UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Leon Luiz Vargas Daros

MODELO DE INTEGRAÇÃO ENTRE AGENTES BDI BASEADOS EM SISTEMAS MULTI-CONTEXTO E ONTOLOGIAS PÚBLICAS PARA REVISÃO DE CRENÇAS

Florianópolis

2019

Leon Luiz Vargas Daros

MODELO DE INTEGRAÇÃO ENTRE AGENTES BDI BASEADOS EM SISTEMAS MULTI-CONTEXTO E ONTOLOGIAS PÚBLICAS PARA REVISÃO DE CRENÇAS

Proposta de Trabalho de Conclusão de Curso submetido ao curso de Sistemas de Informação para a obtenção do Grau de Bacharel em Sistemas de Informação.
Orientador: Prof. Ricardo Azambuja Silveira
Coorientador: Felipe Demarchi

Florianópolis

2019

Leon Luiz Vargas Daros

MODELO DE INTEGRAÇÃO ENTRE AGENTES BDI BASEADOS EM SISTEMAS MULTI-CONTEXTO E ONTOLOGIAS PÚBLICAS PARA REVISÃO DE CRENÇAS

Este Proposta de Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Sistemas de Informação", e aprovado em sua forma final pelo curso de Sistemas de Informação.

Florianópolis, 01 de Novembro 2019.

Prof. Ricardo Azambuja Silveira
Orientador

Felipe Demarchi
Coorientador

Banca Examinadora:

Prof. Elder Rizzon Santos
Presidente

Thiago Angelo Gelaim

AGRADECIMENTOS

Uma vez tendo passado por tudo que passei aqui na UFSC chegou o momento de relembrar e agradecer. Dentre todas as aventuras que tive aqui, a conclusão deste trabalho foi a maior delas. Uma jornada que fez com que eu tivesse que mudar, evoluir e superar.

Por terem me dado o apoio para que eu estivesse aqui, e por continuarem do meu lado aonde quer que eu vá. Gostaria de agradecer aos meus pais, Rosaria Vargas e Gerson Daros. Gostaria de agradecer também à minha vó, a Dona Leonor, que durante a maior parte da faculdade me ajudou a ter o pão de cada dia.

Por sempre estarem comigo neste percurso, gostaria de agradecer aos meus amigos pelos conselhos, pelas aventuras, pelos estresses, pelas assembléias e pelas queimaduras de 2º grau. Em especial a Júlia Nakayama, a Luiza Cardoso, a Cinthya Weigel, a Sabrina Schütz, o Richard Bertozzo, o Bruno Ferreira.

Gostaria de agradecer também ao meu orientador, Ricardo Azambuja Silveira. Pelos momentos de conversa que me traziam mais tranquilidade e me trilhavam um caminho para fora do labirinto. Também agradeço a Thiago Gelaim que teve toda a calma e paciência do mundo em me auxiliar durante o periodo de desenvolvimento deste trabalho.

Por sempre estar do meu lado, por me auxiliar em todos os momentos, por ser a mais paciente e por me aguentar, gostaria de agradecer a Nina Rebolini Ferretti. Obrigado por ser minha parceira no co-op da vida e por me ajudar a chegar até aqui.

E por último, mas não menos importante, gostaria de agradecer a mim mesmo. Gostaria de agradecer ao eu do passado por ter se rodeado das melhores pessoas e por sempre pensar no Leon do futuro. Agradeço a mim por todo esforço e por dar ouvidos as pessoas certas. Agradeço a mim do passado por perceber o momento de parar e cuidar das feridas.

Acima de tudo agradeço a todos que vieram antes de mim, pois neste momento passo a ser a primeira pessoa de toda a linhagem da minha familia a obter um diploma.

Não importa o quanto voce bate, mas sim o quanto aguenta apanhar e continuar. É assim que se ganha

Rocky Balboa

RESUMO

Com a grande quantidade de informação disponível na web, e com o crescente interesse na utilização destes dados, surge a web semântica, que propõe uma estrutura para as informações, de maneira que estes conteúdos passem a existir como bases de conhecimento disponibilizadas como ontologias. Com a utilização de ontologias, a informação que antes se encontrava desordenada, agora possui um padrão, fazendo com que se torne mais fácil as consultas a estes dados, tornando esta área interessante para o estudo de inteligência computacional. Desde então, agentes inteligentes vêm sendo utilizados como forma de percorrer, consultar e analisar estas ontologias e, assim, esta área tem gerado uma série de estudos e modelos. No entanto, para que haja progresso nesta área de pesquisa, sempre é necessário que exista uma comparação entre estes modelos, fazendo com que, tanto os novos modelos de agente quanto os já existentes, evoluam. Com base nesta teoria de prova de conceito, neste trabalho foi desenvolvido um modelo para que agentes com arquitetura baseada em crenças, desejos e intenções (BDI) e que utilizam uma abordagem baseada em sistemas multi-contexto, desenvolvidos na linguagem Sigon, consigam através de um contexto específico, se comunicar com ontologias públicas remotas. Possibilitando assim procedimentos de revisão de crenças que possam influenciar o ciclo de raciocínio e deliberação do agente. O modelo proposto é baseado em modelo já existente desenvolvido na linguagem JASON.

Palavras-chave: Agentes. Ontologias. Web Semântica. Sistema Multi-Contexto.

ABSTRACT

With the large amount of information available on the web, and with the growing interest in the use of this data, the semantic web emerges. Which proposes a structure for information, so that these contents come into existence as knowledge bases made available as ontologies. With the use of ontologies, information that was previously disordered now has a pattern, making it easier to query this data, making this area very interesting for the study of computational intelligence. Since then, intelligent agents have been used as a way to browse, consult and analyze these ontologies, and thus this area has generated a series of studies and models. However, in order to make progress in this area of research, it is always necessary to compare these models, making both new and existing agent models evolve. Based on this proof-of-concept theory in this paper, a model will be developed so that agents with a BDI model that have a multi-context system based approach, developed in the Sigon language can achieve through a specific context, communicate with remote public ontologies. Thus enabling belief review procedures that may influence the agent's reasoning and deliberation cycle. The proposed model is based on an existing model developed in the JASON language.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Hoje em dia vivemos constantemente recebendo e enviando informação através da rede mundial de computadores (WEB). Com o passar do tempo, todo este conteúdo se acumula na internet de maneira desestruturada, dificultando a coleta destas informações. Neste âmbito surge, como proposta para equacionar o problema, a chamada Web Semântica, idealizado por Berners-Lee, Hendler e Lassila (2001). A Web semântica propõe uma maneira de trazer ordem e uma estrutura aos conteúdos das páginas, tornando a busca, prospecção e extração de dados mais fácil de ser executada por agentes inteligentes de software, já que a WEB convencional foi concebida, primordialmente para uso por agentes humanos.

Ao estruturar as informações disponíveis, estes dados passam a se tornar mais próximos de uma representação de conhecimento baseado no conceito de ontologia. O conceito de ontologia se baseia na organização formal de uma informação, através da descrição de conceitos, em um determinado domínio de conhecimento, e sua relação com outros elementos (GUARINO, 1995). Através da utilização de ontologias, a web semântica busca alcançar seu objetivo de fornecer um meio mais legível de disponibilizar a informação para que agentes inteligentes possam analisar e extrair estes dados.

Estes agentes, segundo Russell e Norvig (1995), podem ser definidos como sistemas capazes de obter percepções em relação ao ambiente e então executar ações de resposta. Na perceção de Michael Wooldridge (2000) agentes são sistemas computacionais capazes de agir de maneira autônoma, de forma que alcancem seus objetivos e sejam capazes de decidirem sozinhos como agir, de acordo com uma dada situação.

O modelo de agentes conhecido como BDI (Belief, Desire, Intentions) seria capaz de compreender, de melhor maneira, a forma de representação de conhecimento utilizado na web semântica, por se tratar de uma abordagem de construção de agentes de cunho predominantemente cognitivo. O modelo BDI foi proposto pelo filósofo Michael Bratman (1987) e tem como foco o raciocínio prático humano, baseado nos conceitos de crenças, desejos e intenções (Belief-Desire-Intention) para formar uma linha de pensamento para o mecanismo de deliberação e ação do agente . Segundo Michael Wooldridge (2000) o modelo BDI é particularmente interessante porque combina três componentes diferentes: componente filosófico, componente de arquitetura de software e componente lógico.

Através dos agentes BDI é possível construir sistemas inteligentes nos quais os agentes possam consultar bases de conhecimento estruturadas. Um exemplo de utilização de base estruturada seriam ontologias que estejam disponíveis na WEB para consumo do agente. Através da expansão da base de conhecimento do agente se atende objetivos individuais e coletivos do agente.

Entretanto, a utilização de agentes para o consumo destas bases de conhecimentos pode se mostrar complexa. Estas bases podem possuir estruturas diferentes umas das outras, dificultando não apenas a navegação do agente como a estruturação de suas bases de conhecimento internas. Conforme será discutido mais adiante, neste trabalho, o problema de integração das crenças internas dos agentes com bases públicas de conhecimento ainda possui pontos em aberto que apresentam desafios para pesquisas neste campo de estudo.

Em contrapartida a estes problemas, devido a complexidade do agente, existem outras abordagens, entre elas a multi-contexto. Segundo Casali, Godo e Sierra (2005), ao separar a lógica de uma agente em contextos, obtém-se escalabilidade, eficiência e inclusive diminui a complexidade através da separação de responsabilidades em contextos. Para isso, ao longo deste projeto, o SIGON(GELAIM et al., 2019), um framework focado na criação de agentes multi-contexto foi utilizado.

1.1 PROBLEMA

A utilização de ontologias como base de conhecimento para agentes vem sendo explorada por algumas pesquisas (DIKENELLI; ERDUR; GUMUS, 2005), (MOREIRA et al., 2006), (KLAPISCAK; BORDINI, 2008), (MASCARDI et al., 2011), (CAMPOS, 2014), (FREITAS et al., 2015), (DEMARCHI, 2017). Em meio a estes trabalhos, os projetos de Felipe Demarchi (2017) e Campos (2014) abrangem um ponto importante no modelo idealizado pela web semântica, pois utilizam ontologias como base de conhecimento de maneira remota.

Segundo (CASALI, 2008) para alcançar todo o potencial das abordagens multiagentes, existem alguns desafios importantes apontados no futuro próximo, tais como: o desenvolvimento de novas teorias, arquiteturas e metodologias assim como apoiar os desenvolvedores de sistemas de agentes e estabelecer ligação com outros ramos da ciência computacional e com outras disciplinas.

Acima de todos os desafios encontrados no desenvolvimento de agentes BDI o principal seria o fornecimento de uma lógica de intenção

mais simples, porém significativa na esperança de trazer uma conexão mais estreita entre as implementações do BDI e as lógicas do BDI. As linguagens de programação de agente das implementações de BDI são normalmente restritas, para que os agentes possam reagir a um ambiente dinâmico.(HERZIG et al., 2017)

Os trabalhos citados acima em sua maioria são desenvolvidos utilizando a linguagem AgentSpeak e a plataforma JASON. O desenvolvimento de agentes BDI nestes moldes tende a ser mais restrito. Ao mesmo tempo em que reúnem toda a lógica necessária em si este formato acabam por tornar sua lógica interna complexa devido ao seu objetivo de estar preparado para diferentes objetivos.

Para que haja uma maior preparação de um agente à demanda de execução em múltiplos ambientes torna-se necessário outro tipo de abordagem do problema. No núcleo de pesquisa no qual se desenvolve este projeto, existe uma proposta de framework para a implementação de agentes multi-contexto denominada SIGON (GELAIM et al., 2019). Este projeto já rendeu alguns frutos como o trabalho de Freitas (2018). No entanto ainda não existe aplicação deste modelo integrado a ontologias públicas.

1.2 SOLUÇÃO PROPOSTA

Ao verificar alguns trabalhos na área de agentes inteligentes e utilização de ontologias, se percebe que uma importante abordagem de modelagem de arquitetura de agentes ainda não foi suficientemente explorada com este objetivo: a abordagem através de agentes multi-contexto.

Do ponto de vista da modelagem lógica, existem várias vantagens em adotar uma abordagem multi-contexto. Em primeiro lugar, separando a descrição lógica de um agente em um conjunto de contextos, cada qual com sua própria lógica, obtém-se, efetivamente, uma forma de lógica de várias ordenações. Isso traz para o sistema as vantagens de escalabilidade e eficiência. A segunda vantagem vem do mesmo problema. Essa abordagem possibilita a construção de agentes que usam várias lógicas diferentes de uma maneira que mantém estas lógicas separadas (CASALI, 2008).

Tendo isto em vista, este projeto propõe buscar uma solução para o desenvolvimento desta alternativa. O modelo a ser proposto tem como objetivo permitir que agentes construídos com base em sistemas multi-contexto interajam com ontologias remotas, tornando possível também

comparar os modelos de agentes existentes e o modelo de agente multi-contexto proposto, no que diz respeito ao uso do conhecimento obtido a partir de ontologias publicadas na WEB.

Este trabalho utilizou como principal base de comparação o projeto desenvolvido por Felipe Demarchi (2017), utilizando os avanços feitos por seu estudo até o momento. Para o desenvolvimento do agente multi-contexto será utilizado o framework Sigon, desenvolvido por Gelaim et al. (2019) . Para que seja possível realizar uma avaliação comparativa do modelo proposto neste projeto, será utilizada uma aplicação do modelo em um jogo do tipo Quiz, o qual utilizará informações coletadas pelos agentes BDI através da web semântica, semelhante à aplicação utilizada por Demarchi (2017).

1.3 OBJETIVOS

1.3.1 Objetivo geral

O objetivo deste trabalho é o desenvolvimento de um modelo de agente BDI baseado na abordagem de sistemas multi-contexto capaz de obter informações a partir de ontologias remotas a serem utilizadas no processo de deliberação do agente.

1.3.2 Objetivos específicos

- Compreender o estado da arte quanto a utilização de ontologias como base de conhecimento em sistemas multiagente;
- Avaliar os projetos correlatos desenvolvido;
- Propor um modelo que permita com que agentes Sigon consigam obter informações através de ontologias remotas;
- Desenvolver aplicação que permita a demonstração do modelo proposto;

1.4 MÉTODO DE PESQUISA

Este trabalho utilizará o método indutivo durante a sua execução. Este método consiste na coleta e síntese de informações relaci-

onadas com o tema estudado e, a partir destes estudos, formular um modelo capaz de solucionar o problema encontrado.

Por este trabalho ter como foco o desenvolvimento de um agente multi-contexto, capaz de consumir informações a partir de ontologias, este se encaixa no perfil de uma pesquisa aplicada, uma vez que possui como objetivo a geração de conhecimento prático para a solução de um problema.

Em relação ao método de abordagem, será utilizada a abordagem qualitativa. Este tipo de abordagem não se preocupa com representatividade numérica, mas, sim, com o aprofundamento da compreensão do objeto estudado.

Ao longo deste trabalho, foi realizado um levantamento bibliográfico dos estudos relacionados ao objetivo deste projeto. Entre estes estudos relacionados foi utilizado principalmente o modelo proposto por Gelaim et al. (2019) e o proposto por Felipe Demarchi (2017) empregando, além da pesquisa bibliográfica, a interação direta com os pesquisadores. Desta maneira este projeto também possui um viés exploratório por envolver levantamento bibliográfico e fontes de informação através de pessoas que possuem experiências na área.

2 REFERENCIAL TEÓRICO

2.1 WEB SEMÂNTICA

A maioria do conteúdo disponível na web hoje em dia é voltado apenas para a leitura feita por humanos, desta maneira torna-se difícil a leitura destes conteúdos por parte de computadores e agentes inteligentes.

Em resolução a esta situação Berners-Lee, Hendler e Lassila (2001) idealizou um modelo de estruturação para o conteúdo relevante das páginas web. Desta maneira uma tarefa se torna mais fácil de ser executada por um agente. Neste ambiente, um computador poderia transitar entre diferentes páginas executando determinadas tarefas de maneira mais fácil e eficiente em relação a um ambiente não estruturado.

Este modelo foi idealizado de modo em que ele faça parte da web, funcionando como uma extensão e não como algo a parte. Nesta extensão, as informações dispostas possuem um significado bem definido permitindo que humanos e computadores consigam cooperar. Para que a web semântica funcione de fato, é necessário que suas informações, além de possuir uma estrutura definida, é necessário que também existam regras de inferência que possam conectar as informações para que conduzam a um raciocínio automatizado.(BERNERS-LEE; HENDLER; LASSILA, 2001)

Para que a web semântica alcance seu objetivo, é necessário utilizar uma linguagem que permita, ao mesmo tempo, expressar informações e regras para que futuros agentes possam consumir estes dados. Esta linguagem permite que haja inferência através do relacionamento entre regras e dados, além de suportar informações de qualquer sistema que siga este modelo de estruturação de conteúdo, também conhecido como representação de conhecimento. Através dessa linguagem, se passa a exportar estes conteúdos para a web fazendo com que ela passe a possuir lógica.(BERNERS-LEE; HENDLER; LASSILA, 2001)

Recentemente, iniciativas dentro e fora do World Wide Web Consortium (W3C) têm se baseado na expertise dessas comunidades, desenvolvendo linguagens de representação e anotação de conhecimento sobre a infra-estrutura atual da Web. Isso não apenas permite que o conhecimento recém-codificado seja facilmente disseminado pela Web, mas também fornece uma sintaxe conveniente para a anotação de conteúdo existente.(OSSENBRUGGEN; HARDMAN; RUTLEDGE, 2006)

Berners-Lee, Hendler e Lassila (2001) ainda relata que o verdadeiro poder da web semântica será alcançado a partir do momento em que agentes inteligentes possam processar dados obtidos através da coleta informações de múltiplas fontes além de interagirem uns com os outros trocando informações com base em seus resultados.

