são obtidos a partir de uma *URI* será apresentada ao longo desta e da próxima seção.

4.0.1.2 Base de Objetivos

A base de objetivos pode ser definida como uma coleção de objetivos que nosso agente tenta alcançar, junto com associações destes objetivos à planos que podem ser usados para alcançá-los. Estes objetivos estão relacionados a estados do mundo que o agente gostaria de provocar. Eles determinam qual vai ver a próxima ação a ser tomada por um agente.

Em um agente PySA, esta base é bastante similar a outros arcabouços que seguem o modelo BDI, exceto pelo fato que as crenças utilizadas na definição de um objetivo possuem um referência a dados linkados na Web.

4.0.1.3 Base de Planos

Planos em PySA nada mais são que funções Python que podem ser utilizadas para atingir um objetivo. Estes planos tem acesso livre aos dados da nossa base de crenças, e portanto conseguem acessar dados de fontes exteriores, e relacioná-los com o nosso conhecimento do mundo utilizando as *URIs* que acompanham nossos literais.

4.0.1.4 Motor de Deliberação

O motor de deliberação é responsável por acessar as crenças do agente. Isto pode ser feito manualmente por um usuário, internamente durante a execução de um plano, ou até externamente através de outros agentes. Este motor conhece a relação das nossas crenças com dados semânticos na Internet, e é ele que gerencia a troca de informações entre as crenças e uma *triplestore*.

Esta módulo do agente pode ser usado de duas maneiras, reativamente e proativamente:

- Quando alguma informação é requerida de um agente, isto é feito de maneira reativa e o processamento é imediato.
- Quando um agente termina um plano e está pronto para uma próxima ação, este motor que determina o que vai ser feito (levando em consideração as crenças e objetivos). Neste caso, o processamento é dito proativo.

4.0.1.5 Triplestore

A *triplestore* contém dados semânticos e ontologias que podem ser servidos online através de um formato RDF. Os testes iniciais de PySA foram feitos usando a DBPedia, uma base de dados semânticos que utiliza *triplestores* para armazenamento. Apesar de PySA estar sempre amarrado à DBPedia, é possível criar ferramentas que seguem o mesmo modelo conectadas a qualquer outra *triplestore*, já que elas compartilham o formato RDF para guardar as triplas de dados e possuem *APIs* similares para pesquisa.

Este objeto está representado na arquitetura de PySA pois é muito importante para o arcabouço, mas ele é uma entidade previamente existente na Web e são apenas utilizadas pela ferramenta (nenhum trabalho foi desenvolvido nelas neste projeto).

4.0.2 Guardando dados semânticos

Os dados semânticos mais simples que podemos guardar são representados como literais. Estes são equivalentes a *"things"* na DBPedia, e se obtém conhecimento semântico buscando uma referência da *URI* na base remota, como pode ser visto no exemplo abaixo onde

adicionamos dados semânticos ao agente, criando uma referência à DBPedia:

```
james_dean = URI(
'http://live.dbpedia.org/page/
James_Dean'
)

marylin_monroe = URI(
'http://live.dbpedia.org/page/
Marilyn_Monroe'
)

agent.AddLiterals(
james_dean,
marylin_monroe
)
```

Este agente agora contém dois literais na sua base de crenças. Mesmo com comandos simples como estes, já ganhamos uma enorme quantidade de dados semânticos, pois agora nossa base de crenças tem acesso direto à toda informação associada com aquelas *URIs*. Na próxima seção veremos como isto pode ser acessado.

4.0.3 Pesquisando dados semânticos

Em PySA, a busca por dados semânticos foi projetada para funcionar da maneira mais simples possível, utilizando a RDFlib que faz buscas utilizando a linguagem SPARQL. Uma alternativa para este modelo poderia ser algo baseado na linguagem KQML (FININ et al., 1994) onde a linguagem descrita pelos autores informa explicitamente qual a ontologia que está sendo usada em cada busca, como visto em alguns exemplos do artigo:

Exemplo de uma busca por valores de ações da IBM:

```
(ask-one
:content (PRICE IBM ?price)
:receiver stock-server
:language LPROLOG
:ontology NYSE-TICKS)
```

Nesta mensagem, a comunicação KQML indica que o conteúdo é “price ibm ?price”, que vai resultar no preço das ações; a ontologia

sendo usada é “nyse-ticks”; o alvo da mensagem é um servidor identificado por “stock-server” e a busca está escrita em uma linguagem LPROLOG.