2.2 ONTOLOGIAS

A origem do termo Ontologia, segundo Staab e Studer (2009) trata-se de uma abstração filosófica que se refere à natureza e a estrutura entre elementos, um estudo de atributos que pertencem a elementos devido a sua natureza. Esta estrutura foca na formalização e registro de elementos através da descrição de seus atributos e suas relações com outros elementos. Na ciência da computação, utilizou-se este conceito para nomear como ontologia um tipo específico de estrutura de informação.

Estas ontologias computacionais são um meio de modelar formalmente a estrutura de um sistema, como por exemplo, suas entidades relevantes e suas relações que surgem durante sua observação, entre outros que forem úteis para o propósito da modelagem (STAAB; STUDER, 2009). Esta tecnologia se torna muito útil por permitir que sua estruturação se forme como um modelo de representação de conhecimento. Através da pesquisa feita por Omerovic, Milutinovic e Tomazic (2001) constatou-se que uma representação de conhecimento deveria ser composta de ontologias populadas por conceitos, como indica a Figura 1.

Segundo Berners-Lee, Hendler e Lassila (2001), o tipo mais comum de ontologia para a Web tem uma taxonomia e um conjunto de regras de inferência. A taxonomia define classes de objetos e relações entre eles. Classes, subclasses e relações entre entidades são uma ferramenta muito poderosa para uso na Web. Podemos expressar um grande número de relações entre entidades, atribuindo propriedades a classes e permitindo às subclasses herdarem tais propriedades.

O principal objetivo da introdução de esquemas formais apropriados para conceito e ontologia é estruturar o conhecimento para torná-lo compartilhável entre computadores e pessoas (OMEROVIC; MILUTINOVIC; TOMAZIC, 2001). O computador não compreende, de fato, nenhuma dessas informações. Mas, agora, pode manipular os termos de maneira muito mais eficaz, de maneiras úteis e significativas para o usuário humano (BERNERS-LEE; HENDLER; LASSILA, 2001).

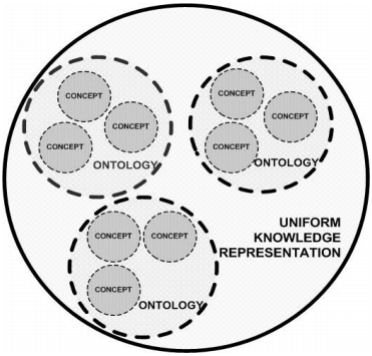

Figura 1 – Modelo de representação de conhecimento uniforme, consistindo em ontologias que são preenchidas por conceitos.
Fonte:(STAAB; STUDER, 2009)

Em Handbook on Ontologies (STAAB; STUDER, 2009) são expostos alguns modelos formais de lógicas descritivas utilizadas para expressar a representação de conhecimento existente em ontologias de maneira com que um agente Inteligente possa utilizar a informação, sendo elas RDF (Resource Description Framework) e OWL (Web Ontology Language).

Em The Semantic Web (BERNERS-LEE; HENDLER; LASSILA, 2001) o autor define RDF como uma maneira de expressar um significado através de um conjunto de triplas, cada tripla sendo um pouco como o sujeito, verbo e objeto de uma sentença elementar. No RDF, um documento faz afirmações de que determinadas coisas (pessoas, páginas da Web ou qualquer outra coisa) possuem propriedades com determinados valores (outra pessoa, outra Web página). Esta estrutura acaba por ser um caminho natural para descrever a grande maioria dos dados processados por agentes. Com base no RDF, se deriva o RDFS (Resource Description Framework Schema), podemos descrevê-lo como um simples indicador de esquema ontológico que suporta apenas hierarquias de classes e propriedades, bem como restrições de domínio e intervalo para propriedades.

RDF e RDFS permitem a representação de algum conhecimento ontológico. As principais primitivas de modelagem de RDF / RDFS dizem respeito à organização de vocabulários em hierarquias tipadas: relações de subclasse e sub-propriedade, domínio e restrições de intervalo e instâncias de classes. No entanto, vários outros recursos estão ausentes. (STAAB; STUDER, 2009)

Devido a ausência de alguns recursos importantes como disjunção de classes e restrições por cardinalidade, passou-se a utilizar como padrão a linguagem OWL para o desenvolvimento de ontologias por se tratar de uma linguagem mais completa e atender melhor às necessidades levantadas pelo W3C, OWL. Baseia-se no RDF e RDF Schema e usa a sintaxe XML do RDF.(STAAB; STUDER, 2009)

2.2.1 OWL

A Web Semântica, durante sua concepção, se tratava uma visão para o futuro da Web, na qual a informação recebe um significado explícito, facilitando para as máquinas processarem e integrarem automaticamente as informações disponíveis na Web. O primeiro nível acima do RDF exigido para a Web Semântica é uma linguagem de ontologia que pode descrever formalmente o significado da terminologia usada em documentos da Web. Se for esperado que máquinas executem tarefas de raciocínio úteis nesses documentos, o idioma deve ir além da semântica básica do RDF. Para atender a essa necessidade foi projetado o OWL, uma linguagem de Ontologia para a Web. (W3C, 2004a)

Segundo a W3C (2004a) a OWL é projetada para uso por aplicativos que precisam processar o conteúdo de informações em vez de apenas apresentar informações aos seres humanos. O OWL facilita uma maior interpretabilidade de máquina do conteúdo da Web do que a suportada pelo XML, RDF e RDF Schema (RDF-S), fornecendo vocabulário adicional junto com uma semântica formal.

Uma ontologia OWL é um grafo RDF, que por sua vez é um conjunto de triplas RDF. Como em qualquer RDF, um grafo de ontologia OWL pode ser escrito em muitas formas diferentes de sintaxe. No entanto, o significado de uma ontologia OWL é determinado apenas pelo RDF. Assim, é permitido usar outros formulários RDF / XML sintáticos, desde que estes resultem no mesmo conjunto subjacente de triplos RDF.(BECHHOFER et al., 2004)

Como um exemplo simples de uma forma sintática alternativa resultando nas mesmos triplas RDF:

A OWL 2.0 fornece alguns perfis para a sua utilização. Estes são subconjuntos do OWL 2 que oferecem vantagens importantes em cenários de aplicativos específicos. Cada um destes perfis utiliza diferentes aspectos de seu poder de expressão em troca de benefícios computacionais. Estes perfis são descritos pela W3C (2012) como:

- OWL 2 EL: É adequado para aplicações em que ontologias muito

```
<owl:Class rdf:ID="Continent"/>

<rdf:Description rdf:about="#Continent">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/
owl#Class"/>
</rdf:Description>
```

Figura 2 – Comparação entre OWL e RDF alcançando o mesmo resultado.

Fonte:(BECHHOFER et al., 2004)

grandes são necessárias e onde poder expressivo pode ser trocado por garantias de desempenho.

- OWL 2 QL: È utilizado para aplicativos em que ontologias relativamente leves são usadas para organizar um grande número de indivíduos e onde é útil ou necessário acessar os dados diretamente por meio de consultas relacionais.
- OWL 2 RL: Assim como a OWL 2 QL, este perfil também é utilizado para aplicações com ontologias leves porém ele se atém a operações diretas em dados na forma de triplos RDF.

2.3 LINKED DATA

De forma geral os dados publicados na Web são disponibilizados como despejos brutos em formatos como CSV ou XML ou marcados como tabelas HTML, sacrificando grande parte de sua estrutura e semântica. Na Web convencional, a natureza do relacionamento entre dois documentos vinculados está implícita, pois o formato dos dados, isto é, HTML, não é suficientemente expressivo. No entanto, nos últimos anos, a Web evoluiu de um espaço global de informações de documentos vinculados para um espaço onde documentos e dados estão vinculados. Por trás dessa evolução, está um conjunto de práticas recomendadas para publicar e conectar dados estruturados na Web, conhecidos como Linked Data.(BIZER; HEATH; BERNERS-LEE, 2009)

Para Bizer, Heath e Berners-Lee (2009), a Linked Data é simplesmente sobre o uso da Web para criar links entre dados de diferentes fontes. Tecnicamente, a Linked Data se refere aos dados publicados na Web de forma que sejam legíveis por máquina e seu significado seja explicitamente definido, vinculado a outros conjuntos de dados exter-

nos e, por sua vez, possa ser vinculado a partir de conjuntos de dados externos.

A Linked Data utiliza como principal elemento a utilização do modelo RDF, o modelo RDF codifica dados na forma de sujeito, predicado e objeto triplos. O assunto e o objeto de uma tripla podem ser representados por duas referências, onde cada uma identifica um recurso ou um URI e uma literal de string, respectivamente. O predicado específica como o assunto e o objeto estão relacionados e também é representado por um URI.(BIZER; HEATH; BERNERS-LEE, 2009)

Segundo Bizer et al. (2008), o exemplo mais visível de adoção e aplicação dos princípios do Linked Data foi o projeto Linking Open Data. O objetivo original e contínuo do projeto é inicializar a Web de dados, identificando os conjuntos de dados existentes disponíveis sob licenças abertas, convertendo-os em RDF de acordo com os princípios de Linked Data e publicando-os na Web.

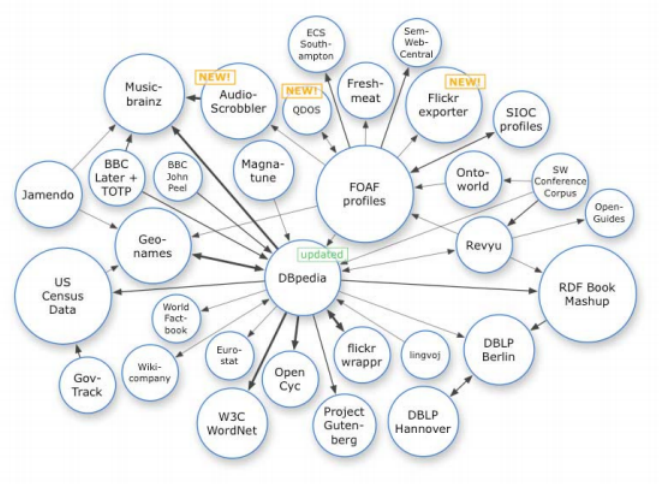

Figura 3 – Amostra da estrutura dados disponibilizada pelo Linking Open Data

Fonte:(BIZER et al., 2008)

2.3.1 SPARQL

A Web Semântica é a iniciativa do W3C de tornar as informações na Web legíveis não apenas por seres humanos, mas também por máquinas. RDF é o modelo de dados para dados da Web Semântica e SPARQL é a linguagem de consulta padrão para este modelo de dados. Em 2004, o Grupo de Trabalho de Acesso a Dados do RDF, parte da Atividade Semântica na Web do W3C, lançou um primeiro rascunho público de trabalho de uma linguagem de consulta para o RDF, chamada SPARQL. Desde então, o SPARQL foi rapidamente adotado como padrão para consultar dados da Web Semântica. Em janeiro de 2008, o SPARQL se tornou uma recomendação do W3C.(ARENAS; PÉREZ, 2011)

Uma consulta SPARQL é composta por: um corpo, que é uma expressão complexa de correspondência de padrão de gráfico RDF e uma cabeça, que é uma expressão que indica como construir a resposta para a consulta. A avaliação de uma consulta é feita em duas etapas. Primeiro, os valores são dados às variáveis de forma que as triplas no corpo da consulta correspondam às triplas no gráfico consultado. Segundo, os valores atribuídos às variáveis são processados para construir a resposta.(ARENAS; PÉREZ, 2011) Um padrão de tripla aplicado a um gráfico corresponde a todos as triplas com termos RDF idênticos para o assunto, predicado e objeto correspondentes. As variáveis no padrão triplo, se houver, são vinculadas aos termos RDF correspondentes nos triplos correspondentes.(W3C, 2004b).

```
SELECT ?Author
WHERE {
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:isPartOf   ?InPods .
  ?InPods     sw:series      conf:pods .
  ?DbpPerson  owl:sameAs     ?Author .
  ?DbpPerson  dbo:birthPlace dbpedia:Oklahoma .  }
```

Figura 4 – Query SPARQL
Fonte:(ARENAS; PÉREZ, 2011)

2.4 AGENTES

Agentes ou, mais especificamente agentes racionais, podem ser considerados uma classe de sistemas. Estes sistemas são chamados agentes devido a sua capacidade de agir de maneira independente e autônoma. Desta forma, eles podem cumprir com seus objetivos (WOOLDRIDGE, 2000). Um agente é algo capaz de perceber o seu ambiente através de sensores e agir neste ambiente através de atuadores, da mesma forma que um agente humano possui olhos e ouvidos que fazem o papel de sensores e mãos e outros membros como atuadores (RUSSELL; NORVIG, 1995). O requisito que exige que os agentes estejam inseridos em um ambiente, seja o ambiente físico ou virtual, implica que muitos dos sistemas desenvolvidos no ramo da IA não sejam categorizados como agente (WOOLDRIDGE, 2000).

Um agente inteligente, durante sua criação, tem o propósito alcançar o melhor resultado possível. Esta definição de agente levanta um incógnita em relação a que tipo de resposta obtida seria considerada a melhor possível. Isso depende de o que pode ser considerado como critério de avaliação desta resposta. A questão da escolha de um parâmetro de avaliação para o sucesso de um agente é um ponto importante a ser levado em consideração. Não podemos culpar um agente por não levar em conta algo que não percebesse, ou por não tomar uma ação que ele é incapaz de tomar (RUSSELL; NORVIG, 1995).

De acordo com a percepção de Russell e Norvig (1995), a definição de um agente racional ideal seria: Para cada possível sequência de percepção, um agente racional ideal deve fazer qualquer ação que seja esperada para maximizar sua medida de desempenho, com base nas evidências fornecidas pela seqüência de percepção e qualquer conhecimento embutido.

Segundo Russell e Norvig (1995), um ponto importante na definição de um agente inteligente é a utilização do conhecimento embutido durante sua criação. Caso todas as ações de um agente sejam baseadas em informações inseridas previamente às suas experiências em seu ambiente de execução, de maneira com que o agente não necessite de captar informações deste ambiente, podemos declarar este agente como não autônomo. O comportamento de um agente pode ser baseado em sua própria experiência e no conhecimento embutido, usado na construção do agente para o ambiente específico em que ele opera. Um sistema é autônomo, na medida em que seu comportamento é determinado por sua própria experiência.

Na perspectiva de Wooldridge (2000) existem propriedades im-

portantes para a definição de um agente inteligente. São elas:

- Autonomia: Um agente autônomo é aquele que possui suas próprias crenças, desejos e intenções e que não se torna submisso em relação a outros agentes. Um agente é considerado autônomo quando possui seus próprios objetivos, estejam estes objetivos concorrendo ou não com outros agentes.
- Reatividade: Como já citado anteriormente, devido a sua concepção, é imprescindível que o agente esteja inserido em uma ambiente. A reatividade é a habilidade de perceber que tipo de informações estão sendo obtidas deste ambiente e responder a elas.
- Proatividade: O agente é dito proativo quando, além de responder aos estímulos externos providos pelo ambiente, ele também possui metas as quais ele pretende alcançar independentemente de estímulos através de uma iniciativa.
- Habilidade Social: É a habilidade que possibilita com que um agente possa se comunicar com outros agentes e, em alguns casos, com humanos.

2.4.1 Agentes BDI

O campo de pesquisas relacionadas à agentes inteligentes têm utilizado conhecimentos pautados pelo estudo da mente humana e o seu comportamento (O'HARE; JENNINGS, 1996). Através destes estudos, emergiram algumas ideias distintas sobre o estudo de sistemas orientados a agentes, dentre estas abordagens se destaca a perspectiva de Michael Bratman (BRATMAN, 1987).

A ideia principal de Bratman (BRATMAN, 1987) é que os desejos e crenças do agente, em certa medida, lhe fornecem razões para agir de várias maneiras naquele momento. O que a racionalidade exige é que sua ação intencional seja pelo menos tão fortemente apoiada por essas crenças de desejos quanto qualquer uma de suas supostas alternativas. Este modelo se apoia nos conceitos de crenças, desejos e intenções (Belief-Desire-Intention) para formar o processo de decisão do agente. Na perspectiva de (RAO; GEORGEFF, 1995) estes conceitos podem ser descritos como:

- Crenças: Representam e armazenam as características do ambiente, as quais são atualizadas apropriadamente após a percepção

de cada ação. O mundo é dinâmico e os sistemas geralmente têm apenas uma visão local do mundo. Portanto, eventos passados precisam ser lembrados e atualizados, ou seja, é necessário armazenar informações importantes a partir dos dados perceptivos básicos

- Desejos: Na literatura filosófica, os desejos podem ser inconsistentes e o agente não precisa conhecer os meios para alcançar esses desejos. Os desejos têm a tendência de guiar o agente em diferentes direções. Eles são insumos para o processo de deliberação do agente, o que resulta no agente escolhendo um subconjunto de desejos que são consistentes e alcançáveis. Tais desejos realizáveis consistentes são geralmente chamados de metas.

- Intenções: Em cada momento do modelo, é atribuido um conjunto de caminhos que o agente é interpretado como tendo selecionado ou preferido. Grosso modo, as intenções são definidas como as condições que inevitavelmente se mantêm em cada um dos caminhos selecionados. Aqui consideramos as intenções de realização, elas possuem objetivos de realizar várias condições para chegar até a intenção objetivo. No entanto, as intenções podem ser definidas para manter certas condições também.

Segundo (WOOLDRIDGE, 2000) as intenções desempenham papel fundamental no processo de raciocínio do agente. Através das intenções o agente começa a possuir um objetivo a alcançar, levando desta forma o agente à ação. Se um agente possui uma intenção, então é esperado que ele haja de acordo com essa intenção, para assim alcançá-la. Desta forma é esperado que o agente siga um curso de ação que se acreditava ser o que mais satisfaria sua intenção. Caso um curso de ação do agente não seja bem sucedido é esperado que haja outra tentativa buscando outras alternativas para satisfazer sua intenção. No entanto, não se deve persistir por muito tempo em uma intenção caso fique claro que ela nunca será alcançada, então é apenas racional abandonar esta intenção. Da mesma forma, se a razão para ter uma intenção desaparece, é racional abandonar a intenção. Uma vez que um agente tenha adotado uma intenção, o próprio fato de ter essa intenção restringirá meu futuro raciocínio prático. Por exemplo, embora tenha alguma intenção em particular, não vou considerar opções incompatíveis com essa intenção.

O processo de raciocínio de um agente BDI (Belief-Desire-Intention) modelado por Wooldridge é separado em sete elementos,como mostrado na Figura 5, são eles:

- Função de revisão de crenças, que recebe um input perceptual e as crenças atuais do agente e, com base nelas, determina um novo conjunto de crenças;
- Um conjunto de crenças atuais, representando as informações que o agente tem sobre seu ambiente atual;
- Função de geração de opções, que determina as opções disponíveis para o agente (seus desejos), com base em suas crenças atuais sobre seu ambiente e suas intenções atuais;
- Um conjunto de desejos atuais, representando possíveis cursos de ações disponíveis para o agente;
- Função de filtro, que representa o processo de deliberação do agente e que determina as intenções do agente com base em suas crenças, desejos e intenções atuais;
- Um conjunto de intenções atuais, representando o foco atual do agente, os estados nos quais ele se comprometeu a tentar alcançar;
- Função de seleção de ação, que determina uma ação a ser executada com base nas intenções atuais.

2.4.2 Agente Multi-Contexto

Modelar diferentes noções por meio de várias modalidades pode ser muito complexo se apenas uma estrutura lógica for usada. A fim de ajudar no projeto de tais sistemas lógicos complexos surge a noção de sistema multi-contexto (MCS).(CASALI; GODO; SIERRA, 2005)

A noção de contexto tem sido estudada em muitas áreas de pesquisa e particularmente na Inteligência Artificial. Os contextos são vistos como uma abordagem importante para representar certos tipos de raciocínio. Por um lado, os contextos são uma ferramenta para formalizar a localidade do raciocínio. Enquanto, por outro lado, os contextos são introduzidos como meio de resolver o problema da generalidade. (CASALI, 2008)

Sistemas multiagentes são sistemas complexos que podem ser bem modelados por MCSs, pois permitem representar seus componentes arquitetônicos e descrever detalhadamente a interação entre eles. Segundo Parsons et al. (2002) ao utilizar uma abordagem multi-contexto, uma arquitetura de agente consiste em quatro tipos básicos de componente:

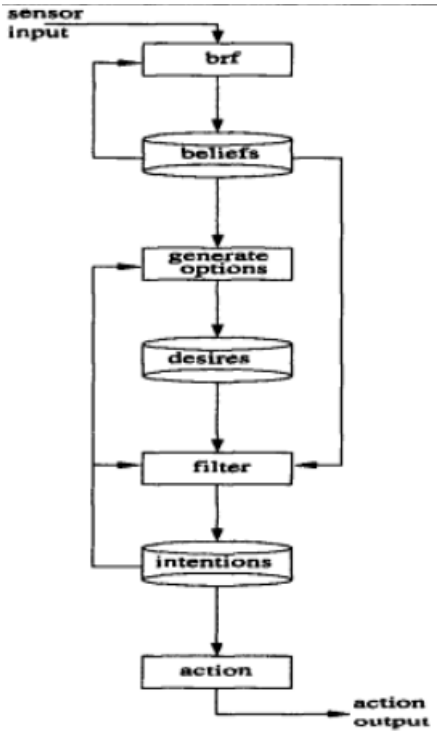

Figura 5 – Processo de Raciocinio de um agente BDI
Fonte:(WOOLDRIDGE, 2000)

- Unidades: entidades estruturais representando os principais componentes da arquitetura.
- Lógicas: Linguagens declarativas, cada uma com um conjunto de axiomas e um número de regras de inferência. Cada unidade tem uma lógica única associada a ela.
- Teorias: conjuntos de fórmulas escritas na lógica associada a uma unidade.
- Regras de ponte: Regras de inferência que relacionam fórmulas em diferentes unidades.

Os contextos contêm o conhecimento de solução de problemas do agente e esse conhecimento é codificado na teoria específica que a unidade encapsula. Em geral, a natureza dos contextos varia entre as

arquiteturas. Segundo o trabalho publicado por Casali (2008) o mecanismo de dedução de sistemas multi-contexto é baseado em dois tipos de regras de inferência, regras internas, dentro de cada unidade; e as regras da ponte, do lado de fora. As regras internas permitem extrair consequências dentro de uma teoria, enquanto as regras de ponte permitem incorporar resultados de uma teoria em outra. Em qualquer arquitetura representada, as regras de ponte definem a interação dos componentes. Essas regras fornecem o mecanismo pelo qual as informações são transferidas entre as unidades. As regras da ponte examinam continuamente as teorias dos contextos que aparecem em suas premissas, procurando novos conjuntos de fórmulas que os combinem. Isso significa que todos os componentes da arquitetura estão sempre prontos para reagir a qualquer alteração (externa ou interna) e que não há elementos centrais de controle.(CASALI, 2008).

2.5 JASON

Jason se trata de uma plataforma para o desenvolvimento de multi-agentes, esta plataforma faz o papel de interpretador para uma extensão da linguagem AgentSpeak, que por sua vez é baseada na arquitetura BDI. Além disso, a plataforma foi feita em Java, o que facilita a realização de pequenas funções para auxiliar os agentes em seu funcionamento. Em relação ao agente desenvolvido a partir do Jason, é importante ressaltar elementos importantes de sua arquitetura, tratam-se de suas bases de crença, planos e objetivos, que por sua vez fazem paralelo aos conceitos de crença, intenção e desejo. (JENSEN; BUCH, 2014)

Hübner, Bordini e Vieira (2014) ressalta algumas funcionalidades, entre elas a aceitação de anotações em seus predicados, Uma anotação é uma lista de termos delimitadas por colchetes que segue imediatamente o predicado. Na base de crenças, anotações são usadas para registrar as fontes das informações. Se destaca também entre suas funcionalidades as ações internas, essas ações internas são distribuídas com o Jason e implementam operações podem ser muito importantes para a programação geral do BDI. Uma ação padrão específica a ser lembrada é a ação .send, que é usada para comunicação entre agentes.

A interpretação feita pelo Jason acaba então afetando diretamente o ciclo de raciocínio do agente. Sobre o ciclo de raciocínio deste agente podemos dizer que o principal pilar para o seu funcionamento é a base de crenças, que se trata de uma lista de literais. Estas crenças

podem ser percebidas através do seu ambiente, comunicação com outros agentes ou inseridas pelo desenvolvedor. Por definição, o agente acredita que esta informações captadas do ambiente são verdadeiras. Os objetivos, por sua vez, funcionam de maneira diferente. Os objetivos expressam as propriedades dos estados do mundo que o agente deseja realizar. Ao representar um objetivo em um programa de agente, isso significa que o agente está comprometido a agir de modo a mudar o mundo para um estado no qual o objetivo é de fato verdade. A partir da definição de um objetivo, o próximo passo é a criação de planos. Eles são o que os agentes usam para reagir às circunstâncias em que se encontram enquanto tentam alcançar seus objetivos.(BORDINI; HüBNER; WOOLDRIDGE, 2007)

Em Programming Multi–Agent Systems in AgentSpeak using Jason(BORDINI; HüBNER; WOOLDRIDGE, 2007) o autor ressalta a distinção entre a arquitetura de um agente e seu programa. A arquitetura de um agente é dita como a estrutura utilizada pelo software na qual um programa de agente é executado. Embora escrevamos o programa que guiará o comportamento do agente muito das ações do agente serão guiadas pela sua arquitetura. Isso se torna mais claro a partir do momento em que o agente reage a eventos através da execução de planos, neste momento um componente da arquitetura do agente se torna responsável pelas mudanças no seu ambiente, baseado nas escolhas de ações feitas pela interpretação do programa. Segundo Bordini, Hübner e Wooldridge (2007) o interpretador Jason executa um programa agente, este agente realiza sua operação de ciclo de raciocínio, como explicado anteriormente, que, no caso de Jason, podemos dividir em 10 etapas principais. Podemos pensar no ciclo de raciocínio efetuado pelo interpretador como algo análogo ao loop de decisão do BDI.

3 TRABALHOS RELACIONADOS

Para esta pesquisa foram utilizados como principais trabalhos relacionados pesquisas resultantes de trabalhos com agentes inteligentes. Os frameworks e funcionalidades criados a partir destas pesquisas servem como base para o desenvolvimento deste projeto.

3.1 UTILIZAÇÃO DE AGENTES COM ONTOLOGIAS REMOTAS PARA A PRODUÇÃO DE CONTEÚDO SIGNIFICANTE A PARTIR DE INFORMAÇÕES DISPONÍVEIS NA WEB SEMÂNTICA

O trabalho desenvolvido por Demarchi (2017) trata-se da utilização de agentes BDI para o acesso de bases de conhecimento remotas, sendo estes agentes implementados através da plataforma Jason. A partir da comunicação destes agentes Jason é possível ao agente adquirir conhecimento através da sua capacidade de comunicação com a web semântica. Podemos considerar estas informações adquiridas pelos agentes como conhecimento uma vez que estes dados captados possuem ligações a outros termos, criando desta forma uma rede de associações. Para que um agente consiga acessar estes dados foi utilizada a abordagem desenvolvida por Campos (2014), desta forma se torna possível o com que crenças possam ser representadas através de URIs, links externos para a base de dados remota.

Para que seja possível a aplicação deste modelo, o autor efetuou alterações no código do Jason de forma que torna-se possível a conexão dos agentes à base de conhecimento chamada DBPedia. As alterações feitas no interpretador não contemplam o seu ciclo de raciocínio. Uma das premissas deste projeto é a utilização de uma estrutura já consolidada e validada, segundo a comunidade científica. A partir desta premissa, a abordagem seguida é a de adição de eventos internos ao código do interpretador, de forma que não se afete o ciclo de raciocínio. Através destes eventos internos o agente passa a ser capaz de utilizar consultas SPARQL. Trata-se de uma forma de busca de informações nas ontologias remotas. Os resultados obtidos através destas consultas inicialmente atualizam a base de crenças do agente. Em seguida, o agente utiliza a sua lista de planos, que necessita fundamentalmente da utilização da ações internas para controlar o acesso à web semântica.

Estas funções utilizadas para que seja possível a construção de consultas SPARQL se dividem em três classes. São elas: SparqlObject,

SparqlResult e SparqlSearch. Em relação a classe SparqlObject, o seu objetivo é a representação de uma tripla em cada uma de suas instâncias a ser utilizada na consulta. Em seguida, a classe SparqlResult tem como função a armazenagem dos resultados de uma consulta efetuada. Por ultimo, a classe SparqlSearch, a qual fará uso de objetos do tipo SparqlObject para construir a query SPARQL e de objetos do tipo SparqlResult para devolver o resultado da consulta.(DEMARCHI, 2017)

A principal ação interna desenvolvida neste projeto seria a searchdbpedia, utilizada ,a priori, como uma forma de validador em relação à construção de consultas. Através de seus parâmetros de entrada, é possível verificar a validação. Esta função lida diretamente com as funções relacionadas ao SPARQL, os parâmetros de entrada utilizados na função searchdbpedia são direcionados justamente para a inicialização da classe SparqlObject e realização da consulta. Em caso das informações inseridas por parâmetro serem insuficientes, a função reconhecerá a operação como inválida, impossibilitando a continuidade da consulta. Caso contrário, a operação será considerada válida e se obterá como retorno uma lista de SparqlResult.(DEMARCHI, 2017)

Segundo Demarchi (2017), este evento interno é a principal demonstração do acesso que um agente pode fazer à DBPedia por intermédio das modificações feitas no código do interpretador Jason. A construção da consulta SPARQL não é feita de forma automática pelo agente, mas sim definida no desenvolvimento do evento interno, o qual pode ser desenvolvido para diferentes contextos.

3.2 SIGON: A MULTI-CONTEXT SYSTEM FRAMEWORK FOR INTELLIGENT AGENTS

A partir do conceito de agentes multi-contexto, Gelaim et al. (2019) desenvolveu um framework para o desenvolvimento deste modelo de agentes. O Sigon, como é chamado, utiliza como principais fontes do seu desenvolvimento os trabalhos desenvolvidos por Parsons et al. (2002) e Casali, Godo e Sierra (2005), desta forma é possível a implementação das principais características de ambos os trabalhos no framework desenvolvido. Por estes estudos possuirem como base o modelo de agente BDI, este framework também possui como foco este modelo.

Segundo Gelaim et al. (2019), o modelo de agente Sigon, baseando-se no modelo de Parsons et al. (2002), possui um contexto de comunicação. Este contexto se torna obrigatório quando se leva em conta

a necessidade da interação entre o agente e o seu ambiente. Para que haja esta interação, este contexto possui uma série de sensores e atuadores. Para que haja de fato um interação entre o agente e o seu ambiente, é necessário que haja ao menos um sensor e um atuador não havendo uma quantidade limite para sua utilização. A integração entre sensores e o contexto de comunicação possui um papel fundamental. Este contexto, ao receber as informações providas pelo sensor, executa suas regras de ponte, o que representa o ciclo de raciocínio. O Atuador por sua vez é responsável por alterar o ambiente e possui relação direta com o contexto de planejamento pois é utilizado para a execução de ações. Ambos os componentes do contexto de comunicação de seus identificadores e ação a ser executada.

O agente Sigon também possui um contexto de planejamento, se baseando, desta vez, no trabalho desenvolvido por Casali, Godo e Sierra (2005), este contexto é composto de planos e ações. Ambas as funções utilizam como parâmetros pré-condições e pós-condições para a sua execução assim como o seu custo. As ações recebem ainda um parâmetro definindo sua nomenclatura, enquanto os planos recebem dois parâmetros adicionais que definem o objetivo a ser alcançado e a ação que deve ser executada para alcançar o objetivo.(GELAIM et al., 2019)

Estes contextos, citados até então podem ser definidos como contextos funcionais. No entanto, este modelo de agente também possui o contexto lógico. Este contexto pode ser utilizado para representar os estados mentais como crenças, desejos e intenções, ele é composto de um conjunto de fórmulas, que podem ser uma cláusulas proposicionais, de primeira ordem ou expressão lógica. O contexto lógico pode ser definido como: C = (L, Ax, E), onde 'L' se trata da linguagem do contexto, 'Ax' é o conjunto de axiomas, e 'E' são as regras de inferência.(GELAIM et al., 2019)

No modelo Sigon, descrito até o momento, percebe-se que não houve imposição de regras de ponte. Desta maneira o desenvolvedor se torna mais livre para ditar o comportamento do seu agente. No entanto, o Sigon possui um agente BDI pré definido como mostra a figura 6.