Várias ideias do KQML foram transportadas para a nossa ferramenta, mas com algumas simplificações. Por exemplo, a linguagem sempre é Python, o alvo da mensagem sempre é o próprio agente que está sendo questionado (apesar de que este pode acessar uma *triplestore* remota caso necessário) e a ontologia está embutida nas *URIs* contidas nos objetos literais e relações.

Como prova de conceito, no exemplo seguinte vamos criar uma relação personalizadas entre literais na nossa base de crenças usando PySA. A maior parte do conhecimento de aplicação feito pelo desenvolvedor será neste formato.

```
dated = CustomRelation('Dated')

agent.AddRelation(dated)

# These two never actually dated, but it
# makes for a better story
agent.Relate(
dated,
james_dean,
marylin_monroe
)

agent.QueryRelation(relation=dated)
>>> ['Dated(James Dean, Marylin Monroe)']
```

Para comparar, em uma linguagem declarativa típica, esta associação se pareceria com isso:

```
dated(jamesDean, marilynMonroe)
```

Este exemplo é bastante superficial, mas é possível ver como pode ser expandido para aplicações reais. Em um clássico exemplo de sistema de recomendação para uma livraria, esta relação poderia representar a presença de um livro em uma loja em particular:

```
some_book = URI('http://live.dbpedia.org/page/Iliad')
some_store = URI('http://live.dbpedia.org/page/BookStore')
```

```
available_at = CustomRelation('Available at')

book_agent.AddRelation(available_at)

book_agent.Relate(
available_at,
some_book,
some_store
)
```

Diferente de KQML (FININ et al., 1994), esta comunicação não foi feita com o objetivo de ser compatível com o padrão FIPA. Este certamente é um objetivo para ser alcançado no futuro.

4.0.4 Inferindo conhecimento

Usando relações presentes nas databases online, é possível inferir conhecimento que nunca foi inserido diretamente na nossa aplicação, e até obter literais (crenças) que não pertencem à base de conhecimento do nosso agente local, basta conhecer as URIs que apontam para a ontologia desejada na DBPedia:

```
birth_date = URI(
'http://dbpedia.org/ontology/birthDate'
)

agent.QueryRelation(
literals=[james_dean],
relation=birth_date
)
>>> ['BirthDate(James Dean, 1931-02-08')]
```

Esta é de longe a aplicação mais interessante deste arcabouço, onde é possível relacionar as crenças dos agentes diretamente a uma base de dados online (neste caso, a data de nascimento de James Dean foi obtida sem este dado ter sido inserido diretamente na nossa aplicação durante o desenvolvimento). Isto significa que o projetista automaticamente ganha uma grande quantidade de trabalho que seria duplicada, mas agora está centralizada em uma *triplestore* pública.

Internamente, PySA se conecta com a DBPedia utilizando a biblioteca *rdflib* (GRIMNES, 2014), e busca todos as triplas utilizando uma

ontologia que lhe é dada. Como as crenças do agente estão associadas a uma *URI*, podemos acessar todas as relações e objetos criados por usuários da DBPedia, e como visto no exemplo anterior, retornar um literal e *URI* na forma de uma data de nascimento (conhecimento que não estava contido no nosso agente).

4.1 CASO DE USO

Esta seção apresenta um caso de uso real que implementa uma aplicação usando o arcabouço PySA.

Nesta aplicação, foi criado um agente historiador, que tem como único objetivo listar todos os predecessores de reis que ele conhece. Aqui é apresentada a definição do agente, seus planos e objetivos e é feita uma demonstração da execução do agente.

4.1.1 O Agente

```
# Create agent
historian = HistorianAgent()

# Give him the name of a single king
historian.belief_base.add(
    resource('Henry_VIII_of_England')
)

# Run the agent, he should track all predecessors of
# known kings (just Henry VIII in this case)
historian.run()
```

Definimos o agente com o mínimo de informação necessária, apenas seu nome, um objetivo (listar todos os predecessores de reis) e um plano para de resolver este objetivo:

```
class HistorianAgent(Agent):
    def __init__(self):
        Agent.__init__(self, "Historian")

        self.goal_base.add(ListPredecessorsGoal)
        self.plan_base.add(ListPredecessorsPlan)
```

É possível observar que na durante a criação do agente (método

__init__) ele adiciona um objetivo e um plano às suas bases de objetivos e planos respectivamente.

Tanto os objetivos quanto os planos são definidos através de classes, que herdam de objetos PySA *"Goal"* e *"Plan"*.