Para que seja possível o desenvolvimento e execução do agente Sigon é necessário introduzir o seu framework. O framework é composto de dois módulos, o primeiro módulo realiza as validações léxicas e sintáticas necessárias no código do agente e possui como responsabilidade a transformação do código fonte do agente em um executável, enquanto o segundo módulo lida com contextos e regras de ponte do agente. As

$$AG_{BDI} = \langle \{CC, BC, DC, IC, PC\}, \Delta_{br} \rangle$$

$$\Delta_{br} = \{$$

$$\frac{CC\colon sense(\varphi) \ and \ PC\colon plan(\varphi,\alpha,Pre,Post,c_a) \ and \ DC\colon \varphi}{BC\colon \varphi}$$

$$\frac{DC\colon \varphi \ and \ BC\colon not \ \varphi \ and \ IC\colon not \ \varphi}{IC\colon \varphi}$$

$$\frac{PC\colon plan(\varphi,\alpha,Pre,Post,c_a) \ and \ IC\colon \varphi \ and \ BC\colon Pre}{CC\colon \alpha}$$

$$\}$$

Figura 6 – Modelo de representação do conhecimento, consistindo em ontologias preenchidas por conceitos.
Fonte:(GELAIM et al., 2019)

classes de contexto de um agente são compostas por um nome único, uma teoria, métodos de verificação e atualização da teoria e regras internas. Cada contexto pode possuir diferentes regras internas. Para que haja uma padronização entre eles, o framework disponibiliza uma interface, que deve ser implementada por todos os contextos, criando assim uma padronização. O framework ainda disponibiliza a utilização de regras internas assim como a utilização de regras de ponte. Podemos definir como regras internas as utilizadas pelo padrão BDI enquanto enquanto as regras de ponte criadas pelo desenvolvedor podem ser utilizadas para criar ou customizar o comportamento do agente.(GELAIM et al., 2019)

4 MODELO PROPOSTO

Este projeto tem como um dos seus principais objetivos a idealização e implementação de um modelo de agente multi-contexto capaz de buscar informações na internet de forma que possa atender à sua intenção de obter conhecimento sobre um determinado assunto. Para isto, este trabalho visa ampliar as funcionalidades de um modelo de agente multi-contexto. Por se tratar de um agente multi-contexto, isolaremos esta funcionalidade em um contexto específico.

Para que seja possível obter os dados que alimentarão o agente, este contexto responderá às intenções do mesmo, de forma que, a partir das intenções, o contexto compreenda que existe uma necessidade de buscar sobre aquele assunto e, então, adicionar este conhecimento obtido ao agente. A fonte de dados utilizada se trata da DBpedia, uma base de dados remota categorizada como ontologia. Esta ontologia, além de permitir o acesso aos dados requisitados, também fornece uma referência aos dados que possuam algum tipo de relacionamento com o elemento pesquisado. Assim, o agente consegue não apenas obter conhecimento sobre o assunto pesquisado, como também passa a possuir consciência da existência de outros assuntos.

O processo de desenvolvimento utilizou o modelo de agente Sigon, desenvolvido por Gelaim et al. (2019), para implementar o contexto ontológico. Também foi utilizado, como ponte de comunicação entre o agente e a ontologia, o pacote semanticWeb desenvolvido por Demarchi (2017) em sua dissertação. Ao unir estes elementos se visa criar um contexto capaz de receber as demandas de conhecimento do agente e a partir da comunicação com a base de dados remota obter conhecimento desejado. Ao longo deste capítulo serão exibidos os conceitos abordados durante o período de desenvolvimento.

4.1 ADIÇÃO DE CONTEXTO

Atualmente o Sigon disponibiliza para o desenvolvedor uma estrutura capaz de suportar a criação de novos contextos. Podemos então utilizar esta estrutura existente para desenvolver uma nova funcionalidade sem que haja necessidade de alterar a arquitetura existente. Por estarmos lidando com uma arquitetura de agentes multi-contexto, nos é disponibilizada a estrutura necessária para criar um novo contexto que não possua relação com os contextos já existentes. Assim, é possível

adicionar um novo contexto que possua sua própria lógica sem que haja necessidade de alterar outros elementos do agente.

Para que exista um padrão entre os contextos do agente é estipulada uma interface ao qual todos os contextos devem implementar. Isto garante que contextos implementem suas funções de acordo com o seu objetivo especificado.

```
public interface ContextService {

    public Theory getTheory();
    public boolean verify(String fact);
    public void appendFact(String fact);
    public void addInitialFact(String fact) throws InvalidTheoryException;
    public String getName();
}
```

Figura 7 – Interface utilizada por todos os contextos.

Podemos então considerar um paralelo entre a adição deste novo contexto na estrutura do agente e a adição de um plugin à uma plataforma com o objetivo de trazer novas características a um ambiente já existente. Baseando-se neste conceito, o projeto desenvolvido visou desenvolver novas funcionalidades, de forma que elas poderiam ser adicionadas futuramente a um agente, de acordo com a necessidade do desenvolvedor. Desta forma, consegue-se trazer um maior dinamismo para as capacidades do agente.

4.2 CICLO DE APRENDIZADO

Para o desenvolvimento deste projeto está sendo usado o modelo de agente BDI. Este tipo de agente possui um ciclo de informações que, por sua vez, é guiado pelas suas crenças, desejos e intenções. Desta forma, é necessário que o contexto ontológico interaja com estes outros contextos para, assim, passar a fazer parte do ciclo de raciocínio do agente. Estas interações entre contextos ocorrem através das chamadas regras de ponte. Ao utilizá-las, o contexto criado passa a receber a informação sobre a qual ele deverá executar uma busca por conhecimento e também passa a possuir a capacidade de emitir quais elementos estão disponíveis para serem pesquisados futuramente.

4.2.1 Captação de Conhecimento

A partir da existência de uma intenção de busca sobre determinado assunto, é possível executar uma regra de ponte capaz de iniciar o evento referente à busca de conteúdo no contexto ontológico. Para que o evento seja iniciado com sucesso, a regra de ponte espera um determinado input para que possa ser inicializado. No nosso caso, apenas um determinado tipo de intenção pode inicializar esta busca. Uma intenção que possua esta finalidade, deve ser especificada na sintaxe especificada, onde X se refere ao elemento ao qual se almeja obter informações e o predicado "questionAbout" demonstra para o agente o tipo de ação que deve ser executada para aquele elemento X, que no nosso caso se limita a buscar informações sobre X.

Uma vez que a regra de ponte cria esta comunicação entre o contexto ontológico e as intenções do agente, inicia-se o processo de busca. Por padrão do Sigon, ao utilizarmos regras de ponte, estamos criando um fluxo de informação entre dois contextos e, desta forma, o contexto receptor da informação precisa lidar com esta entrada. Como exibido anteriormente na imagem 7, o contexto, implementa uma série de funções, entre elas a função appendFact. Nesta função, o contexto recebe esta entrada, onde ocorrerá todo o processo de busca e persistência da informação. O processo de captação de conhecimento é executado através da utilização da estrutura fornecida pelo pacote semanticWeb, desenvolvido por Demarchi (2017), composto por três classes sendo elas: SparqlObject, SparqlResult e SparqlSearch. Ele nos fornece a habilidade de nos comunicarmos com ontologias remotas utilizando requisições SPARQL. Cada uma destas classes possui um papel na comunicação com a base de dados:

- SparqlObject : Primeiramente utilizamos a classe SparqlObject que possui o papel de armazenar os sujeitos, predicados, objetos e filtros que serão utilizados na query SPARQL.

- SparqlSearch: Recebe como entrada um SparqlObject para que com base nesta entrada construir e executar um query que resultará na obtenção de dados referentes aos parâmetros encontrados na classe SparqlObject.

- SparqlResult : Recebe o resultado obtido pela requisição, caso o resultado seja uma lista, o resultado obtido por SparqlSearch será uma lista de SparqlResult. Ela armazena o tipo de resultado obtido sendo ele um elemento literal ou uma referência para outro

elemento.

```
@Override
public void appendFact(String fact) {
    String subject = "";
    String content = getContent(fact);
    if(!checkedResource(content)){
        fact = stringToInputFormat(content);
        subject = "http://dbpedia.org/resource/"+fact;
        removeNewResourceStatus(content);
        for (String predicate : mappedPredicates) {
            List<SparqlResult> result = executeQuery("<"+subject+">", predicat
            for (SparqlResult sr : result){
                String newFact = "knowledge("+formatPredicate(predicate)+","+s
                apprendToProlog(newFact);
            }
        }
    }
}
```

Figura 8 – Função responsável pela obtenção de informações

A execução de requisições e obtenção de informações ocorrerá para todos os predicados que forem do interesse do desenvolvedor. Como houve um foco em obter informações mais relacionadas a localidades e informações geográficas, foi utilizada uma lista de predicados mais relacionados ao tema. Assim, o ciclo de requisições ocorrerá com base na lista de predicados criada. Caso nenhum dos predicados pré-selecionados seja encontrado no sujeito pesquisado, nenhum resultado será obtido.

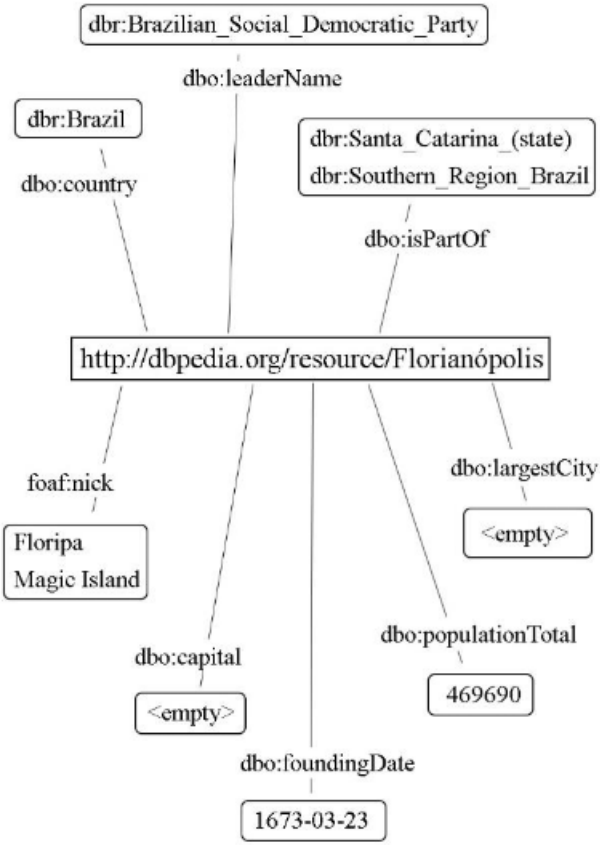

Figura 9 – Informações que podem ser obtidas de um recurso através de seus predicados

Fonte:(DEMARCHI, 2017)

Também faz parte do ciclo, a etapa de formatação das respostas obtidas pela requisição, que ocorre para que o agente não tenha problemas para persistir e compreender a informação. Esta formatação é composta do predicado, que representa o relacionamento entre o sujeito pesquisado e o objeto encontrado como sendo a outra ponta deste relacionamento, além de ser composto também pelos próprios sujeito e objeto envolvidos naquele relacionamento. Além de alterações relacionadas à estrutura da informação, também são removidos quaisquer espaços e caracteres especiais encontrados na sentença. O resultado final deste processo é exibido na imagem 10.

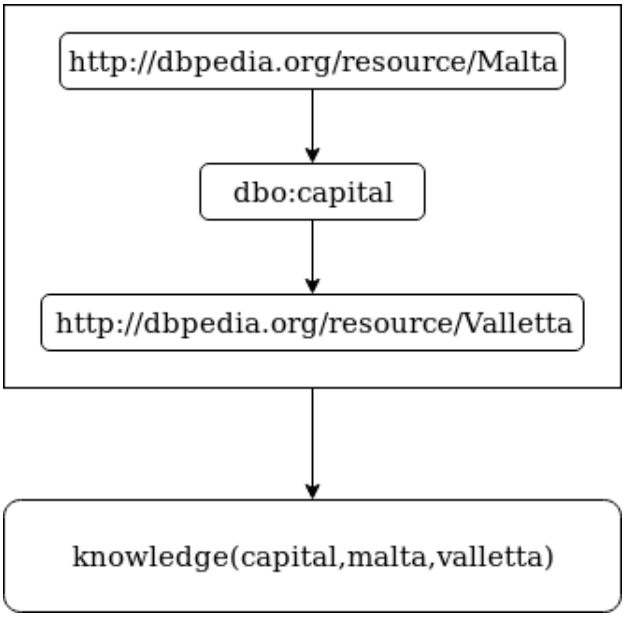

Figura 10 – Processo de composição de conhecimento do agente

Após o processo de formatação, este conhecimento é então adicionado ao contexto ontológico, como mostra a imagem 11. Este novo conhecimento passa por uma validação garantindo que um mesmo conhecimento não seja inserido mais de uma vez.

```
knowledge(type,malta,populatedPlace).

knowledge(populationTotal,malta,445426).

knowledge(country,valletta,malta).

knowledge(isPartOf,valletta,south_Eastern_Region).

knowledge(isPartOf,valletta,districts_of_Malta).

knowledge(lat,valletta,35.8978).

knowledge(long,valletta,14.5125).
```

Figura 11 – Amostra de lista de conhecimentos obtidos

4.2.2 Detecção de Conhecimento

Durante o ciclo de busca de informações que ocorre ao percorrer a lista de predicados, existe um ponto crucial para a continuidade de execução do agente. Ao obtermos o resultado de uma requisição com a utilização do suporte provido pelo pacote semanticWeb, podemos ter diferentes tipos de resultado, que devem receber seu devido suporte. Atualmente, são suportados dois tipos de resultados pela classe SparqlResult. Existem os resultados classificados como literais, que podem armazenar valores numéricos, datas ou textos, e também existem os resultados que podem ser classificados como um recurso da ontologia. Um recurso, ao contrário de um literal, não possui um valor como uma resposta fixa e, na realidade, se trata de uma referência a um elemento complexo da ontologia, que por sua vez possui seus próprios relacionamentos e literais. Ao nos depararmos com uma requisição da qual foi obtido como resposta um objeto do tipo recurso, isso nos diz que o sujeito do qual estamos obtendo informações possui um relacionamento com outro elemento elaborado. Este conceito acaba sendo importante para a compreensão do conceito de ontologia e dados ligados.

```
public String getObject(SparqlResult result){
    String object = "";
    if(result.getResourceResult()!=null){
        object = stringToOutputFormat(getResourceLabel(result.getResourceResult().getURI()));
        appendNewResource(object);
    }
    if(result.getLiteralResult()!=null){
        object = result.getLiteralResult().getString();
    }
    return stringToOutputFormat(object);
}
```

Figura 12 – Função responsável por obter valores de recursos e literais

Através do ciclo de obtenção de informações, parte do processo consiste no tratamento de cada um dos tipos de informação obtida. Devido ao resultado do tipo literal ser um valor fixo não existe necessidade de processamento adicional, além da formatação já citada anteriormente. Por outro lado, o processamento dos valores do tipo resource se tornam vitais para a continuação do ciclo de busca do agente. Por termos como objetivo a obtenção de informações, é necessário que seja obtido proveito dos relacionamentos existentes no sujeito ao qual estamos obtendo informações. Para que este recurso passe a fazer parte do ciclo de captação de informação, é necessário primeiramente adicionar à base de conhecimento do agente a tripla a qual ele faz parte e foi

originalmente encontrado. Por não se tratar de um literal, é necessário extrair a sua nomenclatura a partir de sua URI. Ao executarmos esta extração, podemos então concluir a persistência deste recurso.

Apesar da persistência da tripla ter sido executada com sucesso, ainda é necessário obter proveito das informações que podem ser alcançadas a partir deste recurso. Para isso criou-se o predicado "newResource(X)". A partir deste predicado, conseguimos sinalizar para o agente que determinado elemento se trata de um novo recurso para obtenção de dados, onde X representa o elemento. Desta forma, no próximo ciclo de execução, ele deve ser adicionado à lista de intenções para que futuramente ele venha a ser o sujeito da busca, para então fornecer suas informações ao agente. Este processo é executado pelo agente através da utilização da regra de ponte, o resultado da execução é exibido na imagem 13, ela garante a adição do novo conhecimento a lista de intenções. Assim garante-se que haja uma continuidade do processo.

```
newResource(valletta).

newResource(euro_sign).

newResource(maltese_language).

newResource(maltese_Sign_Language).

newResource(english_language).

newResource(birkirkara).

newResource(place).

newResource(location).

newResource(country).

newResource(musicalArtist).

newResource(populatedPlace).
```

Figura 13 – Lista de futuros conhecimentos a serem obtidos

4.2.3 Busca Dinâmica

Por este trabalho possuir como um de seus principais objetivos a obtenção de informações provenientes de ontologias remotas, se tornou importante que o agente fosse capaz de obter proveito das conexões

existentes entre os elementos pesquisados. Desta forma, podemos então não apenas obter informações relacionadas a determinado elemento como também, a partir de suas conexões, obter informações em relação ao contexto em que determinado elemento está situado. Para que isto seja possível, é necessário guiar o fluxo de execução do agente de forma que também seja executada a busca por informações relacionadas. Devido a este projeto ter sido desenvolvido com a utilização do Sigon, que por sua vez utiliza como base o modelo BDI, utilizamos suas intenções como meio de guiar o agente.

Uma vez que a busca é guiada pelas intenções do agente, podemos então manipulá-las através de regras de ponte. Atualmente este modelo funciona utilizando, como principais elementos, duas regras de ponte. Elas cooperam de forma que, juntas, criam um fluxo de informação entre o contexto ontológico e o contexto de intenções. O ciclo executado pelas regras de ponte se inicia com a adição de novos recursos para pesquisa no contexto de intenções. Estes recursos são oriundos do contexto ontológico. Em seguida, a próxima regra executa a pesquisa destes recursos e, como já citado anteriormente, pode gerar novos recursos para pesquisa. No entanto, neste momento eles são adicionados no contexto ontológico. Desta maneira, consegue-se então criar um ciclo dinâmico de captação de conhecimento.

Por estarmos lidando com ontologias, é natural que a exploração dos relacionamentos entre os recursos seja um caminho de duas vias. Assim, se torna possível que encontremos um mesmo recurso mais de uma vez. Ao utilizarmos um algoritmo dinâmico de obtenção de conhecimento precisamos nos ater à possibilidade de receber múltiplas vezes um mesmo input proveniente da exploração dos relacionamentos de um recurso. Para que não haja possibilidade da criação de um loop, devido a múltiplos recursos estarem indicando uns aos outros foi criado uma validação para a adição de novas intenções de busca.

Esta validação consiste na verificação dos elementos que foram até então adicionados ao contexto ontológico. Utilizamos, como base para a verificação, a checagem da existência de alguma tripla em que o elemento seja o sujeito juntamente de um predicado do tipo “type”. Caso o elemento não seja encontrado como sendo o sujeito em nenhuma tripla, a busca pode seguir normalmente e o predicado “newResouce” que continha este elemento é excluido, pois agora este deixa de ser um recurso novo. Esta mesma validação também é utilizada para a criação de sinalização de novos recursos através do predicado “newResouce”.

Ao final deste processo, devido a utilização de regras de ponte, ele tende a ser reiniciado de forma que permaneça buscando e consumindo

informação. O fluxo de comunicação entre contextos e a sequência da execução de etapas pode ser descrito pela imagem 14.

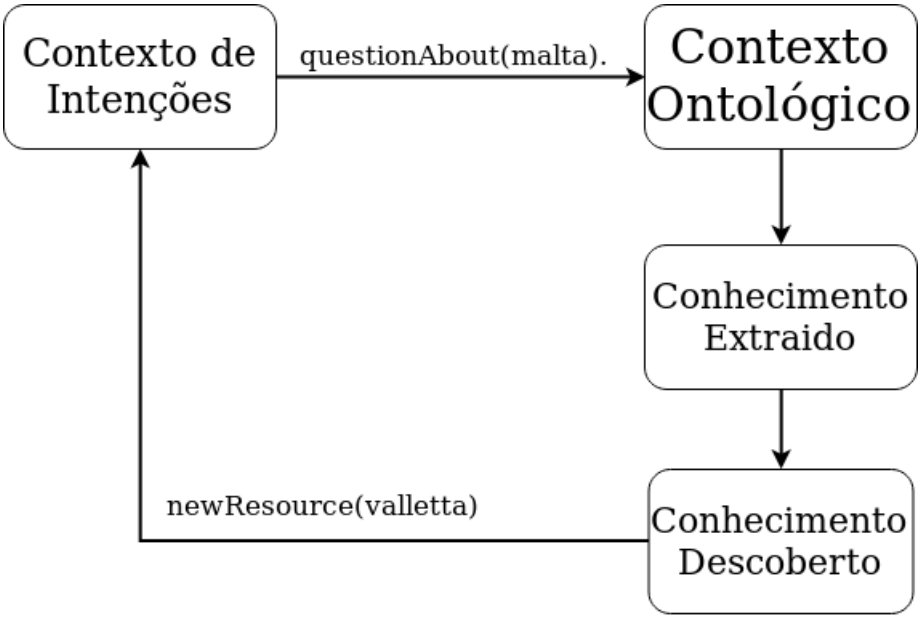

Figura 14 – Fluxo de excução do modelo proposto

A cada execução, do ciclo do agente Sigon, se obtém apenas o conhecimento relacionado ao recurso pesquisado. Considerando então uma pesquisa com apenas um nível de relacionamento, podemos considerar que na execução de um segundo ciclo de execução este nível seria aprofundado. Podemos considerar isto pois a partir das informações obtidas no ciclo anterior houve a possibilidade da execução de um próximo ciclo. Por se tratar de uma busca dinâmica, a pesquisa alcança um nível de profundidade dos relacionamentos igual ao número de ciclos de execução do agente.

4.3 COMPARAÇÃO COM PROJETOS CORRELATOS

Embora este projeto possua conexão direta com ambos os trabalhos relacionados, ainda assim foram necessárias novas abordagens e ajustes para que fosse possível obtermos o resultado final. Em relação

ao projeto desenvolvido por Demarchi (2017), houve a necessidade de algumas modificações, pois o elemento de entrada passou a ser a label do recurso ao invés do próprio recurso. Isso se dá devido à limitações existentes no framework Sigon. Devido a esta limitação, este agente trabalha apenas com literais. Por outro lado, ao recebermos o input diretamente de uma label, podemos construir o link da DBpedia referente àquele recurso, devido a existência de um padrão nos links de recursos. Desta forma acabamos por não necessitar de uma query para obtenção da label.

Outra diferença deste trabalho em relação ao projeto de Demarchi (2017) é sobre o registro de informações já utilizadas previamente. Por seu trabalho não possuir registros sobre quais elementos da ontologia já foram utilizados, existe a possibilidade da criação de um loop. Neste projeto, por trabalharmos com o controle do ciclo de obtenção de informação de maneira mais controlada, através de verificação do conteúdo já obtido, não existe a possibilidade da criação dinâmica de um processo de recursão entre múltiplos recursos.

Em relação ao Sigon, desenvolvido por Gelaim et al. (2019), também houveram algumas diferenças tanto em relação a funcionalidade de contextos quanto na utilidade que havia sido feita do framework até o momento. Por lidarmos com um fluxo constante de informações entre diferentes contextos houve a necessidade de assegurar que não existisse a possibilidade de que um contexto pudesse possuir dois fatos completamente idênticos, afim de possuir um melhor controle sobre quais elemento já haviam passado pelo processo de busca. A funcionalidade de verificação é algo que já existia no contexto de crenças. No entanto, esta verificação não era feita nos contextos de desejos e intenções e, para que este projeto pudesse ser executado de melhor maneira, foram feitas alterações nestes contextos para que passassem a utilizar esta verificação. Em relação à utilização do framework, através de pesquisas sobre projetos que já utilizaram este framework anteriormente, como o projeto de Freitas (2018) e também o projeto desenvolvido por Mello, Gelaim e Silveira (2019), assim como os testes deste framework disponíveis no github, percebeu-se que, o modelo desenvolvido neste projeto foi a primeira utilização do Sigon a se comunicar com um ambiente externo. A aplicação utiliza seus sensores e atuadores como um meio de se comunicar com um ambiente externo que no caso se trata de outra aplicação. Nas demais aplicações, atuadores eram utilizados como uma forma de inserir um novo input nos sensores para executar o próximo ciclo de processamento.

5 ESTUDO DE CASO

Como forma de demonstrar a aplicação do modelo proposto em um cenário prático, criou-se um estudo de caso. Neste estudo se foca mais nos aspectos da utilização do modelo e nos passos seguidos para que a construção deste estudo de caso se tornasse viável. Para a demonstração, optou-se pela criação de uma aplicação nos moldes de um quiz. Nela, o usuário receberá uma afirmação a qual ele deverá responder sobre a sua veracidade. Em relação à arquitetura da aplicação, ela consiste de um servidor que possui a tarefa de executar a troca de mensagens entre o agente e a interface. Desta forma, o agente consegue enviar suas afirmações à interface e a interface consegue enviar a resposta do usuário.

Para que seja possível a inicialização da interface da aplicação é necessário que o agente inicie seu processo de obtenção de informação previamente. Por haver este tipo de demanda da aplicação, o agente utilizado para este caso possui dois fluxos de execução. Cada um destes fluxos possui um objetivo diferente. O primeiro tipo utiliza os conceitos citados até o momento e foca na obtenção de informações assim como o envio de uma destas informações para a interface, onde será utilizada para alimentar a aplicação. O segundo ciclo foca no recebimento da resposta do usuário, checagem da resposta e envio do resultado da checagem da resposta para a interface. Para que esta diferenciação entre os fluxos de execução do agente se tornasse viável foi utilizado o sensor do contexto de comunicação. Assim, o agente consegue perceber inputs externos e agir de acordo com o esperado para cada tipo de input.

Um elemento fundamental para a execução dos diferentes fluxos é a utilização do contexto de comunicação. Seguindo a abordagem desenvolvida por Mello, Gelaim e Silveira (2019), utilizamos atuadores com o objetivo executar ações, de forma que elas respondam aos objetivos do agente através da utilização de regras de ponte. Para esta aplicação foram desenvolvidos dois atuadores, um para cada ciclo. O primeiro atuador, denominado BuildQuestion, pertence ao ciclo de obtenção de informações e tem como principal tarefa estabelecer uma comunicação entre o agente e o servidor, para que o servidor repasse a mensagem para a interface. Esta mensagem se trata de uma das triplas obtidas até o momento. Através do envio de uma tripla, conseguimos criar uma suposição que será utilizada como a questão a qual o usuário deverá responder. Para os fins da aplicação este atuador se limita a obter uma

questão relacionada à capital do local pesquisado.

```
public class BuildQuestion extends Actuator{

    @Override
    public void act(List<String> args) {
        try {
            connect(args);
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }

    public static void connect(List<String> args) throws URISyntaxException{

        String predicate = args.get(0);
        String subject = args.get(1);
        String object = args.get(2);

        Socket socket = IO.socket("http://127.0.0.1:3001");

        // Sending an object
        JSONObject obj = new JSONObject();
        try {
            obj.put("question", "knowledge("+predicate+","+subject+","+object+").");
        } catch (JSONException e) {
            e.printStackTrace();
        }
        socket.emit("sendQuestion", obj);
        socket.connect();
    }
}
```

Figura 15 – Implementação de atuador BuildQuestion

O próximo atuador, denominado ValidateAnswer, se encarrega de executar todo o processamento necessário do segundo ciclo. Este atuador inicia sua ação ao receber uma mensagem do usuário através de um sensor. Esta mensagem consiste na tripla que havia sido enviada anteriormente, juntamente da resposta do usuário em relação a veracidade da informação. Em sua execução, este atuador remonta a tripla e então verifica a existência desta informação. Como o agente possui apenas informações verdadeiras, sabemos que, caso a tripla exista em seus conhecimentos, esta se tratará de uma verdade na visão do agente. Após a busca interna, comparamos o resultado com a resposta dada pelo usuário. Caso haja uma discrepância entre o resultado e a resposta, o agente executa outra busca interna, desta vez com o intuito de retornar qual seria a resposta correta. Ainda, durante a execução do atuador, é enviado para o servidor o resultado da comparação entre a resposta do usuário e a resposta do agente.

Através da implementação deste estudo de caso foi possível abordar de maneira mais prática os conceitos elaborados até então, possuindo como principal elemento a utilização prática do contexto ontológico. Além da utilização do contexto também foi criado um cenário

onde a partir de sua utilização se criariam novas possibilidades, como a utilização das informações obtidas para a construção de outras aplicações como por exemplo um quiz.

```java
public class ValidateAnswer extends Actuator{

    @Override
    public void act(List<String> args) {
        testKnowledge(args);
    }

    @SuppressWarnings("static-access")
    public void testKnowledge(List<String> args){
        PrologEnvironment prologEnvironment = OntologicContextService.getInstance().getPrologEnvironment();
        SolveInfo info = null;
        String predicate = args.get(0);
        String subject = args.get(1);
        String object = args.get(2);
        Boolean answer = Boolean.parseBoolean(args.get(3));
        try {
            info = prologEnvironment.getEngine().solve("knowledge("+predicate+","+subject+","+object+").");
            if(info.isSuccess() != answer){
                info = prologEnvironment.getEngine().solve("knowledge("+predicate+","+subject+",X).");
                System.out.println("Resposta certa: "+info.getBindingVars().get(0).getLink().getTerm());
            }else {
                System.out.println("Resposta Correta");
            }
            connect(info.isSuccess() == answer);
        } catch (MalformedGoalException | URISyntaxException | NoSolutionException e) {
            e.printStackTrace();
        }
    }
```

Figura 16 – Implementação de atuador ValidateAnswer

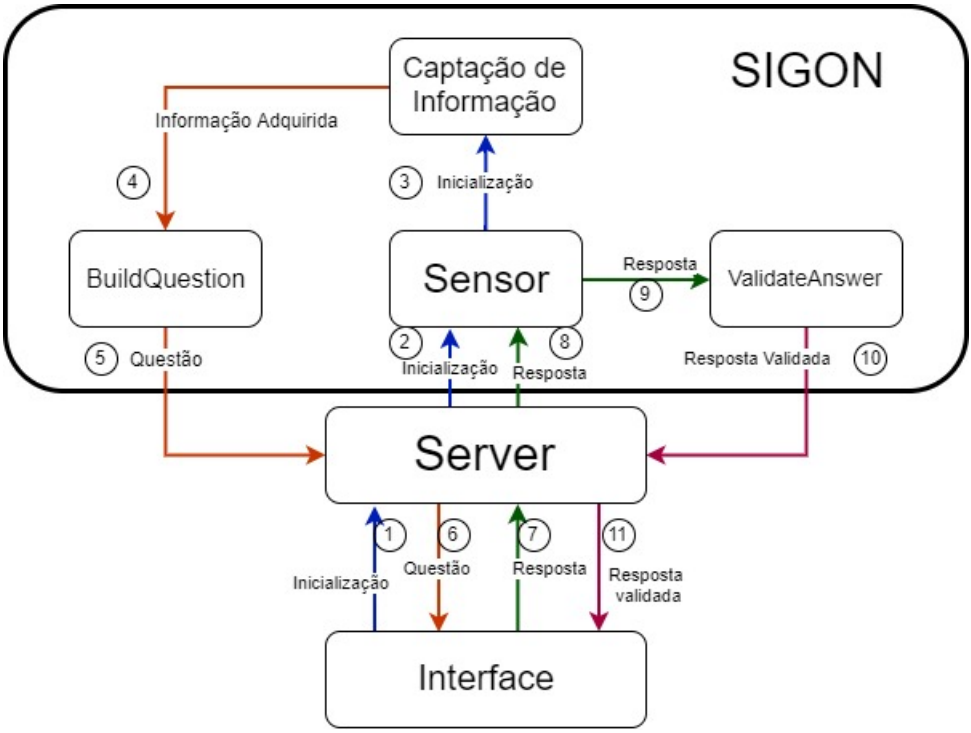

Figura 17 – Fluxo de execução do estudo de caso

Figura 18 – Tela da aplicação

6 CONSIDERAÇÕES FINAIS

Em sua problemática este trabalho estabelece como um dos seus principais pontos a serem desenvolvidos a criação de um modelo de agente que possua a capacidade de obter informações a partir de ontologias remotas, utilizando uma estrutura de agente multi-contexto. Durante a argumentação, foi comentado sobre as vantagens ao se utilizar esta estrutura de agente, pois uma de suas principais vantagens seria a possibilidade de criarmos contextos que contenham suas próprias lógicas internas. Através da definição deste objetivo geral foram criados objetivos específicos que contribuiram para o desenvolvimento deste trabalho. A pesquisa em relação ao estado da arte pode ser definido como um dos objetivos específicos mais importantes pois, a partir, deste passou-se a ter um melhor conhecimento do campo de utilização de agentes juntamente a exploração de ontologias. Ao obter este conhecimento, foi possível criar a base teórica necessária para o desenvolvimento deste projeto.

Como fruto deste período de pesquisa e desenvolvimento foi possível concluir com sucesso as metas fixadas no início deste projeto. Ao fim desse período conseguiu-se obter como principais resultados um modelo de agente funcional que possui a capacidade de obter informações a partir de ontologias e também uma aplicação capaz de utilizar este modelo de forma integrada à aplicação. Através destas aplicações desenvolvidas foi possível também explorar o potencial das regras de ponte criadas através do Sigon. Outro ponto importante em relação à utilização do Sigon seria a utilidade atribuída aos sensores e atuadores no estudo de caso. A partir destes resultados obtivemos não apenas um novo contexto para ser utilizado juntamente ao Sigon, mas também expandimos o campo de possibilidades deste modelo ao adicionarmos uma funcionalidade capaz de trabalhar de maneira dinâmica.