4.1.2 Objetivos

```
class ListPredecessorsGoal(Goal):
    def completed(self):
        known_royalty = list_known_royalty(
            self.belief_base
        )

        for king in known_royalty:

            predecessor = get_predecessor(
                self.belief_base,
                king
            )

            if predecessor is not None and
               predecessor not in known_royalty:
                return False

        return True
```

Basicamente, este objetivo implementa uma função que define se ele está completo ou não. Nesta função obtemos uma lista de todos os reis que estão contidos na nossa base de crenças, e verificamos se o predecessor de cada rei também está presente na base de crenças. Se todos estiverem presentes, o objetivo foi cumprido.

4.1.3 Planos

```
class ListPredecessorsPlan(Plan):
    FULFILLS_GOALS = [
        ListPredecessorsGoal
    ]

    def preconditions(self):
```

```
        return True  # This plan is always usable


    def execute(self):
        for king in list_known_royalty(self.belief_base):
            self.list_predecessors(king)


    def list_predecessors(self, king):
        current_king = king
        while True:
            print current_king

            predecessor = get_predecessor(
               self.belief_base,
               current_king
            )

            if predecessor is None:
                break

            self.belief_base.add(predecessor)
            current_king = predecessor
```

Este plano define alguns campos e funções. O primeiro campo *FULFILLS_GOALS* é uma lista de todos os objetivos que podem ser alcançados por este plano. No nosso caso, apenas um plano. Nesta implementação, este plano está associado ao objetivo na hora que é criado, mas esta lista pode ser manipulada em tempo de execução caso novos planos e objetivos surjam.

Depois, temos uma função *preconditions* que define se o plano está pronto para ser executado, e neste caso, o plano sempre está pronto e portanto retorna um valor verdadeiro sem precisar checar outras condições.

A função *execute* é a principal função dos planos, e é chamada na hora de executá-lo. Aqui, esta função primeiro acha todos os reis que estão contidos na base de crenças, e depois lista todos os seus predecessores.

4.1.4 Funções auxiliares

Internamente, algumas funções auxiliares são usadas tanto no objetivo, quanto no plano. Estas funções servem para listar todos os reis conhecidos, e obter o predecessor de um rei, e estão implementadas da seguinte maneira:

```
def list_known_royalty(belief_base):
    return list(
       belief_base.subjects(
          RDF.type,
          ontology("Royalty")
       )
    )

def get_predecessor(belief_base, king):
    predecessor_list = list(
        belief_base.objects(
            subject=king,
            predicate=ontology('predecessor')
        )
    )

    if not predecessor_list:
        return None

    return predecessor_list[0]
```

list_known_royalty lista todos os objetos da base de crenças que são do tipo *Royalty*. Aqui, tanto o "tipo" quanto o valor *Royalty* são *URIs* da Web Semântica, a função *ontology* usada vai converter o nome simplificado *Royalty* para *http://dbpedia.org/ontology/Royalty*, ou seja, está sendo usada uma ontologia real presente na DBPedia.

get_predecessor lista todos os objetos da base de crenças que atendem ao predicado *predecessor* para um dado rei. Esta função retorna a *URI* associada ao predecessor de um rei, mas não o adiciona à base de crenças do agente (isto é feito externamente no plano de listar predecessores). Assim como na função anterior, vemos a utilização de uma função *ontology* que neste caso converte o nome simplificado *predecessor* para *http://dbpedia.org/ontology/predecessor*.