Desta forma, este trabalho pode vir a contribuir para o campo de pesquisa relacionado à agentes multi-contexto e utilização de ontologias remotas. Este trabalho também poderá ser utilizado como base para uma possível continuação no mesmo segmento de utilização de dados externos ao agente.

7 TRABALHOS FUTUROS

Ao longo do desenvolvimento foram encontradas situações as quais deixam espaço para futuros trabalhos, por este trabalho lidar com informações obtidas da internet é comum que muitos textos possuam acentos ou caracteres especiais. No entanto hoje em dia o Sigon não possui suporte para este tipo de caracteres em sua sintaxe, assim como suporte para Strings em geral. Para que fosse possível superar esta dificuldade o contexto ontológico executa uma formatação em seus conteúdos para que fiquem de acordo com o permitido na sintaxe, porém ao executar essa formatação existe a possibilidade de não ser encontrado nenhuma referência para o conteúdo formatado na DBpedia.

Um segundo ponto seria em relação a continuidade da busca por conteúdo. Na implementação atual, não existe um verificação interna do contexto de forma que perceba-se o momento em que não existem novas informações sendo obtidas. O algoritmo foca em guardar o conhecimento obtido e realimentar seu próprio ciclo, a partir dos recursos encontrados, porém existe a possibilidade de não haverem mais novos recursos ou, em alguns casos não podem ser obtidos devido a formatação executada no conteúdo.

Atualmente não podemos afirmar que ocorrem falhas na execução de uma intenção, caso sua busca retorne nenhum resultado, afinal a pesquisa foi concluida. Porem uma alternativa, para uma implementação futura, seria a criação de planos de contingência nestes casos. Desta forma se evita que não haja obtenção de nenhuma informação a partir do elemento pesquisado.

Até o momento o algoritmo criado utiliza os ciclos do Sigon para criar os ciclos de busca. No entanto para futuros trabalhos algumas abordagens alternativas são interessantes. Como por exemplo, ciclos de busca configuráveis para a obtenção de um determinado número de níveis de relacionamento por busca.

REFERÊNCIAS

ARENAS, M.; PÉREZ, J. Querying semantic web data with sparql. In: *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York, NY, USA: ACM, 2011. (PODS '11), p. 305–316. ISBN 978-1-4503-0660-7. <http://doi.acm.org/10.1145/1989284.1989312>.

BECHHOFER, S. et al. *OWL Web Ontology Language Reference*. 2004. <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. *Scientific American*, v. 284, p. 34–43, 2001.

BIZER, C.; HEATH, T.; BERNERS-LEE, T. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, v. 5, n. 3, p. 1–22, 2009. <https://eprints.soton.ac.uk/271285/>.

BIZER, C. et al. Linked data on the web (ldow2008). *Proceedings of the 17th international conference on World Wide Web (WWW '08)*, 01 2008.

BORDINI, R. H.; HüBNER, J. F.; WOOLDRIDGE, M. *Programming Multi–Agent Systems in AgentSpeak using Jason*. [S.l.]: Wiley-Blackwell, 2007. 292 p.

BRATMAN, M. Intention, plans, and practical reason. Harvard University Press, 1987.

CAMPOS, D. de. Representação de dados semânticos em agentes bdi. 2014. 72 f. *Dissertação (Mestrado) - Curso de Programa de Pós-graduação em Ciência da Computação*, Universidade Federal de Santa Catarina, Florianópolis, 2014.

CASALI, A. *On Intentional and Social Agents with Graded Attitudes*. [S.l.]: Universitat de Girona, 2008. 217 p. ISBN 978-8-4692-1448-0.

CASALI, A.; GODO, L.; SIERRA, C. Graded bdi models for agent architectures. *International Journal of Human and Computer Studies*, Springer, Berlin, Heidelberg, v. 3487, p. 126–143, 2005.

DEMARCHI, F. A integração de agentes com bases ontológicas heterogêneas aplicadas a web semântica. Florianópolis, 2017.

DIKENELLI, O.; ERDUR, R. C.; GUMUS, O. Seagent:a platform for developing semantic web based multi agent systems. In: PECHOUCEK, M.; STEINER, D.; THOMPSON, S. (Ed.). *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. [S.l.]: Acm, 2005. p. 1271–1272. ISBN 1-59593-093-0.

FREITAS, A. et al. Integrating ontologies with multi-agent systems through cartago artifacts. In: *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. Florianópolis: Ieee, 2015. p. 143 – 150. ISBN 978-1-4673-9618-9.

FREITAS, G. S. de. Um modelo de raciocínio sobre sensores para agentes sigon em ambientes de realidade virtual. Florianópolis, 2018.

GELAIM, T. Ângelo et al. Sigon: A multi-context system framework for intelligent agents. *Expert Systems with Applications*, v. 119, p. 51–60, 2019.

GUARINO, N. Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human and Computer Studies*, v. 43, p. 625–640, 1995.

HERZIG, A. et al. Bdi logics for bdi architectures: Old problems, new perspectives. *KI - Künstliche Intelligenz*, Springer Berlin Heidelberg, v. 31, p. 73–83, 2017.

HüBNER, J. F.; BORDINI, R. H.; VIEIRA, R. Introdução ao desenvolvimento de sistemas multiagentes com jason. 2014.

JENSEN, R.; BUCH, B. *Multi-Agent Programming in Jason*. Richard Petersens Plads, Building 324, DK-2800 Kgs. Lyngby, Denmark, compute@compute.dtu.dk: Technical University of Denmark, Department of Applied Mathematics and Computer Science, 2014. DTU supervisor: Jørgen Villadsen, jovi@dtu.dk, DTU Compute. <http://www.compute.dtu.dk/English.aspx>.

KLAPISCAK, T.; BORDINI, R. H. Jasdl: A practical programming approach combining agent and semantic web technologies. In: BALDONI, M. et al. (Ed.). *Declarative Agent Languages and Technologies VI*. Berlin: Springer, 2008. p. 91–110. ISBN 978-3-540-93919-1.

MASCARDI, V. et al. Cool-agentspeak: Enhancing agentspeak-dl agents with plan exchange and ontology services. In: *WI-IAT '11*

Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 01. [S.l.]: IEEE, 2011. p. 109–116. ISBN 978-0-7695-4513-4.

MELLO, R. R. P. de; GELAIM, T. Â.; SILVEIRA, R. A. Negotiating agents: A model based on bdi architecture and multi-context systems using aspiration adaptation theory as a negotiation strategy. In: BAROLLI, L. et al. (Ed.). *Complex, Intelligent, and Software Intensive Systems*. Cham: Springer International Publishing, 2019. p. 351–362. ISBN 978-3-319-93659-8.

MOREIRA Álvaro F. et al. Agent-oriented programming with underlying ontological reasoning. In: BALDONI, M. et al. (Ed.). *Declarative Agent Languages and Technologies III*. Berlin: Springer-verlag, 2006. p. 155–170. ISBN 978-3-540-33106-3.

O'HARE, G. M. P.; JENNINGS, N. R. (Ed.). *Foundations of Distributed Artificial Intelligence*. New York, NY, USA: John Wiley & Sons, Inc., 1996. ISBN 0-471-006750.

OMEROVIC, S.; MILUTINOVIC, V.; TOMAZIC, S. Concepts, ontologies, and knowledge representation. Slovenian Research Agency, p. 42, 2001.

OSSENBRUGGEN, J. van; HARDMAN, L.; RUTLEDGE, L. Hypermedia and the semantic web: A research agenda. *Journal of Digital Information*, v. 3, n. 1, 2006. ISSN 1368-7506. <https://journals.tdl.org/jodi/index.php/jodi/article/view/78>.

PARSONS, S. et al. Agent specification using multi-context systems. In: *Lecture Notes in Computer Science*. [S.l.]: Springer, Berlin, Heidelberg, 2002. p. 205–226.

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.

RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence a Modern Approach*. New Jersey: Alan Apt, 1995. 932 p.

STAAB, S.; STUDER, R. (Ed.). *Handbook on Ontologies*. [S.l.]: Springer-Verlag Berlin Heidelberg, 2009. 811 p. ISBN 978-3-540-92673-3.

W3C. *OWL Web Ontology Language Overview*. 2004. <https://www.w3.org/TR/2004/REC-owl-features-20040210/s1.2>.

W3C. *SPARQL Query Language for RDF*. 2004. <https://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/conventions>.

W3C. *OWL 2 Web Ontology Language Document Overview*. 2012. <https://www.w3.org/TR/owl2-overview/>.

WOOLDRIDGE, M. *Reasoning about Rational Agents*. London: The MIT Press, 2000.

.1 APÊNDICE - ARTIGO

Modelo de Integração Entre Agentes BDI Baseados em Sistemas Multi-contexto e Ontologias Públicas para Revisão de Crenças

Leon Luiz Vargas Daros[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil

leondaros@gmail.com

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

Resumo. *Este meta-artigo descreve o estilo a ser usado na confecção de artigos e resumos de artigos para publicação nos anais das conferências organizadas pela SBC. É solicitada a escrita de resumo e abstract apenas para os artigos escritos em português. Artigos em inglês deverão apresentar apenas abstract. Nos dois casos, o autor deve tomar cuidado para que o resumo (e o abstract) não ultrapassem 10 linhas cada, sendo que ambos devem estar na primeira página do artigo.*

1. Introdução

Hoje em dia vivemos constantemente recebendo e enviando informação através da rede mundial de computadores (WEB). Com o passar do tempo, todo este conteúdo se acumula na internet de maneira desestruturada, dificultando a coleta destas informações. Neste âmbito surge, como proposta para equacionar o problema, a chamada Web Semântica, idealizado por \citeonline{BERNERS-LEE}. A Web semântica propõe uma maneira de trazer ordem e uma estrutura aos conteúdos das páginas, tornando a busca, prospecção e extração de dados mais fácil de ser executada por agentes inteligentes de software, já que a WEB convencional foi concebida, primordialmente para uso por agentes humanos.

Ao estruturar as informações disponíveis, estes dados passam a se tornar mais próximos de uma representação de conhecimento baseado no conceito de ontologia. O conceito de ontologia se baseia na organização formal de uma informação, através da descrição de conceitos, em um determinado domínio de conhecimento, e sua relação com outros elementos \cite{Guarino}. Através da utilização de ontologias, a web semântica busca alcançar seu objetivo de fornecer um meio mais legível de disponibilizar a informação para que agentes inteligentes possam analisar e extrair estes dados.

Estes agentes, segundo \citeonline{RUSSEL}, podem ser definidos como sistemas capazes de obter percepções em relação ao ambiente e então executar ações de resposta. Na perceção de Michael \citeonline{Wooldridge} agentes são sistemas computacionais capazes de agir de maneira autônoma, de forma que alcancem seus

objetivos e sejam capazes de decidirem sozinhos como agir, de acordo com uma dada situação.

O modelo de agentes conhecido como BDI (Belief, Desire, Intentions) seria capaz de compreender, de melhor maneira, a forma de representação de conhecimento utilizado na web semântica, por se tratar de uma abordagem de construção de agentes de cunho predominantemente cognitivo. O modelo BDI foi proposto pelo filósofo Michael \citeonline{BRATMAN} e tem como foco o raciocínio prático humano, baseado nos conceitos de crenças, desejos e intenções (Belief-Desire-Intention) para formar uma linha de pensamento para o mecanismo de deliberação e ação do agente . Segundo Michael \citeonline{Wooldridge} o modelo BDI é particularmente interessante porque combina três componentes diferentes: componente filosófico, componente de arquitetura de software e componente lógico.

Entretanto, a utilização de agentes para o consumo destas bases de conhecimentos pode se mostrar complexa. Estas bases podem possuir estruturas diferentes umas das outras, dificultando não apenas a navegação do agente como a estruturação de suas bases de conhecimento internas. Conforme será discutido mais adiante, neste trabalho, o problema de integração das crenças internas dos agentes com bases públicas de conhecimento ainda possui pontos em aberto que apresentam desafios para pesquisas neste campo de estudo.

Em contrapartida a estes problemas, devido a complexidade do agente, existem outras abordagens, entre elas a multi-contexto. Segundo \citeonline{CASALIArticle}, ao separar a lógica de uma agente em contextos, obtém-se escalabilidade, eficiência e inclusive diminui a complexidade através da separação de responsabilidades em contextos. Para isso, ao longo deste projeto, o SIGON\cite{GELAIM}, um framework focado na criação de agentes multi-contexto foi utilizado.

2. Trabalhos Relacionados

Para esta pesquisa foram utilizados como principais trabalhos relacionados pesquisas resultantes de trabalhos com agentes inteligentes. Os frameworks e funcionalidades criados a partir destas pesquisas servem como base para o desenvolvimento deste projeto.

2.1 UTILIZAÇÃO DE AGENTES COM ONTOLOGIAS REMOTAS PARA A PRODUÇÃO DE CONTEÚDO SIGNIFICANTE A PARTIR DE INFORMAÇÕES DISPONÍVEIS NA WEB SEMÂNTICA

O trabalho desenvolvido por \citeonline{DEMARCHI} trata-se da utilização de agentes BDI para o acesso de bases de conhecimento remotas, sendo estes agentes implementados através da plataforma Jason. A partir da comunicação destes agentes Jason é possível ao agente adquirir conhecimento através da sua capacidade de comunicação com a web semântica. Podemos considerar estas informações adquiridas pelos agentes como conhecimento uma vez que estes dados captados possuem ligações a outros termos, criando desta forma uma rede de associações. Para que um agente consiga acessar estes dados foi utilizada a abordagem desenvolvida por \ citeonline{Campos}, desta forma se torna possível o com que crenças possam ser representadas através de URIs, links externos para a base de dados remota.

Para que seja possível a aplicação deste modelo, o autor efetuou alterações no código do Jason de forma que torna-se possível a conexão dos agentes à base de

conhecimento chamada DBPedia. A abordagem seguida é a de adição de eventos internos ao código do interpretador, de forma que não se afete o ciclo de raciocínio. Através destes eventos internos o agente passa a ser capaz de utilizar consultas SPARQL. Trata-se de uma forma de busca de informações nas ontologias remotas. Os resultados obtidos através destas consultas inicialmente atualizam a base de crenças do agente. Em seguida, o agente utiliza a sua lista de planos, que necessita fundamentalmente da utilização da ações internas para controlar o acesso à web semântica.

2.2. SIGON: A MULTI-CONTEXT SYSTEM FRAMEWORK FOR INTELLIGENT AGENTS

A partir do conceito de agentes multi-contexto, \citeonline{GELAIM} desenvolveu um framework para o desenvolvimento deste modelo de agentes. O Sigon, como é chamado, utiliza como principais fontes do seu desenvolvimento os trabalhos desenvolvidos por \citeonline{Parsons} e \citeonline{CASALIArticle}, desta forma é possível a implementação das principais características de ambos os trabalhos no framework desenvolvido. Por estes estudos possuirem como base o modelo de agente BDI, este framework também possui como foco este modelo.

Segundo \citeonline{GELAIM}, o modelo de agente Sigon, baseando-se no modelo de \citeonline{Parsons}, possui um contexto de comunicação. Este contexto se torna obrigatório quando se leva em conta a necessidade da interação entre o agente e o seu ambiente. Para que haja esta interação, este contexto possui uma série de sensores e atuadores. Para que haja de fato um interação entre o agente e o seu ambiente, é necessário que haja ao menos um sensor e um atuador não havendo uma quantidade limite para sua utilização. A integração entre sensores e o contexto de comunicação possui um papel fundamental. Este contexto, ao receber as informações providas pelo sensor, executa suas regras de ponte, o que representa o ciclo de raciocínio. O Atuador por sua vez é responsável por alterar o ambiente e possui relação direta com o contexto de planejamento pois é utilizado para a execução de ações. Ambos os componentes do contexto de comunicação de seus identificadores e ação a ser executada.

No modelo Sigon, descrito até o momento, percebe-se que não houve imposição de regras de ponte. Desta maneira o desenvolvedor se torna mais livre para ditar o comportamento do seu agente.

3. Modelo Proposto

Este projeto tem como um dos seus principais objetivos a idealização e implementação de um modelo de agente multi-contexto capaz de buscar informações na internet de forma que possa atender à sua intenção de obter conhecimento sobre um determinado assunto. Para isto, este trabalho visa ampliar as funcionalidades de um modelo de agente multi-contexto. Por se tratar de um agente multi-contexto, isolaremos esta funcionalidade em um contexto específico.

Para que seja possível obter os dados que alimentarão o agente, este contexto responderá às intenções do mesmo, de forma que, a partir das intenções, o contexto compreenda que existe uma necessidade de buscar sobre aquele assunto e, então, adicionar este conhecimento obtido ao agente. A fonte de dados utilizada se trata da DBpedia, uma base de dados remota categorizada como ontologia. Esta ontologia, além de permitir o acesso aos dados requisitados, também fornece uma referência aos dados que possuam algum tipo de relacionamento com o elemento pesquisado. Assim, o

agente consegue não apenas obter conhecimento sobre o assunto pesquisado, como também passa a possuir consciência da existência de outros assuntos.

O processo de desenvolvimento utilizou o modelo de agente Sigon, desenvolvido por \citeonline{GELAIM}, para implementar o contexto ontológico. Também foi utilizado, como ponte de comunicação entre o agente e a ontologia, o pacote semanticWeb desenvolvido por \citeonline{DEMARCHI} em sua dissertação. Ao unir estes elementos se visa criar um contexto capaz de receber as demandas de conhecimento do agente e a partir da comunicação com a base de dados remota obter conhecimento desejado.

Ao longo deste capítulo serão exibidos os conceitos abordados durante o período de desenvolvimento.

3.1. Adição de Contexto

Atualmente o Sigon disponibiliza para o desenvolvedor uma estrutura capaz de suportar a criação de novos contextos. Podemos então utilizar esta estrutura existente para desenvolver uma nova funcionalidade sem que haja necessidade de alterar a arquitetura existente. Por estarmos lidando com uma arquitetura de agentes multi-contexto, nos é disponibilizada a estrutura necessária para criar um novo contexto que não possua relação com os contextos já existentes. Assim, é possível adicionar um novo contexto que possua sua própria lógica sem que haja necessidade de alterar outros elementos do agente.

Podemos então considerar um paralelo entre a adição deste novo contexto na estrutura do agente e a adição de um plugin à uma plataforma com o objetivo de trazer novas características a um ambiente já existente. Baseando-se neste conceito, o projeto desenvolvido visou desenvolver novas funcionalidades, de forma que elas poderiam ser adicionadas futuramente a um agente, de acordo com a necessidade do desenvolvedor. Desta forma, consegue-se trazer um maior dinamismo para as capacidades do agente.

3.2. Ciclo de Aprendizado

Para o desenvolvimento deste projeto está sendo usado o modelo de agente BDI. Este tipo de agente possui um ciclo de informações que, por sua vez, é guiado pelas suas crenças, desejos e intenções. Desta forma, é necessário que o contexto ontológico interaja com estes outros contextos para, assim, passar a fazer parte do ciclo de raciocínio do agente. Estas interações entre contextos ocorrem através das chamadas regras de ponte. Ao utilizá-las, o contexto criado passa a receber a informação sobre a qual ele deverá executar uma busca por conhecimento e também passa a possuir a capacidade de emitir quais elementos estão disponíveis para serem pesquisados futuramente.

3.2.1. Captação de Conhecimento

A partir da existência de uma intenção de busca sobre determinado assunto, é possível executar uma regra de ponte capaz de iniciar o evento referente à busca de conteúdo no contexto ontológico. Para que o evento seja iniciado com sucesso, a regra de ponte espera um determinado input para que possa ser inicializado. No nosso caso, apenas um determinado tipo de intenção pode inicializar esta busca. Uma intenção que possua esta finalidade, deve ser especificada na sintaxe especificada, onde X se refere ao elemento ao qual se almeja obter informações e o predicado “questionAbout” demonstra para o

agente o tipo de ação que deve ser executada para aquele elemento X, que no nosso caso se limita a buscar informações sobre X.

Uma vez que a regra de ponte cria esta comunicação entre o contexto ontológico e as intenções do agente, inicia-se o processo de busca. Por padrão do Sigon, ao utilizarmos regras de ponte, estamos criando um fluxo de informação entre dois contextos e, desta forma, o contexto receptor da informação precisa lidar com esta entrada. Como exibido anteriormente na imagem ~\ref{figContextService}, o contexto, implementa uma série de funções, entre elas a função appendFact. Nesta função, o contexto recebe esta entrada, onde ocorrerá todo o processo de busca e persistência da informação. O processo de captação de conhecimento é executado através da utilização da estrutura fornecida pelo pacote semanticWeb, desenvolvido por \ citeonline{DEMARCHI}, composto por três classes sendo elas: SparqlObject, SparqlResult e SparqlSearch. Ele nos fornece a habilidade de nos comunicarmos com ontologias remotas utilizando requisições SPARQL.

Também faz parte do ciclo, a etapa de formatação das respostas obtidas pela requisição, que ocorre para que o agente não tenha problemas para persistir e compreender a informação. Esta formatação é composta do predicado, que representa o relacionamento entre o sujeito pesquisado e o objeto encontrado como sendo a outra ponta deste relacionamento, além de ser composto também pelos próprios sujeito e objeto envolvidos naquele relacionamento. O resultado final deste processo é exibido na imagem abaixo.

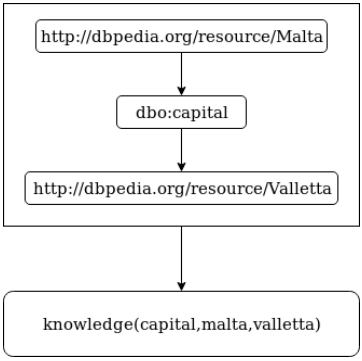

Figura 1. Processo de compisição de conhecimento do agente

3.2.2. Detecção de Conhecimento

Durante o ciclo de busca de informações que ocorre ao percorrer a lista de predicados, existe um ponto crucial para a continuidade de execução do agente. Ao obtermos o resultado de uma requisição com a utilização do suporte provido pelo pacote semanticWeb, podemos ter diferentes tipos de resultado, que devem receber seu devido suporte. Atualmente, são suportados dois tipos de resultados pela classe SparqlResult. Existem os resultados classificados como literais, que podem armazenar valores numéricos, datas ou textos, e também existem os resultados que podem ser classificados como um recurso da ontologia. Um recurso, ao contrário de um literal, não possui um valor como uma resposta fixa e, na realidade, se trata de uma referência a um elemento

complexo da ontologia, que por sua vez possui seus próprios relacionamentos e literais. Ao nos depararmos com uma requisição da qual foi obtido como resposta um objeto do tipo recurso, isso nos diz que o sujeito do qual estamos obtendo informações possui um relacionamento com outro elemento elaborado. Este conceito acaba sendo importante para a compreensão do conceito de ontologia e dados ligados.

Através do ciclo de obtenção de informações, parte do processo consiste no tratamento de cada um dos tipos de informação obtida. Devido ao resultado do tipo literal ser um valor fixo não existe necessidade de processamento adicional, além da formatação já citada anteriormente. Por outro lado, o processamento dos valores do tipo resource se tornam vitais para a continuação do ciclo de busca do agente. Por termos como objetivo a obtenção de informações, é necessário que seja obtido proveito dos relacionamentos existentes no sujeito ao qual estamos obtendo informações. Para que este recurso passe a fazer parte do ciclo de captação de informação, é necessário primeiramente adicionar à base de conhecimento do agente a tripla a qual ele faz parte e foi originalmente encontrado. Por não se tratar de um literal, é necessário extrair a sua nomenclatura a partir de sua URI. Ao executarmos esta extração, podemos então concluir a persistência deste recurso.

Apesar da persistência da tripla ter sido executada com sucesso, ainda é necessário obter proveito das informações que podem ser alcançadas a partir deste recurso. Para isso criou-se o predicado “newResource(X)”. A partir deste predicado, conseguimos sinalizar para o agente que determinado elemento se trata de um novo recurso para obtenção de dados, onde X representa o elemento. Desta forma, no próximo ciclo de execução, ele deve ser adicionado à lista de intenções para que futuramente ele venha a ser o sujeito da busca, para então fornecer suas informações ao agente. Este processo é executado pelo agente através da utilização da regra de ponte, ela garante a adição do novo conhecimento a lista de intenções. Assim garante-se que haja uma continuidade do processo.

3.2.3 Busca Dinâmica

Por este trabalho possuir como um de seus principais objetivos a obtenção de informações provenientes de ontologias remotas, se tornou importante que o agente fosse capaz de obter proveito das conexões existentes entre os elementos pesquisados. Desta forma, podemos então não apenas obter informações relacionadas a determinado elemento como também, a partir de suas conexões, obter informações em relação ao contexto em que determinado elemento está situado. Para que isto seja possível, é necessário guiar o fluxo de execução do agente de forma que também seja executada a busca por informações relacionadas. Devido a este projeto ter sido desenvolvido com a utilização do Sigon, que por sua vez utiliza como base o modelo BDI, utilizamos suas intenções como meio de guiar o agente.

Uma vez que a busca é guiada pelas intenções do agente, podemos então manipulá-las através de regras de ponte. Atualmente este modelo funciona utilizando, como principais elementos, duas regras de ponte. Elas cooperam de forma que, juntas, criam um fluxo de informação entre o contexto ontológico e o contexto de intenções. O ciclo executado pelas regras de ponte se inicia com a adição de novos recursos para pesquisa no contexto de intenções. Estes recursos são oriundos do contexto ontológico. Em seguida, a próxima regra executa a pesquisa destes recursos e, como já citado anteriormente, pode gerar novos recursos para pesquisa. No entanto, neste momento eles

são adicionados no contexto ontológico. Desta maneira, consegue-se então criar um ciclo dinâmico de captação de conhecimento.

Por estarmos lidando com ontologias, é natural que a exploração dos relacionamentos entre os recursos seja um caminho de duas vias. Assim, se torna possível que encontremos um mesmo recurso mais de uma vez. Ao utilizarmos um algoritmo dinâmico de obtenção de conhecimento precisamos nos ater à possibilidade de receber múltiplas vezes um mesmo input proveniente da exploração dos relacionamentos de um recurso. Para que não haja possibilidade da criação de um loop, devido a múltiplos recursos estarem indicando uns aos outros foi criado uma validação para a adição de novas intenções de busca.

Ao final deste processo, devido a utilização de regras de ponte, ele tende a ser reiniciado de forma que permaneça buscando e consumindo informação. O fluxo de comunicação entre contextos e a sequência da execução de etapas pode ser descrito pela imagem abaixo.

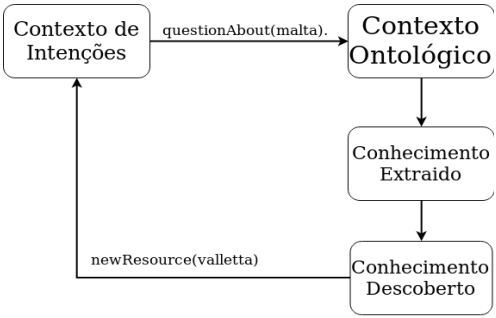

Figura 2. Fluxo de execução do modelo proposto

A cada execução, do ciclo do agente Sigon, se obtém apenas o conhecimento relacionado ao recurso pesquisado. Considerando então uma pesquisa com apenas um nível de relacionamento, podemos considerar que na execução de um segundo ciclo de execução este nível seria aprofundado. Podemos considerar isto pois a partir das informações obtidas no ciclo anterior houve a possibilidade da execução de um próximo ciclo. Por se tratar de uma busca dinâmica, a pesquisa alcança um nível de profundidade dos relacionamentos igual ao número de ciclos de execução do agente.

4. Estudo de Caso

Como forma de demonstrar a aplicação do modelo proposto em um cenário prático, criou-se um estudo de caso. Neste estudo se foca mais nos aspectos da utilização do modelo e nos passos seguidos para que a construção deste estudo de caso se tornasse viável. Para a demonstração, optou-se pela criação de uma aplicação nos moldes de um quiz. Nela, o usuário receberá uma afirmação a qual ele deverá responder sobre a sua veracidade. Em relação à arquitetura da aplicação, ela consiste de um servidor que possui a tarefa de executar a troca de mensagens entre o agente e a interface. Desta forma, o agente consegue enviar suas afirmações à interface e a interface consegue enviar a resposta do usuário.

Para que seja possível a inicialização da interface da aplicação é necessário que o agente inicie seu processo de obtenção de informação previamente. Por haver este tipo de demanda da aplicação, o agente utilizado para este caso possui dois fluxos de execução. Cada um destes fluxos possui um objetivo diferente. O primeiro tipo utiliza os conceitos citados até o momento e foca na obtenção de informações assim como o envio de uma destas informações para a interface, onde será utilizada para alimentar a aplicação. O segundo ciclo foca no recebimento da resposta do usuário, checagem da resposta e envio do resultado da checagem da resposta para a interface. Para que esta diferenciação entre os fluxos de execução do agente se tornasse viável foi utilizado o sensor do contexto de comunicação. Assim, o agente consegue perceber inputs externos e agir de acordo com o esperado para cada tipo de input.

Um elemento fundamental para a execução dos diferentes fluxos é a utilização do contexto de comunicação. Seguindo a abordagem desenvolvida por \ citeonline{10.1007/978-3-319-93659-8_31}, utilizamos atuadores com o objetivo executar ações, de forma que elas respondam aos objetivos do agente através da utilização de regras de ponte. Para esta aplicação foram desenvolvidos dois atuadores, um para cada ciclo. O primeiro atuador, denominado BuildQuestion, pertence ao ciclo de obtenção de informações e tem como principal tarefa estabelecer uma comunicação entre o agente e o servidor, para que o servidor repasse a mensagem para a interface. Esta mensagem se trata de uma das triplas obtidas até o momento. Através do envio de uma tripla, conseguimos criar uma suposição que será utilizada como a questão a qual o usuário deverá responder. Para os fins da aplicação este atuador se limita a obter uma questão relacionada à capital do local pesquisado.

O próximo atuador, denominado ValidateAnswer, se encarrega de executar todo o processamento necessário do segundo ciclo. Este atuador inicia sua ação ao receber uma mensagem do usuário através de um sensor. Esta mensagem consiste na tripla que havia sido enviada anteriormente, juntamente da resposta do usuário em relação a veracidade da informação. Em sua execução, este atuador remonta a tripla e então verifica a existência desta informação. Como o agente possui apenas informações verdadeiras, sabemos que, caso a tripla exista em seus conhecimentos, esta se tratará de uma verdade na visão do agente. Após a busca interna, comparamos o resultado com a resposta dada pelo usuário. Caso haja uma discrepância entre o resultado e a resposta, o agente executa outra busca interna, desta vez com o intuito de retornar qual seria a resposta correta. Ainda, durante a execução do atuador, é enviado para o servidor o resultado da comparação entre a resposta do usuário e a resposta do agente.

Através da implementação deste estudo de caso foi possível abordar de maneira mais prática os conceitos elaborados até então, possuindo como principal elemento a utilização prática do contexto ontológico. Além da utilização do contexto também foi criado um cenário onde a partir de sua utilização se criariam novas possibilidades, como a utilização das informações obtidas para a construção de outras aplicações como por exemplo um quiz.

5. Conclusão e Trabalhos Futuros

Como fruto deste período de pesquisa e desenvolvimento foi possível concluir com sucesso as metas fixadas no início deste projeto. Ao fim desse período conseguiu-se obter como principais resultados um modelo de agente funcional que possui a capacidade de obter informações a partir de ontologias e também uma aplicação capaz de utilizar este modelo de forma integrada à aplicação. Através destas aplicações

desenvolvidas foi possível também explorar o potencial das regras de ponte criadas através do Sigon. Outro ponto importante em relação à utilização do Sigon seria a utilidade atribuída aos sensores e atuadores no estudo de caso. A partir destes resultados obtivemos não apenas um novo contexto para ser utilizado juntamente ao Sigon, mas também expandimos o campo de possibilidades deste modelo ao adicionarmos uma funcionalidade capaz de trabalhar de maneira dinâmica.

Ao longo do desenvolvimento foram encontradas situações as quais deixam espaço para futuros trabalhos, por este trabalho lidar com informações obtidas da internet é comum que muitos textos possuam acentos ou caracteres especiais. No entanto hoje em dia o Sigon não possui suporte para este tipo de caracteres em sua sintaxe, assim como suporte para Strings em geral. Para que fosse possível superar esta dificuldade o contexto ontológico executa uma formatação em seus conteúdos para que fiquem de acordo com o permitido na sintaxe, porém ao executar essa formatação existe a possibilidade de não ser encontrado nenhuma referência para o conteúdo formatado na Dbpedia.

Um segundo ponto seria em relação a continuidade da busca por conteúdo. Na implementação atual, não existe um verificação interna do contexto de forma que perceba-se o momento em que não existem novas informações sendo obtidas. O algoritmo foca em guardar o conhecimento obtido e realimentar seu próprio ciclo, a partir dos recursos encontrados, porém existe a possibilidade de não haverem mais novos recursos ou, em alguns casos não podem ser obtidos devido a formatação executada no conteúdo.

Atualmente não podemos afirmar que ocorrem falhas na execução de uma intenção, caso sua busca retorne nenhum resultado, afinal a pesquisa foi concluida. Porem uma alternativa, para uma implementação futura, seria a criação de planos de contingência nestes casos. Desta forma se evita que não haja obtenção de nenhuma informação a partir do elemento pesquisado.

Até o momento o algoritmo criado utiliza os ciclos do Sigon para criar os ciclos de busca. No entanto para futuros trabalhos algumas abordagens alternativas são interessantes. Como por exemplo, ciclos de busca configuráveis para a obtenção de um determinado número de níveis de relacionamento por busca.

6. Referências

ARENAS, M.; PÉREZ, J. Querying semantic web data with sparql.