4.1.5 Resultado da execução

```
http://dbpedia.org/resource/Henry_VIII_of_England
http://dbpedia.org/resource/Henry_VII_of_England
http://dbpedia.org/resource/Richard_III_of_England
http://dbpedia.org/resource/Edward_V_of_England
http://dbpedia.org/resource/Edward_IV_of_England
http://dbpedia.org/resource/Henry_VI_of_England
http://dbpedia.org/resource/Henry_V_of_England
http://dbpedia.org/resource/Henry_IV_of_England
http://dbpedia.org/resource/Richard_II_of_England
http://dbpedia.org/resource/Edward_III_of_England
http://dbpedia.org/resource/Edward_II_of_England
http://dbpedia.org/resource/Edward_I_of_England
http://dbpedia.org/resource/Henry_III_of_England
http://dbpedia.org/resource/John,_King_of_England
http://dbpedia.org/resource/Richard_I_of_England
http://dbpedia.org/resource/Henry_II_of_England
http://dbpedia.org/resource/Stephen,_King_of_England
http://dbpedia.org/resource/Henry_I_of_England
http://dbpedia.org/resource/William_II_of_England
http://dbpedia.org/resource/William_the_Conqueror
http://dbpedia.org/resource/Edgar_the_%C3%86theling
http://dbpedia.org/resource/Harold_Godwinson
http://dbpedia.org/resource/Edward_the_Confessor
http://dbpedia.org/resource/Harthacnut
http://dbpedia.org/resource/Cnut_the_Great
http://dbpedia.org/resource/Edmund_Ironside
http://dbpedia.org/resource/%C3%86thelred_the_Unready
http://dbpedia.org/resource/Sweyn_Forkbeard
http://dbpedia.org/resource/Harald_Bluetooth
http://dbpedia.org/resource/Gorm_the_Old

====================================
Goal completed: ListPredecessorsGoal
====================================
```

Podemos ver nesta lista todos os predecessores do rei Henrique VIII da Inglaterra. Vale a pena notar que todos estes objetos agora fazem parte da base de crenças do agente, e junto com eles, várias informações associadas, por exemplo a data de nascimento e período

de reinado de cada um dos reis.

Através deste caso de uso, é possível observar os pontos defendidos na proposta. As crenças que compõem a base de crenças do agente estão associadas a *URIs* da DBPedia, o que permite que o agente procure mais informação sobre esta crença diretamente na Web. Note que todos os predecessores que foram encontrados nunca foram declarados na nossa aplicação, mas sim obtidos de uma *triplestore* remota. As crenças que representam as associações entre literais também nunca foram declaradas na aplicação, mas sim obtidas a partir de uma ontologia "predecessor" já existente (e também associada a uma *URI*), é possível usá-la pois sempre que uma crença é adicionada a este agente, esta informação é buscada na DBPedia e inserida na base de crenças.

De maneira geral, foi atingido o objetivo de associar as crenças à dados semânticos (o agente sabe que Henry VIII é um rei), usar uma associação padronizada para referenciar as crenças (na forma de *URI*) e obter uma grande quantidade de informações sobre as crenças do agente sem nunca inserí-las na aplicação (neste caso, a lista de todos os predecessores e informações associadas a eles).

5 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho foi iniciado com as seguintes perguntas:

- Como podemos criar sistemas que operam sobre dados na Web e se comunicam sem ambiguidade?
- Como compartilhar e reutilizar informação com conteúdo semântico entre sistemas deste tipo?

Para respondê-las, este trabalho contribui com um modelo de desenvolvimento de agentes e uma implementação (*PySA*) como prova de conceito para este modelo. Foi possível verificar que o modelo é capaz de estender as capacidades de um sistema *BDI* tradicional adicionando dados semânticos às crenças do agente, o que permite que toda comunicação utilizando estes dados seja feita de forma desambígua. A utilização de *URIs* padronizadas permite a comunicação com outros agentes que compartilham uma mesma *database* online de dados, além de tornar possível inferir informação semântica dos agentes, mesmo que esta nunca tenha sido descrita manualmente pelo criador do agente, e sim herdada de trabalhos anteriores.

Em comparação com trabalhos similares apresentados no capítulo de Revisão da Litetura, foram obtidos alguns resultados que diferenciam este modelo dos anteriores. A principal diferença está na conexão das crenças dos agentes (e suas ontologias) à uma base de dados remota, o que faz com que aplicações seguindo este modelo não precisem declarar novamente as ontologias a serem usadas, e permite que uma grande quantidade de relações e dados que já estão disponíveis na Web Semântica sejam utilizados pelos agentes.

O modelo se mostrou completo em relação às questões levantadas na problemática deste trabalho, mas PySA ainda está em um estágio inicial de desenvolvimento, e é necessário fazer várias melhorias para que uma versão útil e estável seja publicada. Apesar disso, pode-se dizer que com a versão atual já é possível verificar através de exemplos que com uma sintaxe simples somos capazes de estender as habilidades dos agentes sem perder capacidades já existentes hoje, e isto poderia se tornar uma ferramenta bastante útil para aplicações de Web Semântica.

Também foi identificado que trabalho adicional pode ser feito na direção de compartilhar literais e relações personalizadas (aquelas criadas pelo autor do agente, e não obtidas de *triplestores* públicas), de maneira que outras aplicações e agentes possam utilizá-los também.

Este problema é o foco de alguns trabalhos correlatos apresentados anteriormente, e várias técnicas descritas neles poderiam ser utilizadas como ponto de partida para este projeto.

Outro ponto a ser desenvolvido está associado à performance da implementação atual, onde atualmente os dados são constantemente acessados de fontes externas através da Internet. Este processo poderia ser otimizado usando mecanismo de *cache* para estes dados, de maneira que só fossem acessados uma vez. Esta alternativa parece razoavelmente segura, pois a frequência com que os dados na fonte original mudam é relativamente baixa, e poucas atualizações seriam perdidas ao se reutilizar dados do passado.

Com o desenvolvimento deste trabalhos adicionais, espera-se que o modelo (e implementações dele) se torne ainda mais reusável, e que isto permita que novas aplicações possam ser desenvolvidas de maneira mais rápida e confiável.

REFERÊNCIAS

AFSHARCHI, M. et al. Common understanding in a multi-agent system using ontology-guided learning. *Knowledge and Information Systems*, Springer-Verlag, v. 36, n. 1, p. 83–120, 2013. ISSN 0219-1377.

AGGARWAL, D.; SINGH, A. Software agent reusability mechanism at application level. *Global Journal of Computer Science and Technology*, Global Journals Inc. (USA), v. 13, n. 3, 2013. ISSN 0975-4350.

ALVARES, L. O.; SICHMAN, J. Introdução aos sistemas multiagentes. In: *Anais da SBC*. [S.l.]: SBC, 1997. p. 1–38.

ANTONIOU, G.; HARMELEN, F. v. *A Semantic Web Primer, 2nd Edition (Cooperative Information Systems)*. 2. ed. [S.l.]: The MIT Press, 2008. ISBN 0262012421, 9780262012423.

ANTONIOU, G.; VANHARMELEN, F. *A Semantic Web Primer*. Cambridge, MA, USA: MIT Press, 2004. ISBN 0262012103.

BERNERS-LEE, T. *Sir Tim Berners-Lee Talks with Talis about the Semantic Web*. 2008. Disponível em: <talis-podcasts.s3.amazonaws.com/twt20080207_TimBL.html>.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web: Scientific American. *Scientific American*, maio 2001.

BITTENCOURT, G. *Inteligência Artificial: Ferramentas e teorias*. [S.l.]: Editora UFSC, 2001.

BORDINI, R. H.; WOOLDRIDGE, M.; HUBNER, J. F. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. [S.l.]: John Wiley & Sons, 2007. ISBN 0470029005.

BRATMAN, M. *Intention, plans, and practical reason*. Cambridge, MA: Harvard University Press, 1987. ISBN 9780674458185. Disponível em: <http://books.google.de/books?id=I0nuAAAAMAAJ>.

BRATMAN, M. E. *Intention, Plans, and Practical Reason*. Cambridge University Press, 1999. Paperback. ISBN 1575861925. Disponível em: <http://www.amazon.co.uk/exec/obidos/ASIN/1575861925/citeulike00-21>.

BRATMAN, M. E.; ISRAEL, D. J.; POLLACK, M. E. Plans and resource-bounded practical reasoning. *Computational Intelligence*, Blackwell Publishing Ltd, v. 4, n. 3, p. 349–355, 1988.

BRENNER, W.; ZARNEKOW, R.; WITTIG, H. *Intelligent Software Agents: Foundations and Applications*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2012. ISBN 3642804861, 9783642804861.

DICKINSON, I. *BDI Agents and the Semantic Web: Developing User-Facing Autonomous Applications*. Tese (Doutorado) — University of Liverpool, September 2006. Disponível em: <http://www.iandickinson.me.uk/thesis.pdf>.

DICKINSON, I.; WOOLDRIDGE, M. Towards practical reasoning agents for the semantic web. In: *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2003. (AAMAS '03), p. 827–834. ISBN 1-58113-683-8.

DIJKSTRA, E. W. On the role of scientific thought (EWD447). In: *Selected Writings on Computing: A Personal Perspective*. [S.l.: s.n.], 1982. p. 60–66.