In: Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. New York, NY,USA: ACM, 2011. (PODS ’11), p. 305–316. ISBN 978-1-4503-0660-7.<http://doi.acm.org/10.1145/1989284.1989312>.

BECHHOFER, S. et al.OWL Web Ontology Language Reference.2004.<http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web.Scientific American, v. 284, p. 34–43, 2001.

BIZER, C.; HEATH, T.; BERNERS-LEE, T. Linked data - the story so far. International Journal on Semantic Web and Information Systems, v. 5, n. 3, p. 1–22, 2009.<https://eprints.soton.ac.uk/271285/>.

BIZER, C. et al. Linked data on the web (ldow2008). Proceedings of the 17th international conference on World Wide Web (WWW ’08), 01 2008.

BORDINI, R. H.; HüBNER, J. F.; WOOLDRIDGE, M. Programming Multi–Agent Systems in AgentSpeak using Jason. [S.l.]: Wiley-Blackwell, 2007. 292 p.

BRATMAN, M. Intention, plans, and practical reason. Harvard University Press, 1987.

CAMPOS, D. de. Representação de dados semânticos em agentes bdi. 2014. 72 f. Dissertação (Mestrado) - Curso de Programa de Pós-graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2014.

CASALI, A.On Intentional and Social Agents with Graded Attitudes. [S.l.]: Universitat de Girona, 2008. 217 p. ISBN 978-8-4692-1448-0.

CASALI, A.; GODO, L.; SIERRA, C. Graded bdi models for agent architectures. International Journal of Human and Computer Studies, Springer, Berlin, Heidelberg, v. 3487, p. 126–143, 2005.

DEMARCHI, F. A integração de agentes com bases ontológicas heterogêneas aplicadas a web semântica. Florianópolis, 2017.

DIKENELLI, O.; ERDUR, R. C.; GUMUS, O. Seagent:a platform for developing semantic web based multi agent systems. In: PECHOUCEK, M.; STEINER, D.; THOMPSON, S. (Ed.). Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems . [S.l.]: Acm, 2005. p. 1271–1272. ISBN 1-59593-093-0.

FREITAS, A. et al. Integrating ontologies with multi-agent systems through cartago artifacts. In: 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology. Florianópolis: Ieee, 2015. p. 143 – 150. ISBN 978-1-4673-9618-9.

FREITAS, G. S. de. Um modelo de raciocínio sobre sensores para agentes sigon em ambientes de realidade virtual. Florianópolis, 2018.

GELAIM, T. Ângelo et al. Sigon: A multi-context system framework for intelligent agents. Expert Systems with Applications, v. 119, p.51–60, 2019.

GUARINO, N. Formal ontology, conceptual analysis and knowledge representation.

International Journal of Human and Computer Studies, v. 43, p. 625–640, 1995.

HERZIG, A. et al. Bdi logics for bdi architectures: Old problems, new perspectives. KI - Künstliche Intelligenz, Springer Berlin Heidelberg, v. 31, p. 73–83, 2017.

HüBNER, J. F.; BORDINI, R. H.; VIEIRA, R. Introdução ao desenvolvimento de sistemas multiagentes com jason. 2014.

JENSEN, R.; BUCH, B. Multi-Agent Programming in Jason. Richard Petersens Plads, Building 324, DK-2800 Kgs. Lyngby, Denmark, compute@compute.dtu.dk: Technical University of Denmark, Department of Applied Mathematics and Computer Science, 2014. DTU supervisor: Jørgen Villadsen, jovi@dtu.dk, DTU Compute. <http://www.compute.dtu.dk/English.aspx>.

KLAPISCAK, T.; BORDINI, R. H. Jasdl: A practical programming approach combining agent and semantic web technologies. In: BALDONI, M. et al. (Ed.). Declarative Agent Languages and Technologies VI . Berlin: Springer, 2008. p. 91–110. ISBN 978-3-540-93919-1.

MASCARDI, V. et al. Cool-agentspeak: Enhancing agentspeak-dl agents with plan exchange and ontology services. In: WI-IAT ’11Proceedings of the 2011

IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 01. [S.l.]: IEEE, 2011. p. 109–116. ISBN 978-0-7695-4513-4.

MELLO, R. R. P. de; GELAIM, T. Â.; SILVEIRA, R. A. Negotiating agents: A model based on bdi architecture and multi-context systems using aspiration adaptation theory as a negotiation strategy. In: BAROLLI, L. et al. (Ed.). Complex, Intelligent, and Software Intensive Systems. Cham: Springer International Publishing, 2019. p. 351–362. ISBN 978-3-319-93659-8.

MOREIRA Álvaro F. et al. Agent-oriented programming with underlying ontological reasoning. In: BALDONI, M. et al. (Ed.). Declarative Agent Languages and Technologies III. Berlin: Springer-verlag, 2006. p. 155–170. ISBN 978-3-540-33106-3.

O'HARE, G. M. P.; JENNINGS, N. R. (Ed.). Foundations of Distributed Artificial Intelligence. New York, NY, USA: John Wiley & Sons, Inc., 1996. ISBN 0-471-006750.

OMEROVIC, S.; MILUTINOVIC, V.; TOMAZIC, S. Concepts, ontologies, and knowledge representation. Slovenian Research Agency, p. 42, 2001.

OSSENBRUGGEN, J. van; HARDMAN, L.; RUTLEDGE, L. Hypermedia and the semantic web: A research agenda. Journal of Digital Information, v. 3, n. 1, 2006. ISSN 1368-7506.<https://journals.tdl.org/jodi/index.php/jodi/article/view/78>.

PARSONS, S. et al. Agent specification using multi-context systems. In: Lecture Notes in Computer Science. [S.l.]: Springer, Berlin,Heidelberg, 2002. p. 205–226.

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.

RUSSELL, S. J.; NORVIG, P. Artificial Intelligence a Modern Approach. New Jersey: Alan Apt, 1995. 932 p.

STAAB, S.; STUDER, R. (Ed.).Handbook on Ontologies.[S.l.]: Springer-Verlag Berlin Heidelberg, 2009. 811 p. ISBN 978-3-540-92673-3.

W3C. OWL Web Ontology Language Overview. 2004.<https://www.w3.org/TR/2004/REC-owl-features-20040210/s1.2>.

W3C.SPARQL Query Language for RDF. 2004.<https://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/conventions>.

W3C.OWL 2 Web Ontology Language Document Overview. 2012.<https://www.w3.org/TR/owl2-overview/>.

WOOLDRIDGE, M.Reasoning about Rational Agents. London: The MIT Press, 2000.

.2 APÊNDICE - CÓDIGO DO CONTEXTO ONTOLÓGICO

```
package br.ufsc.ine.agent.context.ontologic;

import java.text.Normalizer;
import java.util.ArrayList;
import java.util.List;

import alice.tuprolog.InvalidTheoryException;
import alice.tuprolog.MalformedGoalException;
import alice.tuprolog.NoSolutionException;
import alice.tuprolog.SolveInfo;
import alice.tuprolog.Theory;
import br.ufsc.ine.agent.Agent;
import br.ufsc.ine.agent.context.ContextService;
import br.ufsc.ine.agent.context.LangContext;
import
    br.ufsc.ine.agent.context.ontologic.semanticWeb.SparqlObject;
import
    br.ufsc.ine.agent.context.ontologic.semanticWeb.SparqlResult;
import
    br.ufsc.ine.agent.context.ontologic.semanticWeb.SparqlSearch;
import br.ufsc.ine.utils.PrologEnvironment;

public class OntologicContextService implements ContextService{

   private static OntologicContextService instance = new
       OntologicContextService();
   private static PrologEnvironment prologEnvironment;

   SparqlSearch sparqlSearch;
   List<SparqlObject> sparqlObjects;
   List<String> mappedPredicates;

   public OntologicContextService() {
      prologEnvironment = new PrologEnvironment();
      sparqlSearch = new SparqlSearch();
      sparqlObjects = new ArrayList<SparqlObject>();
      initMappedPredicates();
   }

   @Override
```

```
public Theory getTheory() {
   return prologEnvironment.getEngine().getTheory();
}

@Override
public boolean verify(String fact) {
   SolveInfo solveGoal;
   try {
      solveGoal = prologEnvironment.solveGoal(fact);
      return solveGoal.isSuccess();
   } catch (MalformedGoalException e) {
      System.out.println(fact+" malformed");
      return false;
   }
}

public void ontologic(List<LangContext> langContexts) {
   langContexts.forEach(ctx -> {
      ctx.getClauses().forEach(clause -> {
         try {
            this.addInitialFact(clause);
         } catch (InvalidTheoryException e) {
            e.printStackTrace();
         }
      });
   });
}

@Override
public void appendFact(String fact) {
   String subject = "";
   String content = getContent(fact);
   if(!checkedResource(content)){
      fact = stringToInputFormat(content);
      subject = "http://dbpedia.org/resource/"+fact;
      removeNewResourceStatus(content);
      for (String predicate : mappedPredicates) {
         List<SparqlResult> result =
              executeQuery("<"+subject+">", predicate,
              "?value", getFilter(predicate));
         for (SparqlResult sr : result){
            String newFact =
                "knowledge("+formatPredicate(predicate)+","
               +stringToOutputFormat(fact)+","+getObject(sr)+").";
            apprendToProlog(newFact);
```

```
        }
      }
   }
}

public String getFilter(String predicate){
   return predicate == "rdf:type" ? "filter
       contains(str(?value),'http://dbpedia.org/ontology/')"
       : null;
}

public String getContent(String fact){
   return fact.substring(fact.indexOf("(") + 1,
       fact.lastIndexOf(")"));
}

public String getObject(SparqlResult result){
   String object = "";
   if(result.getResourceResult()!=null){
      object = stringToOutputFormat(getResourceLabel
          (result.getResourceResult().getURI()));
      appendNewResource(object);
   }
   if(result.getLiteralResult()!=null){
      object = result.getLiteralResult().getString();
   }
   return stringToOutputFormat(object);
}

public void appendNewResource(String resource){
   if(!checkedResource(resource)){
      apprendToProlog("newResource("+resource+").");
   }
}

public void apprendToProlog(String newFact){
   boolean update = false;
   try {
      update = this.verify(newFact);
      if (update) {
         prologEnvironment.updateFact(newFact, newFact);
      } else {
         prologEnvironment.appendFact(newFact);
      }
   } catch (InvalidTheoryException e) {
```

```
      e.printStackTrace();
   }
}

public void removeNewResourceStatus(String c){
   String removedFact = "newResource("+c+")";
   if(prologEnvironment.getEngine().getTheory()
       .toString().contains(removedFact)) {
      Agent.removeBelief = true;
   }
   if (Agent.removeBelief) {
      Agent.removeBelief = false;
      try {
         prologEnvironment.removeFact(removedFact);
         return;
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}

public boolean checkedResource(String resouce){
   Theory theory = prologEnvironment.getEngine().getTheory();
   boolean contains =
       theory.toString().contains("type,"+resouce+",");
   return contains;
}

public String stringToOutputFormat(String s) {
   s = firstCharLowerCase(s);
    s = Normalizer.normalize(s, Normalizer.Form.NFD);
    s = s.replaceAll("[\\p{InCombiningDiacriticalMarks}]",
        "");
    s = s.replaceAll("\\s","_");
    return s;
}

public String stringToInputFormat(String s) {
   s = firstCharUpperCase(s);
    return s;
}

public List<SparqlResult> getWrongAnswer(){
   return null;
}
```

```
public List<SparqlResult> searchNewURI(List<SparqlObject>
    objects){
   List<SparqlResult> result =
       sparqlSearch.searchDbpedia(objects);
   clearObjectsList();
   return result;
}

public List<SparqlResult> executeQuery(String subject,String
    predicate, String object, String filter){
   addObjectToList(subject, predicate, object, filter);
   return searchNewURI(getSparqlObjects());
}


public String getURI(String label){
   List<SparqlResult> result = executeQuery("?uri",
       "rdfs:label", "'"+label+"'@en",null);
   if (!result.isEmpty()) {
      for (SparqlResult sparqlResult : result) {
         if(!sparqlResult.getResourceResult().getURI()
             .contains("wikidata")){
            return sparqlResult.getResourceResult().getURI();
         }
      }
   }
   return "";
}

public List<SparqlResult> filterDbpedia(List<SparqlResult>
    list){
   List<SparqlResult> newList = new ArrayList<SparqlResult>();
   for (SparqlResult sparqlResult : list) {
      String uri = sparqlResult.getResourceResult().getURI();
      if(uri.contains("dbpedia") && !uri.contains("Wiki") &&
          !uri.contains("Wiki")){
         newList.add(sparqlResult);
      }
   }
   return newList;
}

public String firstCharUpperCase(String label){
```

```
    return label.substring(0,1).toUpperCase() +
        label.substring(1);
}

public String firstCharLowerCase(String label){
    return label.substring(0,1).toLowerCase() +
        label.substring(1);
}

public String formatPredicate(String predicate){
    return predicate.substring(predicate.lastIndexOf(":") + 1);
}

public String getResourceLabel(String subject){
    return subject.substring(subject.lastIndexOf("/")+1);
}

@Override
public void addInitialFact(String fact) throws
     InvalidTheoryException {
    prologEnvironment.appendFact(fact);
}

@Override
public String getName() {
    return "ontologic";
}

public void initMappedPredicates(){
    mappedPredicates = new ArrayList<String>();

    //Where
    mappedPredicates.add("dbo:country");
    mappedPredicates.add("dbo:isPartOf");
    mappedPredicates.add("dbo:birthPlace");
    mappedPredicates.add("geo:lat");
    mappedPredicates.add("geo:long");

    //When
    mappedPredicates.add("dbo:foundingDate");

    //Who
    mappedPredicates.add("dbo:leaderName");

    //What
```

```
        mappedPredicates.add("foaf:nick");

        mappedPredicates.add("dbo:capital");
        mappedPredicates.add("dbp:capital");

        mappedPredicates.add("dbo:currency");
        mappedPredicates.add("dbo:officialLanguage");
        mappedPredicates.add("dbo:largestCity");
        mappedPredicates.add("rdf:type");

        //How Many
        mappedPredicates.add("dbo:populationalTotal");
        mappedPredicates.add("dbo:populationTotal");
    }

    public void clearObjectsList(){
        sparqlObjects.clear();
    }

    public void addObjectToList(String subject,String predicate,
         String object, String filter){
        SparqlObject triple = new SparqlObject();
        triple.setUri(subject);
        triple.setPredicate(predicate);
        triple.setObject(object);
        triple.setFilter(filter);
        sparqlObjects.add(triple);
    }

    public static OntologicContextService getInstance() {
        return instance;
    }

    public static void setInstance(OntologicContextService
         instance) {
        OntologicContextService.instance = instance;
    }

    public SparqlSearch getSparqlSearch() {
        return sparqlSearch;
    }

    public void setSparqlSearch(SparqlSearch sparqlSearch) {
        this.sparqlSearch = sparqlSearch;
    }
```

```
    public List<SparqlObject> getSparqlObjects() {
        return sparqlObjects;
    }

    public void setSparqlObjects(List<SparqlObject>
         sparqlObjects) {
        this.sparqlObjects = sparqlObjects;
    }

    public List<String> getMappedPredicates() {
        return mappedPredicates;
    }

    public void setMappedPredicates(List<String>
         mappedPredicates) {
        this.mappedPredicates = mappedPredicates;
    }

    public static PrologEnvironment getPrologEnvironment() {
        return prologEnvironment;
    }

    public static void setPrologEnvironment(PrologEnvironment
         prologEnvironment) {
        OntologicContextService.prologEnvironment =
            prologEnvironment;
    }

}
```

.3 APÊNDICE - AGENTE SIGON UTILIZADO NO MODELO PROPOSTO

```
communication:
    sensor("answerSensor",
         "ontologicExperiment.sensors.AnswerSensor").

beliefs:
```

```
desires:

intentions:
    questionAbout(malta).

! intentions questionAbout(X) :- _ontologic newResource(X).

_ontologic:

! _ontologic question(X) :- intentions questionAbout(X).
```

.4 APÊNDICE - AGENTE SIGON UTILIZADO NO ESTUDO DE CASO

```
communication:
    sensor("answerSensor",
        "ontologicExperiment.sensors.AnswerSensor").
    actuator("buildQuestion",
        "ontologicExperiment.actuators.BuildQuestion").
    actuator("validateAnswer",
        "ontologicExperiment.actuators.ValidateAnswer").

beliefs:

desires:

intentions:
    questionAbout(malta).

! intentions questionAbout(X) :- _ontologic newResource(X) &
     communication sense(I).

_ontologic:

! _ontologic question(X) :- intentions questionAbout(X) &
     communication sense(I).

! communication buildQuestion(capital,Y,Z) :- _ontologic
     knowledge(capital,Y,Z) & communication sense(I).
```

```
! communication validateAnswer(X,Y,Z,A) :- communication
    sense(X,Y,Z,A).
```