FININ, T. et al. Kqml as an agent communication language. In: *Proceedings of the Third International Conference on Information and Knowledge Management*. New York, NY, USA: ACM, 1994. (CIKM '94), p. 456–463. ISBN 0-89791-674-3. Disponível em: <http://doi.acm.org/10.1145/191246.191322>.

GESSLER, D. et al. Sswap: A simple semantic web architecture and protocol for semantic web services. *BMC Bioinformatics*, v. 10, n. 1, p. 309, 2009. ISSN 1471-2105. Disponível em: <http://www.biomedcentral.com/1471-2105/10/309>.

GRIMNES, G. A. *RDFLib*. 2014. Disponível em: <https://github.com/RDFLib/rdflib>.

HALAC, T. G.; EKINCI, E. E.; DIKENELLI, O. Description logic based bdi implementation for goal-directed semantic agents. In: *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 02*. Washington, DC, USA: IEEE Computer Society, 2011. (WI-IAT '11), p. 62–65. ISBN 978-0-7695-4513-4. Disponível em: <http://dx.doi.org/10.1109/WI-IAT.2011.192>.

HEBELER, J. et al. *Semantic Web Programming*. Wiley, 2011. ISBN 9781118080603. Disponível em: <http://books.google.com.br/books?id=rtcIFmXkysEC>.

KARDAS, G.; DEMIREZEN, Z.; CHALLENGER, M. Towards a dsml for semantic web enabled multi-agent systems. In: *Proceedings of the International Workshop on Formalization of Modeling Languages*. New York, NY, USA: ACM, 2010. (FML '10), p. 5:1–5:5. ISBN 978-1-4503-0532-7. Disponível em: <http://doi.acm.org/10.1145/1943397.1943402>.

KLAPISCAK, T.; BORDINI, R. H. Declarative agent languages and technologies vi. In: BALDONI, M. et al. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. JASDL: A Practical Programming Approach Combining Agent and Semantic Web Technologies, p. 91–110. ISBN 978-3-540-93919-1.

KOBAYASHI, M.; TAKEDA, K. Information retrieval on the web. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 32, n. 2, p. 144–173, jun. 2000. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/358923.358934>.

LAPLANTE, P. A. *What Every Engineer Should Know About Software Engineering (What Every Engineer Should Know)*. Boca Raton, FL, USA: CRC Press, Inc., 2007. ISBN 0849372283.

LEHMANN, J. et al. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2014.

LI, L.; WU, B.; YANG, Y. Agent-based ontology integration for ontology-based applications. In: *PROC. OF AUSTRALASIAN ONTOLOGY WORKSHOP (AOW 2005), THE 18TH AUSTRALIAN JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, CONFERENCES IN RESEARCH AND PRACTICE IN INFORMATION TECHNOLOGY (CRPIT) SERIES BY AUSTRALIAN COMPUTER SOCIETY, VOL 58*. [S.l.: s.n.], 2005. p. 53–59.

LIU, C.-H.; CHEN, J. J.-Y. Using ontology-based bdi agent to dynamically customize workflow and bind semantic web service. *JSW*, v. 7, n. 4, p. 884–894, 2012.

MASCARDI, V. et al. CooL-AgentSpeak: Enhancing AgentSpeak-DL Agents with Plan Exchange and Ontology Services. In: *Proceedings of*

the 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2011. [S.l.: s.n.], 2011. p. 109–116.

NAKAYAMA, K.; HARA, T.; NISHIO, S. An agent system for ontology sharing on www. In: *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*. New York, NY, USA: ACM, 2005. (WWW '05), p. 964–965. ISBN 1-59593-051-5. Disponível em: <http://doi.acm.org/10.1145/1062745.1062820>.

RAO, A. S. Agentspeak(l): Bdi agents speak out in a logical computable language. In: *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1996. (MAAMAW '96), p. 42–55. ISBN 3-540-60852-4. Disponível em: <http://dl.acm.org/citation.cfm?id=237945.237953>.

RAO, A. S.; GEORGEFF, M. P. Bdi agents: From theory to practice. In: *IN PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS (ICMAS-95*. [S.l.: s.n.], 1995. p. 312–319.

RUSHER, J. *Triple Store*. 2003. Disponível em: <http://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html>.

RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 2. ed. [S.l.]: Pearson Education, 2003. ISBN 0137903952.

W3C. *Linked Data*. 2014. Disponível em: <http://www.w3.org/DesignIssues/LinkedData.html>.

W3C. *The Resource Description Framework (RDF)*. 2014. Disponível em: <http://www.w3.org/TR/REC-rdf-syntax/>.

W3C. *W3C*. 2014. Disponível em: <http://www.w3.org/>.

WOOLDRIDGE, M. *An Introduction to MultiAgent Systems*. 2nd. ed. [S.l.]: Wiley Publishing, 2009. ISBN 0470519460, 9780470519462.

ANEXO A – Documentação de classes e métodos do arcabouço PySA

Neste capítulo são descritas as classes e métodos disponíveis no arcabouço PySA, desenvolvido durante este projeto. A figura 5 apresenta uma visão geral da arquitetura do arcabouço, e detalhes são apresentados nas seções seguintes.

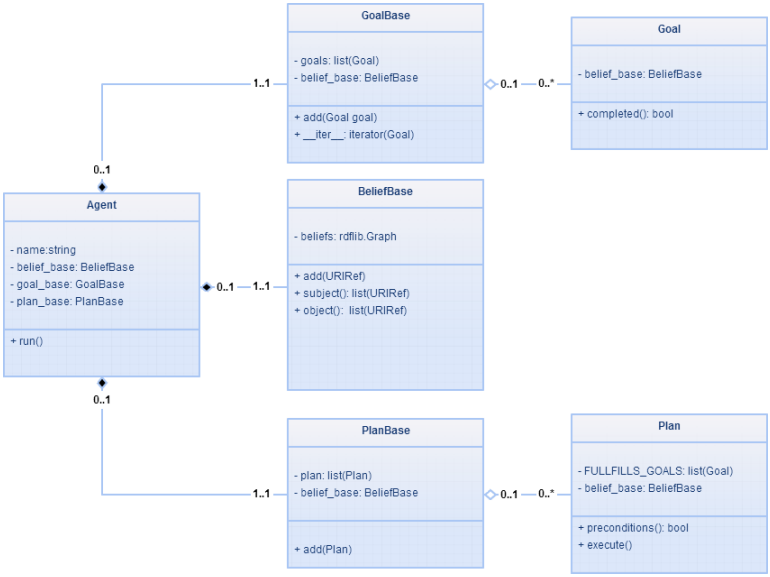

Figura 5 – Visão geral da arquitetura PySA

A.1 CLASSES

A.1.1 Agent

A classe *Agent* (figura 6) representa um agente PySA. Este agente possui um nome (*name*) pelo qual é identificado e 3 bases de dados que serão descritas nas seções seguintes:

- belief_base(BeliefBase): A base de crenças do agente.
- goal_base(GoalBase): A base de objetivos do agente.
- plan_base(PlanBase): A base de planos do agente.

Figura 6 – PySA Agent

O método *run* inicia o ciclo de vida do agente. Durante este ciclo, o agente procura objetivos dentro de *goal_base* que podem ser alcançados, e executa planos de sua *plan_base* para completá-los. Este ciclo continua até que o agente não tenha mais nenhum objetivo para cumprir.

A.1.2 BeliefBase

A classe *BeliefBase* (figura 7) representa a base de crenças do agente. A base é composta primariamente por um campo *beliefs*, que é um grafo de crenças obtido da RDFLib. Neste grafo é armazenado todo o conhecimento do agente, e é possível interagir com esta base através de 3 métodos:

- add(URIRef): Adiciona uma nova URI à base de crenças do agente.

- subject(predicate, object): Lista todos os sujeitos dentro de triplas RDF que atendem à mascara (_, predicate, object).

- object(subject, predicate): Lista todos os objetos dentro de triplas RDF que atendem à mascara (subject, predicate, _).

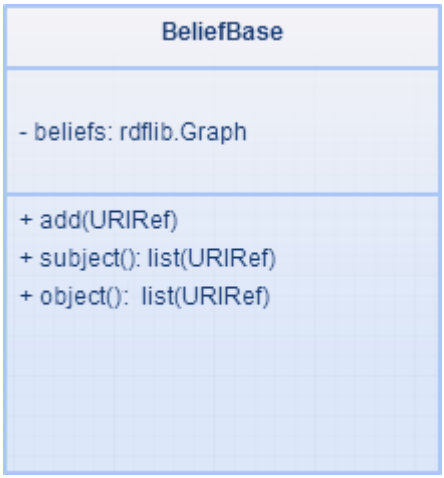

Figura 7 – PySA BeliefBase

A.1.3 GoalBase

A classe *GoalBase* (figura 8) representa a base de objetivos do agente. Esta base é composta por vários *Goals* que serão descritos na próxima seção. Além disso, a base de objetivos disponibiliza dois métodos para se interargir com a base: *add(Goal)* para adicionar novos objetivos e *__iter__* para se iterar sobre os objetivos existentes.

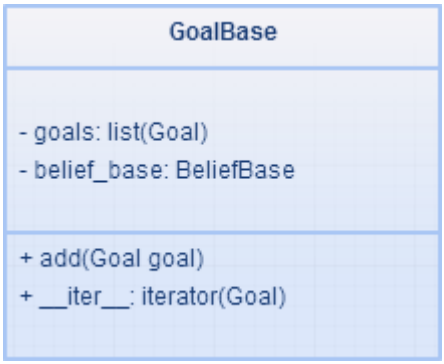

Figura 8 – PySA GoalBase

A.1.4 Goal

A classe *Goal* (figura 9) representa cada objetivo individual do agente. Os objetivos ficam contidos na *goal_base* descrita acima e possuem apenas um método *completed*, que indica se o objetivo foi cumprido ou não. Para determinar isto, um objetivo mantém uma referência direta à *belief_base*, e pode operar sobre as crenças do agente para determinar a situação atual (e se falta algo para se completar este objetivo).

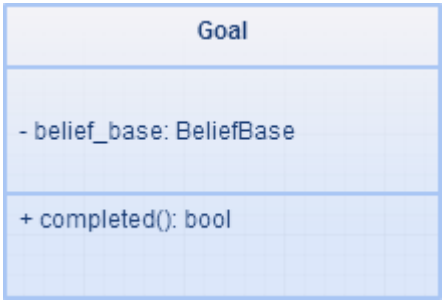

Figura 9 – PySA Goal

A.1.5 PlanBase

A classe *PlanBase* (figura 10) representa a base de planos do agente. Esta base é composta por vários *Plans* que serão descritos na próxima seção. A base de planos disponibiliza apenas um método *add* para adicionar novos planos à base. *plan_base* também mantém uma conexão com a base de crenças, que é passada para os planos que são inseridos nesta base.

Figura 10 – PySA PlanBase

A.1.6 Plan

A classe *Plan* (figura 11) representa cada plano que pode ser executado pelo agente. Os planos ficam contidos na *plan_base* e mantém uma referência direta à *belief_base*, para que possa operar sobre as crenças do agente para determinar e alterar a sua percepção da situação atual. Além disso, cada plano possui uma lista de objetivos que podem ser atingidos ao se executar este plano (*FULLFILLS_GOALS*).

Os planos possuem dois métodos: *preconditions* retorna um valor verdadeiro ou falso indicando se este plano pode ser executado no momento (ou seja, atende à um conjunto de pré-condições); *execute* executa o plano em si. A execução geralmente acarreta na modificação das crenças do agente, ou na atuação sobre outros agentes e o ambiente.

Figura 11 – PySA Plan

A.2 MÉTODOS AUXILIARES

A.2.1 ontology

O método *ontology* pode ser usado em qualquer ponto de uma aplicação PySA e serve para facilitar o acesso à uma relação da ontologia da DBPedia. Este método recebe como parâmetro uma variável de texto, e retorna um objeto do tipo *URIRef* que referencia uma URI da DBPedia.

Para exemplificar, note que estas duas chamadas são equivalentes:

```
uriref = ontology('Royalty')

uriref = rdflib.term.URIRef('http://dbpedia.org/ontology/Royalty')
```

A.2.2 resource

O método *resource* pode ser usado em qualquer ponto de uma aplicação PySA e serve para facilitar o acesso à um recurso (ou *thing*) da DBPedia. Este método recebe como parâmetro uma variável de texto, e retorna um objeto do tipo *URIRef* que referencia uma URI da DBPedia.

Para exemplificar, note que estas duas chamadas são equivalentes:

```
uriref = resource('Henry_VIII_of_England')

uriref = rdflib.term.URIRef(
'http://dbpedia.org/resource/Henry_VIII_of_England'
)
```